AN INTEGER WAVELET TRANSFORM, IMPLEMENTED ON A PARALLEL TI TMS320C40 PLATFORM

Francis Decroos[1,2], Peter Schelkens[1,2], Gauthier Lafruit[2], Jan Cornelis[1], Francky Catthoor[2,3]

[1] Vrije Universiteit Brussel, Department ETRO, Pleinlaan 2, B-1050 Brussel, Belgium
[2] IMEC v.z.w., Department DESICS, Kapeldreef 75, B-3001 Leuven, Belgium
[3]Katholieke Universiteit Leuven, Department ESAT, Kardinaal Mercierlaan 94, B-3001 Leuven, Belgium

Corresponding author: Tel.: ++ 32 2 629 39 55; fax: ++ 32 2 629 28 83; e-mail: frdecroo@etro.vub.ac.be

Abstract – We present the implementation of the lifting scheme - an integer wavelet transform - on a parallel TI TMS320C40-platform. While classically, the optimization is mostly driven by arithmetic optimizations, our selected methodology uses power, speed and memory size figures to explore the design space. Among several implementation approaches, the classical row-column approach is selected, based on the memory constraints imposed by the C40 implementation platform. The use of an in-place wavelet organization eliminates the need for intermediate buffers. To minimize data transfer times, the DMA coprocessor is used to perform data transfers in parallel with the wavelet processing. The obtained implementation, fully written in parallel C-code, uses one processor to perform the wavelet decomposition/reconstruction of the luminance (Y) frames (256 by 256) and one for the horizontally subsampled chrominance (U and V) frames. Frame rates of 19.9 fps and 13.2 fps were achieved for a one level, respectively a three level 9/7 Daubechies wavelet transform.

I. INTRODUCTION

Wavelet-based image and video compression has gained in importance during recent years. These techniques, relying on the discrete wavelet transform, have an improved compression behavior by delivering a higher performance in a rate-distortion sense at different bit rates (i.e. the mean number of bits used to code one pixel). In contrast to compression algorithms based on the block-based Discrete Cosine Transform (DCT), they support lossy-to-lossless compression, progressive transmission (quality and resolution scalability) and region-of-interest coding and decoding (ROI). In this paper, we will only focus on the implementation of the wavelet transform. The implementation of the quantization and entropy encoding stages of e.g. an Embedded Zero-Tree (EZT) Wavelet coder [Shap93] is subject of other publications [Sche99a] [Sche99b].

The followed implementation approach is different from the generally introduced arithmetic optimizations of the wavelet transform [Chak93] [Parh93] [Vish94]. Since image and video processing algorithms involve high amounts of data transfers, we opted for an implementation path that considered the critical data transfer and storage (DTS). Therefore, we have adopted the lower steps of a known systematic methodology [Catt98], improving the algorithm's memory related behavior and thus also the overall performance. The specific results obtained on this application are however fully new.

This optimization approach is targeted on reducing the load on the communication channels between the different components of the system. Hence, reducing the amount of memory accesses, and even more, keeping the utilized memory as close as possible to the central data path (better exploiting the available local register files and on-chip memory), boosts the performance of the implementation in terms of low-power behavior, speed and memory size. These three implementation parameters will be of continuously growing importance in the coming years. We have explored the design space by introducing memory and speed optimizations typically requiring temporal locality of data transfer in memory. Since we aim at a parallel implementation in a DSP development environment, we did not consider power estimates, although they are strongly correlated with the memory-related behavior of the algorithm.

The utilized design space exploration can be subdivided in five main stages: (1) a global data-flow analysis, (2) global transformations (e.g. loop transformations) in order to localize memory accesses, (3) the introduction of an abstract data reuse hierarchy, (4) the effective memory allocation and assignment and finally, (5) the in-place memory optimizations to reduce the amount of memory needed. Thus, we will propose an efficient memory organization and code mapping of the wavelet transform module on a ten processor TI TMS320C40 development system (figure 1) in which we have two processors at our disposal to perform the forward wavelet-transform of a color image in YUV-format, and two for the backward transform. The other processors are needed for frame grabbing, displaying and EZT encoding and decoding (includes arithmetic coding). The parallel system is programmed with a multitasking and multithreading real-time operating system (ThreeL Diamont RTOS, formerly Parallel C), built around the Texas Instruments TMS320C4x Compiler.

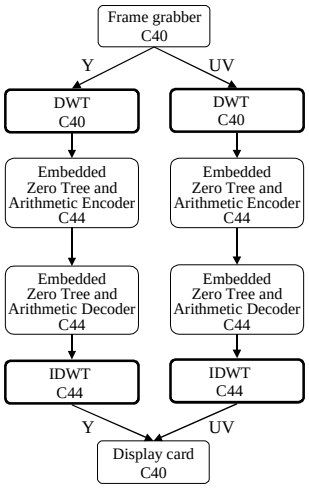


Figure 1 – The parallel DSP environment based on Texas Instruments TMS320C40 and C44 processors. The figure also illustrates the proposed distribution of the compression algorithm over the different components of the environment.

II. ALGORITHMIC DESCRIPTION

The wavelet transform is implemented using the lifting scheme [Swel95], due to the fact that this scheme offers a lossless transform. This is not the case for the floating-point fast wavelet transform, since it introduces rounding errors due to the floating-point arithmetic. Additionally, the lifting scheme, applying integer arithmetic, reveals a lower computational complexity than the classical wavelet filters due to the reuse of the data delivered by the high-pass filtering stage. Actually, the lifting scheme is a 3-step procedure, existing out of a lazy wavelet splitting stage (subdivision in even and odd samples), a prediction pass (delivering the high pass coefficients) and an update stage (delivering the low-pass coefficients) (figure 2). Important to note is that the produced data can be immediately stored back in the buffer out of which the image/wavelet coefficients were extracted. This is referred to as the in-place wavelet coefficient organization (related to the in-place memory optimization issues). Its counterpart is called the Mallat organization. The latter is the most popular decomposition organization. However, we will opt for the first one because of its interesting implementation cost properties.

The lifting scheme consists of a first step, virtually subdividing the input S_{i-1} (i indicating the wavelet level) into two subsets S_i and D_i, by extracting the even and odd numbered data samples:

$$s_{i,l} = s_{i-1,2l} \qquad d_{i,l} = s_{i-1,2l+1} \qquad (1)$$

Next, the set of samples S_i is used to predict the odd numbered samples $d_{i,l}$. However, the predicted value itself is not coded, but only the difference between the exact and the predicted value. The samples $d_{i,l}$ are replaced by the differences. In practice, those differences correspond to the high-pass wavelet coefficients.

$$d_{i,l} = d_{i,l} - \sum_k p_k s_{i,l-k} \qquad (2)$$

The update step corrects the even numbered data set S_i using the high-pass data set D_i.

$$s_{i,l} = s_{i,l} + \sum_k u_k d_{i,l-k} \qquad (3)$$

The data S_i now delivers the low-pass wavelet coefficients. The inverse transform can easily be calculated by just inverting the operations order (addition becomes subtraction, and vice versa) [Swel95].

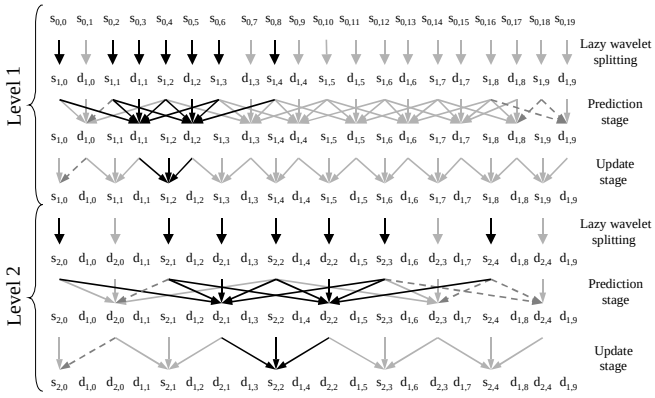


Figure 2 – The lifting scheme is a 3-step procedure, existing out of a lazy wavelet splitting stage, a prediction stage and an update stage. The prediction stage delivers the high-pass wavelet coefficients, while the update stage generates the low-pass coefficients. The illustration shows a 2-level wavelet transform (Daubechies 9-7 filters) on a 20 sample wide 1D data set. The black arrows indicate the data necessary to obtain one low-pass value in one wavelet level, the light gray arrows those for the other values. The dashed gray arrows highlight the coefficients that are considered twice during one calculation, however with different filter weights. This is due to the mirroring approach, applied to surpass the edge problems.

III. DESIGN SPACE EXPLORATION

III.1 Global transformations

In this step, we will be merely interested in localizing the memory accesses by introducing loop transformations.

With figure 2 in mind, the obvious way to obtain the wavelet transformed data set is to consecutively calculate and store: (1) the high-pass coefficients in the prediction stage and (2) all low-pass coefficients in the update stage. However, the low-pass coefficients are processed using the high-pass coefficients that are retrieved again from the background memory. By merging the two filter loops (figure 3), temporal locality is introduced, reducing the bandwidth necessary to transport the data: when enough prediction values are obtained, the next update value is calculated. Depending on the size of the prediction and update filter lengths, respectively $2M_p^F + 1$ and $2M_u^F + 1$, either situation (a) $M_p^F > M_u^F$, or situation (b) $M_p^F \le M_u^F$ is encountered (figure 3.a and b). The first case requires an extra delay between the prediction and the update stages in

order to maintain the correct input values for the prediction stage (see black arrows).

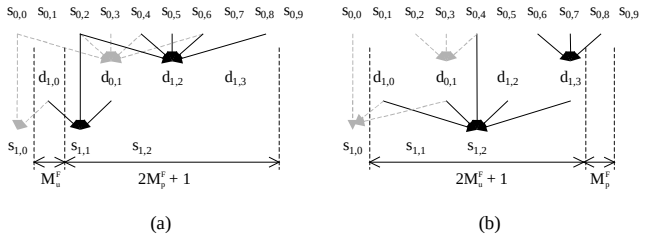


Figure 3 – Merging the prediction and the update stages increases the data locality. Depending on the size of the prediction and update filter lengths, respectively $2M_p^F + 1$ and $2M_u^F + 1$, either situation (a) $M_p^F > M_u^F$, or situation (b) $M_p^F \leq M_u^F$ is encountered. For situation (a), the prerequisite for updating $s_{1,1}$ is that $d_{1,2}$ is calculated in the prediction step. Similarly, for situation (b), the prerequisite for updating $s_{1,2}$ is that $d_{1,3}$ is calculated in the prediction step.

Apart from the previous enhancement, other improvements on the wavelet transform control and loop flow are possible. Starting with the classical implementation, i.e. filtering the image on (i) a row-column level-by-level basis (RC), other possibilities are (ii) row-row (level-by-level, RR), (iii) row-row with modified recursive pyramid algorithm (RPA) [Vish94a], (iv) block-based [Lafr99]. We will briefly discuss these techniques and refer for an elaborate discussion (VLSI implementation oriented) to [Lafr99].

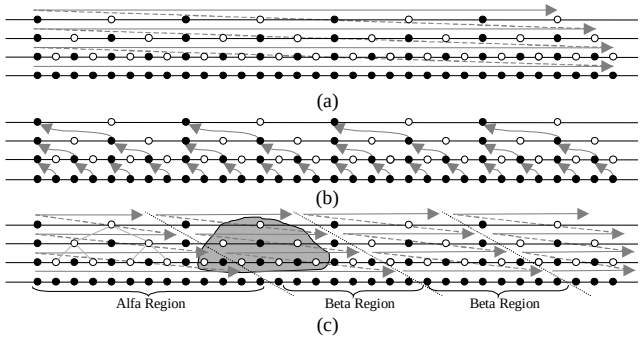


Figure 4 – 1D-Representation of the different traversal schemes, with (a) the horizontal traversal schemes (RC and RR), (b) vertical traversal schemes (RPA and modified RPA) and the mixed traversal scheme (MTA). Due to image edge effects, the first block – the α-region - is bigger than the β-region. The gray region groups the nodes required to obtain one tree for embedded zero-tree coding.

(i) The RC-technique first calculates the rows and then the columns (illustrated for the one-dimensional case in Figure 4.a). After the first level transform is processed, the second level is calculated. The disadvantage of this approach is the bursty (irregular) tree generation. For a multi-level wavelet transform, useful data is only released when the processing of the columns starts at the highest level, imposing for the EZT-module to be idle during a significant time.

(ii) The RR-approach partially solves this problem by trying to merge horizontal and vertical filtering stages. Once enough rows are processed, a first part of the columns can already be processed, resulting in an earlier data release on individual level basis. Still the output is bursty, due to the level-by-level approach.

(iii) The (modified) recursive pyramid algorithm RPA exploits the data dependency even further by initiating the higher level transforms as soon as sufficient data is released by the feeding level (Figure 4.b). This results in a band-bursty data release, i.e. data is released at constant time intervals. The first two methods (i.e. the RC and RR approaches) are addressed as horizontal traversal activation (HTA) schedules, the latter (i.e. the RPA approach) as vertical traversal activation (VTA) schedule.

(iv) The block-based method combines the advantages of the HTA and VTA approaches. It is referred to as mixed traversal activation (MTA), and generates a block-based output, which is interesting if we want to pipeline it with other modules (Figure 4.c).

Before taking a final decision, we need to browse through the succeeding steps of the methodology.

III.2 Memory Hierarchy and Data Reuse

Our aim is to minimize on-chip memory and off-chip memory access, in order to meet the constraints of real-time implementations. In the previous paragraph, we have already augmented the data locality, and in this step we will try to exploit the introduced loop transformations. If the data reuse factor is bigger than one, we will move the data up in the memory hierarchy and try to select the best solution based on memory size and memory access cost. These statistics directly relate to power and area. In a first phase, all evaluations are still on an abstract level, and no decision is taken concerning the final implementation platform. Within this step we will introduce the concept of intermediate data copies. In a second phase however, the decision was steered by the properties of the chosen implementation platform. Normally, this decision is postponed to the memory allocation and assignment step.

Depending on the wavelet transform chosen, different buffering approaches have to be applied. The RC-technique typically performs a filtering of one line (row or column), which is addressed several times. The introduction of an intermediate data copy of that line, certainly improves the implementation cost. The RR, VTA and MTA techniques require for each wavelet level an intermediate data representation, containing the currently processed data, plus two additional buffers to store the vertical and/or horizontal overlap data. Typically, for all classical methods, the original image is stored in the main buffer, and is successively updated with the filtered data. In the MTA-approach this is not the case, and data is immediately written in a rather large tree interface buffer, since edge effects cause the top and left border blocks (α-regions) to be larger than the other ones (β-regions) (Figure 4.c). An extensive study of the properties for the different

approaches, related to memory size and transfers can be found in [Sche99a][Lafr99b].

We could conclude that the RPA scheme requires the smallest amount of total memory size, but suffers from a large number of total memory accesses. The block-based scheme has less memory accesses but at the cost of a higher memory size. The RC-technique has the least memory accesses but needs the largest memory size. Using the in-place property of the wavelet transform however, we were able to reduce this amount of memory dramatically (see § III.4.1).

The RC-technique requires an intermediate data copy (in on-chip memory) of the currently processed line or column (figure 5) to reduce the amount of memory transfers to the off-chip memory. All the other schemes require the storage of extra lines and/or blocks in the on-chip memory, raising the required cache size far above the available 2 kWord. Seen the limited availability of on-chip memory for the TI TMS320C40, we selected the RC-technique. Although rejected here on the basis of memory constraints of the implementation platform, the other techniques remain attractive alternatives for ASICs and processors with larger on-chip memories.

As explained in previous paragraph, the RC-scheme only starts releasing complete trees, when processing the highest level of the wavelet pyramid. This introduces an important, fixed delay in the pipeline of the EZW coder and decoder.

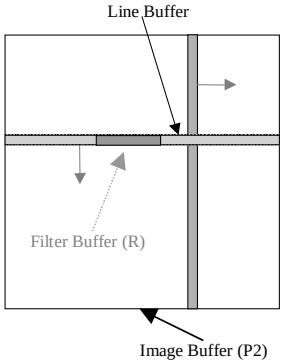


Figure 5 – Different buffers necessary to perform the wavelet transform for the RC-method.

III.3 Memory Allocation/Assignment

The memory used by a C program is divided into four logical areas, which are mapped at load time into the available physical memories of the implementation environment. The logical areas are (1) the stack storage, (2) the heap storage, (3) the static storage and (4) the code storage [Thre95]. The physical memories are the two on-chip memory blocks of 1 kWord (32-bit) featuring separate address busses and two off-chip EDRAM blocks of 1 MWord (C40) or 512 kWord (C44). The first two are capable of handling two reads or one write during each processor cycle, while the latter two possess separated address busses and a multiplexed data bus, allowing two reads or one write for both off-chip memories, during each cycle [SPRU96]. Even though, when page misses do occur, throughput is decreased.

In the ThreeL Parallel C programming environment, one can choose to define data objects as (1) arrays on the stack, (2) dynamically allocated on the heap, (3) static or (4) assigned to an address (e.g. the start address of an on-chip area). After compilation, a configuration process guided by user directives inserts mapping information into the application file. The bootstrap and loading software uses this information to map the logical areas into the available memories.

In the CPU core, two data operands from the processor data bus are accessed in one cycle by two dedicated CPU busses and released for the multiplier, ALU or register file. At the same time, two register busses are able to deliver two data values from the register file to the multiplier or the ALU.

The temporal locality introduced in paragraph III.2 does not assure that the compiler will keep two calculated high-pass values in registers near the CPU in order to use them immediately in the calculation of the low-pass value. On the contrary, the values will be stored and retrieved again. A way to impose register usage is to store the high-pass values in temporal variables (see the intermediate example code in figure 6). When programming C, one has to rely on the compiler to assign temporary variables to registers. Only hand coding in assembly ensures full user control of the assignment to registers (only directives are possible in C). If however not enough registers are available (e.g. small register files), a significant influence on the performance will be noticed. This becomes apparent for the inverse transform, where four supplementary variables are needed to store the values from the inverse update step in order to calculate the inverse prediction step.

```
for(row=edge;row<h+edge;row++)
{
  CopyImageToLine(image,line);
  x=edge-1;
  dcol=edge-1;
  // Prediction step
  high[row][dcol]=line[x]-
       nearest(HP_3*line[x-3] +
               HP_1*line[x-1] +
               HP1 *line[x+1] +
               HP3 *line[x+3],HS);
  dcol++;
  for(x=edge+1;x<w+edge;x+=2)
  {
    // Prediction step
    high[row][dcol]=line[x]-
       nearest(HP_3*line[x-3] +
               HP_1*line[x-1] +
               HP1 *line[x+1] +
               HP3 *line[x+3],HS);
    // Update step
    low[row][dcol]=line[x-1]+
       nearest(LP_1*line[x-2] +
               LP1 *line[x],LS);
    dcol++;
  }
}
```

(a)

```
for(row=edge;row<h+edge;row++)
{
  CopyImageToLine(image,line);
  x=edge-1;
  dcol=edge-1;
  // Prediction step
  tmp1=line[x]-
       nearest(HP_3*line[x-3] +
               HP_1*line[x-1] +
               HP1 *line[x+1] +
               HP3 *line[x+3],HS);
  dcol++;
  for(x=edge+1;x<w+edge;x+=2)
  {
    // Prediction step
    tmp2= line[x]-
       nearest(HP_3*line[x-3] +
               HP_1*line[x-1] +
               HP1 *line[x+1] +
               HP3 *line[x+3],HS);
    high[row][dcol]=tmp2;
    // Update step
    low[row][dcol]=line[x-1]+
       nearest(LP_1*tmp1+
               LP1 *tmp2,LS);
    tmp1=tmp2;
    dcol++;
  }
}
```

(b)

Figure 6 : Simplified code example with (a) the original and (b) the modified implementation using temporary variables.

III.4 In-place Optimisations

Some extra optimizations can further reduce the amount of consumed memory and speed up the implementation.

III.4.1 In-place Wavelet Organisation

During the prediction and update stages the odd/even sample data can be replaced by the calculated prediction and update values (see §II). This in-place wavelet organization [Swel95] is generally considered as an inter-array storage order optimization (figure 7) in the context of the DTS-methodology of [Catt98]. The advantage is that the obsolete memory units (i.e. the locations where the update coefficients are stored) can be reused to store the newly calculated data (i.e. output from succeeding prediction and update stages), seen the irrelevance of the ancient data during the further calculation of the wavelet tree.

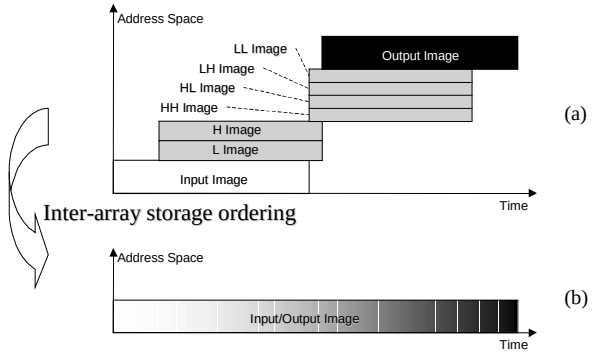


Figure 7 – Required memory size and buffers in case of the Mallat-organization (a) and the in-place Sweldens-organization (b) where the number of buffers is reduced to one single buffer of input image size.

III.4.2 Line Buffering

The data transfer time from the off-chip memory to the on-chip memory (line buffer) and backwards, has to be considered too. Therefore, it is certainly useful that while one line is processed, the next line is already copied into the on-chip memory, and the previous processed line is copied back to the off-chip memory partition (figure 8). This is an application of the data reuse decision step of our methodology [Catt98] for which an efficient search space exploration has been performed. For example, line i is copied from off-chip memory into buffer 1 during a first timeframe, then processed in the next timeframe and finally copied back to the off-chip memory. In the meanwhile, during the first timeframe, buffer 2 is processed and buffer 3 is copied into the off-chip image buffer. Of course, these "background" transfers should not interfere with the CPU. In our development environment, the parallel DMA-processor of the TMS320C40 was exploited for this task.

III.4.3 Frame buffering

The previous data reuse technique has also been applied on the frame level for inter-chip frame buffering. Since the parallel development system does not offer shared memory, complete frames had to be moved continuously from one processor to another through slow inter-processor communication ports. Nevertheless, it turned out to be possible to increase the throughput of the entire application by carrying out these frame transfers in parallel with the wavelet transform.

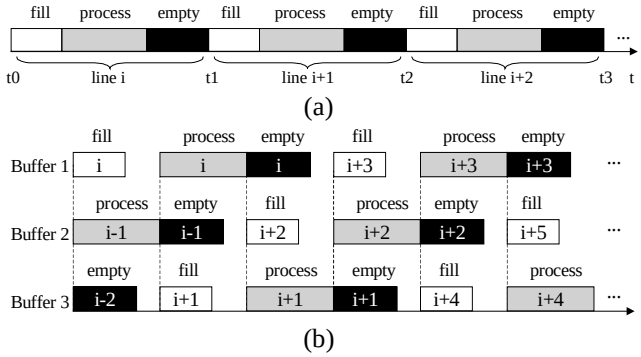


Figure 8 – (a) Sequential line buffering: the line buffer is filled before and emptied after processing while in (b) parallel line buffering, lines are processed while the previous processed line is stored into an off-chip image buffer and the next one is retrieved from another off-chip image buffer.

As a result, this has lead to additional input and output buffers. The transport of the frame buffers is again performed by the DMA-coprocessor. We have chosen a setup with four frame buffers (figure 9). In parallel, the following action had to take place in the different buffers: (1) receive an image in buffer 1 via the communication port from the frame grabber, (2) use buffer 2 as input for the wavelet transform, (3) use buffer 3 as output for the wavelet transformed data, (4) send buffer 4 to the next processor port. If more than one wavelet level needs to be calculated, buffer 3 is used as input and output buffer for the levels higher than 1 (No bottleneck occurs since for the next wavelet levels the input image is successively subsampled by a factor 4). In the next timeframe, the function of the buffers is permutated, as it is the case for the line buffering. Note also, that although the latency (i.e. the difference between image capture and display), caused by the pipeline structure remains unaffected by carrying out frame transfers in parallel with the wavelet transform, the frame rate is increased. Nevertheless, the capabilities of the DMA-coprocessor are limited since it has single data and address busses. Thus, the balancing of the four parallel DMA-processes and the CPU is a critical issue in relation to the obtained performance.

III.5 Memory organization

The resulting memory organization for the considered processor is shown in figure 10. The four frame buffers are stored in local off-chip memory and the three line buffers in the on-chip memory. Remark that the frame buffers are contained by two off-chip memory partitions, having a common data bus but separate address busses.

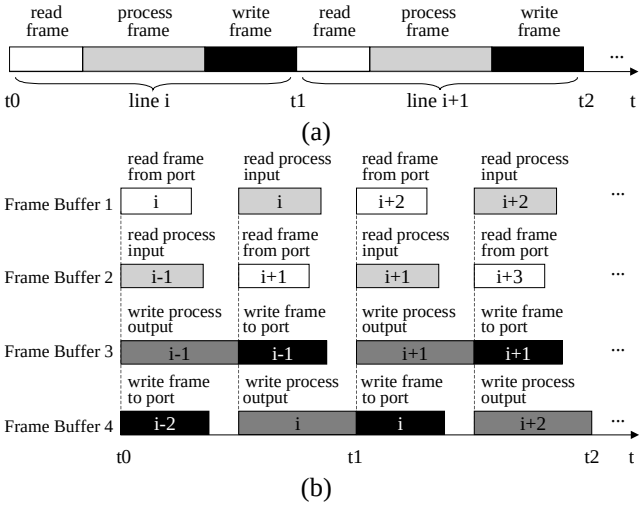


Figure 9 – (a) Sequential frame buffering where a frame has to be read from a communication port before processing and the result is written to another port while in (b) parallel frame buffering, processing goes on while a next frame is read and a previous one is stored.

Finally, the system was implemented for the transformation of color images. The image format is YUV.Since two processors were available, one was appointed to transform the Y-frame and the other to transform the chrominance frames. Since the latter are subsampled before processing, the load of the second processor was not significantly higher than the load of the first one.

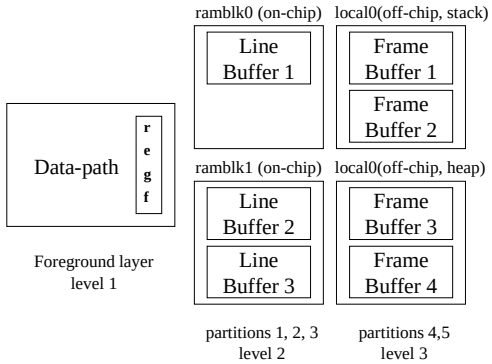
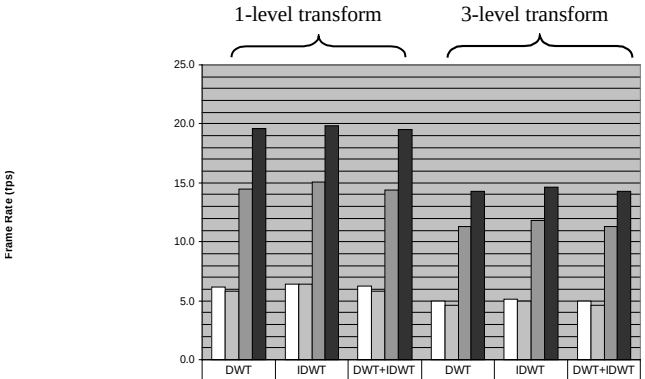


Figure 10 – Final memory hierarchy with three hierarchy levels and different memory partitions which are spread over the available physical memory partitions.

IV. RESULTS

The obtained frame rates for the forward and inverse wavelet transform are shown in Figure 11. The results are displayed for both a 1-level and a 3-level wavelet transform and a frame size of 256-by-256 pixels (YUV-color image). Remark that the introduction of the line buffers in the sequential mode initially did reduce the frame rate slightly, due to the increased number of memory transfers. The time gained by localizing the memory transfers is lost again by the extra copy operations. However, paralleling the data transfer then becomes feasible by exploiting the DMA-channels (line and frame buffering). This allowed a performance boost above a factor three. Frame rates of 19.9 fps for a 1-level transform, and 14.3 fps for a 3-level transform were obtained. However, the drawback of those extra buffers is the increased amount of needed on-chip and off-chip memory.



	DWT	IDWT	DWT+IDWT	DWT	IDWT	DWT+IDWT
Original lifting scheme	6.2	6.4	6.2	5.0	5.1	5.0
Line buffer	5.8	6.4	5.8	4.6	5.0	4.6
Triple line buffer (DMA-access)	14.5	15.1	14.4	11.3	11.8	11.3
Quadruple frame buffer (DMA-access)	19.6	19.9	19.5	14.3	14.6	14.3

Figure 11 - Achieved frame rates on two TI TMS320C40-processors for a 1- and 3-level wavelet transform (256x256 YUV frames; DWT: discrete wavelet transform; IDWT: inverse DWT).

V. CONCLUSIONS

A 3-level wavelet transform for 256x256 YUV color images was implemented on 2 TI TMS320C40-processors. Due to the limited amount of available on-chip memory, the design space exploration resulted in the selection of the classical row-column approach. By exploiting the DMA-facilities of the processor, a satisfactory (near real-time) performance was obtained (14,3 fps for a 3 level wavelet transform).

ACKNOWLEDGEMENTS

The presented work results from the research carried out in the framework of the IWT-IT - TeleVision project, funded by the IWT (Belgium).

REFERENCES

[Catt98] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, A. Vandecappelle, "Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design", Kluwer Academic Publishers, Boston, 1998.

[Chak93] C. Chakrabarti, M. Vishwanath, R. Owens, "Architectures for Wavelet Transforms", VLSI Signal Processing VI, IEEE special publications, New York, pp. 507-515, 1993.

[Lafr99a] G. Lafruit, P. Schelkens, J. Bormans, "Proposal of weighting factors for Scalable Texture Objects", MPEG-4 Meeting, Seoul, ISO/IEC JTC1/SC29/WG11 MPEG98/M4454, March 1999

[Lafr99b] G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, I. Bolsens, "Optimal Memory Organization for Scalable Texture Codecs in MPEG-4", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 9, No. 2, pp. 218 –243, March 1999.

[Mall89] S. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.11, No.7, pp. 674-693, 1989.

[Parh93] K.K. Parhi, T. Nishitani, "VLSI architectures for Discrete Wavelet Transforms", IEEE Transactions on VLSI Systems, Vol.1, No.2, pp. 191-202, 1993.

[Sche99a] P. Schelkens, G. Lafruit, F. Decroos, J. Cornelis, F. Catthoor, "Power Exploration For Embedded Zero-Tree Wavelet Encoding", IRIS internal report TR 0057, Vrije Universiteit Brussel, 1999.

[Sche99b] P. Schelkens, F. Decroos, G. Lafruit, F. Catthoor, J. Cornelis, "Efficient Implementation of Embedded Zero-Tree Wavelet Encoding", accepted for IEEE International Conference on Electronics, Circuits and Systems (ICECS'99), Paphos, Cyprus, September 5-8, 1999.

[Shap93] J.M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients", IEEE Transactions on Signal Processing, Vol. 41, No.12, pp. 3445-3462, 1993.

[SPRU96] TMS320C40 User Guide, Texas Instruments Inc., Houston,1996.

[Swel95] W. Sweldens, "The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions", SPIE Conference 1995, Vol. 2569, pp. 68-79, 1995.

[Thre95] ThreeL parallel C User Guide V2.0, 3L Ltd., Edinburgh, pp. 385-394,1995.

[Vish94] M. Vishwanath, "The Recursive Pyramid Algorithm for the Discrete Wavelet Transfer", IEEE Transactions on Signal Processing, Vol. 42, No. 3, pp. 673-676, 1994.