WAVELET IMAGE COMPRESSION FOR MOBILE/PORTABLE APPLICATIONS

M. Ravasi [1], *M. Mattavelli* [1], *D. J. Mlynek* [1], *A. Buttar* [2], *S. Soudagar* [2]

[1] **Swiss Federal Institute of Technology, Integrated Systems Center C3I, CH-1015 Lausanne Switzerland.**

[2] **Motorola**
207 Route de Ferney, CH-1218 Grand Saconnex, Geneva Switzerland.

Abstract

This paper presents some examples of architectures for the implementation of wavelet codecs. Compared to previous works [1, 2, 3, 4, 5], the novelty of the approach is based on a joint optimization of both the algorithmic and architectural features, according to the overall system hardware and software strategy.

The results of the analysis, although general for the different considered architectures, allow specific optimization of system performance for both dedicated ASIC design and embedded software implementation based on available system resources, such as execution speed and cache performance, while minimizing power consumption.

1 Introduction

Digital camera and mobile wireless communications are two market segments which are currently exhibiting rapid growth and efficient and flexible image data compression is required for storage on removable media or transmission over communication networks. Recent advances in low cost image sensor technology have greatly contributed to the quality of digital camera products. Mobile communications are emerging from primarily voice-based services, towards data services which open up many opportunities for transmission of images to terminals, such as smart phones and personal digital assistants, with color display capability.

Within this framework, wavelet transform is playing a leading role for its better performances in terms of signal analysis, multiresolution features and improved compression compared to existing methods such as the DCT based compression schemes adopted in the old JPEG standard. This success is testified by the fact that the wavelet transform has now been adopted by MPEG-4 and will be the base of JPEG-2000. Indeed superior performance at low bit-rates and transmission of data according to client display parameters are particularly interesting for mobile applications. The wavelet transform shows better results because, thanks to its time-scale representation, it's intrinsically well suited to non-stationary signal analysis, such as images. Although it is a rather simple transform, its implementation may lead to critical requirements in terms of memory size and bandwidth yielding to costly implementations. Thus different solutions must be investigated to find specifically optimized implementations being able to derive the best solution fitting a given system scenario.

2 An overall system optimization approach

Because of the sub-band tree decomposition of the wavelet transforms, the coding/decoding process has to be performed on several layers. Practical system limitations faced by the designer include memory size and memory bandwidth for the storage of the temporary data for the intermediate layers, with efficient use of both on-chip and off-chip storage [1, 2]. In this paper we show that redesigning the data processing scheduling and the memory storage scheme we can obtain a joint optimization of the algorithmic and architectural features according to specific system requirements. The optimum choice of these factors can be achieved by analyzing different strategies, referred to in this paper as "Classical", "Sliding-Windows Layer-by-Layer on *N* Stripes", "Sliding-Windows on *N* layers", "Block-by-Block with External Tree Memory" and "Block-by-Block with Internal Tree Memory". Each of these strategies corresponds to an implementation characterized in parametric form in terms of generic architectural features such as:

- on-chip memory size
- on-chip data-path bandwidths
- overall filter complexity
- external memory size
- external data-path bandwidths.

The rest of the paper is organized as follows: section 3 reports the description of different studied strategies, section 4 presents parametrical results and the corresponding numeric values for some example design cases, while section 5 reports the conclusions about the comparison of the different strategies.

3 Different strategies for Discrete Wavelet Transform coding sample processing

According to Discrete Wavelet Transform (DWT) theory, we can split a 1D discrete signal by first filtering it with two FIR filters p and q (analysis filters), together referred to as "*wavelet kernel*", and then down-sampling filters' output by a factor of 2. Filter p and q are respectively a low-pass and a band-pass filter, referred to as basic scaling function and wavelet basis, and their down-sampled outputs, that is the encoded half-length signals, are referred to as the approximation and the detail signal. We can reconstruct the original signal by up-sampling the two coded sub-signals and then filtering them with two FIR filters p_0 and q_0 (reconstruction filters) as shown in Figure 2. We have to choose the four filters to form a biorthogonal set, to allow the perfect reconstruction of the original signal from its encoded version.

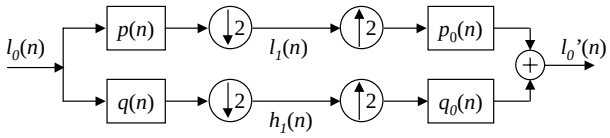

Figure 1: *By using a set of biorthogonal filters, the perfect reconstruction of $l_0(n)$ from its decomposed signals $l_1(n)$ and $h_1(n)$* is guaranteed: $l_0'(n)=l_0(n)$.

Hereafter, when we refer to p and q filters we refer both to filters only and to filters and down-sampling together. We explicitly state what is referred to in order to avoid ambiguity.

According to wavelet theory, we have to choose, among all possible biorthogonal filter sets, a couple of filters p and q local both in time and frequency. This representation clearly still preserves the temporal aspect of the information carried by the signal because, for the given short length of analysis filters, each coded sample depends upon only few neighboring samples of the input signal.

Both the approximation and the detail signal can be recursively split into sub-signals, yielding the tree decomposition typical of DWT coding. It can be easily shown that the results of the down-sampling stages is an encoded signal composed by the same number of samples of the original signal. The example in Figure 2 shows the Mallat tree decomposition where only the approximation signals are recursively coded. Typically tree decompositions commonly adopted in image processing applications differ from the Mallat decomposition in splitting the detail signals only in the very first layers. The reason is that there's little correlation among samples in detail signals hence there is no need of further splitting them.

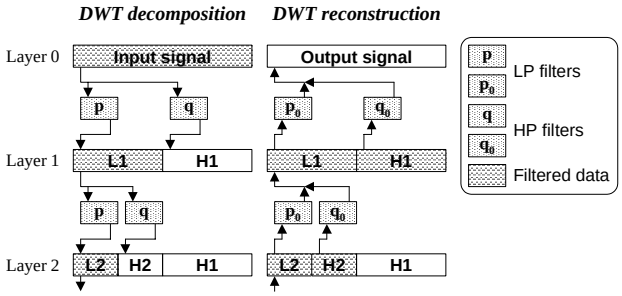

Figure 2: *1D DWT with Mallat tree decomposition. The number of samples of the encoded signal is equal to the one of the input signal.*

The tree decomposition yields a frequency-dependant representation of the signal. Filtering down-sampled version of the original signal can be seen as filtering the signal with dilated versions of the filters p and q. This scaling in time of the filters is equivalent to scaling their frequency response and proportionally moving their center frequency. Thus, layer by layer, the width of the analysis filters doubles and correspondingly the width of analyzed bands halves together with the center frequency of the wavelet basis. Furthermore, layer by layer the number of coded samples in each sub-signal halves, efficiently adapting to the reduction of the analyzed band.

Therefore, the DWT representation of a signal is actually a joint time-frequency representation and, because of the scaling of the filters' support intervals, is referred to as a time-scaling representation. This representation is able to focus on signal details and discontinuities, thus it's well suited for non-stationary signals' analysis such as sounds and images. It shows better performances in non-stationary signals' analysis and compression with respect to full frequency representations, that lose any information about the typical temporal aspect of non-stationary signals, or with respect to existing time-frequency representations like block DCT.

2D DWT coding is usually based on separable basic scaling functions and wavelet bases such that it can be performed iterating two orthogonal 1D DWT. Each image is first processed in a direction and then in the orthogonal one, thus obtaining 4 images whose size is a quarter of the size of the original, as shown in Figure 3. A detailed and more complete description of Wavelet Transforms and Discrete Wavelet Transforms can be found in [6].

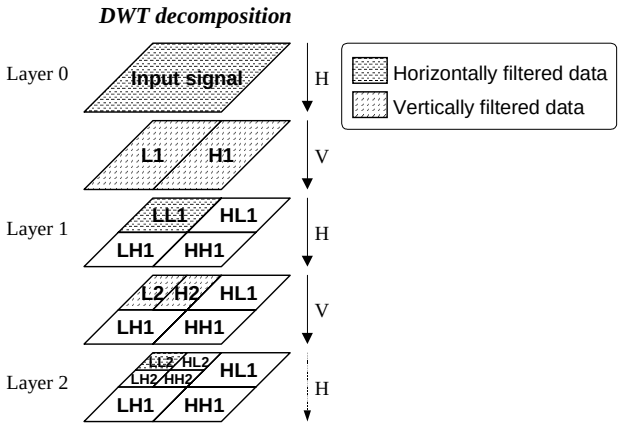

Figure 3: *2D DWT with Mallat tree decomposition. The size of intermediate layers decreases twice faster than in 1D case and the amount of data to be filtered tends asymptotically to 4/3 of the size of the input signal. Since data must be filtered both horizontally and vertically, the total amount of filtered samples tends to 8/3 of the size of input signal.*

In Figure 2 we can see that, due to the presence of temporary samples in intermediate layers, the total amount of processed samples along all layers tends asymptotically to twice the size of the input samples. Like in the case of 1D signals, the Mallat decomposition for 2D signals is obtained by recursively splitting only approximation samples, see Figure 3. Keeping in mind that samples must be processed twice because of the 2D separable coding, the total amount of processed samples tends to 8/3 of the number of the input samples. The results is that in addition to the bandwidth needed for the input and output of samples, an additional bandwidth of 1.67 sample is needed for temporary samples. Furthermore, temporary samples need an extra memory to be stored in and read from. The best choice, from a performance point of view, is to build this memory on-chip to obtain fast accesses. If the size of the required temporary memory is too large to be implemented on-chip, we need to implement it off-chip, thus compromising the performance with external power costly and slow memory accesses.

Because of the small length of analysis and reconstruction filters, the computational load results small compared to the data transfer load for all external and internal I/O operations. Thus, the main bottleneck in wavelet coding and decoding is the management of temporary data dealing with the trade-off between off-chip memory bandwidth and on-chip memory size.

In sections 3.1, 3.2 and 3.3 we show some solutions for the implementation of 2D wavelet codec. They are characterized by implementing different data-paths along the tree decomposition. We consider an input signal of $W \cdot H$ samples, a Mallat tree decomposition on L layers, and the following notation:

EM_s External Memory size. It is the memory required to store temporary samples. Notice that this memory is never used for input or output signal.

EM_r External Memory bandwidth used to read temporary samples.

EM_w External Memory bandwidth used to write temporary samples.

IM_s Size of Internal (On-Chip) Memory.

IM_r Internal Memory bandwidth used to read temporary samples.

IM_w Internal Memory bandwidth used to write temporary samples.

EB_r External Bandwidth used to read both input and temporary samples. It's the sum of EM_r and WH:

$$EB_r = EM_r + WH \quad (1)$$

EB_w External Bandwidth used to write both output and temporary samples. It's the sum of EM_w and WH:

$$EB_w = EM_w + WH \quad (2)$$

$F_\#$ Filter Number. Number of p-q filters pair implemented on chip.

F_w Filter Width. It's the width of the joint support interval of filters p and q. Since these filters are usually centered on the origin, that is the support interval of one includes the support interval of the other, F_w is equal to the maximum of the orders of both filters plus one.

F_s Filter Memory. Memory size necessary to store the input samples required by a couple of p-q filters to compute their output samples, without counting the new input sample. F_s depends on filters' implementation: in next sections we consider lifting-scheme implementations [7, 8] that minimize both the required filter memory size and the number of operations required to compute output samples.

3.1 Classical

The classical approach to 2D wavelet coding processes one layer at the time in the tree decomposition, as shown in Figure 2. On each layer the vertical and horizontal processing are performed successively one by one. It is a very simple implementation because all the process is carried out by a single 1D wavelet coding/decoding that operates on a layer at a time, first in one direction, then in the orthogonal one.

While coding the first layer we need to store data between the vertical and horizontal processing, thus the size of the temporary memory must be proportional to the number of the input image samples. For common high resolution image sizes, the amount of required memory is

too large to be conveniently built on-chip,

$$EM_{s,Classical} = WH\ . \tag{3}$$

Only a couple of filters is required for processing because no parallel operations are performed. No on-chip memory is required because data are read and written only in the in external memory:

$$IM_{s,Classical} = 0, \tag{4}$$

$$IM_{r,Classical} = IM_{w,Classical} = 0, \tag{5}$$

$$F_{\#,Classical} = 1. \tag{6}$$

Each layer must be read and written twice, because of the 2D separable filtering, and keeping in mind that the layer's size decreases by a factor or 4 layer by layer,

$$EB_{r,Classical} = EB_{w,Classical} = 2\sum_{l=1}^{L}\frac{WH}{4^{l-1}} = \frac{2}{3}\frac{4^{L}-1}{4^{L-1}} < \frac{8}{3}WH \tag{7}$$

and, according to expressions (1) and (2), we obtain

$$EM_{r,Classical} = EB_{r,Classical} - WH < \frac{5}{3}WH\ . \tag{8}$$

$$EM_{w,Classical} = EB_{w,Classical} - WH < \frac{5}{3}WH\ . \tag{9}$$

This simple implementation has the disadvantage of requiring the highest external memory bandwidth and size because a large amount of temporary samples must be stored both between two successive layers and between vertical and horizontal processing.

3.2 Sliding-Windows

The main idea behind Sliding-Windows approach is to exploit dependencies among different layers and among vertical and horizontal processing in order to try to use temporary samples as soon they are available. If by minimizing the lifetime of temporary samples, we can reduce the size of the required temporary memory and, if such memory results small enough to be implemented conveniently on chip, we have also reduced the bandwidth of the (slow) external memory.

In Figure 4 we can observe that we can exploit the inter-layer dependencies in order to remove the time gap between the creation and the consumption of temporary samples of intermediate layers. As soon as samples are produced by the low-pass filter processing a layer, we use these samples to feed the filters on the following layer. Thus no time gap occurs between creation and consumption of temporary samples.

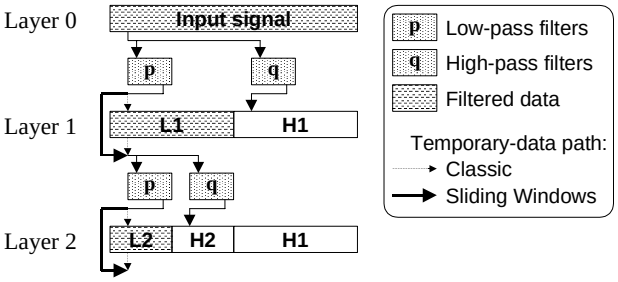

Figure 4: *1D Sliding Windows. Temporary data of intermediate layers (L1, L2, …) have do not need to be stored and read if they are processed as a soon as they are available.*

The scheme of Figure 5 shows how to manage temporary samples between horizontal and vertical filtering. Once again we try to reuse the temporary samples between the two filtering processes as soon as they are produced. Let's suppose that we first filter horizontally. Horizontal filters produce samples along rows, while vertical filters needs input samples along columns. To produce these columns of samples with horizontal filters, we could use a set of horizontal filters, with a couple of filters for each line. In this way, scheduling the horizontal filters line by line, we are able to produce columns of temporary data capable of feeding vertical filters. We actually do not need to implement a couple of filters for each line because they never work in parallel, only one line is active at a time. We just need to store 2 columns of samples needed as input by all the virtual horizontal filters and read them line by line to load them in parallel in a true real pair of horizontal *p-q* filters. These two columns behave like two windows sliding over the input image, because they cache successive columns of input samples. Obviously, to reduce the external memory bandwidth, the memory to store horizontal filters' input samples must be built on-chip. Within each layer, only input and output samples are exchanged outside the layer, all the temporary samples are managed by on-chip memory. Thus, within each layer we achieve the minimum necessary external bandwidth.

The horizontal filtering produces at the same time the low-pass and high-pass samples. Thus so as to get the final output samples we need two couples of vertical filters, to process in parallel the low-passed and high-passed temporary samples coming from the horizontal filtering.

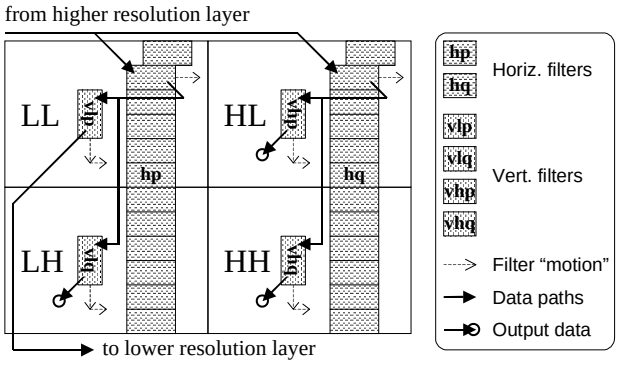

Figure 5: *2D Sliding Windows on a layer. Temporary samples between horizontal and vertical filtering are processed with two "columns" of horizontal filters whose output is used to feed vertical filters.*

If we extend the scheme of Figure 4 to 2D signals, substituting each 1D layer and its corresponding filters with the scheme of Figure 5, we obtain the Sliding Windows on All Layers (SW All L) implementation. No external memory is required because all temporary samples are processed on chip and the external bandwidth is thus limited to the minimum required to read the input signal and store the output signal:

$$EM_{s,SW.All.L} = 0\,, \tag{10}$$

$$EM_{r,SW.All.L} = EM_{w,SW.All.L} = 0\,, \tag{11}$$

$$EB_{r,SW.All.L} = EB_{w,SW.ALL.L} = WH\,. \tag{12}$$

For each layer we need to implement three couples of filters:

$$F_{\#,SW.All.L} = 3L\,. \tag{13}$$

Furthermore, for each layer we need the on-chip memory for sliding windows temporary samples. The total width of the two columns is equal to F_s, because we need to store on each line all the old input samples necessary to the corresponding filter pair to compute their output. Noting that the height of the *LL* signals decreases by a factor of two from one layer to the next, we obtain:

$$IM_{s,SW.All.L} = \sum_{l=1}^{L} \frac{H}{2^{l-1}} F_s = \frac{2^L - 1}{2^{L-1}} HF_s < 2HF_s\,. \tag{14}$$

Obviously, to minimize this memory size we have to dispose the sliding-windows memory along input signal smaller dimension.

We know that each line in the sliding windows has to store the input samples of a couple of filters. Thus each line seems to behave like a FIFO line. Furthermore, looking at the scheme of Figure 1, we notice that we compute two output samples after having loaded two new input samples in the two filters. This means that to produce two output samples we need to write a sample in memory, read F_s samples and write another sample. Since we're considering the lifting-scheme implementation for the filters, we must be aware that each two input samples more than two samples change in the filter memory. If we consider the worst case all samples change and have to be written into the filter memory. Thus, to produce two output samples we need to write a sample in memory, read F_s samples and write F_s samples. For synchronization, it is better if for each column of input samples we produce a column of output samples. We can achieve this result by delaying one of the two simultaneous output samples of the lifting-scheme. We do not need an extra memory for this delay because we can exploit the delay line of the lifting-scheme filters and thus find the delayed sample in the sliding-windows memory. All we have to do is to read one more sample from on-chip memory. Thus, we find that for each two samples produced on a layer, we need to read and write F_s+1 samples:

$$IM_{r,SW.All.L} = IM_{w,SW.All.L} = \sum_{l=1}^{L} \frac{WH}{4^{l-1}} \cdot \frac{F_s + 1}{2} = \frac{4^L - 1}{3 \cdot 4^{L-1}} \cdot \frac{F_s + 1}{2} WH < \cdot \frac{2}{3}(F_s + 1)WH \tag{15}$$

It's a relatively costly implementation because of both the quite large amount of required on-chip memory and the complex scheduling needed to synchronize all the filters working in parallel on all layers. If we exploit only the scheme of Figure 5 to process a layer at a time, we deal with a simpler solution, referred to as "Sliding-Windows Layer-by-Layer on 1 Stripe" (SW LbL 1S). This solution avoids the storage of temporary samples between horizontal and vertical filtering but needs to store inter-layer temporary samples. Since we are processing a layer at a time, the temporary memory for inter-layer temporary samples can be reused layer after layer. Thus the size of this memory has to be equal to the maximum amount of such temporary samples. It results equal to the size of the first LL sub-signal,

$$EM_{s,SW.LbL.1S} = \frac{HW}{4} \cdot \tag{16}$$

LL sub-signals for layers from 1 to *L*-1 must be written and read in this memory only once:

$$EM_{r,SW.LbL.1S} = EM_{w,SW.LbL.1S} = \sum_{l=1}^{L-1} \frac{WH}{4 \cdot 4^{l-1}} = \frac{WH}{4} \cdot \frac{4^L - 4}{3 \cdot 4^{L-1}} < \frac{WH}{3}\,, \tag{17}$$

$$EB_{r,SW.LbL.1S} = EB_{w,SW.LbL.1S} < \frac{4}{3}WH \,. \quad (18)$$

Since we process only one layer at a time, we need only three couples of filters,

$$F_{\#,SW.LbL.1S} = 3, \quad (19)$$

and enough internal memory to process the largest layer. Since we can reuse it for successive smaller layers it results that:

$$IM_{s,SW.LbL.1S} = HF_s \,, \quad (20)$$

while its bandwidth is the same as in the SW All L case because we are still processing all the same samples (in SW All L approach, inter-layer temporary samples are used as soon as available, thus they need no temporary memory and do not influence the internal memory bandwidth), therefore:

$$IM_{r,SW.LbL.1S} = IM_{w,SW.LbL.1S} < \frac{2}{3}(F_s + 1)WH \,. \quad (21)$$

With respect to SW ALL L, this solution has the advantage of being much simpler. It requires only three filters and almost half internal memory and needs no complex inter layer scheduling. The increase of external bandwidth is realtively small but the amount of required external memory is quite large.

We can also develop intermediate solutions between SW All L and SW LbL 1S, referred to as "Sliding Windows on N Layers" (SW N L), using the same approach of SW All L but applied only to *N* layers out of *L*, as shown in Figure 6 for 1D signals.

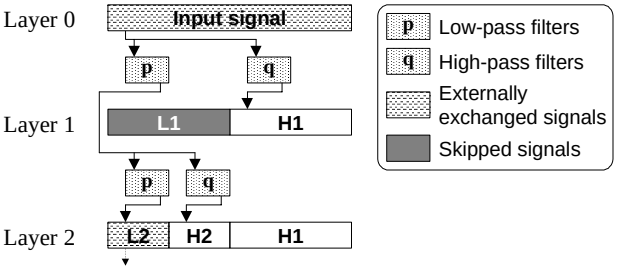

Figure 6: *1D Sliding-Windows on 2 Layers. We apply the scheme of SW LbL 1S only on 2 layers: we skip the signal L1 because we reuse output samples of first p filter as soon as they are available, but we need to store (and read) signal L2 in external temporary memory.*

With respect to SW LbL 1S, we reduce the amount of required external memory and its bandwidth, because the first LL sub-signal that we need to store on off-chip memory is obviously smaller than the first LL layer (like in SW LbL 1S), but we need more on-chip memory, enough to process the first larger *N* layers:

$$IM_{s,SW.N.L} = \sum_{l=1}^{N} \frac{H}{2^{l-1}} F_s = \frac{2^N - 1}{2^{N-1}} HF_s \,, \quad (22)$$

and more filters to process N layers at a time,

$$F_{\#,SW.N.L} = 3N \,, \quad (23)$$

while, for the same reasons discussed for the SW LbL 1S case, the internal memory bandwidth results equal to both previous cases,

$$IM_{r,SW.N.L} = IM_{w,SW.N.L} < \frac{2}{3}(F_s + 1)WH \,. \quad (24)$$

The required external memory is equal to the size of the first externally stored LL sub-signal samples,

$$EM_{s,SW.N.L} = \frac{HW}{4^N} \,, \quad (25)$$

and we can compute the resulting bandwidth like we did in the SW LbL 1S case, keeping in mind that we read and write a LL sub-signal only a layer out of *N*,

$$EM_{r,SW.N.L} = EM_{w,SW.N.L} = \sum_{l=1}^{N \cdot l < L} \frac{WH}{4^{N \cdot l}} < WH \frac{4^L - 1}{4^L (4^N - 1)} < \frac{WH}{4^N - 1} \,, \quad (26)$$

$$EB_{r,SW.N.L} = EB_{w,SW.N.L} < \frac{4^N}{4^N - 1} WH \,. \quad (27)$$

We can try to reduce the amount of internal memory required by the SW LbL 1S approach, by using the same scheme but applying it only to stripes of the input signal, obtaining the "Sliding-Windows Layer-by-Layer on N Stripes" implementation (SW LbL *N*S). If, for instance, we process half input signal at a time, that is we process two horizontal stripes separately, we need just half internal memory because we need to implement the Sliding-Windows only on half height, as shown in Figure 7.

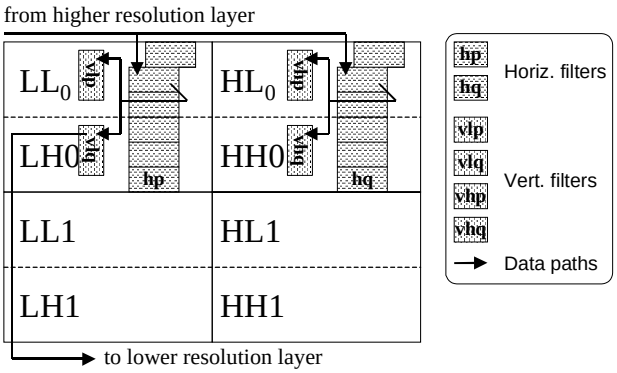

Figure 7: *2D Sliding-Windows Layer-by-Layer on 2 Stripes. We reduce the amount of internal memory processing only a stripe of input signal at a time.*

With N stripes we have N-1 inter-stripe borders. Reminding that the joint support interval of both filters p and q spans F_w samples, we need to filter, for each inter-layer border, some lines below the border for the upper stripe and some lines above for the lower stripe, for a total of F_w-1 lines per border and (F_w-1) (N-1) lines per input signal. These lines are processed twice, because they are needed by both the adjacent stripes. The best choice is to save the temporary samples between horizontal and vertical processing relative to these lines, to be able to retrieve them later to start processing the next stripe. We need an extra temporary memory of (F_w-1) W samples that, considering that F_w-1≤F_s (they are equal with standard filter implementation) and that W>H/N (we are considering horizontal signals with W≥H), is larger than the reduction of internal memory. Thus if we want to reduce the on-chip memory, we need to put this extra memory off chip,

$$EM_{s,SW.LbL.NS} = EM_{s,SW.LbL.1S} + (F_w - 1)W = \frac{HW}{4} + (F_w - 1)W, \quad (28)$$

and we need more external memory bandwidth to write and read these new temporary data,

$$EM_{r,SW.LbL.NS} = EM_{w,SW.LbL.NS} = EM_{r,SW.LbL.1S} + \sum_{l=1}^{L}(F_w - 1)(N-1)\frac{W}{2^{l-1}} < \frac{WH}{3} + 2(F_w - 1)(N-1)W, \quad (29)$$

$$EB_{r,SW.LbL.NS} = EB_{w,SW.LbL.NS} < \frac{4}{3}WH + 2(F_w - 1)(N-1)W. \quad (30)$$

The internal memory bandwidth does not change because samples are never filtered horizontally twice,

$$IM_{r,SW.LbL.NS} = IM_{w,SW.LbL.NS} = IM_{r,SW.LbL.1S} = IM_{w,SW.LbL.1S} < \frac{2}{3}(F_s + 1)WH, \quad (31)$$

while its size decreases with the number of stripes,

$$IM_{s,SW.LbL.NS} = \frac{HF_s}{N}. \quad (32)$$

Finally, the number or required couples of filters is three like in SW LbL 1S because we process one stripe at a time thus, once again, a layer at a time,

$$F_{\#,SW.LbL.NS} = 3. \quad (33)$$

3.3 Block-by-Block

The Block-by-Block approach preserves the simplicity of Classical approach while exploiting data dependencies like the other optimized solutions. The main idea is to use the Classical scheme to process blocks of the input image signal instead of the whole image itself. Even if we are using the Classical scheme, splitting the input image in blocks allows us to still exploit data dependencies. Processing a local area of the input image, we reduce the lifetime of temporary data both between horizontal and vertical processing and between two successive layers. Thus we do exploit intra-layer and inter-layer data dependencies. Like in the Classical scheme, a temporary memory, referred to as Tree Memory, is required to store these temporary samples. To avoid any blocking effect due to input signal splitting, we need an extra temporary memory, referred to as Inter-Block Memory, to process the temporary data shared by adjacent blocks.

In Figure 8 and Figure 9 we see how to apply the Block-by-Block approach respectively to 1D and 2D signals. Each block processing is extended along the whole decomposition tree and is performed according to the Classical approach. In the 2D case, considering signals wider in the horizontal dimension , we need to extend the inter-block temporary memory along the whole vertical dimension of the input signal to store temporary samples between two full columns of blocks.

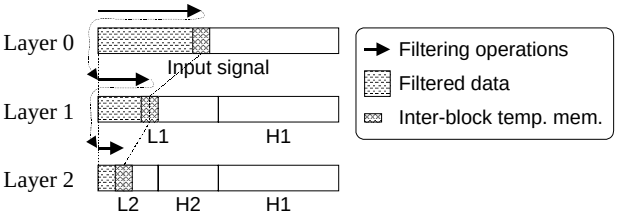

Figure 8: *1D Block-by-Block. We divide the input signal in segments that are separately processed as in the Classical approach. We need some extra memory to store inter-block temporary sample.*

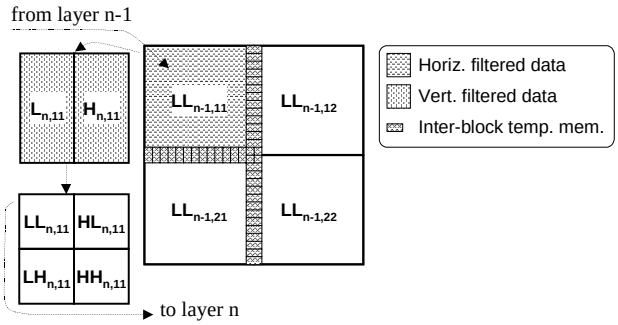

Figure 9: *2D Block-by-Block - scheme for one layer. It is the straightforward 2D extension of its 1D counterpart. We divide the input image in square blocks. Processing the blocks column by column and, in each column, row by row, we need a full input signal's height column and only a block's width row of inter-block temporary samples.*

We present the results for two solutions related to this approach, the first with external Tree Memory, referred to as Block-by-Block with External Tree Memory (BbB ETM), and the second with internal Tree Memory, referred to as Block-by-Block with Internal Tree Memory (BbB ITM). For a detailed discussion about this approach, refer to the Block Based approach discussed in [1]. The following equations are derived from [1], with N_L=1, N=H and M=(F_w-1)/2:

$$IM_{s,BbB.ETM} = 2(F_w - 1)H + (F_w - 1)\sigma_L , \quad (34)$$

$$IM_{r,BbB.ETM} = IM_{w,BbB.ETM} < \frac{4(F_w - 1)}{2^L} WH , \quad (35)$$

$$EM_{s,BbB.ETM} = \left(\frac{\alpha_L - \alpha_1}{2} + 1 \right)^2 , \quad (36)$$

$$EM_{r,BbB.ETM} = EM_{w,BbB.ETM} = \frac{5}{3} WH , \quad (37)$$

$$EB_{r,BbB.ETM} = EB_{w,BbB.ETM} = \frac{8}{3} WH , \quad (38)$$

$$IM_{s,BbB.ITM} = 2(F_w - 1)H + (F_w - 1)\sigma_L + \left(\frac{\alpha_L - \alpha_1}{2} + 1 \right)^2 , \quad (39)$$

$$IM_{r,BbB.ITM} = IM_{w,BbB.ITM} < \left(\frac{4(F_w - 1)}{2^L} + \frac{5}{3} \right) WH , \quad (40)$$

$$EM_{s,BbB.ITM} = 0 , \quad (41)$$

$$EM_{r,BbB.ITM} = EM_{w,BbB.ITM} = 0 , \quad (42)$$

$$EB_{r,BbB.ITM} = EB_{w,BbB.ITM} = WH , \quad (43)$$

where

$$\alpha_i = \frac{F_w - 1}{2}(2^i - 1) + 1 , \quad (44)$$

$$\sigma_L = \frac{F_w - 1}{2}(2^L - L - 1) + L . \quad (45)$$

Both solutions require only a couple of filters because samples are processed like in Classic approach:

$$F_{\#,BbB.ETM} = F_{\#,BbB.ITM} = 1 \quad (46)$$

4 Results

Figure 10 reports the results of the different approaches described in the previous sections for a 1600x1200 grayscale image, coded with JPEG-2000 13x7 wavelet kernel implemented with lifting-scheme, with a 6 layers Mallat tree decomposition.

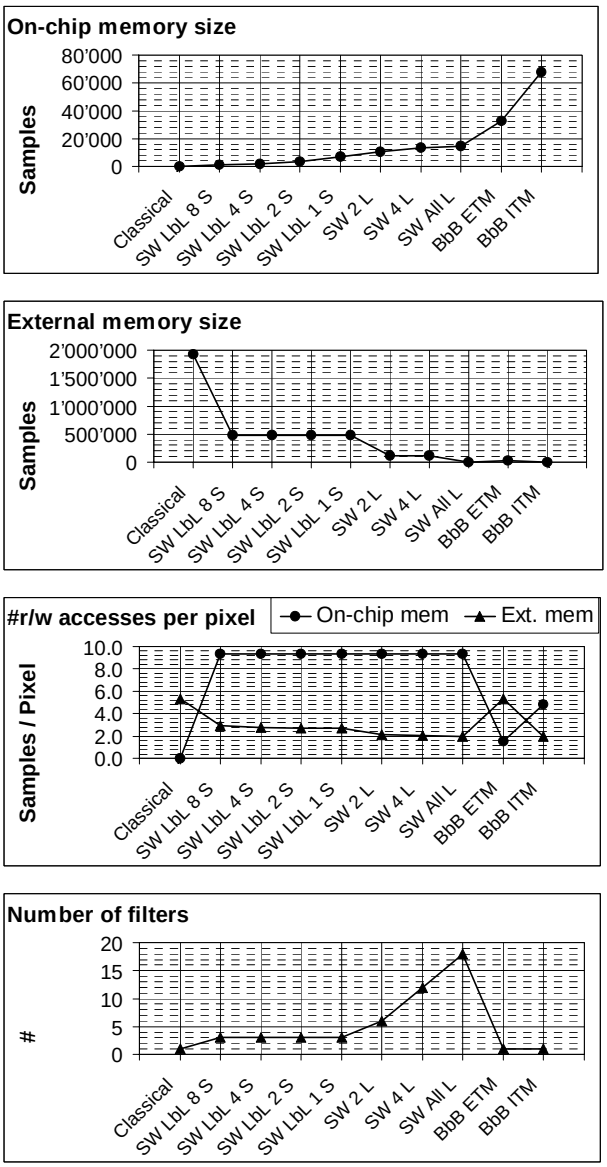

Figure 10: *Results for 1600x1200 grayscale image, JPEG2000 13x7 wavelet kernel implemented with lifting-scheme, 6 layers Mallat tree decomposition.*

5 Conclusions

The development of new strategies and new results of a joint optimization of algorithmic and architectural aspects of wavelet codec implementations allows system optimization in a variety of hardware and software configurations, differing for their requirements of on-chip and off-chip memory, both in terms of size and bandwidth, for their computational complexity and for their performance.

6 References

[1] G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, I. Bolsens, “Optimal memory organization for scalable texture codecs in MPEG-4”, to appear in IEEE Transaction on Circuits and Systems for Video Technology, special issue on SNHC coding 1999.

[2] C. Chakrabarti, M. Vishwanath, R. Owens, “Architectures for Wavelet Transforms”, VLSI Signal Processing VI,IEEE special publications, NY, pp. 507-515, 1993.

[3] M. Vishwanath, “The Recursive Pyramid Algorithm for the Discrete Wavelet Transform”, IEEE Transactions on Signal Processing, Vol. 42, No. 3, pp. 673-676, March 1994.

[4] T.C. Denk, K.K. Parhi, “Calculation of minimum number of registers in 2-D discrete wavelet transforms using lapped block processing”, IEEE Int. Symposium on Circuit and Systems, Vol. 3, pp. 77-80, London, England, May 1994.

[5] G. Lafruit, J. Bormans, “Graceful degradation parameters for a scalable wavelet codec”, ISO/IEC JTC1/SC29/WG11/MPEG97/m2655, Fribourg, October 1997.

[6] Y. Sheng, Wavelet Transform, Chapter 10 in “The Transforms and Applications Handbook”, A.D. Poularikas, CRC Press, 1996.

[7] I. Daubechies, W. Sweldens, “Factoring Wavelet Transforms into Lifting Steps”, J. Fourier Anal. Appl., Vol. 4, Nr. 3, pp. 247-269, 1998.

[8] W.Sweldens, P. Schröder, “Building your own wavelets at home”, In ``Wavelets in Computer Graphics", ACM SIGGRAPH Course Notes, pp. 15-87, 1996

7 Biographies

Massimo RAVASI was born in Lecco, Italy, on December 16, 1968. In December 1997 he received his the Dipoma in Electrical Engineering from the "Politecnico di Milano". Since October 1997 he worked as a part-time collaborator at "Laboratorio S.I.A." of the "Politecnico di Milano. Since April 1998 he joined the "Integrated Systems Center" (C3i) of the "Swiss Federal Institute of Technology" (EPFL) as a research assistant where his main activity is the development of a hardware JPEG 2000 codec.

Marco MATTAVELLI received his Diploma of Electrical Engineering from the Politecnico di Milano, Milano, Italy in March 1987. Then he joined the "Philips Research Laboratories" of Eindhoven. Main research activities regarded channel and source coding for optical recording electronic photography, and signal processing of TV and HDTV signals. Since October 1991 he joined the "Signal Processing Laboratory" (LTS) of the "Swiss Federal Institute of Technology" (EPFL) where he got his PhD degree in 1996. Then he joined the Integrated Circuit Laboratory of EPFL where he is currently Scientific Advisor. Main research interests currently are: architectures and system for video coding, the application and implementation of combinatorial optimization techniques for image analysis, and tools for the aid to architecture design of complex systems. He is also involved in ISO-MPEG standardization activities where currently chairs the MPEG Implementation Study Group.

Daniel J. MLYNEK obtained his Ph.D. degree from the University of Strasbourg, France in 1972. He joined ITT Semiconductors in 1973 as a Design Engineer for MOS circuits in the Telecommunication field. He was with ITT Semiconductors until 1989 and held several positions in the R&D, including that of the Technical Director in charge of the IC developments and the associate technologies. The main activities in the design were in the area of Digital TV Systems where ITT was a World leader an in some of the advanced HDTV concepts. He has several patents on digital TV Systems. Dr. Mlynek was awarded the Eduard Rhein Price for his innovation in signal processing principles that have been implemented in the digital TV system "Digit 2000". In June 1989, Dr. Mlynek joined the Swiss Federal Institute of Technology (EPFL), Lausanne Switzerland, where he is a Professor responsible for the VLSI Integrated Circuits.

Alistair G. BUTTAR holds a Ph.D. degree from the University of Edinburgh, and joined Motorola, Geneva in Dec. 1984. Since then, he has held several positions in IC design, system design and strategic marketing in digital TV and telecommunications, and is presently responsable for a strategic multimedia communications market development and systems R&D group. Dr. Buttar has represented Motorola in various international standards bodies and consortia including ETSI, DVB, ISO, DAVIC, WAP and the UK Digital TV Group.

Salma SOUDAGAR graduated from the Swiss Federal Institute of Technology in Lausanne (EPFL) in April 93, where she received a masters degree in Computer Science Engineering. She then joined Motorola Semiconductor Products Sector, in Geneva, Switzerland, initially in the development of IC design technologies, and for the past 2 years in the design and implementation of multimedia algorithms. Her current research interests are in the area of multimedia communications, image coding and transmission, video processing. She is also active in the ISO -JPEG committee for the definition of the next generation still image compression standard (JPEG2000), she chairs the Error Resilience subgroup of the JPEG committee.