

Performance comparison of two-dimensional discrete wavelet transform computation schedules on a VLIW digital signal processor

K. Masselos, Y. Andreopoulos and T. Stouraitis

Abstract: The two-dimensional discrete wavelet transform (2D DWT) is becoming one of the standard tools for image and video compression systems. Various input-traversal schedules have been proposed for its computation. Here, major schedules for 2D DWT computation are compared with respect to their performance on a very long instruction-word (VLIW) digital signal processor (DSP). In particular, three popular transform-production schedules are considered: the row-column, the line based and the block based. Realisations of the wavelet transform according to the considered schedules have been developed. They are parameterised with respect to filter pair, image size and number of decomposition levels. All realisations have been mapped on a VLIW DSP, as these processors currently form an attractive alternative for the realisation of signal, image and video processing systems. Performance metrics for the realisations for a complete set of parameters have been obtained and compared. The experimental results show that each realisation performs better for different points of the parameter space.

1 Introduction

Because of its capability of providing a time-frequency representation of the signal (required when the time localisation of the frequency components is needed), the discrete wavelet transform (DWT) has been widely used in several signal processing applications. The DWT passes the time-domain signal through a multirate structure of high-pass and low-pass filters. In two dimensions, the classical design of this process is performed along the rows and columns of the input signal [1], as shown in Fig. 1. A multilevel (multirate) decomposition consists of iterating the design of Fig. 1 for the row-wise and column-wise low-pass filtered data. In general, two approaches exist for the realisation of the filtering: the conventional convolution-based implementation [1] and the lifting-based implementation [2], which employ both low-pass and high-pass filters through a ladder-structure of adders and multipliers. For optimised implementations, the lifting-based implementation is generally preferable as it reduces the required arithmetic operations [2].

One application area, where the 2D DWT has demonstrated good algorithmic performance, is image

compression [3]. This fact resulted in the inclusion of the DWT into image and video compression standards, namely JPEG-2000 [4] and MPEG-4 (visual texture coding) [5]. In this work, the lifting implementation of two filter-pairs included in JPEG-2000 was considered, namely the 5/3 and 9/7 filter pairs. The lifting equations of the 9/7 filter-pair are [4]

$$Y_{2n+1}^1 = X_{2n+1} + \alpha \times (X_{2n} + X_{2n+2}) \quad (1)$$

$$Y_{2n}^1 = X_{2n} + \beta \times (Y_{2n-1}^1 + Y_{2n+1}^1) \quad (2)$$

$$Y_{2n+1}^2 = Y_{2n+1}^1 + \gamma \times (Y_{2n}^1 + Y_{2n+2}^1) \quad (3)$$

$$Y_{2n}^2 = Y_{2n}^1 + \delta \times (Y_{2n-1}^2 + Y_{2n+1}^2) \quad (4)$$

$$Y_{2n+1}^3 = K \times Y_{2n+1}^2 \quad (5)$$

$$Y_{2n}^3 = \frac{Y_{2n}^2}{K} \quad (6)$$

The approximated values used for the constants are: $\alpha = -1.586134342$, $\beta = -0.052980118$, $\gamma = 0.882911075$, $\delta = 0.443506852$ and $K = 1.230174105$.

The 1D application of the transform occurs as follows: First, (1) and (2) are applied to the input samples of each decomposition level $\{\dots, X_{2n-1}, X_{2n}, X_{2n+1}, X_{2n+2}, \dots\}$. The output coefficients $\{\dots, Y_{2n-1}^1, Y_{2n}^1, Y_{2n+1}^1, Y_{2n+2}^1, \dots\}$ are further modified (lifted) by (3) and (4) into coefficients $\{\dots, Y_{2n-1}^2, Y_{2n}^2, Y_{2n+1}^2, Y_{2n+2}^2, \dots\}$. Finally, a scaling operation formulates the final values (5) and (6). Each pair of equations formulates a predict-and-update step. In practice, the final scaling step of (5) and (6) is incorporated within (3) and (4), leading to $S = 2$ passes for the application of the lifting transform. In two dimensions, the transform is separately applied along the input rows and columns of each decomposition level.

© IEE, 2006

IEE Proceedings online no. 20045218

doi:10.1049/ip-vis:20045218

Paper first received 20th October 2004 and in revised form 6th June 2005

K. Masselos is with the Department of Electrical and Electronic Engineering, Imperial College, Exhibition Road, South Kensington SW7 2BT, UK

Y. Andreopoulos is with the Department of Electrical Engineering (EE), University of California Los Angeles, 54-147 Engineering IV Building, 420 Westwood Plaza, Los Angeles, CA 90095-1594, USA

T. Stouraitis is with the Department of Electrical and Computer Engineering (ECE), University of Patras, Rio 26500, Greece

E-mail: yandreop@ee.ucla.edu

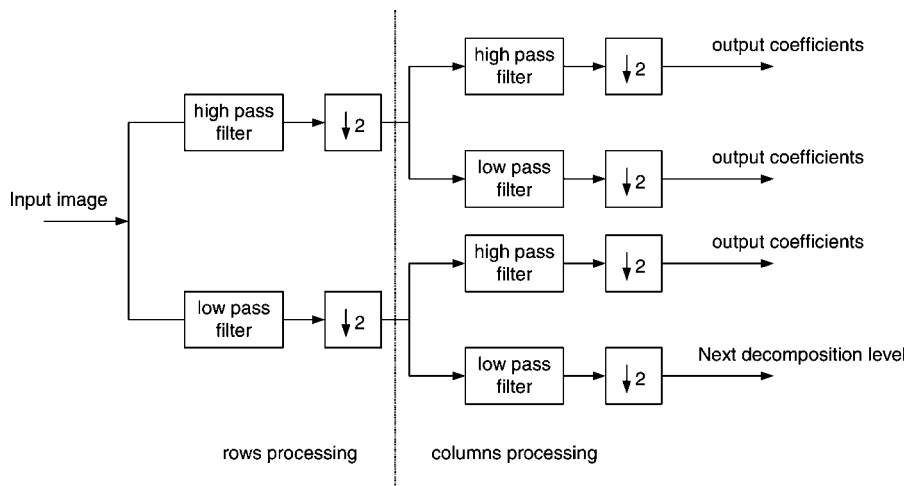


Fig. 1 One level of the 2D wavelet decomposition

Concerning the 5/3 filter pair, the lifting equations are [4]

$$Y_{2n+1}^1 = X_{2n+1} - \left\lfloor \frac{X_{2n} + X_{2n+2}}{2} \right\rfloor \quad (7)$$

$$Y_{2n}^1 = X_{2n} + \left\lfloor \frac{Y_{2n-1}^1 + Y_{2n+1}^1 + 2}{4} \right\rfloor \quad (8)$$

Notice that the 5/3 filter pair is applied in $S = 1$ pass and the floor operator $\lfloor g \rfloor$ provides an integer-to-integer (reversible) transform [4].

Because of its computational complexity, early proposals for the realisation of the wavelet transform called for application-specific integrated circuit (ASIC) realisations [6, 7]. Process technology and architecture developments led to programmable (instruction set) digital signal and multimedia processors with high clock rates and processing capabilities (in terms of number of operations executed per second). An important advantage of (domain specific) digital signal processors (DSPs) compared to ASICs and general-purpose microprocessors is the good balance between flexibility and implementation efficiency as shown in Fig. 2. Flexibility is related to the capability of given hardware resources to execute different algorithms/tasks. Flexibility can be exploited both for post-shipment functionality upgrading of electronic systems and area/size optimisation through sharing of the same hardware resources among various tasks. Implementation efficiency is related to the silicon area and power required for the execution of a task within specific performance constraints.

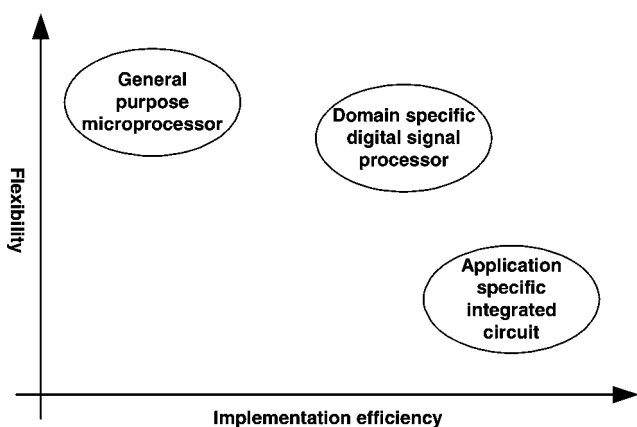


Fig. 2 Trade-off between implementation efficiency and flexibility for various implementation options

Furthermore, the application development cycles for DSPs are far smaller than those for ASICs. For the above reasons (domain specific), DSPs (e.g. Philips TriMedia, Texas Instruments TMS320C6x) have been representing an attractive alternative for the realisation of (multidimensional) signal processing systems for quite sometime [8].

In this paper, a comparison of the performance of various schedules for the computation of the 2D DWT on a programmable very-long instruction-word (VLIW) DSP is presented. Comparisons of 2D DWT computation schedules have been presented in [9–11]. The comparison presented in [9] is abstract and does not take into consideration any implementation-related parameters. The comparison of [10] is based on a theoretical analysis, which is ASIC oriented. Some implementation parameters mainly related to memory-organisation are taken into consideration. However, the analysis stops at a high level and important issues, such as interconnect and control, are not taken into account. Furthermore, issues related to programmable architectures, such as hardware-controlled caches, are not taken into consideration. As a result, the analysis of [10] is not easily applicable to such architectures. In [11], a comparison of wavelet transform schedules on programmable architectures is presented. The various schedules are compared with respect to data-cache performance, which is related to overall performance but not always in a straightforward manner. Furthermore, the comparison is performed (a) on a simple scalar-processor simulator and (b) on a general-purpose processor. Consequently, although well suited for such type of algorithms, a DSP was not considered in the analysis of [11]. The major contribution of our work is in filling this gap by presenting the performance comparison of wavelet transform computation schedules on a real VLIW DSP architecture that forms an attractive alternative for the realisation of multimedia-processing algorithms.

2 Wavelet transform computation schedules

Three major computation schedules have been proposed for the 2D DWT, namely the row-column (RC) [11], the line-based (LB) [12] and the block based (BB) [13, 14]. The RC approach exploits the fact that the 2D DWT is a separable transform and thus it can be computed along the rows and columns of each decomposition level. This was shown in Fig. 1. According to the RC computation schedule, the transform proceeds to the next decomposition level only when all the input data to the current level have been processed [1].

The pseudocode for the RC schedule is shown in Fig. 3. Based on the analysis of [11], we assume that the input image is stored in array IMG (of size $N_1 \times N_2$) and two line-buffers are used (TMP_IMG_1 and TMP_IMG_2) to perform the filtering operations on each row or column. The final result is stored back in array IMG with coefficient re-ordering [11], that is, separating the low- and high-frequency wavelet coefficients in two sets. In the pseudocode of Fig. 3, operator $A \rightarrow B$ assigns values taken from function, array or variable A to array or variable B , whereas $A \leftarrow B$ exchanges the memory pointers of A and B , thus, in the rest of the execution, B denotes the memory area of A and vice versa. For simplicity, the reading or writing from/to sequential array positions is sometimes denoted with $[\cdot]$, whereas $[a:b:c]$ indicates array accessing from position a to position c with step b . The design of Fig. 3 is parametrical to the number of predict-and-update passes (S) through each row or column for the application of the lifting-based transforms shown in Section 1, the input-image dimensions ($N_1 \times N_2$ pixels) and the number of decomposition levels ($LEVELS$). Thus, the $Apply_lifting()$ function of Fig. 3 applies the equations of one predict-and-update lifting step, that is, (7) and (8) for $S = 1$ and (1)–(6) for $S = 2$. In practice, special treatment (mirroring) is applied at the image borders, in order to initiate and terminate the application of the transform [4].

Both LB [12] and BB [13] approaches have been proposed to improve the issues of memory utilisation and memory-access locality of the conventional RC approach. The LB-approach operates with a streaming input of non-overlapping sets of input image lines to produce the wavelet coefficients at all the decomposition levels. The BB-approach operates with a raster scan of the input image with non-overlapping blocks and produces each block's wavelet coefficients at all the decomposition levels. As a result, the main difference between the two methods is the selected image traversal method (based on complete image rows or on blocks).

The pseudocode for the LB and BB schedules is presented in Figs. 4 and 5, respectively. The general flow of our design is following the architecture introduced in [14]. Initially, each new input component is written from the input image (IMG) to an interpass memory (IPM of Fig. 4). For the BB-design of Fig. 5, IPM is a 2D array of $2^{LEVELS} \times 2^{LEVELS}$ samples (or wavelet coefficients); for the LB-design (Fig. 4), a 2D array of size $2^{LEVELS} \times N_2$ is used. Whenever the filtering is interleaved, the intermediate

```

For [0:1:LEVELS-1] → L
{
  For [0:1:N1/2L-1] → row
  {
    IMG[row][·] → TMP_IMG1[·]
    For [1:1:S] → lift_pass
    {
      For [0:2:N2/2L-1-1] → col
      {
        Apply_lifting(lift_pass, TMP_IMG1[·], col) → TMP_IMG2[col:col+1]
        TMP_IMG1 ← TMP_IMG2
      }
      TMP_IMG1[0:2:N2/2L-2-1] → IMG[row][0:1:N2/2L+1-1] /* R */
      TMP_IMG1[1:2:N2/2L-1-1] → IMG[row][N2/2L+2:1:N2/2L-1] /* R */
    }
  }
  For [0:1:N2/2L-1] → col
  {
    IMG[·][col] → TMP_IMG1[·]
    For [1:1:S] → lift_pass
    {
      For [0:2:N1/2L-1-1] → row
      {
        Apply_lifting(lift_pass, TMP_IMG1[·], row) → TMP_IMG2[row:row+1]
        TMP_IMG1 ← TMP_IMG2
      }
      TMP_IMG1[0:2:N1/2L-2-1] → IMG[0:1:N1/2L+1-1][col] /* R */
      TMP_IMG1[1:2:N1/2L-1-1] → IMG[N1/2L+2:1:N1/2L-1][col] /* R */
    }
  }
}

```

Fig. 3 RC computation schedule pseudocode

The comment $/*R*/$ denotes that the data are written in the destination matrix in reordered form, that is, separating the low- and high-frequency wavelet coefficients

```

For [0:2LEVELS:N1-2LEVELS-1] → six
{
  IMG[six:1:six+2LEVELS-1][0:1:N2-1] → IPM[0:1:2LEVELS-1][0:1:N2-1]
  1 → cstep;
  For [0:1:LEVELS-1] → L
  {
    For [0:cstep:2LEVELS-1] → row
    {
      For [0:cstep:N2-1] → col
      {
        IPM[row][2·col·cstep:cstep:(2·col+1)·cstep] → FF1[·]
        For [1:1:S] → lift_pass
        {
          Apply_lifting(lift_pass, FF1[·], 0) → FF2[·]
          FF1 ← FF2
        }
        FF1[·] → IPM[row][2·col·cstep:cstep:(2·col+1)·cstep]
      }
    }
    For [0:cstep:N2-1] → col {
      OM_COLS[L][col/cstep][·] → FF1[·]
      For [0:cstep:2LEVELS-1] → row
      {
        IPM[2·row·cstep:cstep:(2·row+1)·cstep+1][col] → FF1[·]
        For [1:1:S] → lift_pass
        {
          Apply_lifting(lift_pass, FF1[·], 0) → FF2[·]
          FF1 ← FF2
        }
        FF1[·] → IPM[2·row·cstep:cstep:(2·row+1)·cstep+1][col]
      }
      FF1[·] → OM_COLS[L][col/cstep][·]
    }
    cstep·2 → cstep;
    IPM[1:cstep:2LEVELS-1][1:cstep:N2-1] → ext_memory /*HFCE*/
  }
  IPM[0][0] → ext_memory /*LFCE*/
}

```

Fig. 4 Pseudocode for the LB computation schedule

The comments $/*LFCE*/$ and $/*HFCE*/$ denote low- and high-frequency coefficient extraction to external memory, respectively

results in the row or column direction of every decomposition level are stored in overlap memories, denoted as OM_ROWS and OM_COLS . Conversely, whenever the filtering restarts with a neighbouring component (block or group of lines), the coefficients stored in the overlap memories are reloaded.

```

For [0:2LEVELS:N1-2LEVELS-1] → six
{
  0 → omcols_offset
  For [0:2LEVELS:N2-2LEVELS-1] → siy
  {
    IMG[six:1:six+2LEVELS-1][siy:1:siy+2LEVELS-1] → IPM[·][·]
    1 → cstep; 2LEVELS → ipm_hs
    For [0:1:LEVELS-1] → L
    {
      For [0:cstep:2LEVELS-1] → row
      {
        OM_ROWS[L][row][·] → FF1[·]
        For [0:cstep:2LEVELS-1] → col
        {
          IPM[row][2·col·cstep:cstep:(2·col+1)·cstep] → FF1[·]
          For [1:1:S] → lift_pass
          {
            Apply_lifting(lift_pass, FF1[·], 0) → FF2[·]
            FF1 ← FF2
          }
          FF1[·] → IPM[row][2·col·cstep:cstep:(2·col+1)·cstep]
        }
      }
      FF1[·] → OM_ROWS[L][row][·]
    }
    For [0:cstep:2LEVELS-1] → col {
      OM_COLS[L][omcols_offset+2LEVELS-L+1+col][·] → FF1[·]
      For [0:cstep:2LEVELS-1] → row
      {
        IPM[2·row·cstep:cstep:(2·row+1)·cstep][col] → FF1[·]
        For [1:1:S] → lift_pass
        {
          Apply_lifting(lift_pass, FF1[·], 0) → FF2[·]
          FF1 ← FF2
        }
        FF1[·] → IPM[2·row·cstep:cstep:(2·row+1)·cstep][col]
      }
      FF1[·] → OM_COLS[L][omcols_offset+2LEVELS-L+1+col][·]
    }
    cstep·2 → cstep; omcols_offset+1 → omcols_offset
    IPM[1:cstep:2LEVELS-1][1:cstep:2LEVELS-1] → ext_memory /*HFCE*/
  }
  IPM[0][0] → ext_memory /*LFCE*/
}

```

Fig. 5 Pseudocode for BB computation schedule

The comments $/*LFCE*/$ and $/*HFCE*/$ denote low- and high-frequency coefficient extraction to external memory, respectively

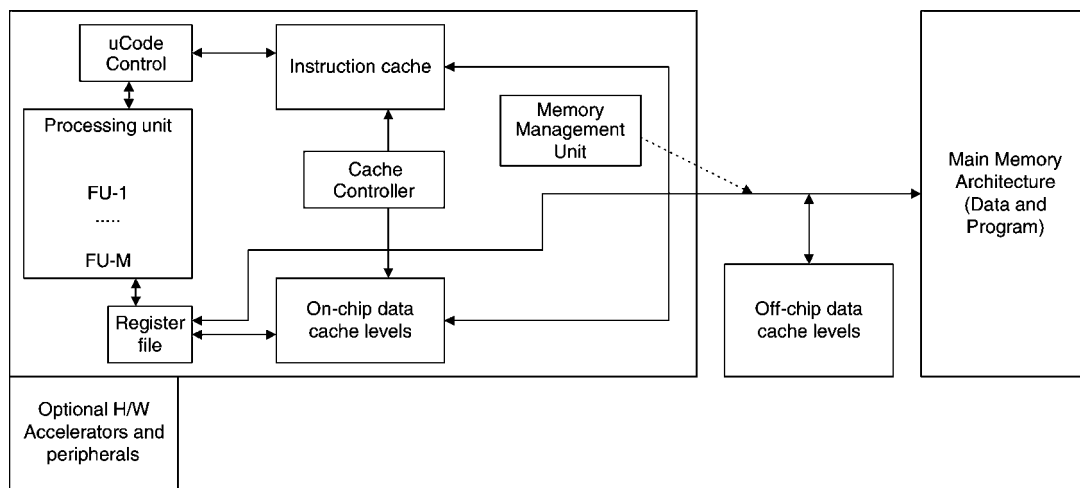


Fig. 6 Generic targeted processor architecture template

OM_ROWS is a 3D array (decomposition level-row of IPM -overlap coefficient) of (maximally) $2^{LEVELS+1} \cdot (2S - 1)$ samples (or coefficients) and OM_COLS is a 3D array (decomposition level-image column-overlap coefficient) that has (maximally) $2 \cdot N_2 \cdot (2S - 1)$ elements. For the typical case of $(2S + 1)/(2S - 1)$ filter pairs [4], the actual application of the lifting transform is performed using two small 1D arrays of $(2M + 1)$ elements, called Filtering-first-in first-out (FIFOs) (FF_1 and FF_2) [11, 14] to emphasise the inherent functionality. Thus, each new pair of input samples enters the FF_1 memory from IPM in FIFO manner and after the transformation is applied (Apply_lifting() function), the resulting low- and high-frequency coefficients are written in-place in IPM . After the completion of each decomposition level within the current unit (set-of-lines or block), the high-frequency coefficients are extracted to a memory that serves as the intermediate buffer between the transform production and the coding engine (ext_memory). The size of this memory depends on the specific compression algorithm [3, 4]. Since the link between transform and coding is not studied in this paper, ext_memory has $N_1 \times N_2$ wavelet coefficients similar to the RC computational schedule.

3 Target architecture

The performance evaluation of the 2D DWT computation schedules targets programmable (domain specific) DSPs. As mentioned, in many occasions, these processors form the most attractive implementation alternative for (multidimensional) signal processing algorithms. In many cases, the architecture of such processors is a highly

parallel VLIW architecture (e.g. Philips TriMedia and TI TMS320C6x families of processors). The data path includes multiple functional units, which are, in principle, programmed through the program memory. Large register files with appropriate I/O bandwidth reside close to the functional units. The data memory hierarchy may include several cache levels, with at least one of them being on-chip. Caches can be either hardware or software controlled. The main off-chip memory is typically accessed over a single (wide) bus. The data transfer is controlled by the hardware (Memory Management Unit). A cache by-pass may also be present, connecting directly the main memory to the register files (Fig. 6).

In our work, the TriMedia TM1 processor [15] introduced by Philips has been used for the realisation of the different variants of the 2D DWT. TriMedia is a 32-bit dedicated media processor for high performance multimedia applications that deal with high-quality video and audio. A summary of the key features of the TM1 architecture is given in Table 1.

4 Performance comparisons

In this section, the performance of the RC, LB and BB 2D DWT computation schedules on the TriMedia TM1 processor is analysed and compared. ANSI C codes have been developed for the three different implementations. The developed C codes are parameterised with respect to the filter pair, the number of decomposition levels and the input image size. The C codes have been mapped on the TriMedia TM1 processor (using the processor's compiler).

Table 1: Major architecture features of TriMedia TM1

Feature	Value	Feature	Value
Clock frequency	100 MHz	SW controlled part of data cache	up to 8 kB
Maximum performance	4 GOPS	On-chip instruction cache size	32 kB
Number of parallel data paths	27	Off-chip main memory size range	0.5–64 MB
Maximum number of data paths activated by the same instruction	5	External highway width and speed	32 bits to 400 MB/s
Register file size/bandwidth	128 32-bit registers with 15 read and 5 write ports	Packed instruction width	32 bits
On-chip data cache size/bandwidth	16 kB/dual port	Decompressed instruction width	220 bits

Table 2: Relative performance results for the 9/7 filter pair under a convolution-based and a lifting-based implementation for typical cases of image sizes and decomposition levels (RC computation schedule)

Image size	Number of decomposition levels	Lifting	Convolution
256 × 256	4	65	100
	5	68	100
512 × 512	4	62	100
	5	66	100

Their performance for the processing of (various sizes of) four test images has been measured (and averaged) using processor’s simulator.

The performance of the schedules has been evaluated for lifting-based implementations as:

- (a) the lifting-based realisation of (7) and (8) provides a reversible transform for the 5/3 filter pair, which corresponds to the suggested usage of this filter pair in the JPEG-2000 standard [4];
- (b) these realisations were found to provide lower execution times than convolution-based realisations.

An example of the relative performance of convolution and lifting-based realisations for the typical case of the RC computational schedule is presented in Table 2, where the value of 100 was always assigned to the worst performance of each case. In total, lifting-based implementations appear to offer an improvement in the order of 35–40% in execution time against convolution-based implementations.

The following parameters were investigated in the evaluation studies of this paper: (a) both the 5/3 and 9/7 filter pairs (included in the JPEG 2000 standard [4]); (b) for (greyscale) image sizes of 128 × 128, 256 × 256, 512 × 512 and 1024 × 1024 pixels and (c) for decomposition levels ranging from 3 to 6. The algorithmic space that has been explored is pictorially shown in Fig. 7. For the smaller image sizes and for a large number of decomposition levels, the size of the wavelet subbands in the last levels may become equal to the filter size. For such cases, no performance results have been obtained.

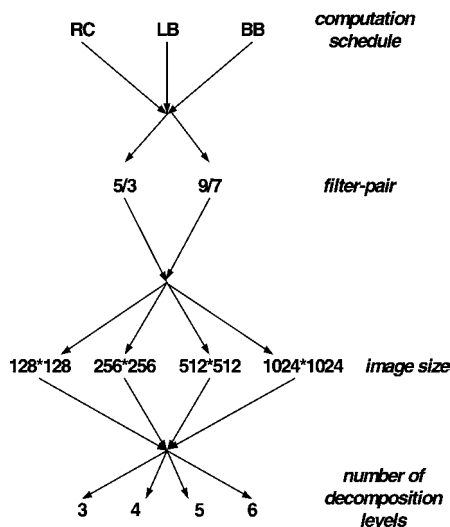


Fig. 7 Algorithmic space explored

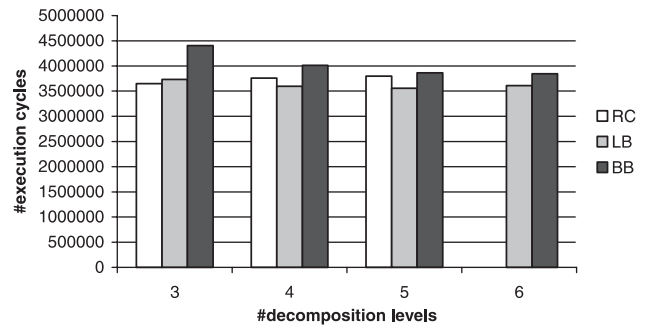


Fig. 8 Performance results for the 5/3 filter pair implementations based on image size of 128 × 128 pixels

The detailed comparisons of the performance of various computation schedules for various image sizes are presented in Figs. 8–15. The corresponding relative performance results (with the value of 100 always assigned to the worst performance of each case) of the various schedules are presented in Tables 3–10.

4.1 128 × 128-pixel images (Figs. 8 and 9 and Tables 3 and 4)

In this case, the BB schedule always leads to worst performance for both the 5/3 and 9/7 filter pairs. The LB schedule always leads to the best performance except one case (5/3 filter pair and 3 decomposition levels). For both the 5/3 and 9/7 filter pairs, as the number of decomposition levels increases, the performance of the RC-schedule becomes worse, whereas that of the BB schedule improves. The performance of the LB schedule improves as the number of decomposition levels increases up to 5 and

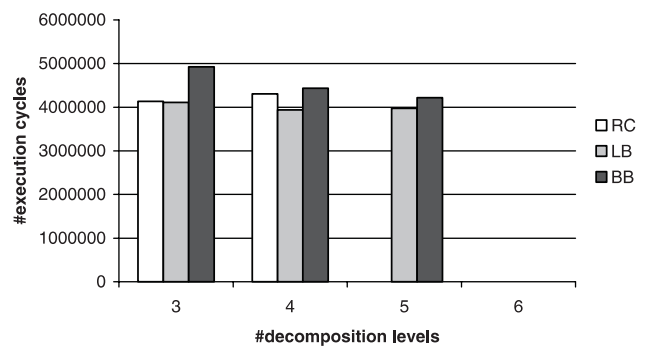


Fig. 9 Performance results for implementations based on the 9/7 filter pair image size of 128 × 128 pixels

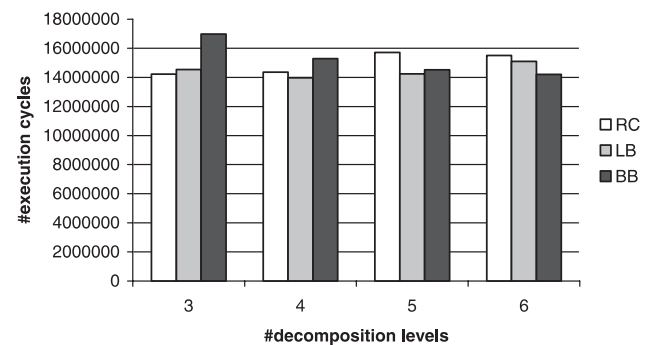


Fig. 10 Performance results for the 5/3 filter pair implementations based on image size of 256 × 256 pixels

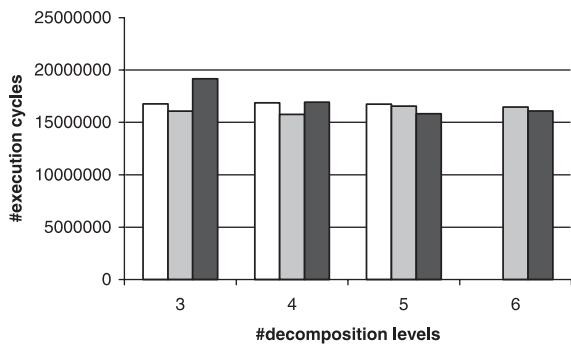


Fig. 11 Performance results for the 9/7 filter pair implementations based on image size of 256×256 pixels

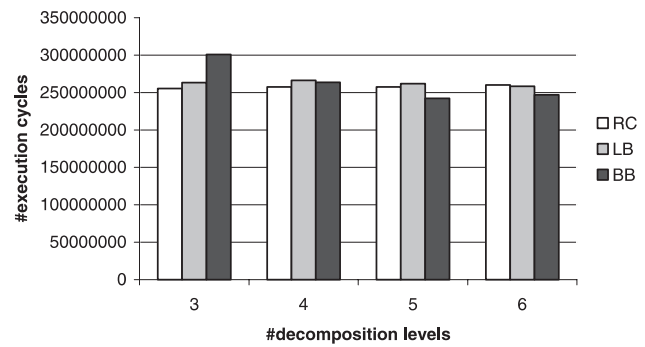


Fig. 15 Performance results for the 9/7 filter pair implementations based on image size of 1024×1024 pixels

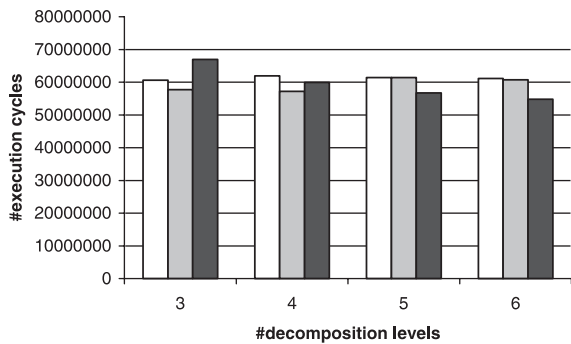


Fig. 12 Performance results for the 5/3 filter pair implementations based on image size of 512×512 pixels

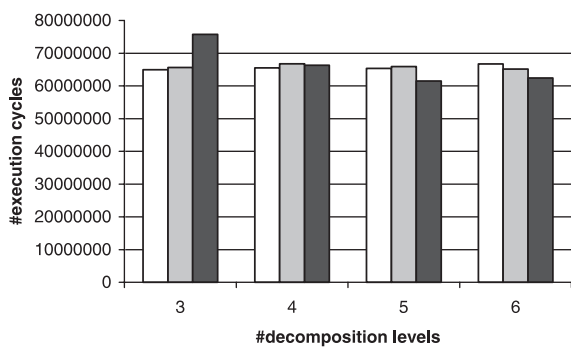


Fig. 13 Performance results for the 9/7 filter pair implementations based on image size of 512×512 pixels

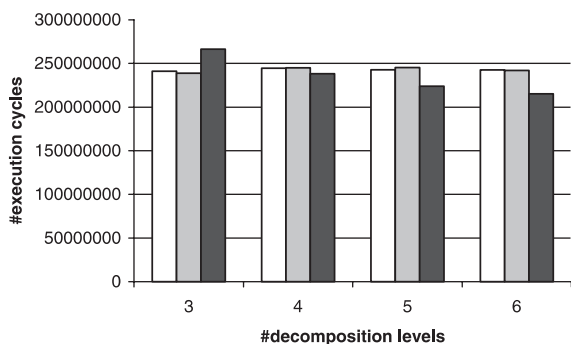


Fig. 14 Performance results for the 5/3 filter pair implementations based on image size of 1024×1024 pixels

then degrades. On average, for the various numbers of decomposition levels, the LB schedule is 3.3 and 11.3% faster than the RC and BB schedules, respectively, for the 5/3 filter pair; for the 9/7 filter pair, the LB-schedule is

Table 3: Relative performance results for the 5/3 filter pair implementations based on image size of 128×128 pixels

Image size	Number of decomposition levels	RC	LB	BB
128×128	3	80	82	100
	4	93	89	100
	5	98	92	100
	6	N/A	94	100

Table 4: Relative results for the 9/7 filter pair implementations based on image size of 128×128 pixels

Image size	Number of decomposition levels	RC	LB	BB
128×128	3	81	80	100
	4	97	88	100
	5	N/A	94	100
	6	N/A	N/A	N/A

5.2 and 13% faster than the RC and BB schedules, respectively.

4.2 256×256 -pixel images (Figs. 10 and 11 and Tables 5 and 6)

In this case, for the 5/3 filter pair, the performance of the RC is best for 3 decomposition levels; the performance of the LB is best for 4 and 5 decomposition levels, whereas the performance of the BB is best for 6 decomposition levels. For the 9/7 filter pair, the performance of the LB is best for 3 and 4 decomposition levels, whereas the BB is best for 5 and 6 decomposition levels. The performance of the RC degrades as the number of decomposition levels increases (in almost all cases). In principle, the performance of the BB improves as the number of decomposition levels increases (for the case of the 9/7 filter pair up to the 5 decomposition levels). There is no clear conclusion for performance change against the number of decomposition levels for the LB for both 5/3 and 9/7-filter pair realisations. On average, for the various numbers of decomposition levels, the LB is 3.5 and 5.5% faster than the RC and BB, respectively, for the 5/3 filter pair. For the 9/7 filter pair, the LB is 3.6 and 4.9% faster than the RC and BB, respectively (faster, on average, for the various numbers of decomposition levels).

Table 5: Relative performance results for the 5/3 filter pair implementations based on image size of 256×256 pixels

Image size	Number of decomposition levels	RC	LB	BB
256×256	3	80	82	100
	4	97	94	100
	5	100	89	91
	6	100	97	91

Table 6: Relative performance results for the 9/7 filter pair implementations based on image size of 256×256 pixels

Image size	Number of decomposition levels	RC	LB	BB
256×256	3	85	81	100
	4	99	93	100
	5	100	99	94
	6	N/A	100	98

Table 7: Relative performance results for the 5/3 filter pair implementations based on image size of 512×512 pixels

Image size	Number of decomposition levels	RC	LB	BB
512×512	3	89	84	100
	4	100	92	96
	5	100	100	92
	6	100	99	88

4.3 512×512 -pixel images (Figs. 12 and 13 and Tables 7 and 8)

The BB has the best performance for 5 and 6 decomposition levels for both the 5/3 and 9/7 filter pairs. The LB has the best performance for 3 and 4 decomposition levels for the 5/3 filter pair, whereas the RC has the best performance for 3 and 4 decomposition levels for the 9/7 filter pair. In principle, the performance of the BB improves as the number of decomposition levels increases (for the case of the 9/7 filter pair up to the 5 decomposition levels). There is no clear relation between performance changes and increase of the number of decomposition levels for the RC and the LB for both 5/3 and 9/7 filter pair realisations. On average, for various numbers of decomposition levels, the LB is 3.4 and 5.6% faster than the RC and BB, respectively, for the 5/3 filter pair. For the 9/7 filter pair, the RC is 0.4 and 1.3% faster than the LB and BB, respectively (faster, on average, for the various numbers of decomposition levels).

4.4 1024×1024 -pixel images (Figs. 14 and 15 and Tables 9 and 10)

In this case, the BB transform has the best performance for the larger number of decomposition levels, that is, for 4, 5 and 6 decomposition levels for the 5/3 filter pair and for 5 and 6 decomposition levels for the 9/7

Table 8: Relative performance results for the 9/7 filter pair implementations based on image size of 512×512 pixels

Image size	Number of decomposition levels	RC	LB	BB
512×512	3	83	84	100
	4	98	100	100
	5	99	100	93
	6	100	97	93

Table 9: Relative performance results for the 5/3 filter pair implementations based on image size of 1024×1024 pixels

Image size	Number of decomposition levels	RC	LB	BB
1024×1024	3	88	88	100
	4	100	100	97
	5	99	100	90
	6	100	100	87

Table 10: Relative performance results for the 9/7 filter pair implementations based on image size of 1024×1024 pixels

Image size	Number of decomposition levels	RC	LB	BB
1024×1024	3	82	85	100
	4	97	100	99
	5	98	100	92
	6	100	99	95

filter pair. The LB has the best performance for 3 decomposition levels for the 5/3 filter pair. The RC has the best performance for 3 and 4 decomposition levels for the 9/7 filter pair. As in the case of 512×512 pixel images, the performance of the BB improves as the number of decomposition levels increases in all cases except one (for the case of the 9/7 filter pair when the number of decomposition levels increases from 5 to 6). No clear relation between performance and the number of decomposition levels for the RC and the LB can be identified for both the 5/3 and 9/7 filter pair realisations. On average, for the various numbers of decomposition levels, the BB is 2.9 and 3% faster than the RC and the LB, respectively, for the 5/3 filter pair. For the 9/7 filter pair, the RC is 1.9 and 2.3% faster than the LB and BB, respectively (faster, on average, for the various numbers of decomposition levels).

5 Conclusions

The three major 2D DWT computation schedules have been compared with respect to their performance on a VLIW DSP. The three schedules produce better performance for various points of the algorithmic space explored (filter pair, image size, number of decompositions levels) and no schedule dominates completely even for sets of cases with common value for one algorithmic parameter. This fact proves that the development of parametrical software code

Table 11: Best wavelet transform implementation option for all possible sets of image size, number of decompositions levels and filter pair

Image size	Number of decomposition levels	Best implementation for 5/3 filter pair	Best implementation for 9/7 filter pair
128 × 128	3	RC	LB
128 × 128	4	LB	LB
128 × 128	5	LB	LB
128 × 128	6	LB	N/A
256 × 256	3	RC	LB
256 × 256	4	LB	LB
256 × 256	5	LB	BB
256 × 256	6	BB	BB
512 × 512	3	LB	RC
512 × 512	4	LB	RC
512 × 512	5	LB	BB
512 × 512	6	BB	BB
1024 × 1024	3	LB	RC
1024 × 1024	4	BB	RC
1024 × 1024	5	BB	BB
1024 × 1024	6	BB	BB

that can be configured to realise all the major computation schedules, depending on the application and the related algorithmic parameters, is of large importance. The same is true for custom hardware realisations and the development of parametrical cores in a hardware description language (most commercially available wavelet transform HDL cores realise one of the three major schedules).

A summary of the best wavelet transform implementation for all the possible sets of image sizes, number of decompositions levels and filter pair is shown in Table 11. The LB has the best performance in 14 cases, the BB in 11 cases, and the RC in 6 cases.

We conclude with some guidelines for the selection of the best implementation with respect to the values of the algorithmic parameters.

- The BB schedule performs better for large number of decomposition levels (mainly 6 but also 5) and large images (1024 × 1024 pixels).
- The LB schedule performs well for decomposition levels below 6, mainly for image sizes below 1024 × 1024 pixels.
- The RC performs well for the 9/7 filter pair and larger image sizes (512 × 512, 1024 × 1024 pixels) for small numbers of decomposition levels (3, 4), whereas, for the 5/3 filter pair, it performs well for small images (128 × 128, 256 × 256 pixels) and small number of decomposition levels (3).
- Independent of the computation schedule, the execution time may improve with an increase of the number of decomposition levels.

The last point is an interesting and somewhat counter-intuitive result, since one expects the execution time to increase with the number of decomposition levels. However, this can be explained in the LB and BB schedules based on the increased locality of the processing.

6 Acknowledgment

This work was supported in part by the Pythagoras II program of the EPEAEK II project of the Greek Government.

7 References

- 1 Mallat, S.G.: 'A theory for multiresolution signal decomposition: the wavelet representation', *IEEE Trans. Pattern Anal. Mach. Intell.*, 1989, **11**, (7), pp. 674–693
- 2 Daubechies, I., and Sweldens, W.: 'Factoring wavelet transforms into lifting steps', *J. Four. Anal. Appl.*, 1998, **4**, (3), pp. 247–269
- 3 Shapiro, J.M.: 'Embedded image coding using zerotrees of wavelet coefficients', *IEEE Trans. Signal Process.*, 1993, **41**, pp. 3445–3462
- 4 ISO/IEC FCD15444-1: 2000. 'JPEG 2000 image coding system', March 2000
- 5 ISO/IEC JTC1/SC29/WG11, FCD 14496-1. 'Coding of moving pictures and audio', May 1998
- 6 Parhi, K., and Nishitani, T.: 'VLSI architectures for discrete wavelet transforms', *IEEE Trans. VLSI Syst.*, 1993, **1**, (2), pp. 191–202
- 7 Vishwanath, M., Owens, R.M., and Irwin, M.J.: 'VLSI architectures for the discrete wavelet transform', *IEEE Trans. Circuits Syst. II*, 1995, **42**, (5), pp. 305–316
- 8 Pirsch, P., Stolberg, H.-J., Chen, Y.-K., and Kung, S.Y.: 'Implementation of media processors', *IEEE Signal Process. Mag.*, 1997, **14**, (4), pp. 48–51
- 9 Chakrabarti, C., Vishwanath, M., and Owens, R.M.: 'Architectures for wavelet transform: a survey', *J. VLSI Signal Process.*, 1996, **4**, (2), pp. 171–192
- 10 Zervas, N.D., Anagnostopoulos, G.P., Spiliotopoulos, V., Andreopoulos, Y., and Goutis, C.E.: 'Evaluation of design alternatives for the 2D discrete wavelet transform', *IEEE Trans. Circuits Syst. Video Technol.*, 2001, **11**, (2), pp. 1246–1262
- 11 Andreopoulos, Y., Schelkens, P., and Cornelis, J.: 'Analysis of wavelet transform implementations for image and texture coding applications in programmable platforms'. Proc. IEEE Workshop on Signal Processing Systems, 2001, vol. 1, pp. 273–284
- 12 Chrysafis, C., and Ortega, A.: 'Line based, reduced memory, wavelet image compression', *IEEE Trans. Image Process.*, 2000, **9**, (3), pp. 378–389
- 13 Lafruit, G., Nachtergaele, L., Bormans, J., Engels, M., and Bolsens, I.: 'Optimal memory organization for scalable texture codecs in MPEG-4', *IEEE Trans. Circuits Syst. Video Technol.*, 1999, **9**, (2), pp. 218–243
- 14 Lafruit, G., Nachtergaele, L., Vanhoof, B., and Catthoor, F.: 'The local wavelet transform: a memory-efficient, high-speed architecture optimized to a region-oriented zero-tree coder', *Integr. Comput. Aided Eng.*, 2000, **7**, (2), pp. 89–103
- 15 <http://www.support.trimedia.philips.com>