Galiano *et al. EURASIP Journal on Advances in Signal Processing* 2013, **2013**:84
http://asp.eurasipjournals.com/content/2013/1/84

EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

Multicore-based 3D-DWT video encoder

Vicente Galiano, Otoniel López-Granado, Manuel P Malumbres and Hector Migallón*

Abstract

Three-dimensional wavelet transform (3D-DWT) encoders are good candidates for applications like professional video editing, video surveillance, multi-spectral satellite imaging, etc. where a frame must be reconstructed as quickly as possible. In this paper, we present a new 3D-DWT video encoder based on a fast run-length coding engine. Furthermore, we present several multicore optimizations to speed-up the 3D-DWT computation. An exhaustive evaluation of the proposed encoder (3D-GOP-RL) has been performed, and we have compared the evaluation results with other video encoders in terms of rate/distortion (R/D), coding/decoding delay, and memory consumption. Results show that the proposed encoder obtains good R/D results for high-resolution video sequences with nearly in-place computation using only the memory needed to store a group of pictures. After applying the multicore optimization strategies over the 3D DWT, the proposed encoder is able to compress a full high-definition video sequence in real-time.

Keywords: 3D-DWT, Video coding, Multicore, Wavelets, Performance

1 Introduction

Currently, most of the popular video compression technologies operate in both intra and inter coding modes. Intra mode compression operates in a frame-by-frame basis while inter mode achieves compression by applying motion estimation and compensation between frames and taking advantage of the temporal correlation between frames. Inter mode compression is able to achieve increased coding efficiency over intra mode schemes. However, in video content production stages, digital video-processing applications require fast-frame random access to perform an undefined number of real-time decompressing-editing-compressing interactive operations, without a significant loss of original video content quality. Intra-frame coding is desirable as well in many other applications like video archiving, high-quality high-resolution medical and satellite video sequences, applications requiring simple real-time encoding like video-conference systems or even for professional or home video surveillance systems [1], and digital video recording systems, where the user equipment is usually not as powerful as the head end equipment.

There is another video encoding approach that may be also considered as an inter coding approach but without the use of motion estimation/compensation. In this approach, known as three-dimensional (3D) coding, a video sequence is considered as a three-dimensional data set where each pixel has two spatial and one temporal coordinates. Most of the 3D encoders proposed in the literature are based on the three-dimensional wavelet transform (3D-DWT), mainly used in watermarking [2] and video coding applications (e.g., compression of volumetric medical data [3], multispectral images [4], or 3D model coding [5]). So, 3D-DWT-based encoders could be an intermediate approximation between intra and inter coding modes, because it avoids motion estimation and compensation, and the decoding latency will depend on the GOP size.

For example, Taubman and Zakhor presented a full-color video coder based on a 3D subband coding with camera pan compensation [6]. Podilchuk, et al. utilized a 3D spatio-temporal subband decomposition and geometric vector quantization [7]. Chen and Pearlman [8] extended to 3D improved embedded zero-tree wavelet (IEZW) for video coding the two-dimensional (2D) embedded zero-tree wavelet (EZW) method [9] and showed promise of an effective and computationally simple video coding system without motion compensation, obtaining excellent numerical and visual results. In [10], instead of the typical quad-trees of image coding, a tree with eight descendants per coefficient is used to extend

*Correspondence: hmigallon@umh.es
Physics and Computer Architecture Department Miguel Hernández University, Elche, 03202, Spain

the set partitioning in hierarchical trees (SPIHT) image encoder to 3D video coding. In [11], a fast SPIHT version is presented using a Huffman-based entropy encoder instead of a context-adaptive arithmetic encoder. However, the proposed image encoder has not been extended to the 3D version. Also in [12], an extension of the fast backward coding of wavelet trees (BCWT) image encoder [13] is presented, reporting a coding speed of 32 frames per second for a common intermediate format (CIF) resolution video sequence. The BCWT image encoder offers high coding speed, low memory usage, and a similar rate/distortion (R/D) performance than the SPIHT encoder. The key of the BCWT encoder is its unique one-pass backward coding, which starts from the lowest level of subbands and travels backwards. Maximum quantization levels of descendants (MQD) map calculation and coefficient encoding are all carefully integrated inside this pass in such a way that there is as little redundancy as possible for computation and memory usage. A 3D zero-tree coding through modified EZW has also been used with good results in compression of volumetric images [14].

In this work, we present a fast 3D-DWT-based encoder with a run-length core coding system. The proposed encoder requires less memory than 3D SPIHT [10] and has a good R/D behavior. Furthermore, we present an in-depth analysis of the use of multicore strategies to accelerate the 3D-DWT. Using these strategies, the proposed encoder is able to compress a full high-definition (HD) video sequence in real-time.

The rest of the paper is organized as follows: section 2 presents the proposed 3D-DWT-based encoder. In section 3, a performance evaluation in terms of R/D, memory requirements, and coding time is presented. Section 4 describes several optimization proposals based on multicore processing strategies applied to the 3D-DWT computation while in section 4.2, we analyze their performance. Furthermore, in section 4.3, we present a pipeline strategy to speed up the proposed encoder. Finally, in section 5, we show the performance of the improved proposed encoder against other state-of-the-art encoders while in section 6, some conclusions are drawn.

2 Encoding system

In this section, we present a 3D-DWT-based encoder with low complexity and good R/D performance. As our main concern is fast encoding process, no R/D optimization, motion estimation/motion compensation (ME/MC) or bitplane processing is applied. This encoder is based on both 3D-DWT and run-length encoding (3D-GOP-RL), and it is able to compress an ITU-D1 (576p30) video sequence at 40 frames per second.

In Figure 1, the whole encoding system scheme is shown. First of all, the 3D-DWT is applied to a group of pictures (GOP) in such a way that a combination of a 2D spatial DWT and a 1D temporal DWT is applied and the temporal DWT absorbs motion in the GOP. The temporal DWT is carried out on the pixel values of the same location along the time axis. Our 3D-DWT implementation, as how 3D-SPIHT and 3D-BCWT are done, uses the Daubechies 9/7F filter for both spatial and temporal domains because this filter has shown good results for lossy compression [15].

After that, all wavelet coefficients are quantized, and then, subband frames are passed from the lowest frequency subband LLL_n to the highest frequency subband HHH_1 to the run-length encoding system which compresses the input data, and we obtain the final bit-stream corresponding to that GOP. As in the 3D-BCWT encoder [12], only one pass is applied over the GOP to encode the coefficients, but contrary to the 3D-BCWT encoder,

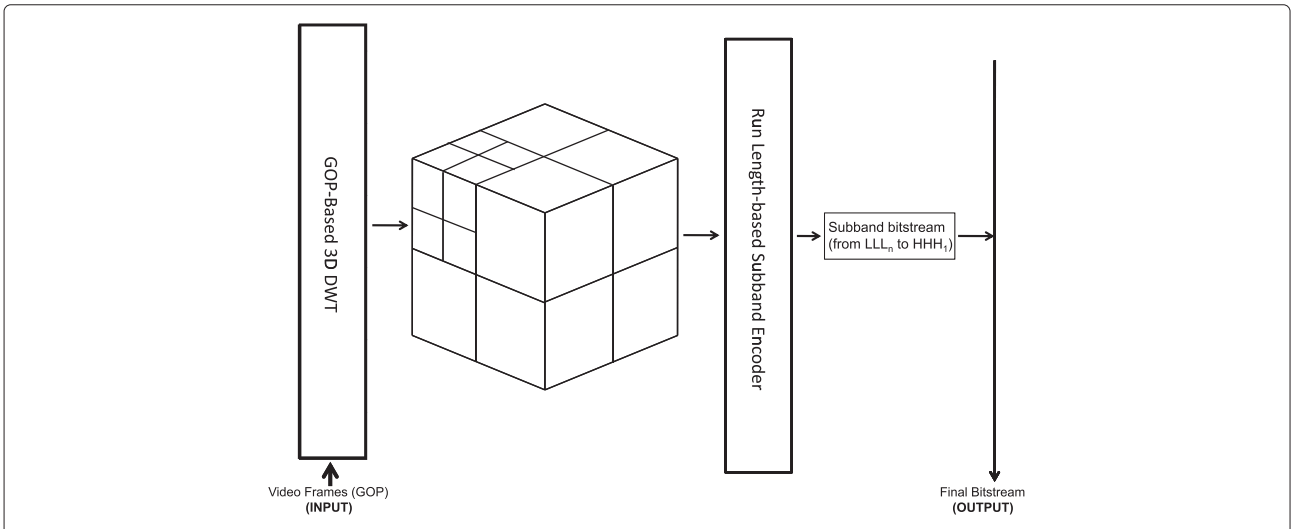


Figure 1 Overview of the proposed run length-based encoder.

Table 1 Configuration parameters of the evaluated encoders

Parameters/ Codec	GOP size	Sequence type	Profile
3D-SPIHT [16]	16	I	-
H.264 (JM16.1 version) [17]	15	IBBPBBP…	High profile
H.263 [18] (ffmpeg-r25117)	15	IPPPPP…	Profile 0
	(No B frames supported in this version)		
MPEG-2 (ffmpeg-r25117)	15	IBBPBBP…	Main profile
MPEG-4 part 2 (ffmpeg-r25117)	15	IBBPBBP…	Simple advanced profile
×**264** (mingw32-lib×x264 r1713–1) [19]	15	IBBPBBP…	High quality preset
×**264 Intra** (mingw32-lib×264 r1713–1) [19]	-	IIIII…	High quality preset

the compressed bit-stream generated by our encoder is ordered in such a way that the decoder obtains the bit-stream in the correct order.

2.1 Fast run-length coding

In the proposed encoder, the quantization process is performed by two strategies: one coarser and another finer. The finer one is done by applying a scalar uniform quantization to the wavelet coefficients using the Q parameter. The coarser one is done by removing bit planes from the least significant part of the wavelet coefficients. We define *rplanes* as the number of less significant bits to be removed, and we call significant coefficient to those coefficients $c_{i,j}$ that are different to zero after discarding the least significant *rplane* bits, in other words, if $c_{i,j} \geq 2^{rplanes}$.

In the proposed coding algorithm, the wavelet coefficients are encoded as follows: the quantized coefficients in the subband buffer are scanned row by row (to exploit their locality). For each coefficient in that buffer, if it is not significant, a run-length count of insignificant symbols at this level is increased (run_length_L). However, if it is significant, we encode both the count of previous insignificant symbols and the significant coefficient, and run_length_L is reset.

A significant coefficient is encoded by means of a symbol indicating the number of bits required to represent that coefficient. An arithmetic encoder with two contexts is used to efficiently store that symbol. As coefficients in the same subband have similar magnitude, an adaptive arithmetic encoder is able to represent this information in a very efficient way. After that, the significant bits and sign of the wavelet coefficient are raw-encoded to speed up the execution time.

In order to encode the count of insignificant symbols, we use a *RUN* symbol. After encoding this symbol, the run-length count (run_length_L) is stored in a similar way as in the case of significant coefficients. First, the number of bits needed to encode the run value is arithmetically encoded (with a different context). Afterwards, the bits are raw-encoded.

Instead of using run-length count symbols, we could have used a single symbol to encode each insignificant coefficient. However, we would need to encode a larger amount of symbols, and therefore, the complexity of the algorithm would increase (most of all, in the case of a large number of insignificant contiguous symbols, which usually occurs in moderate-to-high compression ratios). However, the compression performance is increased if a specific symbol is used for every insignificant coefficient since an arithmetic encoder processes more efficiently many likely symbols than a lower amount of less likely symbols. So, for short run-lengths, we encode a *LOWER* symbol for each insignificant coefficient instead of coding a run-length count symbol for all the sequence. The threshold to enter the run-length mode and start using run-length count symbols is defined by the *enter_run_mode* parameter. The formal description of the depicted algorithm can be found in Algorithm 1.

Algorithm 1. Run-length coding of the wavelet coefficients

function *RLW_Code_Subband(Buffer, L)*
  Scan Buffer in horizontal raster order
  for each $C_{i,j}$ *in Buffer*
    $nbits_{i,j} = \lceil log_2 \left(\left| C_{i,j} \right| \right) \rceil$
  if $nbits_{i,j} \leq rplanes$
    increase run_length_L
  else
    if $run\_length_L \leq enter\_run\_mode$
      repeat run_length_L **times**
        arithmetic_output *LOWER*
    else
      arithmetic_output *RUN*
      $rbits = \lceil log_2 \left(run\_length_L \right) \rceil$
      arithmetic_output *rbits*
    output $bit_{nbits_{(i,j)}-1} \left(\left| C_{i,j} \right| \right) \ldots bit_{rplane+1} \left(\left| C_{i,j} \right| \right)$
    output $\text{sign}(c_{i,j})$
end of function

Note: $bit_n(C)$ *is a function that returns the* n^{th} *bit of* C

Table 2 Memory requirements for evaluated encoders (kB)

Format/ Codec	QCIF	CIF	ITU-D1	Full-HD
H.264	35,824	86,272	227,620	489,960
×264	10,752	18,076	36,600	178,940
MPEG-2	4,696	6,620	**9,164**	32,820
MPEG-4	5,160	6,868	9,324	**31,192**
3D-GOP-RL	**1,611**	**6,390**	20,576	123,072
3D-SPIHT	10,152	34,504	118,460	645,720

3 Performance evaluation

In this section, we will compare the performance of our proposed encoder (3D-GOP-RL) using the Daubechies 9/7F filter for both spatial and temporal domains and a GOP size of 16 with the video encoders presented in Table 1.

The performance metrics employed in the tests are R/D performance, coding and decoding delay, and memory requirements. All the evaluated encoders have been tested on an Intel PentiumM Dual Core 3.0 GHz processor (Santa Clara, CA, USA) with a 2-Gbyte RAM memory.

The test video sequences used in the evaluation are the Foreman (QCIF and CIF) 300 frames, container (QCIF

Table 3 Average peak signal-to-noise ratio (PSNR) (dB) with different bit rate and coders

Codec/Bit-rate Kbps/dB	H.264	x264	x264 Intra	MPEG-2	MPEG-4	H.263	3D SPIHT	3D GOP-RL
				Foreman (CIF)				
3,040	45.46	45.32	39.95	40.74	41.38	40.41	40.32	41.05
1,520	42.28	41.74	35.29	37.10	37.90	36.38	36.42	36.48
760	39.75	38.61	31.43	34.09	35.15	35.15	33.35	33.01
380	36.85	35.29	28.15	31.59	32.81	29.86	30.78	30.41
190	34.14	31.75	25.07	29.32	30.53	28.45	28.53	28.36
				Container (CIF)				
3,040	47.64	47.16	37.97	43.59	42.70	40.41	47.82	45.88
1,520	43.69	43.36	33.04	40.43	41.41	36.38	43.99	40.57
760	42.00	39.85	29.22	37.19	38.44	35.15	39.54	35.54
380	38.46	36.38	25.88	34.48	36.01	29.86	35.20	31.66
190	35.40	33.00	23.27	32.05	33.85	28.45	31.10	28.75
				Hall (CIF)				
3,040	45.76	44.38	41.19	42.29	42.77	42.56	44.68	44.49
1,520	42.68	41.17	36.60	39.89	40.71	40.24	42.27	41.03
760	40.05	39.09	31.89	37.95	38.92	37.58	40.11	37.51
380	38.55	37.12	27.32	35.95	37.21	32.62	37.39	33.57
190	35.84	34.38	23.88	33.59	35.43	30.04	33.56	30.22
				Mobile (ITU-D1)				
6,400	41.86	40.26	35.56	37.82	38.66	38.05	38.24	36.32
3,598	40.66	38.62	32.53	36.09	37.11	36.10	35.07	33.85
2,100	38.71	37.26	30.12	34.37	35.84	34.55	32.53	32.22
1,142	36.90	35.13	27.87	32.58	34.46	32.63	30.52	30.44
542	35.34	31.57	25.65	30.68	32.16	30.00	28.82	28.74
				Ducks (Full-HD) 50 fps				
98,304	37.77	36.82	36.26	**38.49**	35.67	35.49	37.77	38.08
49,152	34.74	34.02	32.62	35.27	32.46	32.20	35.39	34.74
24,576	33.00	32.01	29.16	32.28	30.55	29.04	33.68	32.69
12,288	31.24	29.86	26.43	29.32	27.64	27.39	31.63	30.69
6,144	29.00	27.71	24.19	27.82	27.11	27.10	28.99	29.09

and CIF) 300 frames, news (QCIF and CIF) 300 frames, hall (QCIF and CIF) 300 frames, mobile (ITU D1 576p30) 40 frames, station2 (HD 1024p25) 312 frames, Ducks (HD 1024p50) 130 frames, and Ducks (SHD 2048p50) 130 frames.

It is important to remark that the H.263, MPEG-2, MPEG-4, and ×264 are evaluated by implementations that are fully optimized, using CPU capabilities like multimedia extensions (MMX2, SSE2Fast, SSSE3, etc.) and multithreading, whereas 3D-SPIHT and 3D-GOP-RL had non-optimized C++ implementations.

3.1 Memory requirements

In Table 2, the memory requirements of different encoders under test are shown. Obviously, the H.263 encoder, only using P frames, requires to keep in memory just two frames to accomplish the ME/MC stage, whereas encoders based on 3D-DWT like 3D-SPIHT and 3D-GOP-RL need to keep more frames in memory to apply the time filter. The 3D-GOP-RL encoder running over a GOP size of 16 frames uses up to 6 times less memory than 3D-SPIHT, up to 22 times less memory than H.264 for QCIF sequence resolution, and up to 6 times less memory than ×264 which is an optimized implementation of H.264, for small sequence resolutions. It is important to remark that 3D-SPIHT keeps the compressed bit-stream of a 16-GOP size in memory until the whole compression is performed, while encoders like MPEG-2, MPEG-4, H.263, H.264, 3D-GOP-RL, and ×264 output the bit-stream inline. Block-based encoders like MPEG-2 and MPEG-4 require less memory than the other encoders, specially at high-definition sequences. Also, the memory requirements in the proposed encoder (3D-GOP-RL) are doubled as the GOP size is doubled.

3.2 R/D performance

Regarding R/D, in Table 3, we can see the R/D behavior of all evaluated encoders for different sequences. As shown, both H.264 and ×264 are the ones that obtain the best results for sequences with high movement, mainly due to the exhaustive ME/MC stage included in these encoders, which is contrary to 3D-SPIHT and 3D-GOP-RL that do not include any ME/MC stage. The R/D behavior of 3D-SPIHT and 3D-GOP-RL is similar for images with moderate-high motion activity, but for sequences with low movement, 3D-SPIHT outperform 3D-GOP-RL, showing the power of its tree encoding system. The proposed encoder (3D-GOP-RL) has a similar behavior to H.263 and MPEG-2 and a slightly lower performance than MPEG-4. Also, we can see the improvement of 3D-GOP-RL and 3D-SPIHT when compared to ×264 in intra mode (up to 11 dB). This R/D improvement is accomplished by exploiting only the temporal redundancy among video frames when applying the 3D-DWT. It is also interesting that the behavior of the 3D-DWT-based encoder for high frame rate video sequences like Ducks. As it can be seen, all 3D-DWT-based encoders have a similar behavior than the other encoders, even better than ×264.

3.3 Encoding time

In Figure 2, we present the coding speed (excluding I/O) of all evaluated encoders and for different sequence resolutions. As it can be seen, MPEG-2 and MPEG-4 encoders are the fastest ones due to their block-based processing algorithm. Regarding 3D-DWT-based encoders, the proposed encoder 3D-GOP-RL is up to seven times as fast as 3D-SPIHT and up to six times as fast as the ×264 encoder.

Also, in Figure 3a, we present the total coding time of a frame for different video sequence resolutions as a

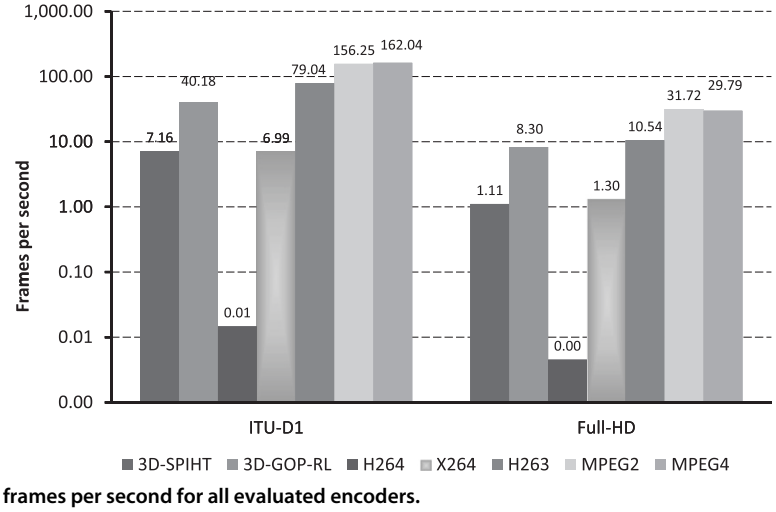
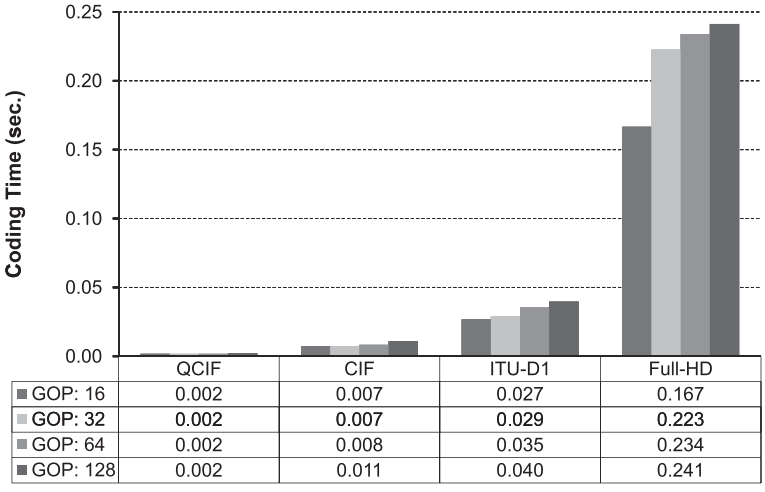
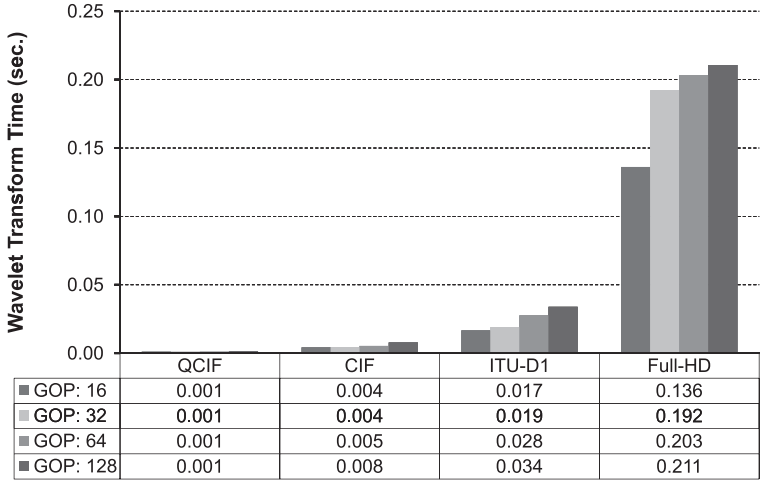


Figure 2 Coding speed in frames per second for all evaluated encoders.



	QCIF	CIF	ITU-D1	Full-HD
GOP: 16	0.002	0.007	0.027	0.167
GOP: 32	0.002	0.007	0.029	0.223
GOP: 64	0.002	0.008	0.035	0.234
GOP: 128	0.002	0.011	0.040	0.241

(a) Total coding time



	QCIF	CIF	ITU-D1	Full-HD
GOP: 16	0.001	0.004	0.017	0.136
GOP: 32	0.001	0.004	0.019	0.192
GOP: 64	0.001	0.005	0.028	0.203
GOP: 128	0.001	0.008	0.034	0.211

(b) Wavelet time

Figure 3 Total coding and wavelet transform times of the 3D-GOP-RL encoder for different video sequence resolutions. (**a**) Total coding time and (**b**) wavelet time.

function of the GOP size. As it can be seen, for low resolution sequences, there are nearly no differences in the total coding time, but for high-resolution video sequences, the total coding time will increase up to 40% as the GOP size increases. Furthermore, it is interesting to see that the required time to perform the 3D-DWT stage ranges between 45% and 80% of the total coding time depending on the GOP size, as seen in Figure 3b. So, improvements in the 3D-DWT computation will drastically reduce the total coding time of the proposed encoder.

4 3D-DWT optimizations

As 3D-DWT computation requires more than 45% and up to 80% of the total coding time in the proposed encoder. In this section, we present several parallel strategies to improve the 3D-DWT computation time.

4.1 Multicore 3D wavelet transform

In the proposed encoder (3D-GOP-RL), the Daubechies 9/7 filter, proposed in [20], has been used to perform the regular filter-bank convolution in order to develop the parallel 3D-DWT algorithm. In [21], we proposed the convolution-based parallel 2D-DWT using an extra memory space in order to perform a nearly in-place computation, avoiding the requirement of twice the image size to store the computed coefficients. This strategy has been also followed to develop the parallel 3D-DWT algorithm.

We want to remark that we use four decomposition levels in order to compute the 3D-DWT, and the computation of each wavelet decomposition level is divided into two main steps: in the first step, the 2D-DWT is applied to each frame of the current GOP, and in the second step, the 1D-DWT is performed to consider the temporal axis.

Table 4 Amount of extra memory size

Frame size	Processes	Extra memory size	Increment (%)
		Pixel size	GOP: 32
352 × 288	1	360	0.0110
	2	720	0.0221
	4	1,440	0.0443
	6	2,160	0.0665
	10	3,600	**0.1109**
1,280 × 640	1	1,288	0.0024
	2	2,576	0.0049
	4	5,152	0.0099
	6	7,728	0.0148
	10	12,880	0.0247
1,920 × 1,024	1	1,928	0.0016
	2	3,856	0.0032
	4	7712	0.0065
	6	11,568	0.0098
	10	**19,280**	0.0164

We have used the symmetric extension technique in order to avoid the border effects on both the frame borders and the GOP borders.

If we consider the first step (i.e., the 2D-DWT applied to each video frame), the extra memory size depends on both the row size or column size (the larger one), and the number of processes in the parallel algorithm. The extra memory stores, the frame row/column pixels, plus the pixels are required to perform the symmetric extension. For the Daubechies 9/7 filter, we must extend the row/column with four elements on both borders.

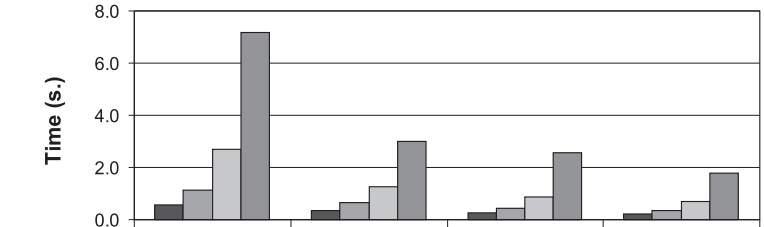
Table 4 shows the extra memory size (in pixels) and the percentage of memory increase for several video frame resolutions and number of processes used in the parallel algorithm. Note that each process stores its own working pixels which are not shared with other processes. The worst case in Table 4, attending to memory increase, is a very small value equal to 0.1109%. If the GOP size is larger than the row or column size, the amount of required extra memory is fixed by the GOP length. Percentage values in Table 4 have been obtained considering a GOP size equal to 32. In the second step of the 3D-DWT (i.e., the temporal 1D-DWT), we perform the symmetric extension in order to avoid the border effects in the temporal domain. In all performed experiments, the maximum GOP size considered is 128; therefore, the extra memory used in the first step is enough to be reused in the second step.

We have used the OpenMP [22] paradigm in order to develop the parallel 3D-DWT algorithm. The multicore platforms used in our tests are as follows:

- Intel Core 2 Quad Q6600 2.4 GHz, with four cores.
- HP Proliant SL390 G7 (HP, Palo Alto, CA, USA) with two Intel Xeon ×5,660, each CPU with six cores at 2.8 GHz.

4.2 Performance evaluation of the multicore 3D-DWT

In this section, we discuss the behavior of the parallel algorithm described in the previous section. Figure 4 presents the 3D-DWT computational times for a video frame resolution of $1,280 \times 640$ varying the GOP size and the number of processes. In the 3D-DWT, there is an intensive use of memory; therefore, the improvement in the use of the cache memory and data locality justifies efficiencies greater than 1. Values shown in Figure 4 correspond to the executions on the multicore Q6600 platform. However, efficiencies greater than 1 are not observed for the multicore HP Proliant SL390 due to the higher memory access performance respect to the multicore Q6600. The HP Proliant SL390 architecture provides a high-bandwidth memory access, through the Intel QPI Speed 64GT/s; therefore, the global performance improvement



	1 Proc.	2 Proc.	3 Proc.	4 Proc.
GOP: 16	0.57	0.36	0.26	0.21
GOP: 32	1.15	0.64	0.45	0.36
GOP: 64	2.70	1.27	0.89	0.71
GOP: 128	7.17	3.01	2.59	1.80

Number of Processes

Figure 4 3D wavelet algorithm. Compiler: GCC. Compiler flags: -O3 -fopenmp. Frame size: $1,280 \times 640$. Multicore Q6600.

is less significant than in the Q6600 platform. In Figure 5, we also present the computational times for the multicore HP Proliant SL390. The efficiencies obtained on both platforms are similar. However, comparing data obtained from video frames of different resolutions, we can conclude that the behavior on the multicore Q6600 becomes worse than on the multicore HP Proliant SL390, as the GOP size increases, i.e., when the global memory size increases.

The GOP size is an important parameter in the 3D-DWT computation, when applied to video coding because the average video quality increases as we increase the GOP size due to the minor GOP boundary effect. However, the computational load and memory requirements increase. Ideally, the GOP size would be equal to the total number of video frames. Since this is not possible due to the device memory restrictions, we must select the GOP size attending to both the video quality and the computational time. As we can see in Figures 4 and 5, the computational time increases as the GOP size increases. The minimum GOP size in our algorithm is 16 due to the four wavelet decomposition levels performed in the 3D-DWT (2^4).

In Figure 6, we present the computational time per frame.

We can observe that the parallel algorithm improves its behavior when both the number of processes and the GOP size increase. We want to remark that upon setting the GOP size equal to 256, for medium- and high-resolution video frames, the results obtained are not good due to the global memory size requirement. The optimal GOP size values are 64 and 128. Setting the GOP size to 128 reduces the border effects while setting the GOP size to 64 reduces the memory requirements. Both GOP size values obtain the best results in terms of computation time per frame, as seen in Figure 6.

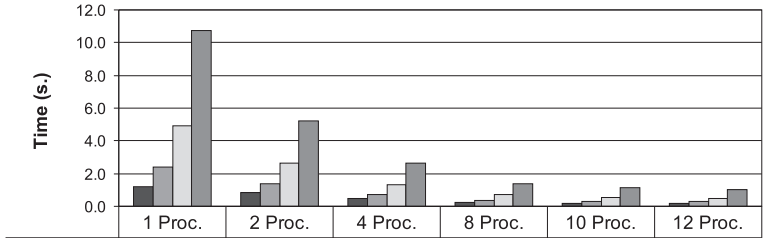
4.3 Overlapping the 3D-DWT stage and the coding stage

In section 4, we have analyzed the behavior of the parallel 3D-DWT for multicores, and we have presented a parallel algorithm that obtains good efficiencies using up to the maximum number of available cores (12 cores in the HP Proliant SL390). Furthermore, we have reduced the computational time of the 3D-DWT stage, but the time of the coding stage has not been considered at this time. So, in order to improve the global coding time, we consider implementing a two-phase pipeline strategy considering both the 3D-DWT and the coding stage. Note that there are no dependencies between these two stages if the working frame of the GOP is not the same.

As we have said, in the pipeline strategy proposed, we overlap the 3D-DWT computation and the coding stage, where both stages process different GOPs. In Figure 7, we show that the pipeline strategy developed. At each step, we simultaneously compute the 3D-DWT of one GOP and encode the GOP transformed in the previous step. At the initial step, we only perform the 3D-DWT transform of the first GOP, and the last GOP is encoded at the final step without overlapping the task.

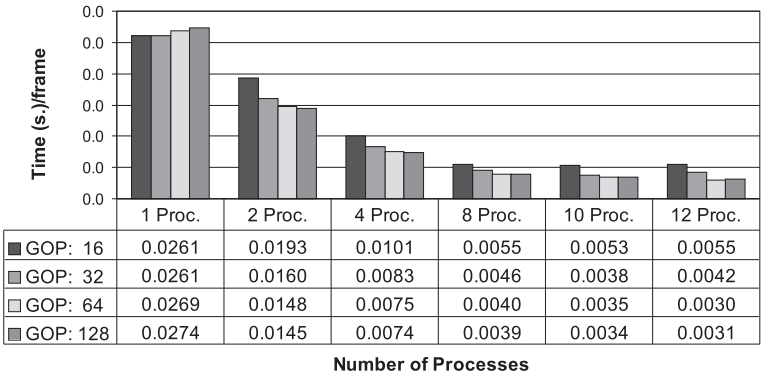
Firstly, in order to implement this pipeline procedure, we consider a multicore algorithm with two processes: the first one computes the 3D-DWT, and the second one computes the coding stage. There exists an inherent penalty in this type of algorithms at both the initial step and the final step. This penalty causes that the computational time reduction will be slightly lower than the optimal value equal to 50%. Considering the optimal GOP size values (64 or 128 frames), the ideal computational time reductions are 46.9% and 48.5%, respectively. We want to remark that our algorithm achieves these ideal values, obtaining, therefore, efficiencies equal to 0.94 and 0.97, respectively.

The previous conclusions are drawn considering that the computational time for both phases, the 3D-DWT stage and the coding stage, is similar. In Figure 8, we analyze the behavior of the computational time for both stages for the container (CIF) video sequence. As we can observe, the assumption that computational times for both stages are similar is only valid for very low compression rates. We can extend the behavior showed in



	1 Proc.	2 Proc.	4 Proc.	8 Proc.	10 Proc.	12 Proc.
GOP: 16	1.20	0.85	0.45	0.23	0.19	0.17
GOP: 32	2.38	1.40	0.72	0.38	0.32	0.30
GOP: 64	4.89	2.64	1.34	0.69	0.56	0.49
GOP: 128	10.71	5.20	2.65	1.38	1.14	1.03

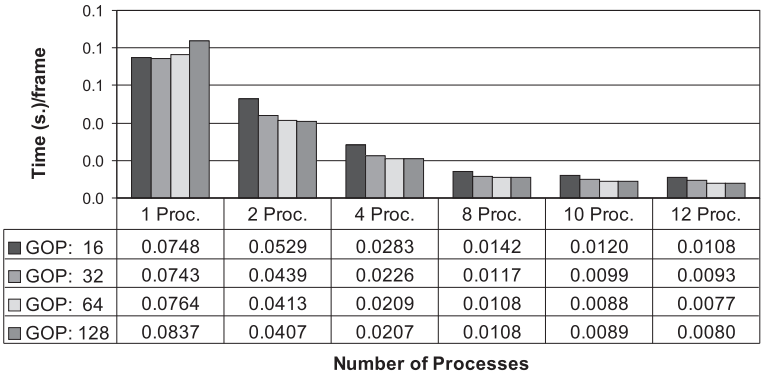
Figure 5 3D wavelet algorithm. Compiler: ICC. Compiler flags: -fast -fopenmp. Frame size: $1,920 \times 1,024$. Multicore HP Proliant SL390.



	1 Proc.	2 Proc.	4 Proc.	8 Proc.	10 Proc.	12 Proc.
GOP: 16	0.0261	0.0193	0.0101	0.0055	0.0053	0.0055
GOP: 32	0.0261	0.0160	0.0083	0.0046	0.0038	0.0042
GOP: 64	0.0269	0.0148	0.0075	0.0040	0.0035	0.0030
GOP: 128	0.0274	0.0145	0.0074	0.0039	0.0034	0.0031

Number of Processes

(a) Frame size: 1280 × 640



	1 Proc.	2 Proc.	4 Proc.	8 Proc.	10 Proc.	12 Proc.
GOP: 16	0.0748	0.0529	0.0283	0.0142	0.0120	0.0108
GOP: 32	0.0743	0.0439	0.0226	0.0117	0.0099	0.0093
GOP: 64	0.0764	0.0413	0.0209	0.0108	0.0088	0.0077
GOP: 128	0.0837	0.0407	0.0207	0.0108	0.0089	0.0080

Number of Processes

(b) Frame size: 1920 × 1024

Figure 6 Computational time per frame. Compiler ICC, Compiler flags -fast -openmp, Multicore HP Proliant SL390. (**a**) Frame size 1,280 × 640. (**b**) Frame size 1,920 × 1,024.

Figure 8 to the rest of the video sequences. Therefore, it is necessary to apply the parallel optimizations presented in section 4 in order to achieve ideal efficiencies. We want to remark that the improvements are focused on the 3D-DWT computation. To obtain the ideal efficiencies (using more than two processes), we must achieve both goals, reduce at maximum the 3D-DWT computational time in the first step (at this step there is no overlapping), and reduce the 3D-DWT computational time in the following steps in order to obtain a time lower or equal to the coding time (the other overlapped task).

Therefore, there are four different conditions in the parallel computation of the first GOP of a video sequence. In the initial step, we only compute the 3D-DWT transform of the first frame of the GOP. In the following steps, in which there are overlapped tasks, we must adapt the 3D-DWT computation in order to obtain the optimal number of processes used in the 3D-DWT computation. In the third stage, we compute the 3D-DWT using the optimal number of processes obtained and the coding stage using one process. As we have said, the fourth step is the computation of the coding stage of the last GOP. Both the

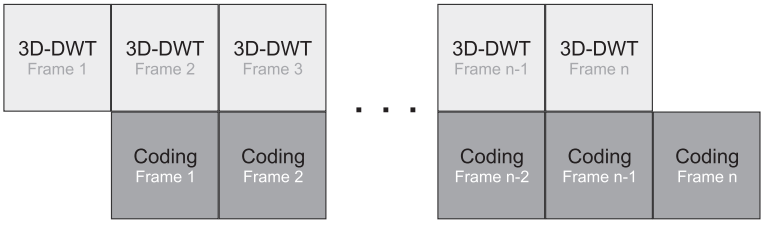


Figure 7 Multicore pipeline strategy.

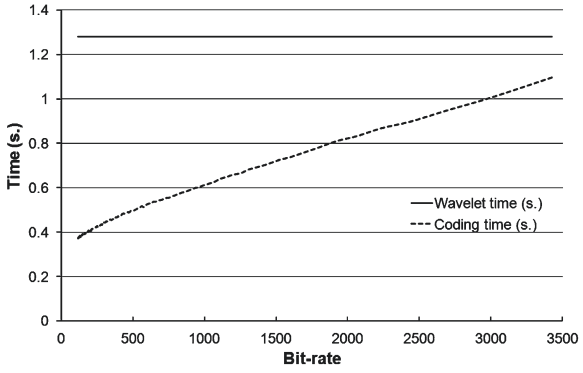


Figure 8 Computational time for the 3D-DWT and coding stages.

fork-join model of parallelism and the nested parallelism, offered by OpenMP, are used to implement these four discussed stages.

The fork-join parallelism refers to a method of specifying the parallel execution of a program whereby the program flow diverges into two or more flows that can be executed concurrently, and then, all flows come back together into a single flow when all of the parallel work is completed. In the nested parallelism, each flow can diverge into a new flow with two or more processes. In Figure 9, we show the structure of the parallel model developed using the fork-join model and the nested parallelism. In the first step, we use the maximum number of processes in order to accelerate at maximum the initial 3D-DWT computation. In the following steps (see Figure 7), the flow diverges into two processes where the first one computes the 3D-DWT of the following GOP and the second one computes the coding stage of the previous GOP. The flow that computes the 3D-DWT must adapt the number of processes in order to obtain a 3D-DWT computational time lower than the computational time of the coding stage. We set the number of processes to compute the 3D-DWT of the second GOP equal to half the maximum number of processes. In the following steps, the algorithm varies in the number of processes, depending on the measured time for both 3D-DWT and coding tasks, until the optimal value is found. Once we have obtained the optimal value of processes to compute the 3D-DWT, this value remains unchanged for the rest of the GOPs. The maximum number of processes used to compute the 3D-DWT is equal to the number of cores available minus one since this core (or process) is used to compute the coding stage. As we can see in Figure 8, the coding stage time is between two and four times lower, depending on the bit rate. Therefore, the optimal number of processes to compute the 3D-DWT depends on the bit rate, varying between 2 and 6.

Using the proposed strategy, we increase the efficiency of the pipeline structure up to 0.97 and up to 0.98 for GOP sizes 64 and 128, respectively. Moreover, the optimal value of processes is lower than the number of available processes, specially for the HP Proliant SL390 platform. The developed pipeline structure allows us to have idle cores, depending on the compression rate, and therefore, we can analyze the parallelization of the coding stage to improve the results in the future work.

Also, it is important to remark that upon joining the presented parallel strategies and the overlapping technique, we nearly reached the ideal speedups, where the bound of the speedup is determined by the computational time of the coding stage. Typical values of the speedup achievable are between 3 and 5.

5 Global performance evaluation

After analyzing both the performance of the multicore approach for the 3D-DWT computation and the aforementioned pipeline structure, we will present a comparison of the proposed encoder against the other test encoders in terms of coding delay.

In Figure 10, we present the coding speed (excluding I/O) in frames per second of all evaluated encoders and for different sequence resolutions. Now, our proposal uses the previously presented multicore optimization to perform the 3D-DWT in section 4. As it can be seen, MPEG-2 and MPEG-4 encoders still are the fastest ones. However, now, the 3D-GOP-RL encoder is up to four times as fast as

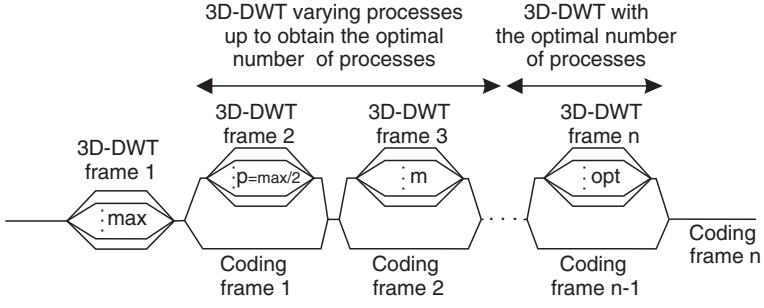


Figure 9 Fork-join with nested parallelism strategy.

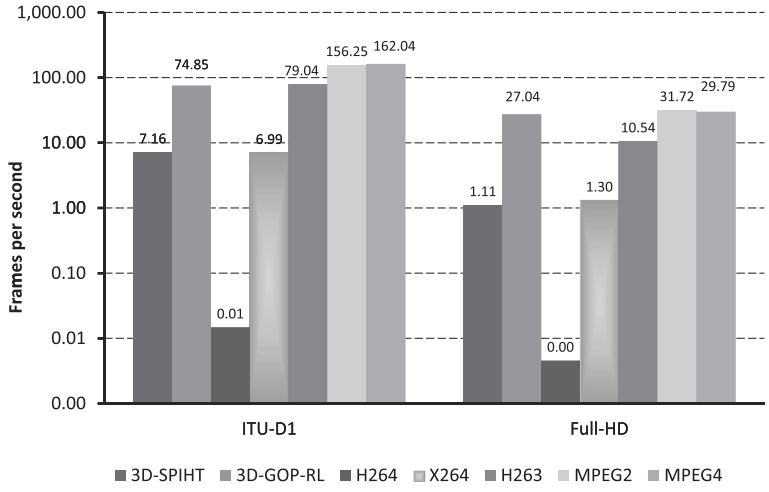


Figure 10 Coding speed in frames per second after multicore optimization of proposed encoder.

the non-multicore version of the proposed encoder, being able to compress a full-HD sequence in real-time.

Although, the multicore version of the 3D-GOP-RL encoder has been speeded up to four times, now, the bottleneck in the encoder is the coding stage after computing the 3D-DWT transform, specially at low compression rates, where there are lots of significant coefficients to encode. Considering the overlapping strategy presented in section 4.3, the 3D-DWT computation is hidden and the total coding time will be due only to the coding stage, except for the first GOP. Of course, that extra memory for the second GOP is required in this approach. As it can be seen in Figure 11, using this technique, the proposed encoder is the fastest one for full-HD video resolutions. Remark, that the optimizations performed are due only to multicore strategies while other encoders like ×264, H.263, MPEG-2, and MPEG-4 are fully optimized implementations, using CPU capabilities like multimedia extensions (MMX2, SSE2Fast, SSSE3, etc.) and multithreading.

6 Conclusions

In this paper, we have presented the 3D-GOP-RL, a fast video encoder based on 3D wavelet transform and efficient run-length coding. We have compared our algorithm against 3D-SPIHT, H.264, ×264, H.263, MPEG-2, and MPEG-4 encoders in terms of R/D, coding delay, and memory requirements.

Regarding R/D, our proposal has a similar behavior to MPEG-2 and H.263 and a slightly lower performance than MPEG-4. When compared with 3D-SPIHT, our proposal has a similar behavior for sequences with medium and high movements but lower performance for sequences

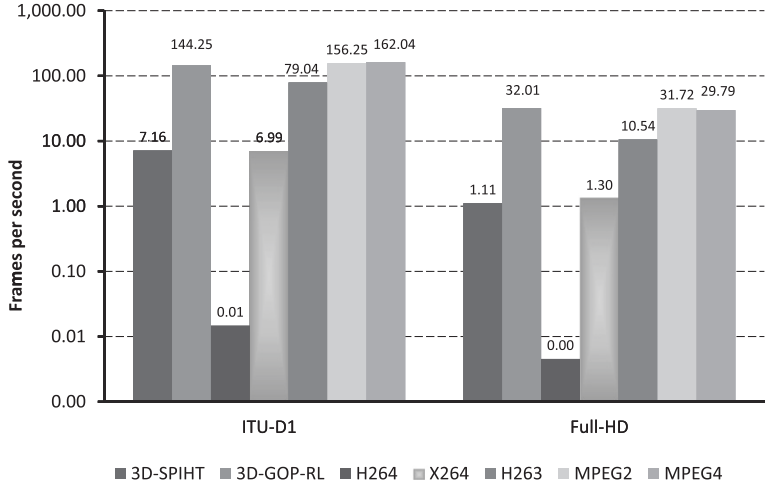


Figure 11 Coding speed in frames per second for all evaluated encoders after multithreading approach.

with low movement, like that of the container. However, our proposal requires six times less memory than the 3D-SPIHT. Both 3D-DWT-based encoders (3D-SPIHT and 3D-GOP-RL) outperform ×264 in intra mode (up to 11 dB), exploiting only the temporal redundancy among video frames when applying the 3D-DWT. It is also important to see the behavior of 3D-DWT-based encoders when applied to high frame rate video sequences, which is obtaining even better PSNR than ×264 in inter mode.

In order to speed up our encoder, we have presented an exhaustive analysis of the parallel strategies to compute the 3D-DWT transform. As we have seen, the parallel algorithm obtains good efficiencies, with the proper parameter setting, using the available cores, up to 12 in the multicore HP Proliant SL390 and up to 4 in the multicore Q6600. Even more, we have applied multithreading strategies to hide the 3D-DWT computational time. Using these strategies, the proposed encoder (3D-GOP-RL) is the fastest encoder for full-HD video resolutions, being able to compress a full-HD video sequence in real-time.

The fast coding/decoding process and the fact of avoiding the use of motion estimation/motion compensation algorithms make the 3D-GOP-RL encoder a good candidate for applications where the coding/decoding delay is critical for proper operation or for applications where a frame must be reconstructed as soon as possible. 3D-DWT-based encoders could be an intermediate solution between pure intra encoders and complex inter encoders.

In the future work, we intend to apply parallel strategies to speed up the encoder even more, but this time, we are focusing on the coding stage.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
This research was supported by the Spanish Ministry of Science and Innovation under grant numbers TIN2011-26254 and TIN2011-27543-C03-03.

Received: 30 October 2012 Accepted: 27 March 2013
Published: 20 April 2013

References

1. R Jang-Seon, K Eung-Tea, Fast intra coding method of H.264 for video surveillance system. Int. J. Comput. Sci. Netw. Secur. **7**(10), 76–81 (2007)
2. P Campisi, A Neri, in *IEEE International Conference on Image Processing*. Video watermarking in the 3D-DWT domain using perceptual masking (IEEE, NY, 2005), pp. 997–1000
3. P Schelkens, A Munteanu, J Barbariend, M Galca, X Giro-Nieto, J Cornelis, Wavelet coding of volumetric medical datasets. IEEE. Trans. Med. Imaging. **22**(3), 441–458 (2003)
4. PL Dragotti, G Poggi, Compression of multispectral images by three-dimensional SPITH, algorithm. IEEE Trans. Geoscience Remote Sensing. **38**(1), 416–428 (2000)
5. M Aviles, F Moran, N Garcia, Progressive lower trees of wavelet coefficients: efficient spatial and SNR scalable coding of 3D models. Lect. Notes Comput. Sci. **3767**, 61–72 (2005)
6. D Taubman, A Zakhor, Multirate 3-D subband coding of video. IEEE Trans. Image Process. **3**(5), 572–588 (1994)
7. CI Podilchuk, NS Jayant, N Farvardin, Three dimensional subband coding of video. IEEE Trans. Image Process. **4**(2), 125–135 (1995)
8. Y Chen, WA Pearlman, in *Visual Communications and Image Processing*, vol. 2727. Three-dimensional subband coding of video using the zero-tree method (SPIE, Bellingham, 1996), pp. 1302–1309
9. JM Shapiro, Embedded image coding using zerotrees of wavelet coefficients. IEEE Trans. Signal Process. **41**(12), 3445–3462 (1993)
10. BJ Kim, Z Xiong, WA Pearlman, Low bit-rate scalable video coding with 3D set partitioning in hierarchical trees (3D SPIHT). IEEE Trans. Circuits Syst. Video Tech. **10**, 1374–1387 (2000)
11. M Bibhuprasad, S Abhishek, M Sudipta, A high performance modified SPIHT for scalable image compression. Int. J. Image Process. **5**(4), 390–402 (2011)
12. L Ye, T Karp, B Nutter, S Mitra, J Guo, in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*. Three-dimensional subband coding of video with 3-D BCWT (IEEE, NY, 2006), pp. 401–405
13. J Guo, S Mitra, B Nutter, T Karp, in *Proceedings of the Data Compression Conference*. A fast and low complexity image codec based on backward coding of wavelet trees (IEEE, NY, 2006), pp. 292–301
14. J Luo, X Wang, CW Chen, KJ Parker, in *Visual Communications and Image Processing*, vol. 2727. Volumetric medical image compression with three-dimensional wavelet transform and octave zerotree coding (SPIE, Bellingham, 1996), pp. 579–590
15. BM Sunil, CP Raj, in *Emerging Trends in Engineering and Technology (ICETET), 2010 3rd International Conference on*. Analysis of wavelet for 3d-dwt volumetric image compression (IEEE, NY, 2010), pp. 180–185
16. BJ Kim, WA Pearlman, in *Proceedings of the Data Compression Conference, 1997*. An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT) (IEEE, NY, 1997), pp. 251–260
17. ISO/IEC 14496–10 and ITU Rec H.264. Coding of audio-visual objects - Part 10, Advanced Video Coding (2003)
18. ITU-T Recommendation H.263, Video coding for low bit rate communication (2005)
19. FFmpeg (2010). Available on http://ffmpeg.zeranoe.com
20. SG Mallat, A theory for multi-resolution signal decomposition: The wavelet representation. IEEE Trans. Pattern. Anal. Mach. Intell. **11**(7), 674–693 (1989)
21. V Galiano, O López, MP Malumbres, H Migallón, in *proceedings of International Conference on Computational and Mathematical Methods in Science and Engineering*. Improving the discrete wavelet transform computation from multicore to gpu-based algorithms (J. Vigo-Aguiar, Salamanca, 2011), pp. 544–555
22. OpenMP application program interface, version 3.1. OpenMP Architecture Rev. Board (2011). Available on http://www.openmp.org

doi:10.1186/1687-6180-2013-84
Cite this article as: Galiano *et al.*: Multicore-based 3D-DWT video encoder. *EURASIP Journal on Advances in Signal Processing* 2013 **2013**:84.