SCHEDULING STRATEGIES FOR 2D WAVELET CODING IMPLEMENTATIONS

M. Ravasi, M. Mattavelli, D. J. Mlynek
Swiss Federal Institute of Technology, Integrated Systems Laboratory LSI,
CH-1015 Lausanne, Switzerland
Tel: +41 21 6936978; Fax: +41 21 6934663
e-mail: massimo.ravasi@epfl.ch

ABSTRACT

Wavelet image compression adopted in the JPEG-2000 and MPEG-4 standards offers several advantages over existing methods based on DCT. This paper presents some wavelet codec scheduling strategies obtained by the joint optimization of both the algorithmic part and the architectural features, according to the target system implementation. Results are presented allowing optimization of system performance either for dedicated ASIC design or for embedded software implementations based on software/hardware system resources partitioning. The optimization can target different features such as execution speed, external and internal cache memory performance, power dissipation, number of parallel wavelet filters.

1 INTRODUCTION

Texture coding based on wavelet transform is playing a leading role for its better performances in terms of signal analysis, multi-resolution 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 for still texture coding [10] 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.

Because of the sub-band decomposition of wavelet transforms, the coding/decoding process of images has to be performed on several layers as shown in Figure 1 and Figure 2 respectively for 1D e 2D case. Practical system limits encountered by the designer include memory size and bandwidth for the storage of the temporary data, with efficient use of both on-chip and off-chip storage [1, 2, 3, 4, 5, 6, 7, 8, 9].

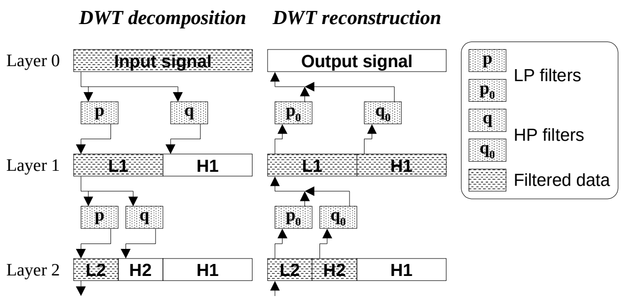

Figure 1: *1D DWT with Mallat tree decomposition. The number of samples of the encoded signal is equal to the one of the input signal.*

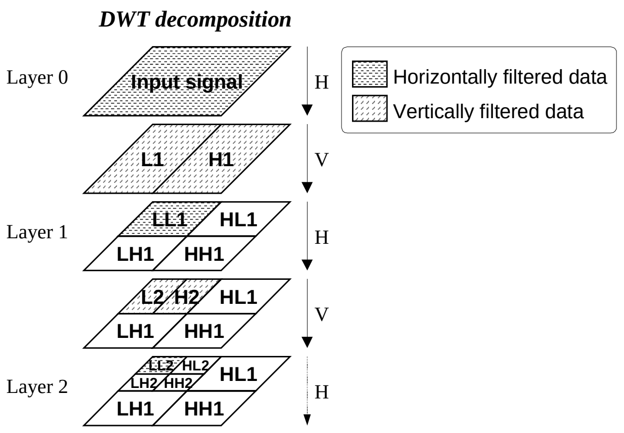

Figure 2: *2D DWT with Mallat tree decomposition. The size of intermediate layers decreases twice faster than in 1D case and the amount of data to be filtered tends asymptotically to 4/3 of the size of the input signal. Since data must be filtered both horizontally and vertically, the total amount of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

Redesigning the data processing scheduling and the memory storage scheme allows 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. 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.

2 DIFFERENT STRATEGIES FOR WAVELET CODING

2.1 Classical

The classical approach to 2D wavelet coding (see Figure 2) processes each layer in the tree decomposition separately and on each layer the vertical and horizontal processing are performed successively one by one. It's a very simple implementation but it requires high external memory bandwidth and size because a great amount of temporary data must be stored both between two successive layers and between vertical and horizontal processing.

2.2 Sliding-Windows

The main idea behind Sliding-Windows approach is to exploit data 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 3 we observe that we can immediately use the samples produced by the low-pass filter processing a layer to feed the filters on the following layer. No time gap occurs between creation and consumption of temporary samples.

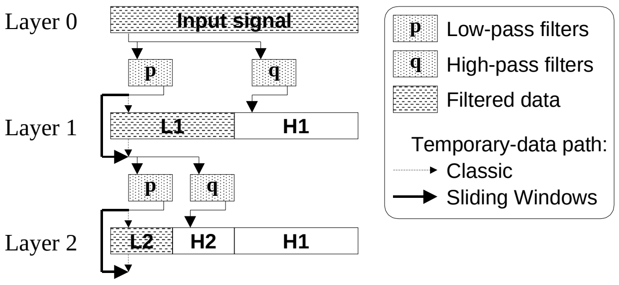

Figure 3: *1D Sliding Windows. Temporary data of intermediate layers (L1, L2, …) have not to be stored and read if they are used as a soon as they are available.*

The scheme of Figure 4 shows how to manage temporary samples between horizontal and vertical filtering to reduce their lifetime. 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. If we put the sliding-windows memory on chip, only input and output samples are exchanged with the external memory, while all the temporary samples are managed by on-chip memory.

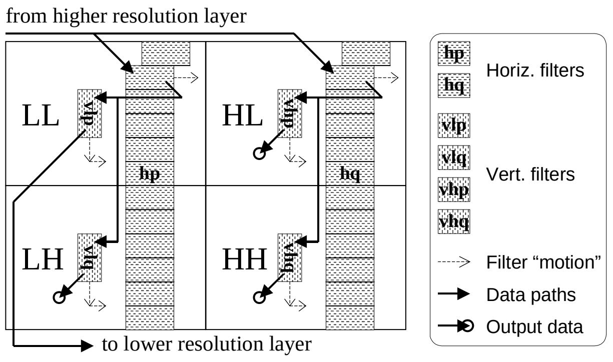

Figure 4: *2D Sliding Windows on a layer. Temporary data between horizontal and vertical filtering are managed with two "columns" of horizontal filters whose output is used to feed vertical filters.*

If we extend the scheme of Figure 3 to 2D signals, substituting each 1D layer and its corresponding filters with the scheme of Figure 4, 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. Furthermore, for each layer we need the on-chip memory for sliding windows temporary samples corresponding to the two columns of horizontal filters of Figure 4. 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 4 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 in an extra temporary memory. Since we process one layer at a time, both inter-layer temporary memory and on-chip memory can be reused layer after layer and their sizes depend on the size of the largest layer to process. On-chip memory 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).

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 relatively 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 5 for 1D signals.

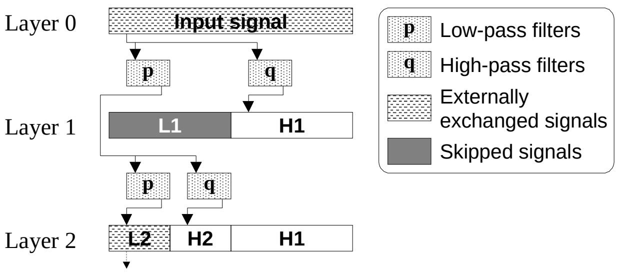

Figure 5: *1D Sliding-Windows on 2 Layers. We apply the scheme of SW LbL 1S only on 2 layers: we skip the signal L1 because we reuse output samples of first p filter as soon as they are available, but we need to store (and read) signal L2 in external temporary memory.*

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, but we need more on-chip memory, enough to process the first larger *N* layers and more filters to process *N* layers at a time. For the same reasons discussed for the SW LbL 1S case, the internal memory bandwidth results equal to both previous cases. The required external memory is equal to the size of the first externally stored LL sub-signal.

We can 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).

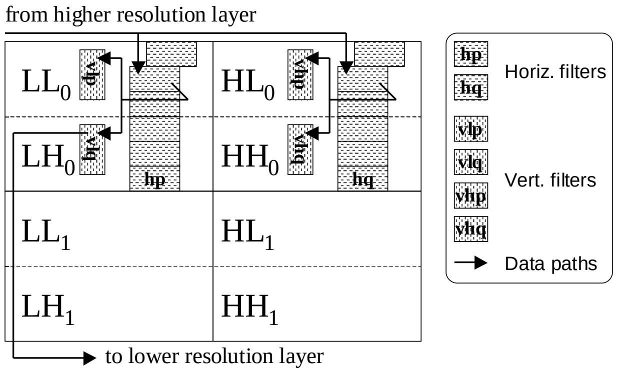

Figure 6: *2D Sliding-Windows Layer-by-Layer on 2 Stripes. We reduce the amount of internal memory processing only a stripe of input signal at a time.*

If, for instance, 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 6. To manage temporary data between successive stripes, we need an extra off-chip temporary memory yielding a corresponding small increase of external memory bandwidth.

The internal memory bandwidth does not change because samples are never filtered horizontally twice, while its size decreases with the number of stripes. 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.

2.3 Block-by-Block.

This approach still exploits inter-layer data dependencies like Sliding-Windows approach but using the Classical scheme to code blocks of image.

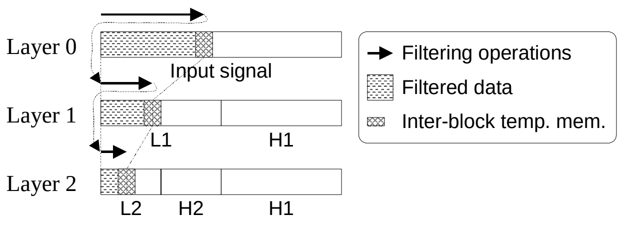

Figure 7: *1D Block by Block. The input signal is divided in segments that are separately coded as in the Classical approach. Some extra memory is required to hold inter-block temporary data.*

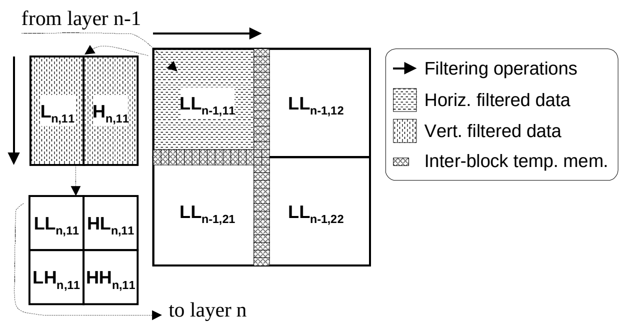

Figure 8: *2D Block by Block. The input signal is divided in blocks. It's the straight extension of the 1D coder.*

A temporary memory, referred to as tree memory, is required and an extra temporary memory is also required to manage inter-block data and avoid any blocking effect.

Two different solutions have been studied, either with off-chip or on-chip tree memory and referred to as "Block-by-Block with External Tree Memory" (BbB ETM) and "Block-by-Block with Internal Tree Memory" (BbB ITM). For a detailed discussion about this approach, refer to the Block Based approach discussed in [2]

3 RESULTS

Figure 9 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 six-layers Mallat tree decomposition. These results example can easily be extended to color images and other wavelet transform kernels, as part of a variety of image capture, transmission and display applications.

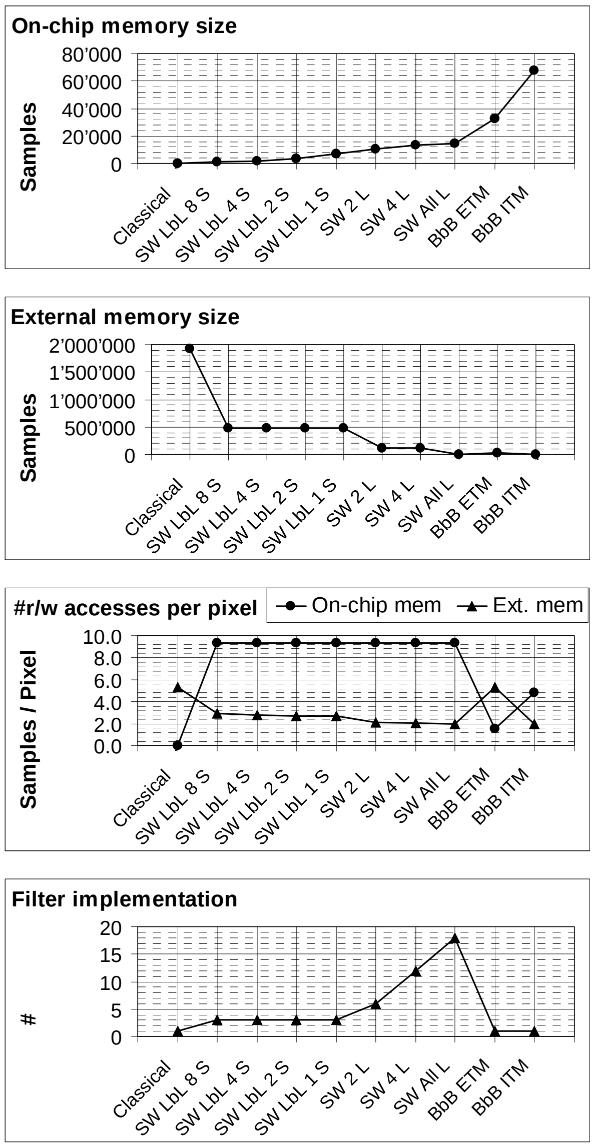

Figure 9: *Results for 1600x1200 grayscale image, JPEG2000's 13x7 wavelet kernel implemented with lifting-scheme, 6 layers Mallat tree decomposition.*

4 CONCLUSIONS

This paper reports new scheduling strategies and new results of wavelet codec implementations. Such results allow a joint optimization of algorithmic and architectural aspects yielding 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, and for their computational complexity and for their performance. Depending on the available system resources, the designer can select, for the implementation, the wavelet scheduling strategy that better matches the desired system cost-performance trade-off.

Acknowledgements

This research activity has been developed as part of a joint project in collaboration with Motorola SPS, Geneva, Switzerland.

References

[1] M. Ravasi, M. Mattavelli, D. J. Mlynek, A. Buttar, S. Soudagar, "Wavelet image compression for mobile/portable applications", IEEE Trans. on Consumer Electronics, Vol. 45, No. 3, August 1999, pag. 794-803.

[2] 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.

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

[4] 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.

[5] 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.

[6] G. Lafruit, J. Bormans, "Graceful degradation parameters for a scalable wavelet codec", ISO/IEC JTC1/SC29/WG11/MPEG97/M2655, Fribourg, October 1997.

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

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

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

[10] ISO/IEC, "Information technology – Generic coding of audio-visual objects – Part 2: Visual", 14496-2 FPDAM 1, JTC 1/SC 29/WG 11 N2802, Vancouver, July 1999.