EURASIP Journal on Applied Signal Processing 2002:9, 981–989

Wavelet Kernels on a DSP: A Comparison Between Lifting and Filter Banks for Image Coding

Stefano Gnavi
CERCOM, Center for Multimedia Radio Communications, Dipartimento di Elettronica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
Email: gnavi@mail.tlc.polito.it

Barbara Penna
CERCOM, Center for Multimedia Radio Communications, Dipartimento di Elettronica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
Email: penna@mail.tlc.polito.it

Marco Grangetto
CERCOM, Center for Multimedia Radio Communications, Dipartimento di Elettronica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
Email: grangetto@polito.it

Enrico Magli
CERCOM, Center for Multimedia Radio Communications, Dipartimento di Elettronica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
Email: magli@polito.it

Gabriella Olmo
CERCOM, Center for Multimedia Radio Communications, Dipartimento di Elettronica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
Email: olmo@polito.it

Received 30 August 2001 and in revised form 30 April 2002

We develop wavelet engines on a digital signal processors (DSP) platform, the target application being image and intraframe video compression by means of the forthcoming JPEG2000 and Motion-JPEG2000 standards. We describe two implementations, based on the lifting scheme and the filter bank scheme, respectively, and we present experimental results on code profiling. In particular, we address the following problems: (1) evaluating the execution speed of a wavelet engine on a modern DSP; (2) comparing the actual execution speed of the lifting scheme and the filter bank scheme with the theoretical results; (3) using the on-board direct memory access (DMA) to possibly optimize the execution speed. The results allow to assess the performance of a modern DSP in the image coding task, as well as to compare the lifting and filter bank performance in a realistic application scenario. Finally, guidelines for optimizing the code efficiency are provided by investigating the possible use of the on-board DMA.

Keywords and phrases: wavelet, lifting scheme, filter bank, JPEG2000, DSP.

1. INTRODUCTION

A huge number of applications use the discrete wavelet transform (DWT) [1] as a means to extract relevant features from signals. Examples are reported in the fields of mathematics, physics, numerical computing, and engineering, including image classification, feature detection, image denoising, image registration, and image compression, just to mention a few. Especially in the engineering field, there has been a considerable interest in using wavelet transforms for image and video coding applications [2, 3]. As a result, the ISO/ITU-T has selected the DWT as the transform coding kernel for the new image compression standard, namely JPEG2000 [4], which will be released during 2001. Consequently, fast and cost-effective implementations of DWT kernels, compliant with JPEG2000 specifications, are called for in order to make its diffusion as widespread as possible.

While the wavelet transform of an image can be fairly

easily computed by means of a general-purpose personal computer, there obviously exist contexts where more compact, light-weight and less power-demanding computing devices are required. A recent trend [5] fosters the design of reconfigurable systems that make use of digital signal processors (DSPs) and field-programmable gate array (FPGA). An example is given by the transmission of images from scientific space missions, where the images collected by the onboard sensors may undergo wavelet-based compression (e.g., Rosetta Osiris [6]), with a DSP-based system being used as computational core. DSPs are also very often used to handle image and video processing tasks in consumer electronics [7]. The importance of wavelets on a DSP is witnessed by the number of implementations proposed in the literature (cf. [8, 9]). In this paper, we focus on the study of the DSP-based implementation of a wavelet kernel; the target application is image coding with JPEG2000, with its extension to intraframe video coding (Motion-JPEG2000).

Until recently, DWT implementations were based on the so-called filter bank scheme [1], which computes the DWT of a signal by iterating a sequence of highpass and lowpass filtering steps, followed by downsampling. In 1997 Sweldens proposed a new scheme, called lifting scheme (LS), as an alternative way to compute the DWT [10]. The LS has immediately obtained a noteworthy success, as it provides several advantages with respect to the filter bank scheme. The most interesting ones from the implementation standpoint are that

(i) the LS requires less operations than the filter bank scheme, with a saving of up to one half for very long filters;
(ii) the LS allows to compute an integer wavelet transform (IWT), that is, a wavelet transform that maps integers to integers [11], thus enabling the design of embedded lossless and lossy image encoders [12, 13].

This paper is focused on the development of a wavelet kernel based on the LS, and using a DSP as the computational core. The interest of this work is manifold. Firstly, from a pure implementation perspective, the performance evaluation of an optimized implementation of such a kernel on a modern DSP indicates the maximum sustainable processing rate. This can be used to estimate the number of images per second that can be processed by, for example, a compression engine such as JPEG2000, or the video frame rate that can be sustained by a Motion-JPEG2000 encoder/decoder in DSP-based applications, for example, videoconferencing by means of a PC card. Secondly, since the LS can be used to design a progressive lossy-to-lossless compression algorithm [13], it is important to evaluate the execution speed of the IWT with respect to the DWT. Thirdly, and distinctively novel in this paper, from a signal processing point of view, there is a strong interest in finding out to which degree the theoretically lower complexity of the LS translates into reduced execution speed; in fact, it is likely that the DSP architecture affects the performance of lifting and filter bank wavelet cores in a different fashion. In this work all aspects are considered, that is, an optimized DSP implementation of the LS is presented, and its performance is then compared with that of the filter bank scheme; both DWT and IWT are considered. The results allow to assess the impact of the DSP architecture on the performance of both algorithms, thus providing useful guidelines for the architectural design and implementation of a wavelet-based processing system.

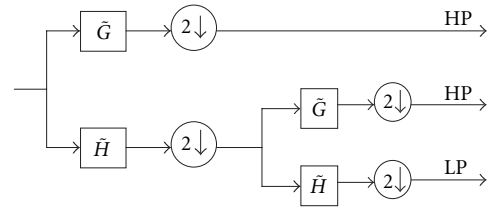

Figure 1: Block diagram of the filter bank scheme.

This paper is organized as follows. In Section 2, we briefly review the wavelet transform, focusing on the filter bank scheme and the LS in Sections 2.1 and 2.2, respectively. The DSP implementations of both algorithms are described in Section 3. In Section 4, a performance evaluation of such implementations is proposed; in particular, results related to the execution speed are reported in Section 4.1, and a comparison between LS and filter bank scheme is presented in Section 4.2. The possibility of improving performance by means of direct memory access (DMA) is discussed in Section 4.3. Finally, in Section 5 conclusions are drawn.

2. WAVELET TRANSFORM

As already stated, the two main algorithms used to compute the DWT are the filter bank scheme and the LS, which are briefly reviewed in Sections 2.1 and 2.2, respectively.

2.1. Filter bank scheme

The filter bank scheme (see [1]) is sketched in Figure 1, where the operations needed to compute the DWT of a one-dimensional signal are depicted. One level of decomposition involves that the input sequence is highpass and lowpass filtered by the analysis filters $\tilde{H}(z)$ and $\tilde{G}(z)$; the two resulting sequences are then downsampled by a factor two. More decomposition levels can be obtained by iterating this procedure on the lowpass branch, as shown in Figure 1. The two-dimensional extension is achieved by filtering and downsampling first along the rows, and then along the columns. The inverse transform is achieved performing a similar sequence of filtering and upsampling operations (see [1]).

2.2. Lifting scheme

As well known [1], a discrete-time filter can be represented by its polyphase matrix, which is built from the $\mathcal{Z}$ transforms of the even and odd samples of its impulse response. The LS stems from the observation [14] that the polyphase matrix can be factorized, leading to the implementation of one step of the filter bank scheme as a cascade of shorter filters, which act on the even and odd signal samples, followed by a normalization. In particular, the LS performs a sequence of primal and dual lifting steps, as described in the following and

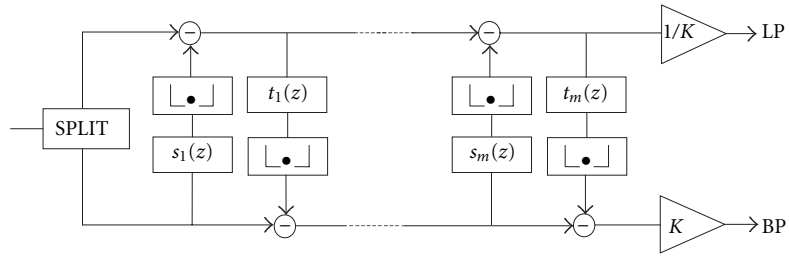

FIGURE 2: Block diagram of the LS.

reported in the block diagram of Figure 2. The inverse transform is achieved performing the same steps in reversed order [14].

The polyphase representation of a discrete-time filter $H(z)$ is defined as

$$H(z) = H_e(z^2) + z^{-1}H_o(z^2), \tag{1}$$

where $H_e(z)$ and $H_o(z)$ are respectively obtained from the even and odd coefficients of $h[n] = \mathcal{Z}^{-1}\{H(z)\}$, where $\mathcal{Z}$ denotes the zeta transform. The synthesis filters $H(z)$ and $G(z)$ (lowpass and highpass, respectively) can thus be expressed in terms of their polyphase matrix

$$P(z) = \begin{bmatrix} H_e(z) & G_e(z) \\ H_o(z) & G_o(z) \end{bmatrix} \tag{2}$$

and $\tilde{P}(z)$ can be analogously defined for the analysis filters.

The Euclidean algorithm [14] can be used to decompose $P(z)$ and $\tilde{P}(z)$ as

$$\begin{aligned} P(z) &= \prod_{i=1}^{m} \begin{bmatrix} 1 & s_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_i(z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & \frac{1}{K} \end{bmatrix}, \\ \tilde{P}(z) &= \prod_{i=1}^{m} \begin{bmatrix} 1 & 0 \\ -s_i(z^{-1}) & 1 \end{bmatrix} \begin{bmatrix} 1 & -t_i(z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{K} & 0 \\ 0 & K \end{bmatrix}. \end{aligned} \tag{3}$$

This factorization leads to the sequence of primal and dual lifting steps shown in Figure 2.

The filters $H_e(z)$, $H_o(z)$, $G_e(z)$, and $G_o(z)$, along with their analysis counterparts, are Laurent polynomials [14]. Since the set of all Laurent polynomials exhibits a commutative ring structure, within which polynomial division with remainder is possible, long division between two Laurent polynomials is not a unique operation [14]. Therefore, several different factorizations (i.e., pairs of $\{s_i(z)\}$ and $\{t_i(z)\}$ filters) may exist for each wavelet. However, in case of DWT implementation, all possible choices are equivalent.

An IWT, mapping integers onto integers, can be very simply achieved rounding off the output of the $s_i(z)$ and $t_i(z)$ filters right before adding or subtracting [11]; the rounding operation introduces a nonlinearity in each filter operation. As a consequence, in the IWT the choice of the factorization impacts on both lossless and lossy compression, so making the transition from the DWT to the IWT not straightforward [15].

TABLE 1: Computational cost (number of multiplications plus additions) of lifting versus filter banks.

Filter	Standard algorithm	Lifting scheme
LEG(5, 3)	$4(N + M) + 2$	$2(N + M + 2)$
DB(9, 7)	$4(N + M) + 2$	$2(N + M + 2)$
SWE(13, 7)	$3(N + \tilde{N}) - 2$	$3/2(N + \tilde{N})$

As already stated, the LS requires fewer operations than the filter bank scheme. The latter algorithm corresponds to merely applying the polyphase matrix: only the samples that are not discarded by the subsequent downsampling operation are actually filtered. In order to compare the two algorithms, one can use the number of multiplications and additions required to output a pair of samples, one on the lowpass and one on the highpass branch. As shown in [14], the cost of lifting tends, asymptotically for long filters, to one-half of the cost of the standard algorithm. Table 1 reports the formulas, presented in [14], to compute the cost of the two algorithms for the filters used in this work (see also Section 3). Here $|h|$ and $|g|$ are the degree of the highpass and lowpass filter (i.e., the number of coefficients minus one); in the case that $|h|$ and $|g|$ are even, we set $|h| = 2N$ and $|g| = 2M$. Note that the filter SWE(13,7), being an interpolating filter, has a different formula, which also involves the number of vanishing moments $\tilde{N}$.

3. IMPLEMENTATION

The LS and filter bank scheme have been implemented on the floating-point Texas Instruments TMS320C6711 DSP board. The board comprises a 150 MHz floating-point processor, two memory regions, namely *on-chip* and *off-chip*, a direct memory access (DMA) controller, and some peripherals interfaces. The CPU core includes two sets of 16 registers (*register file A, register file B*), the on-chip memory divided into two cache memories (*L1, L2*), and the arithmetic and logical units (see [16]). Figure 3 shows the block diagram of the DSP architecture.

In the following, we outline some features of our implementation of the two algorithms, including the filters and types of boundary extensions used. The LS implementation is compliant with the specifications of the Final Committee Draft of JPEG2000 Part I (core coding systems) [4], which is,

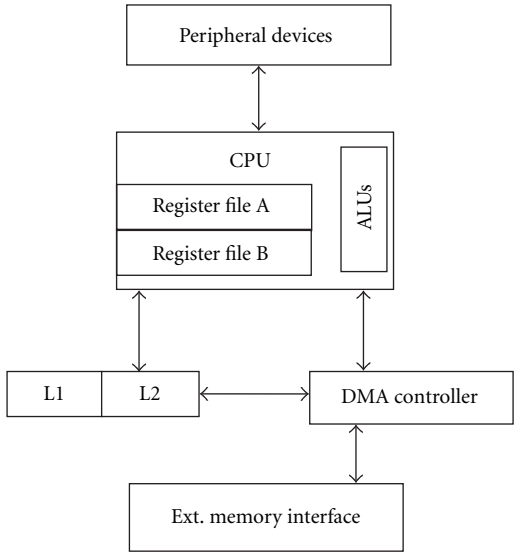

Figure 3: Block diagram of the DSP architecture.

at the time of this writing, the latest publicly available document describing the standard.

The code profiling results for the two algorithms, reported in Section 4, have been obtained demanding the optimization of the assembler code to the C compiler, which is known to nearly achieve the same efficiency as an expert programmer. For this reason, the code has been written in a simple and plain style, so as to facilitate compiler optimization. Therefore, in this section we only give an overview of the implementations of the two algorithms, whereas we rather concentrate on the profiling results (Section 4), which represent the main contribution of this article. Of course, one could achieve some performance improvement by constraining the implementations, for example, to support a limited number of filters (even only one); nevertheless, this approach would negatively impact on generality of application, which in this work has been preserved as far as possible.

As for boundary extension at the borders of the input signal, which is necessary because the wavelet filters are noncausal, two possible extensions are considered.

(i) Symmetric extension: it performs mirroring of the signal samples outside the signal support. If used with biorthogonal symmetric filters, it allows to achieve perfect reconstruction also at the image borders.

(ii) Zero padding: it consists in adding zeros before and after the signal. This extension is not supported by the JPEG2000 standard, but is very simple, and hence sometimes used.

The filters supported by this implementation have been selected according to the JPEG2000 standard:

- LeGall I(5,3) (LEG(5,3) in the following);
- Daubechies (9,7) (DB(9,7) in the following);
- Sweldens (13,7) (SWE(13,7) in the following).

The first two filters are explicitly embodied in JPEG2000, for the reversible and nonreversible transform, respectively.

Table 2: Factorization of LEG(5,3).

$s_i(z), t_i(z) = a_0 z^{d_M} + a_1 z^{d_{M-1}} + a_2 z^{d_{M-2}} + \cdots$						
Filter: LEG(5,3)						
	d_M	a_0	a_1	a_2	a_3	K
$s_1(z)$	0	0	0	0	0	1
$t_1(z)$	1	0.5	0.5	0	0	
$s_2(z)$	0	−0.25	−0.25	0	0	

Table 3: Factorization of DB(9,7).

$s_i(z), t_i(z) = a_0 z^{d_M} + a_1 z^{d_{M-1}} + a_2 z^{d_{M-2}} + \cdots$						
Filter: DB(9,7)						
	d_M	a_0	a_1	a_2	a_3	K
$s_1(z)$	0	0	0	0	0	1.2302
$t_1(z)$	1	−1.5861	−1.5861	0	0	
$s_2(z)$	0	−0.0530	−0.0530	0	0	
$t_2(z)$	1	0.8829	0.8829	0	0	
$s_3(z)$	0	0.4436	0.4436	0	0	

Table 4: Factorization of SWE(13,7).

$s_i(z), t_i(z) = a_0 z^{d_M} + a_1 z^{d_{M-1}} + a_2 z^{d_{M-2}} + \cdots$						
Filter: SWE(13,7)						
	d_M	a_0	a_1	a_2	a_3	K
$s_1(z)$	0	0	0	0	0	1
$t_1(z)$	2	0.0625	−0.5625	−0.5625	0.0625	
$s_2(z)$	1	−0.03125	0.28125	0.28125	−0.3125	

Note that the last filter is not supported by JPEG2000 Part I. However, it has been considered because, being a long filter, it allows to verify the asymptotic complexity of the LS.

The selection of the factorization of the wavelet filters, to be used in the LS, has been made following the directives of the JPEG2000 standard, and is reported in Tables 2, 3, and 4. The filter length, deducible from the acronym, allows to easily identify the N and M parameters previously defined.

As for the filter bank scheme, the input signal is filtered by the same kernels listed above, but using the expanded rather than the factorized representation. Notice that, while performing the convolution between the signal and the filter impulse response, the samples that would be discarded by downsampling are not computed at all. For completeness, Tables 5, 6, and 7 report the coefficients of the filters employed, up to the fourth decimal digit.

4. EXPERIMENTAL RESULTS

As stated in Section 1, the objective of this work is manifold; in particular, experimental tests have been carried out with the following goals.

(1) To evaluate the absolute running time of an LS-based wavelet kernel on a modern DSP; in particular, in view of

Table 5: LEG(5,3) filter.

i	h_0	h_1
0	0.75	1
±1	0.25	−0.5
±2	−0.125	0

Table 6: DB(9,7) filter.

i	h_0	h_1
0	0.6029	1.1151
±1	0.2669	−0.5913
±2	−0.0782	−0.0575
±3	−0.0169	0.0913
±4	0.0267	0

Table 7: SWE(13,7) filter.

i	h_0	h_1
0	0.6797	1
±1	0.2813	−0.5625
±2	−0.1230	0
±3	−0.0313	0.0625
±4	0.0352	0
±5	0	0
±6	−0.0020	0

the implementation of an embedded lossy-to-lossless image compression system, to understand to which degree embodying an IWT capability may penalize the execution speed. This matter is discussed in Section 4.1.

(2) To find out how close to the theoretical value is the actual performance gain of the LS with respect to the filter bank scheme, in terms of execution speed. This matter is discussed in Section 4.2.

(3) To study the possibility of exploiting an available onboard DMA, in order to speed up code execution. This matter is discussed in Section 4.3.

The results shown in the following, and the comparison between LS and filter bank scheme, have been reported (see Sections 4.1 and 4.2) in terms of the time needed to perform one level of transform on one image row; this has been done so as to facilitate the interpretation of results. The results have been parameterized on the length of the input data vector, and execution times for dyadic lengths are reported. It has been found that the sum of such dyadic values yields a very accurate estimate of the multilevel transform. Of course, computing the wavelet transform of an image requires to perform both rowwise and columnwise filtering. However, it has been found that the time needed to compute a columnwise filtering is the same as for rowwise filtering. Even though this behavior might seem surprising at a first glance, it can be reasonably justified by the efficient management of memory accesses performed by the cache memory; a more detailed explanation is given in Section 4.3.

4.1. Absolute running times

The graphs in Figures 4, 5, and 6 report the absolute running times achieved by the LS (in the DWT and IWT mode, respectively) and the filter bank scheme, in order to compute the one-level wavelet transform of a one-dimensional data vector, contiguously stored in the external memory. The boundary extension used is the symmetric one.

The results reported on the graphs can be used to estimate the number of images per second that can be processed by these algorithms. Employing the LEG(5,3) filter and the symmetric extension for a complete 2D one-level decomposition on a 256×256 grey-scale image, the LS allows to process between 7 and 8 images per second, whereas the filter bank scheme only sustains between 4 and 5 images per second. Note that computing the IWT, rounding off the filtered coefficients in the LS, leads to slower operation. The running times of the IWT are from 10% to 25% larger than the DWT using the LS.

If the wavelet kernel is thought of as the core of a JPEG2000 encoder, it is worth recalling that the wavelet transform is responsible for a significant part of the total encoder and decoder running time. Some figures have been obtained by profiling the Jasper reference JPEG2000 implementation, and have been reported in [17]. It turns out that, for progressive lossless coding, the wavelet transform is responsible of about 30% of the overall encoder and decoder running time. In the progressive lossy case this percentage is increased to about 50% at the encoder, and 70% at the decoder. This implies that it should be possible to encode/decode, with a single DSP, about 2 256×256 images per second in the integer lossless mode (using the LEG(5,3) filter), and encode and decode about 2 images per second in the lossy mode using the DB(9,7) filter. While this figures are suitable for an image coding application, it turns out that more powerful hardware, such as a multi-DSP system or an FPGA, is required to sustain real-time Motion-JPEG2000 video.

4.2. Comparison between lifting and filter bank

As stated, in [14] it is claimed that the LS requires asymptotically half the number of operations with respect to the filter bank scheme. We have compared the running time of our LS and filter bank implementations, in order to understand how the DSP architecture impacts on the performance gain. In particular, Table 8 reports the ratios between the running time of the filter bank scheme and the LS. Comparing these figures with the theoretical results, it can be noticed that these ratios are different from the theoretical values.

This behavior can be explained considering the architectural features of the processor employed. The DSP used in this work has an efficient pipeline, which can dispatch 8 parallel instructions per cycle. Parallel instructions proceed simultaneously through each pipeline phase, whereas serial instructions proceed through the pipeline with a fixed relative phase difference between instructions. Every time a jump to an instruction not belonging to the pipeline occurs, the pipeline must be emptied and reloaded. Thus, in this case, the filtering operations that frequently update the pipeline

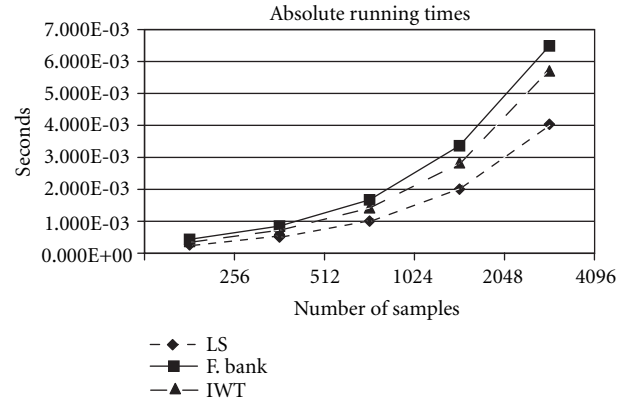

Figure 4: LEG(5,3): absolute running times.

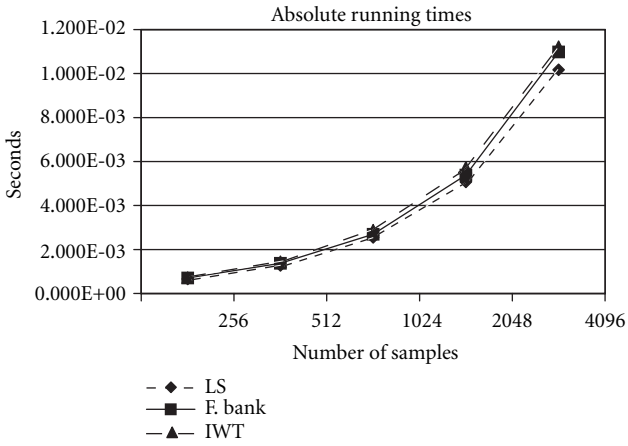

Figure 5: DB(9,7): absolute running times.

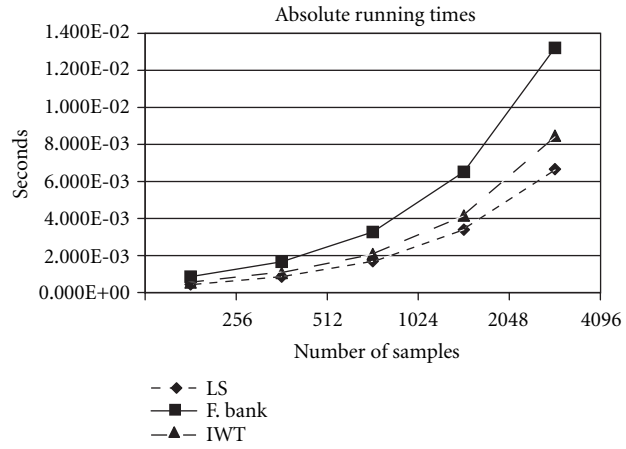

Figure 6: SWE(13,7): absolute running times.

contents, turn out to be disadvantaged. The effect on the computation of the wavelet transform is that, in general, the convolution with a long kernel can be optimized more efficiently than several convolutions with short kernels. Therefore there is a trade-off, in that the filter bank must perform twice as many operations as the LS with long filters; on the other hand, the use of the pipeline tampers with the LS operation, since the factorizations of long filters may consist of numerous short filters. The best results, with regard to the gain, are obtained with the SWE(13,7) filter: even though the filter is long, its factorization consists of only 2 filters, with 4 coefficients each.[1] The opposite occurs with the DB(9,7) filter, whose factorization consists of 4 filters with 2 coefficients each. The gain that comes from the inferior number of operations in LS is thus lost in emptying and reloading the pipeline. The LEG(5,3) filter has an intermediate behavior. Note that, for an increasing number of samples, the ratio between the running times of the two algorithms is not constant, nor it increases linearly. This behavior is due to the way the processor manages the cache memory and the data transfer from external to internal memory.

Table 8: Ratios between the running times of filter bank scheme and LS.

Filter bank running time/LS running time			
Samples	LEG(5,3)	DB(9,7)	SWE(13,7)
256	1.650	1.069	1.969
512	1.678	1.077	1.937
1024	1.657	1.058	1.929
2048	1.679	1.062	1.917
4096	1.607	1.080	1.982
Theoretical value			
	1.4	1.666	1.833

4.3. Optimization with DMA

The results in Section 4.2 have shown that the LS is faster than the filter bank scheme as for the computation of the wavelet transform. These results have been obtained using implementations which demand to the CPU the data transfer from the external memory to the CPU itself for performing the convolutions. In the following, we focus on the architectural features of the DSP employed, investigating the possibility of improving the LS performance by exploiting the properties of the DMA, typically available on a DSP board.

The LS program previously described filters a vector of coefficients allocated on a region of external (off-chip) memory. On the other hand, the DSP has a two-level internal (on-chip) cache, with significantly lower access time than the external one. The second-level cache (L2) can be configured as internal memory, and can be used to store and filter the image pixels values, with an expected speedup due to the reduced memory access time.

Since the size of an image is usually larger than the L2 size, it is necessary to transfer the data in small blocks from

[1] It is worth noticing that even higher gains (nearly 3) have been found with the SWE(13,7) filter, using a fixed-point implementation that is not addressed in this paper. This is not surprising, for the upper bound of 2 on the LS gain [14] is computed in the case of worst case factorization, while the SWE(13,7) filter also admits shorter factorizations.

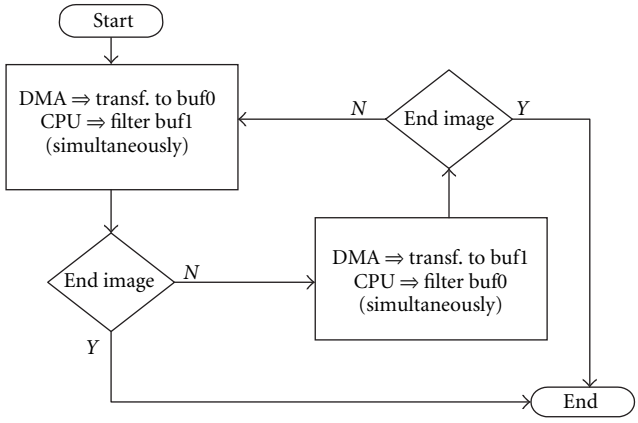

Figure 7: Ping-pong buffering.

Table 9: Comparison between the running times of the LS without and with using the DMA.

Absolute running times [in seconds]	
Standard LS	LS with DMA
1.222	1.651

Table 10: Comparison between the running times of the LS without and with using the DMA: transfer of several rows simultaneously.

Absolute running times [in seconds]		
	Standard LS	LS with DMA
2 rows	1.222	1.174
4 rows	1.222	1.166

the external to the internal memory. The device used to this purpose is the DMA controller. In this work, the DMA has been configured so as to transfer a row (or column) of the image into the on-chip memory, while at the same time the CPU filters the data transferred at the previous step. In this way, the CPU never accesses the off-chip memory, since both the stack and the temporary variables are allocated in L2.

The use of L2 as a shared resource between DMA and CPU involves the need of synchronizing these devices. The data can be corrupted if the accesses to L2 do not take place in the correct order. To avoid this problem, we have employed four software interrupts to regulate the sequence of operations. Moreover, the two concurrent devices are set to operate on two different buffers, which are swapped at each filtering cycle with a ping-pong buffering mechanism (see Figure 7).

We have run this version of the LS on a 512×512 grey-scale image, performing one complete 2D decomposition level with the DB(9,7) filter. This has led to the results shown in Table 9, where the running time of the standard LS algorithm is also reported for comparison. It can be noticed that the synchronization of the devices and the reconfiguration of the DMA after each transfer leads to a higher running time. In order to make the employment of DMA and L2 advantageous, it is necessary to reduce the number of DMA reconfigurations. This can be done by transferring more than one row or column at one time. Table 10 shows that transferring, for example, 2 or 4 rows simultaneously yields an improvement of the LS performance. However, the gain is not as high as expected, and hardly pays back for the additional complexity. The reason for such low gain using the DMA lies in the efficiency of the DSP cache memory.

In fact, every time the CPU needs a datum stored in the external memory, 32 consecutive bytes are transferred from the memory to L1. If the offset between two data processed sequentially by the CPU is fewer than 32 bytes (i.e., 8 floating-point coefficients), the CPU accesses the memory only once, since the second datum will be already cached in L1. The advantage of accessing a faster memory is apparent only when the weight of the memory accesses is high overall. We assume that the image pixel values are stored in the external memory as floating-point values in row major order. As for rowwise filtering, one access to the external memory is sufficient to retrieve eight samples of the to-be-filtered data. As far as columnwise filtering is concerned, once a complete image column has been retrieved from the external memory, the subsequent seven columns are also placed in the cache.[2] Moreover, in the specific case of the wavelet transform, most of the time is spent by the processor in computing the convolution, that is, sums and products between the filter coefficients and the image pixel values; the filtering routine is computationally heavy, so that the weight of the access operations is not very high. Therefore the actual number of accesses to the external memory turns out to be quite limited, and their weight on the program running time accordingly low. In summary, the performance improvement in the wavelet transform computation, which can be obtained by employing the DMA, is limited because of the efficiency of the on-chip cache.

5. CONCLUSIONS

In this paper, we have addressed the development of wavelet cores on a DSP, compatible with the JPEG2000 specifications. The wavelet transform has been implemented according to the filter bank scheme and the lifting scheme; in this latter case the integer-transform option has also been considered. The code has been profiled so as to evaluate the efficiency of the implementation and, more interestingly, to allow a comparison between the LS and the filter bank scheme. Moreover, the use of the DMA has also been considered as a possible way to improve data throughput.

The results have highlighted some aspects of DSP-based implementations of the wavelet transform, which are discussed in the following.

(1) The DSP considered in this work is able to compute up to 8 complete 2D one-level wavelet transforms per second on a grey-scale 256×256 image. This figure can be used to

[2] This holds provided that the cache memory is large enough to store eight columns, as usually happens in practice.

evaluate the number of JPEG2000 frames that a single DSP is able to code or decode, for example, using the JPEG2000 profiling results reported in [17].

(2) A performance comparison between lifting and filter banks has been carried out. We have found that the LS always runs faster than the filter bank scheme. However, the performance gain differs from the theoretical results in [14], because the DSP architecture has a different impact on code optimization for the two algorithms. In particular, convolutions with long filters, which are typical of the filter bank scheme, tend to benefit from the DSP pipelined architecture. On the other hand, the LS gain is higher for long filters. In the end, the actual gain heavily depends on the number and length of the factorized filters used in the LS.

(3) It has turned out that employing the DMA to transfer data from the external to the internal memory (and vice versa), while the CPU concurrently filters the previously transferred data, may provide very little advantage, if any at all, in terms of execution speed. This is due to the fact that the on-chip cache memory is able to very efficiently manage the data transfer operations, for both rowwise and columnwise filtering.

ACKNOWLEDGMENT

This work was partially developed under the Texas Instruments Elite program.

REFERENCES

[1] M. Vetterli and J. Kovačević, *Wavelets and subband coding*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1995.

[2] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing*, vol. 1, no. 2, pp. 205–230, 1992.

[3] D. Lazar and A. Averbuch, "Wavelet-based video coder via bit allocation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 7, pp. 815–832, 2001.

[4] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards, and Practice*, Kluwer Academic Publishers, Dordrecht, Netherlands, 2001.

[5] J. Eyre and J. Bier, "The evolution of DSP processors," *IEEE Signal Processing Magazine*, vol. 17, no. 2, pp. 43–51, 2000.

[6] B. Fiethe, P. Rüffer, and F. Gliem, "Image processing for rosetta osiris," in *6th International Workshop on Digital Signal Processing Techniques for Space Applications*, vol. 144, ESTEC, Noordwijk, The Netherlands, September 1998.

[7] J. Eyre, "The digital signal processor derby," *IEEE Spectrum*, vol. 38, no. 6, pp. 62–68, 2001.

[8] K. Haapala, P. Kolinummi, T. Hamalainen, and J. Saarinen, "Parallel DSP implementation of wavelet transform in image compression," in *Proc. IEEE International Symposium on Circuits and Systems*, pp. 89–92, Geneva, Switzerland, May 2000.

[9] B. Yiliang, W. Houng-Jyh, C.-C. J. Kuo, and R. Chung, "Design of a memory-scalable wavelet-based image codec," in *Proc. IEEE International Conference on Image Processing*, Chicago, Ill, USA, October 1998.

[10] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," *Siam J. Math. Anal*, vol. 29, no. 2, pp. 511–546, 1997.

[11] R. C. Calderbank, I. Daubechies, W. Sweldens, and B. Yeo, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332–369, 1998.

[12] A. Bilgin, P. Sementilli, F. Sheng, and M. Marcellin, "Scalable image coding using reversible integer wavelet transforms," *IEEE Trans. Image Processing*, vol. 9, no. 11, pp. 1972–1977, 2000.

[13] M. Grangetto, E. Magli, and G. Olmo, "Efficient common-core lossless and lossy image coder based on integer wavelets," *Signal Processing*, vol. 81, no. 2, pp. 403–408, 2001.

[14] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl.*, vol. 4, no. 3, pp. 247–269, 1998.

[15] M. Grangetto, E. Magli, and G. Olmo, "Minimally nonlinear integer wavelets for image coding," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, Istanbul, Turkey, June 2000.

[16] Document SPRU189F, "TMS320C6000 CPU and instructions set reference guide," October 2000, www.ti.com.

[17] M. D. Adams and F. Kossentini, "JasPer: A software-based JPEG-2000 codec implementation," in *Proc. of IEEE International Conference on Image Processing*, vol. 2, pp. 53–56, Vancouver, BC, Canada, October 2000.

Stefano Gnavi was born in Biella, Italy, in March 1976. He received the degree in electrical engineering at Politecnico di Torino, Italy, in July 2001. Since March 2002 he is a researcher under grant with the Center for Wireless Multimedia Communications (CERCOM), at the Department of Electronics, Politecnico di Torino. His research interests are in the field of image communication, video processing and compression, as well as hardware implementation. Currently he is working on very low bit rate video coding techniques.

Barbara Penna was born in Castellamonte, Italy, in May 1976. She received the degree in electrical engineering at Politecnico di Torino, Italy, in July 2001. Since September 2001 she is a researcher under grant with the Signal Analysis and Simulation (SAS) group, at the Department of Electronics, Politecnico di Torino. Her research interests are in the field of data compression. Currently she is working on novel SAR raw data compression algorithms based on wavelet transforms.

Marco Grangetto received the "summa cum laude" degree in electrical engineering at Politecnico di Torino in 1999, where he is currently pursuing a Ph.D. degree. His research interests are in the field of digital signal processing and multimedia communications. In particular, he is working at the development of efficient and low complexity lossy and lossless image encoders based on wavelet transforms. Moreover, he is challenging the design of reliable multimedia delivery systems for tetherless lossy packet networking. He was awarded the Premio Optime by "Unione industriale di Torino" in September 2000, and a Fulbright grant in 2001 for a research period at the Center for Wireless Communications (CWC) at UCSD.

Enrico Magli received the degree in electronics engineering in 1997, and the Ph.D. degree in electrical and communications engineering in 2001, from Politecnico di Torino, Turin, Italy. He is currently a Post-Doctoral researcher at the same university. His research interests are in the field of robust wireless communications, compression of remote sensing images, superresolution imaging, and pattern detection and recognition. In particular, he is involved in the study of compression and detection algorithms for aerial and satellite images, and of signal processing techniques for environmental surveillance from unmanned aerial vehicles. From March to August 2000 he was a visiting researcher at the Signal Processing Laboratory of the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland.

Gabriella Olmo received the Laurea Degree (cum laude) and the Ph.D. in electronic engineering at Politecnico di Torino in 1986 and 1992, respectively. From 1986 to 1988 she was researcher with CSELT (Centro Studi e Laboratori in Telecomunicazioni), Turin, working on network management, non hierarchical models and dynamic routing. From 1991, she has been Assistant Professor at Politecnico di Torino, where she is member of the Telecommunications group and the Image Processing Lab. Her main recent interests are in the field of wavelets, remote sensing, image and video coding, resilient multimedia transmission, joint source-channel coding, stratospheric platforms. She has joined several national and international research programs under contracts by Inmarsat, ESA (European Space Agency), ASI (Italian Space Agency), European Community. She has coauthored more than 80 papers in international scientific journals and conference proceedings.