Algorithms and architectures for 2D discrete wavelet transform

Article *in* The Journal of Supercomputing · November 2012

DOI: 10.1007/s11227-012-0790-x

CITATIONS
7

READS
1,112

1 author:

Asadollah Shahbahrami
University of Guilan
105 PUBLICATIONS **651** CITATIONS

SEE PROFILE

J Supercomput (2012) 62:1045–1064
DOI 10.1007/s11227-012-0790-x

Algorithms and architectures for 2D discrete wavelet transform

Asadollah Shahbahrami

Published online: 22 May 2012

Abstract The 2D Discrete Wavelet Transform (DWT) is an important function in many multimedia applications, such as JPEG2000 and MPEG-4 standards, digital watermarking, and content-based multimedia information retrieval systems. The 2D DWT is computationally intensive than other functions, for instance, in the JPEG2000 standard. Therefore, different architectures have been proposed to process 2D DWT. The goal of this paper is to review and to evaluate different algorithms and different kinds of architectures such as application-specific integrated circuits, field programmable gate array, digital signal processors, graphics processing units, and General-Purpose Processors (GPPs) that are used to process 2D DWT. In addition, we implement the 2D DWT using different algorithms on GPPs enhanced with multimedia extensions. The experimental results show that the largest speedup of the vectorized 2D DWT over the scalar implementation is about 2.8 for first level decomposition. Furthermore, the characteristics of the 2D DWT and disadvantages of the existing architectures such as GPPs enhanced with SIMD instructions are discussed.

Keywords Multimedia standards · Discrete wavelet transform · Processor architecture · Multimedia extensions

1 Introduction

The Discrete Wavelet Transform (DWT) provides a time-frequency representation of a signal. The DWT is a multiresolution technique that can analyze different frequencies by different resolutions. The wavelet representation of a discrete signal X consisting of N samples can be computed by convolving X with the low-pass and

A. Shahbahrami (✉)
Department of Computer Engineering, Faculty of Engineering, University of Guilan, P.O. Box 3756-41635, Rasht, Iran
e-mail: shahbahrami@guilan.ac.ir

high-pass filters and down-sampling the output signal by 2, so that the two frequency bands each contains $N/2$ samples. With the correct choice of filters, this operation is reversible. This process decomposes the original image into two sub-bands: the lower and the higher band [1]. This transform can be extended to multiple dimensions by using separable filters. In other words, the DWT can be implemented as 1D, 2D, or 3D that depends on input signal dimensions.

The JPEG2000 and MPEG-4 multimedia standards employ 2D DWT instead of the Discrete Cosine Transform (DCT) that is used in the JPEG and MPEG-2 multimedia standards. The reason for this is that the image quality is much higher at lower bit rates. In addition, the 2D DCT based compression standards partition an image into discrete 8×8 pixel blocks. This generates blocking artifacts in the output image. The 2D DWT operates on a complete image or a large part of an image and this avoids the artifact problem. Consequently, it needs more memory than the 2D DCT. In addition, 2D DWT has been used in other applications such as digital watermarking [2, 3], Content-Based Multimedia Information Retrieval (CBMIR) systems [4–8], medical image processing [9, 10], and GPS satellite signal acquisition [11].

Furthermore, the 2D DWT is computationally intensive than other functions, for instance, in the JPEG2000 standard. For example, presented results in [12] showed that the 2D DWT consumes on average 46 % of the encoding time for lossless compression. For lossy compression, the 2D DWT even requires 68 % of the total encoding time on average. Therefore, different architectures have been proposed to process 2D DWT.

We make the following contributions compared to other research works. First, we provide a detailed review of different algorithms and approaches to implement the 2D DWT. Second, we provide a review of different kinds of architectures such as application-specific integrated circuits, field programmable gate array, digital signal processors, graphics processing units, and general-purpose processors that are used to process 2D DWT. The advantages and disadvantages of these architectures are discussed. Third, in order to improve the performance of the 2D DWT, we exploit the available Data-Level Parallelism (DLP) using Single Instruction on Multiple Data (SIMD) which are available on recent processors such as Pentium 4. The experimental results show that the largest speedup of the vectorized 2D DWT over the scalar implementation is about 2.8 for first level decomposition. Finally, we discuss the characteristics of the 2D DWT in order to show that new designs are needed to process the DWT, for instance, the collaboration of reconfigurable architectures with grid computing and combination of the fine-grained reconfigurable architectures with coarse-grained reconfigurable architectures.

The paper is organized as follows. Section 2 describes some background information about the DWT. We present different approaches to implement the DWT in Sect. 3. Characteristics of the 2D DWT are discussed in Sect. 4. Different processors architecture which have been proposed to process DWT are presented in Sect. 5. We discuss some limitations of current processors in Sect. 6. Finally, the conclusions of the paper are drawn in Sect. 7.

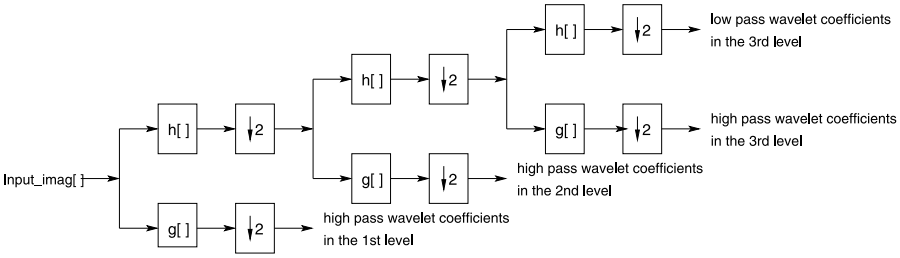

Fig. 1 Three level 2D DWT decomposition of an input image using filtering approach. The h and g variables denote the low-pass and high-pass filters, respectively. The notation of ($\downarrow 2$) refers to downsampling of the output coefficients by two

2 Background

In this section, we describe the 2D discrete wavelet transform and different algorithms to traverse an image to implement the 2D DWT.

2.1 2D discrete wavelet transform

The 2D DWT is computed by performing low-pass and high-pass filtering of the image pixels as shown in Fig. 1. In this figure, the low-pass and high-pass filters are denoted by h and g, respectively. This figure depicts the three levels of the 2D DWT decomposition. At each level, the high-pass filter generates detail image pixels information, while the low-pass filter produces the coarse approximations of the input image. At the end of each low-pass and high-pass filtering, the outputs are downsampled by two ($\downarrow 2$). The 2D DWT is separable and obtains from two 1D DWT. In other words, a 2D DWT can be performed by first performing a 1D DWT on each row, which is referred to *horizontal filtering*, of the image followed by a 1D DWT on each column, which is called *vertical filtering*.

Figure 2 illustrates the first decomposition level ($d = 1$) of 2D DWT implementation on an image. At this level, the original image is decomposed into four subbands that carry the frequency information in both the horizontal and vertical directions. In order to form multiple decomposition levels, the algorithm is applied recursively to the LL subband. Figure 3 illustrates the second ($d = 2$) and third ($d = 3$) decomposition levels as well as the layout of the different bands.

There are different algorithms to traverse an image to implement 2D DWT, namely *Row-Column Wavelet Transform* (RCWT) and *Line-Based Wavelet Transform* (LBWT) [13–17]. These approaches are discussed in the following sections.

2.2 Row-column wavelet transform

In the RCWT approach, the 2D DWT is divided into two 1D DWTs, namely horizontal and vertical filtering. The horizontal filtering processes the rows of the original image and stores the wavelet coefficients in an auxiliary matrix. Thereafter, the vertical filtering phase processes the columns of the auxiliary matrix and stores the results back in the original matrix. In other words, this algorithm requires that all lines are

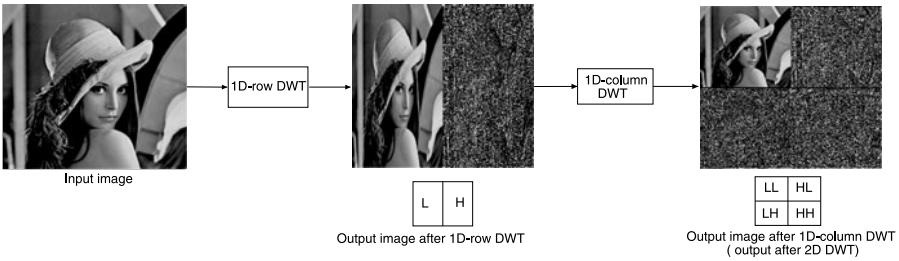

Fig. 2 Different sub-bands after first decomposition level of 2D DWT implementation on an image

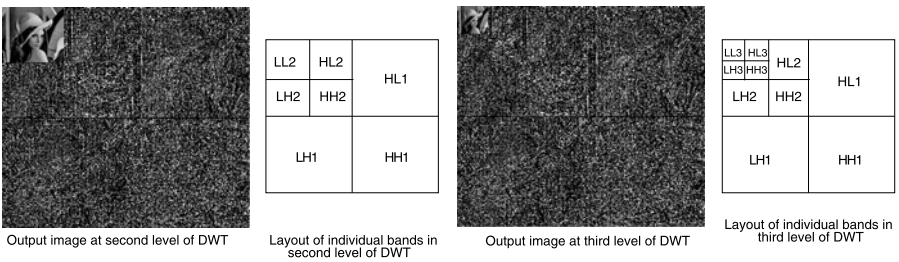

Fig. 3 Different subbands after second and third decomposition levels of 2D DWT implementation on an image

horizontally filtered before the vertical filtering starts. The computational complexity of both horizontal and vertical filtering is the same. Figure 2 depicts both horizontal and vertical filtering. Each of these filtering is applied separately. Each of these $N \times M$ matrices requires NMc bytes of memory, where c denotes the number of bytes required to represent one wavelet coefficient.

2.3 Line-based wavelet transform

In the line-based wavelet transform approach, the vertical filtering starts as soon as a sufficient number of lines, as determined by the filter length, has been horizontally filtered. In other words, the LBWT algorithm uses a single loop to process both rows and columns together. This technique computes the 2D DWT of an $N \times M$ image by the following stages. First, the horizontal filtering filters L rows, where L is the filter length, and stores the low-pass and high-pass values interleaved in an $L \times M$ buffer. Thereafter, the columns of this small buffer are filtered. This produces two wavelet coefficients rows, which are stored in different subbands in an auxiliary matrix in the order expected by the quantization step. Finally, these stages are repeated to process all rows and columns. Figure 4 illustrates the LBWT algorithm.

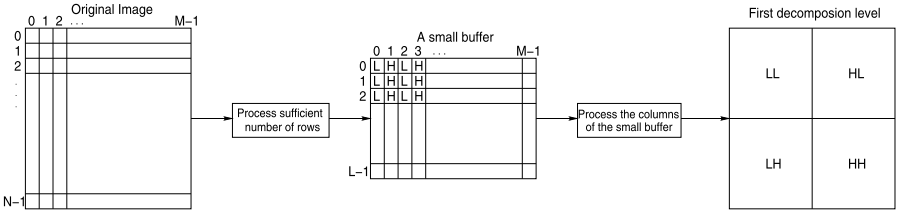

Fig. 4 The line-based wavelet transform approach processes both rows and columns in a single loop

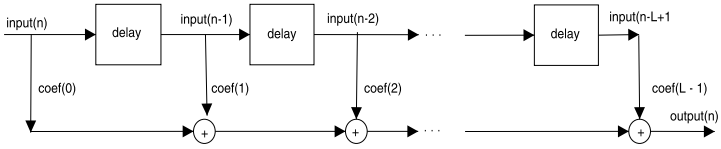

Fig. 5 Direct structure of an FIR filter

3 Different approaches to implement 2D DWT

There are different approaches to implement the DWT such as traditional convolution-based and lifting scheme methods. These approaches are discussed in the following sections.

3.1 Convolution-based

The convolutional methods apply filtering by multiplying the filter coefficients with the input samples and accumulating the results. Their implementations are the same as the implementation of Finite Impulse Response (FIR) filters. The FIR filter is a continuing computation over time of the form:

$$output(n) = \sum_{k=0}^{L-1} input(n-k) * coef(k) \tag{1}$$

Each output value requires L multiplications and $L-1$ additions. The filter coefficients are denoted $coef(k)$ for $k = 0, \ldots, L-1$, where L is the filter length, and the signal length is n samples. Figure 5 depicts the signal-flow diagram of the FIR filter [18]. It represents the direct calculation of Eq. (1) and is called a direct structure [19].

There are different FIR filters to implement the DWT. The Daubechies' transform with four coefficients (Daub-4) [20] and the Cohen, Daubechies and Feauveau 9/7 filter (CDF-9/7) [21] are examples of this category. For instance, the CDF-9/7 transform has nine low-pass filter coefficients $h = \{h_{-4}, h_{-3}, h_{-2}, h_{-1}, h_0, h_1, h_2, h_3, h_4\}$ and seven high-pass filter coefficients $g = \{g_{-2}, g_{-1}, g_0, g_1, g_2, g_3, g_4\}$. Both filters are symmetric, i.e., $h_{-i} = h_i$. This kind of implementation needs a large number of computations. The number of low-pass filter and high-pass filter coefficients in the Daub-4 transform equals four. Figure 6 and Fig. 7 depict C implementations of

```
void Daub_4_horizontal() {
int i, j, jj;
float low[] ={-0.1294, 0.2241, 0.8365 , 0.4830};
float high[]={-0.4830, 0.8365, -0.2241, -0.1294};
for (i=0; i<N; i++)
  for(j=0, jj=0; jj<M; j++, jj +=2)
    {
     tmp[i][j]      = img[i][jj]      * low[0]   +  img[i][jj + 1] *
                      low[1] +
                      img[i][jj + 2] * low[2]   +  img[i][jj + 3] *
                      low[3];

   tmp[i][j + M/2] = img[i][jj]      * high[0] + img[i][jj  + 1] *
                     high[1] + img[i][jj + 2] * high[2] +
                     img[i][jj  + 3] * high[3];
  }
}
```

Fig. 6 C implementation of horizontal filtering for Daub-4 transform using the RCWT approach

```
void Daub_4_vertical() {
int i, j, jj;
float low[] ={-0.1294, 0.2241, 0.8365 , 0.4830};
float high[]={-0.4830, 0.8365, -0.2241, -0.1294};
for (i=0, ii=0; ii<N; i++, ii +=2)
  for (j=0; j<M; j++) {
    img[i][j]= tmp[ii][j]   *low[0]+tmp[ii+1][j]*low[1] +
               tmp[ii+2][j] *low[2]+tmp[ii+3][j]*low[3];

    img[i+N/2][j]= tmp[ii][j]  *high[0]+tmp[ii+1][j]*high[1] +
                   tmp[ii+2][j]*high[2]+tmp[ii+3][j]*high[3];
  }
}
```

Fig. 7 C implementation of vertical filtering for Daub-4 transform using the RCWT approach

horizontal and vertical filtering for Daub-4 transform using the RCWT approach, respectively. In addition, Fig. 8 depicts the C implementation of the Daub-4 transform using the LBWT approach. As can be seen, both rows and columns are processed in a single program together.

3.2 Lifting scheme

The lifting scheme has been proposed for the efficient implementation of the 2D DWT. The lifting approaches need 50 % less computations than the convolution-based approaches. The basic idea of the lifting scheme is to use the correlation in the image pixels values to remove the redundancy [22, 23]. The lifting scheme has three phases, namely: split, predict, and update, as illustrated in Fig. 9.

In the split stage, the input sequence is split into two subsequences consisting of the even and odd samples. In the predict and update stages, the high-pass and low-pass values are computed, respectively. One example of this group is the integer-to-integer $(5, 3)$ lifting scheme ($(5, 3)$ *lifting*). The lifting operation for the $(5, 3)$ filter bank is depicted in Fig. 10. The sequences $\{s_i^0\}$ and $\{d_i^0\}$ denote the even and

```
void Daub_4_Transform() {
 // processing both rows and columns in a single loop.
 int i, j, jj, ii, k;
 float buffer[4][M] = {0};
 float low[] ={-0.1294, 0.2241, 0.8365 , 0.4830};
 float high[]={-0.4830, 0.8365, -0.2241, -0.1294};
 for (i=0, ii=0; i<N; ii++, i +=2) {
   k = ( ii % 2 ) * 2;
   for(j=0, jj=0; jj<M; j++, jj +=2) {
     buffer[k][jj]     = img[i][jj]     * low[0]  + img[i][jj + 1]  *
                         low[1] +
                         img[i][jj + 2] * low[2]  + img[i][jj + 3]  *
                         low[3];
     buffer[k][jj + 1] = img[i][jj]     * high[0] + img[i][jj  + 1] *
                         high[1] +
                         img[i][jj + 2] * high[2] + img[i][jj  + 3] *
                         high[3];

     buffer[k+1][jj]     = img[i+1][jj]    * low[0]  +
                           img[i+1][jj + 1] * low[1] +
                           img[i+1][jj + 2]* low[2]  +
                           img[i+1][jj + 3] * low[3];
     buffer[k+1][jj + 1] = img[i+1][jj]    * high[0] +
                           img[i+1][jj  + 1]* high[1] +
                           img[i+1][jj + 2]* high[2] +
                           img[i+1][jj  + 3]* high[3];
     }
     for(j=0, jj=0; jj<M; j++, jj +=2) {
       tmp[ii][j]  =  buffer[0][jj]     * low[0]  +
                      buffer[1][jj] * low[1] +
                      buffer[2][jj]     * low[2]  +
                      buffer[3][jj] * low[3];
       tmp[ii + N/2][j] = buffer[0][jj] * high[0] +
                          buffer[1][jj] * high[1] +
                          buffer[2][jj] * high[2] +
                          buffer[3][jj] * high[3];

       tmp[ii][j + M/2]= buffer[0][jj + 1]   * low[0] +
                         buffer[1][jj + 1]* low[1] +
                         buffer[2][jj + 1]   * low[2] +
                         buffer[3][jj + 1]* low[3];
       tmp[ii+N/2][j + M/2]=buffer[0][jj + 1]* high[0]+
                            buffer[1][jj + 1] * high[1] +
                            buffer[2][jj + 1]* high[2]+
                            buffer[2][jj + 1] * high[3];
     }
   }
 }
```

Fig. 8 C implementation of the Daub-4 transform using the LBWT approach

odd input sequence and the outputs $\{s_i^1\}$ and $\{d_i^1\}$ are the low-pass and high-pass output coefficients of the DWT filter, respectively. In the first step, the original 1D input signal is split into a subsequence consisting of the even-numbered input values $\{s_i^0\}$ and a subsequence containing the odd-numbered input values $\{d_i^0\}$. Thereafter,

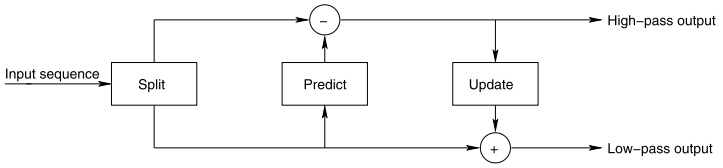

Fig. 9 Three different phases in the lifting scheme

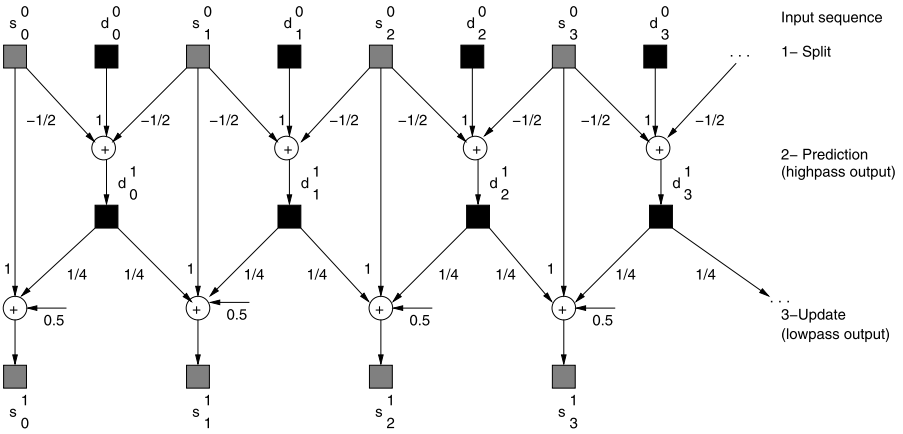

Fig. 10 One prediction and update stage in the lifting scheme of the (5, 3) lifting transform

the prediction stage produces the high-pass output values $\{d_i^1\}$ and the update stage generates the low-pass output values $\{s_i^1\}$.

The equations that are used in the prediction and update stage of the (5, 3) lifting transform are given by

$$d_i^1 = d_i^0 - \left\lfloor \frac{s_i^0 + s_{i+1}^0}{2} \right\rfloor \tag{2}$$

$$s_i^1 = s_i^0 + \left\lfloor \frac{d_{i-1}^1 + d_i^1 + 2}{4} \right\rfloor \tag{3}$$

Figure 11 depicts the C code of the horizontal and vertical filtering of the (5, 3) lifting transform for an $N \times M$ image.

4 Discrete wavelet transform characteristics

The 2D DWT has certain requirements that separate it from other functions such as 2D DCT, color space conversion, and entropy coding [24]. This is because of the following reasons. First, the 2D DWT has been widely used in a variety of applications and standards such as JPEG2000 and MPEG-4, digital watermarking, content-based multimedia information retrieval, and medical image processing. Each application

```
void Lifting53_horizontal() {
for (i=0; i< N; i++)
  for (j=1, jj=0; j<M; jj++, j +=2) {
    // calculation of high-pass values
    tmp[i][jj + M/2] = img[i][j] - ((img[i][j-1] +
                      img[i][j+1]) >> 1);
    // calculation of low-pass values
    tmp[i][j] = img[i][j-1] + ((tmp[i][jj + M/2] +
                tmp[i][jj + M/2 - 1] + 2) >> 2);
  }
}

void Lifting53_vertical() {
for (j=1, jj=0; j<N; jj++, j +=2)
  for (i=0; i<M; i++) {
    // calculation of high-pass values
    img[jj + N/2][i] = tmp[j][i] -  ((tmp[j-1][i] +
                       tmp[j+1][i]) >> 1);
    //calculation of low-pass values
    img[jj][i] = tmp[j-1][i] +  ((img[jj + N/2][i] +
                 img[jj + N/2 -1][i] + 2) >> 2);
  }
}
```

Fig. 11 The C implementations of the horizontal and vertical filtering of the (5, 3) lifting scheme

needs a specific implementation of the 2D DWT based on algorithm, wavelet filter length, and decomposition levels. Second, the 2D DWT is used in different environments ranging from desktop systems to mobile systems to process different media data such as text, handwritten data, image, video, and 3D data. Third, the 2D DWT processes a complete image in both horizontal and vertical directions. That means that it needs much more memory than other functions such as the 2D DCT. Vectorization of vertical filtering is much easier than vectorization of horizontal filtering. The 2D DWT has a high level of the DLP. In order to prove this, we have implemented it using SIMD instructions in the following section.

A well-known technique to increase the performance is to execute as many as possible parts of the 2D DWT in parallel. Therefore, the architecture executing the 2D DWT has to exploit the potential parallelism. The 2D DWT has a huge number of calculations in parallel and in a regular manner. This is because both low- and high-pass filters are applied modularly and regularly to regions of each image. The processing of each region of an image can be performed independently from the other regions. This feature means that the 2D DWT has a large potential for parallelism in terms of data computation. Finally, the 2D DWT can be implemented using either fixed-point or floating-point arithmetic operations and different data types [25]. In addition, as was mentioned in the Introduction section, the 2D DWT is computationally intensive than other functions, for example, in the JPEG2000 standard. Furthermore, different constant coefficients in low-pass and high-pass filtering are used to implement the 2D DWT and these coefficients can be kept in cache or registers in order to improve performance.

Based on the above characteristics and requirements, 2D DWT has been implemented on different platforms and different architectures have also been proposed to

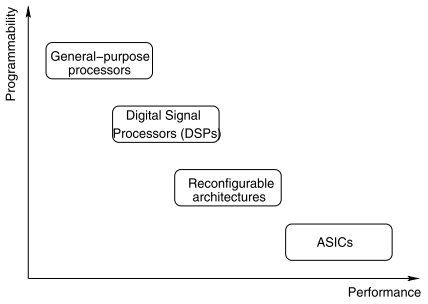

Fig. 12 Different proposed architectures for processing of 2D DWT

improve the performance of the 2D DWT. Some of these platforms and architectures are discussed in the following section.

5 Processor architectures

Some architectures that have been proposed to process DWT are depicted in Fig. 12 in the performance-programmability space. As this figure shows, implementation of 2D DWT has evolved through ASICs [23, 26–30], FPGA [31–33], DSPs [34, 35], and GPPs [25, 36, 37]. In addition, recently GPUs have been used for parallel implementation of 2D DWT [38]. These architectures are discussed in the following sections.

5.1 DSP based approaches

Schelkens et al. [34] have implemented an integer DWT on a TMS320C40 platform. They have focused on memory optimization and code mapping of an integer lifting scheme using the RCWT approach. Cho et al. [35] have implemented the overlapped block-based lifting scheme using a fixed-point DSP chip. The overlapped block-based approach partitions entire image memory into blocks to fit into the cache size and reorders the sequence of the wavelet lifting. Gnavi et al. [39] have presented a performance comparison between traditional convolution-based and lifting scheme approaches on a programmable DSP platform. They observed that the lifting scheme transforms were always faster than their convolution-based counterpart in the context of the JPEG2000. The performance improvement of lifting scheme techniques depended on the wavelet filters length and the number of lifting scheme steps.

Most of the DSP implementations have been focused on integer DWT to achieve high-performance, while floating-point representation of 2D DWT coefficients is usually required in some applications. In addition, DSP architectures do not offer sufficient flexibility and high-performance for different filter bank lengths of the DWT. This is because those processors do not have suitable microarchitectures to exploit available parallelism for high- performance.

5.2 ASIC based approaches

The ASIC implementations have been extensively used in order to provide high-performance for DWT. ASIC based architectures to support DWT can be classified

into different categories depends on some metrics such as style of the computation and topology of processing elements. Based on the computational style, different architectures can be categorized in to serial and parallel computations. In serial architectures, in each single time step, each Multiply-ACcumulate (MAC) unit performs one multiplication and one addition. Parallel architectures have L multipliers, one for each wavelet coefficient. The multiplications are performed in parallel. The DWT can be implemented as 1D, 2D, or 3D, therefore, different ASIC implementations have been provided. Each architecture is often designed for a specific application in mind. Some 1D architectures have been discussed in [40–42] and some 2D and 3D architectures have also been explained in [23, 26, 27, 43] and [29], respectively. In addition, Xiong et al. [28] have presented an efficient array architectures for multidimensional implementation of the lifting scheme of 2D DWT. Liao et al. [30] have proposed some architectures for 1D and 2D lifting scheme wavelet transform. Weeks et al. [44] have discussed different ASIC architectures to support DWT and compared them based on design type, latency, area, memory, and number of multipliers and adders. Most ASIC architectures are based on integer wavelet transform, especially on lifting scheme. However, this choice might has negative impact on precision.

ASIC implementations are a direct mapping of a filter bank length to hardware. The implemented hardware is optimized to provide high-performance with low power for fixed filter bank length and fixed wavelet decomposition levels and it cannot be used for various filter bank lengths and various transform levels with different precisions which are used in DWT implementation. In other words, flexibility is a key requirement to implement DWT within existing applications and to follow new generations of multimedia applications and standards.

5.3 GPUs based approaches

The GPUs have been designed as application-specific units with control and communication units that enable the use of many ALUs. Several researchers have implemented DWT on the GPUs [38, 45–47]. Hopf and Ertl [46] have developed an OpenGL-based model for implementation of traditional convolution-based techniques on graphics hardware. Wang et al. [47] have implemented the DWT targeted on the Jasper software on the GPUs. However, those implementations cannot efficiently exploit all the hardware resources available in GPUs. For example, Hopf and Ertl's implementation is not a direct mapping on hardware, therefore, it limits the efficiency of that implementation. Tenllado et al. [38, 45] have mapped the parallel implementation of the traditional convolution-based and lifting scheme approaches of the DWT on GPUs.

The GPUs have been designed as application-specific and have been optimized for some applications, for instance the 3D scene rendering. This means that implementing other applications such as DWT on the GPUs is a complicated task as mentioned by researchers in [38].

5.4 FPGA based approaches

In order to provide a flexibility for the DWT implementation based on wavelet filter length and wavelet decomposition structure, some reconfigurable hardwares have

been proposed in [31–33]. In [31, 32] Tseng et al. have proposed a reconfigurable architecture to process the DWT. Their proposed architecture consists of reconfigurable processing element array and reconfigurable address generator. A reconfigurable architecture for an integer 8-point Haar wavelet transform was proposed in [33]. However, function-specific reconfigurable hardware implementations are not flexible enough to support different implementation of 2D DWT. In addition, some of those implementations have focused on Fine-Grained Reconfigurable Architectures (FGRA) such as FPGAs. Those FGRAs use more logic blocks to implement the circuit. The resulting interconnection delays hinder the performance.

On the other hand, Coarse-Grained Reconfigurable Architecture (CGRA) decreases total number of logic blocks and, therefore, interconnect complexity is reduced by localizing the connections. MATRIX and RaPiD [48, 49] are some CGRA which have been proposed to overcome the drawbacks of FGRAs. MATRIX consists of 2D array of 8-bit FUs and each FU consists of an 8-bit ALU, memory, and control logic, while RaPiD composes of 1D array of FUs. While fewer configuration bits are needed for the PEs, fully utilizing the functionality of each PE is difficult, leading to significant underutilization of CGRAs. This can be avoided by either Medium-Grained Reconfigurable Architectures (MGRAs) or by adapting the reconfigurable architecture to the computational characteristics of the algorithm with application-specific grained processing elements. In addition, reconfigurable implementations should provide flexibility for variable wavelet filter lengths, variable wavelet decomposition levels, and support both convolution-based and lifting-based approaches.

5.5 GPPs and GPPs enhanced with multimedia extension

In this section, we first discuss the general-purpose processors enhanced with new instructions, Single-Instruction Multiple-Data (SIMD), and then we explain the implementation of the DWT on GPPs and GPPs enhanced with SIMD extension.

5.5.1 GPPs enhanced with SIMD extension

In order to increase the performance of multimedia applications such as JPEG and JPEG2000 standards, GPPs vendors have extended their instructions set architecture with SIMD extensions [50]. The first multimedia extensions are Intel's MMX [51, 52], Sun's Visual Instruction Set (VIS) [53], Compaq's Motion Video Instructions (MVI) [54], MIPS Digital Media eXtension (MDMX) [55, 56], and HP's Multimedia Acceleration eXtension (MAX) [50, 57]. These extensions supported only integer data types and were introduced in the mid-1990s. 3DNow [58] was the first to support floating-point media instructions. It was followed by Streaming SIMD Extension (SSE) and SSE2 from Intel [59, 60]. Motorola's AltiVec [61, 62] supports integer as well as floating-point media instructions. In addition, high-performance processors also use SIMD processing. An excellent example of this is the cell processor [63–65] developed by a partnership of IBM, Sony, and Toshiba. Cell is a heterogeneous chip multiprocessor consisting of a PowerPC core that controls eight high-performance Synergistic Processing Elements (SPEs). Each SPE has one SIMD computation unit that is referred to as Synergistic Processor Unit (SPU). Each SPU has 128 128-bit registers. SPUs support both integer and floating-point SIMD instructions.

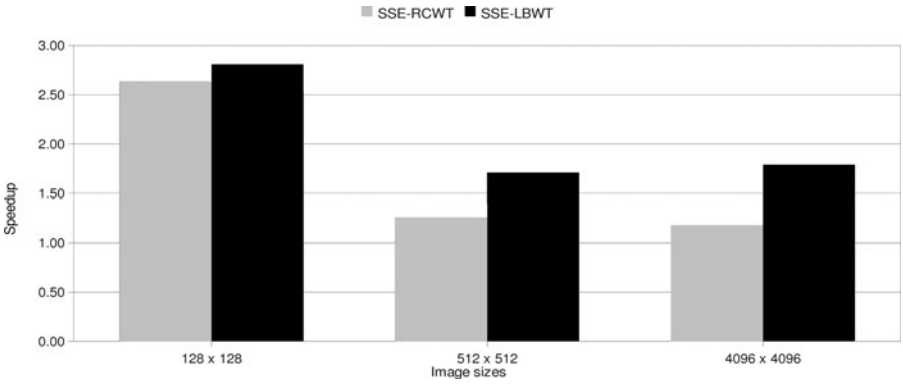

Fig. 13 Speedup of the SSE-RCWT over C-RCWT as well as the speedup of the SSE-LBWT over C-LBWT for the first level decomposition of the 2D DWT of Daub-4 using different image sizes

Table 1 summarizes the common and distinguishing features of existing multimedia instruction set extensions [24, 66–69].

5.5.2 Implementations of 2D DWT on GPPs

There are different implementations of the DWT on GPPs such as [25, 36, 37]. However, implementation of the DWT on GPPs is not efficient. The main reason for this is the discrepancies between the memory access patterns of the two main parts of the 2D DWT, horizontal and vertical filtering; especially, when a large image is processed by both horizontal and vertical filtering, one of them causes to exhibit poor data locality [12]. Therefore, a lot of research have been done on cache-aware DWT implementations [13–15, 70–74].

5.5.3 Implementations of 2D DWT on GPPs Enhanced with SIMD Extension

Some research work [37, 75–77] have mapped the 2D DWT on GPPs enhanced with multimedia extensions such as MMX [51, 52] and SSE [59, 60].

In addition, we have implemented the 2D DWT using SIMD instructions [12, 25, 78, 80, 81]. In recent published paper [80], the SIMD implementation of the RCWT and LBWT have been compared to each other. One example, of this comparison is depicted in Fig. 13. This figure shows the speedup of the SIMD implementation of the RCWT (SSE-RCWT) and LBWT (SSE-LBWT) over the corresponding scalar versions (C-RCWT and C-LBWT) for the first level decomposition of the 2D DWT of Daub-4 transform. We found that the performance of the C-LBWT program is almost the same as the C-RCWT. The performance improvement of the SSE-LBWT is larger than the SSE-RCWT for all image sizes. This is because the SSE-LBWT implementation exploits more DLP of the 2D DWT than the SSE-RCWT implementation. In order to show this, we explain vectorization of the LBWT.

The C implementation of the LBWT approach for Daub-4 transform has already been mentioned in Fig. 8. For SIMD implementation of this algorithm, a circular queue buffer of size $L \times M$ has been used, where L is the filter length and M is the

Table 1 Summary of available multimedia extensions. *Sn* and *Un* indicate *n*-bit signed and unsigned integer packed elements, respectively. Values *n* without a prefix *U* or *S* in the last row, indicate operations work for both signed and unsigned values. [1] Note that 68 instructions of the 144 SSE2 instructions operate on 128-bit packed integer in XMM registers, wide versions of 64-bit MMX/SSE integer instructions

GPP with	AltiVec/	MAX-1/2	MDMX	MMX/	VIS	MMX/	SSE	SSE2	SPU ISA
Multimedia Extension	VMX			3DNow		SIMD			
ISA Name									
Company	Motorola/IBM	HP	MIPS	AMD	Sun	Intel	Intel	Intel	IBM/Sony/ Toshiba
Instruction set	Power PC	PARISC2	MIPS-V	IA32	P. V.9	IA32	IA64	IA64	–
Processor	MPC7400	PA RISC	R1000 PA8000	K6-2	Ultra Sparc	P2	P3	P4	Cell
Year	1999/2002	1995	1997	1999	1995	1997	1999	2000	2005
Datapath width	128-bit	64-bit	64-bit	64-bit	64-bit	64-bit	128-bit	128-bit	128-bit
Size of register file	32 × 128b	(31)/32 × 64b	32 × 64b	8 × 64b	32 × 64b	8 × 64b	8 × 128b	8 × 128b	128 × 128b
Dedicated or shared with	Dedicated	Int. Reg.	FP Reg.	Dedicated	FP Reg.	FP Reg.	Dedicated	Dedicated	Dedicated
Integer data types:									
8-bit	16	–	8	8	8	8	8	16	16
16-bit	8	4	4	4	4	4	4	8	8
32-bit	4	–	–	2	2	2	2	4	4
64-bit	–	–	–	–	–	–	–	2	2
Shift right/left	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Multiply-add	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Shift-add	No	Yes	No	No	No	No	No	No	No
Floating-point	Yes	No	Yes	Yes	No	No	Yes	Yes	Yes
Single-precision	4 × 32	–	2 × 32	4 × 16 2 × 32	–	–	4 × 32	4 × 32	4 × 32
Double-precision	–	–	–	1 × 64	–	–	–	2 × 64	2 × 64
Accumulator	No	No	1 × 192b	No	No	No	No	No	
# of instructions	162	(9) 8	74	24	121	57	70	144[1]	213
# of operands	3	3	3–4	2	3	2	2	2	2/3/4
Sum of absolute-differences	No	No	No	Yes	Yes	No	Yes	Yes	Yes
Modulo addition/ subtraction	8, 16, 32	16	8, 16	8, 16 32	16, 32	8, 16 32, 64	8, 16 32, 64	8, 16 32, 64	8, 16 32, 64
Saturation addition/ subtraction	U8, U16, U32 S8, S16, S32	U16, S16	S16	U8, U16 S8, S16	No	U8, U16 S8, S16	U8, U16 S8, S16	U8, U16 S8, S16	– –

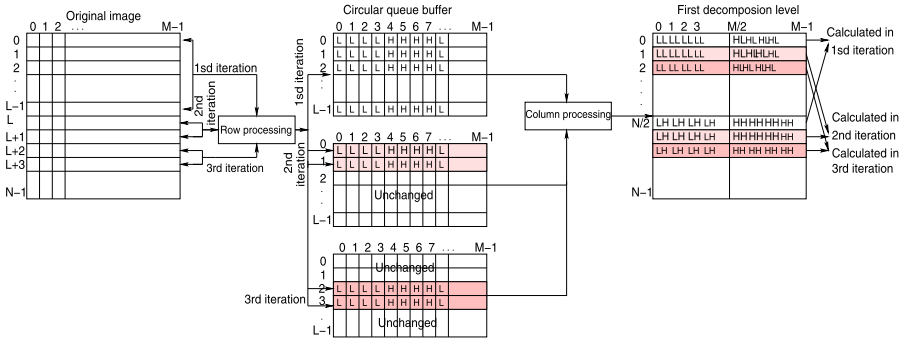

Fig. 14 Three iterations of vectorization of the LBWT approach using 4-way parallelism

image height. $L-2$ rows of the input image are horizontally processed in the prolog part and stored in the $L-2$ rows of the buffer. In the main loop of the program, first, two extra rows of the input image are horizontally processed and stored in the last two rows of the buffer. Second, the filled buffer is vertically filtered and the calculated wavelet coefficient are stored in an auxiliary matrix in the order expected by the quantization step. In the next iteration of the main loop, only two rows of the input matrix are horizontally processed and stored in the first two rows of the buffer. This is because the remaining $L-2$ rows from 2 to $L-1$ are reused for the next iteration. In addition, for another iteration, rows from 4 to $L-1$ and rows 0 and 1 are reused. We have used this buffer as a circular queue and in each iteration two rows are sequentially replaced with wavelet coefficient, which have been horizontally obtained. Figure 14 depicts three iterations of this algorithm. As this figure shows, each four low-pass and four high-pass values are interleaved. This is because of the 4-way SIMD implementation.

As Fig. 13 shows the speedup of the SSE-RCWT implementation for 128×128 image size is 2.63, while the speedup for image sizes larger than 128×128 is 1.22 on average. The main reason for this is that for small image sizes, almost all reads hit the L1 data cache except for compulsory misses. Therefore, the speedup obtained is the speedup resulting from SIMD vectorization. For larger image sizes, the speedup decreases because this SIMD implementation has become memory-bound. In other words, in the vertical filtering, four input rows that are needed to compute one output row can be kept in cache for small image sizes, while for larger image sizes they cannot. This means that for larger image sizes, in addition to compulsory misses, there are capacity misses.

6 Discussion

As already discussed in Sect. 4 the 2D DWT has certain requirements that separate it from other functions. Those requirements of the 2D DWT have been not matched well with the ability of the existing architectures such as ASICs, DSPs, GPPs, and FPGAs.

Table 2 Comparison of different architectures

Architectures	Performance	Flexibility	Power	Cost	Density	Design effort
ASIC	High	Low	Low	High	Medium	High
Reconfigurable hardware	Medium	High	High	Medium	Medium	Medium
Digital signal processors	Medium	High	Medium	Medium	Medium	Medium
GPPs with multimedia extensions	Low	High	Medium	Low	High	Low

Table 2 compares different architectures on some metrics, performance, flexibility, power, cost, density, and design effort. The ASIC approaches offer the advantages of high-performance and low power, but their design and debugging phases involve a significant amount of time. Because the development cost cannot be spread across multiple applications, the cost of ASICs are generally higher than, for example, conventional microprocessor-based solutions. In addition, they are suitable only for mapping of a filter bank length to hardware.

The DSP approaches do not offer sufficient flexibility and high-performance for different filter bank lengths of the 2D DWT. Reconfigurable architectures are more flexible than ASIC designs, while their performance is less than ASIC approaches. In addition, implementing 2D DWT on graphics processing units is still a complicated task.

Although, SIMD-enhanced GPPs are flexible and programmable, they have limited performance. This is because of the following reasons [24]. First, there is a mismatch between the computational format and the storage format of the DWT data. This is because the intermediate coefficients are larger than the original input data. In other words, implementing the DWT on GPPs enhanced with SIMD extensions needs many overhead instructions for converting between different packed data types. Second, existing SIMD computational instructions cannot efficiently exploit DLP of the 2D DWT. As was discussed, the 2D DWT consists of two 1D transforms called horizontal and vertical filtering in the RCWT approach. The horizontal filtering processes the rows while vertical filtering processes the columns. SIMD vectorization of the vertical filtering is straightforward, since the corresponding data of each column are adjacent in memory. Therefore, several columns can be processed without any rearranging of the subwords. For horizontal filtering on the other hand, corresponding elements of adjacent rows are not continuous in memory. In order to employ SIMD instructions, data rearrangement instructions are needed to transpose the matrix. This step takes a significant amount of time. For example, transposing an 8×8 block of bytes, requires 56 MMX/SSE instructions, if the elements are two bytes wide, then 88 instructions are required. As another example, presented results in [24, 79] showed that the dynamic number of instructions of the horizontal filtering is 40 % more than the dynamic number of instructions of the vertical filtering for the $(5, 3)$ lifting scheme.

Finally, there is a mismatch between the reusing of the 2D DWT data and the number of media registers in SIMD architectures. The 2D DWT algorithms usually have regular and constant access patterns. For instance, some 2D DWT implementations use some coefficients that are unmodified through the full execution of the function.

The number of these coefficients is usually small and they can be stored in media registers rather than the memory hierarchy. On the other hand, the number of the SIMD registers is small, and keeping the intermediate results and constants values in the registers is not possible. This increases the number of memory operations.

7 Conclusions and future work

In this paper, we reviewed the concept, implementation, and the different architectures such as ASICs, DSPs, GPPs, FPGAs, and GPUs to process the 2D discrete wavelet transform. The advantages and disadvantages of these architectures for implementation of the 2D DWT have been discussed. The 2D DWT has certain characteristics compared to other tools. For instance, the 2D DWT has been widely using in different applications. The 2D DWT has different filter banks with different filter lengths. Each application needs to implement different decomposition levels. In addition, the DWT has different kinds of parallelism: task- and data-level parallelism. These requirements of 2D DWT have been not matched well with the ability of the discussed architectures to provide both performance and flexibility. In order to show the limitations of the GPPs enhanced with SIMD extensions, we have implemented different algorithms of the 2D DWT, RCWT, and LBWT using SIMD instructions. The experimental results showed that existing SIMD instructions cannot exploit the available DLP efficiently. The largest speedup was 2.8 for the first level decomposition implementation.

New architectures are needed to provide both high-performance with flexibility. For example, reconfigurable architectures offer a compromise between the performance advantages of ASIC and the flexibility of DSPs. In order to increase the performance of time-consuming functions, reconfigurable architectures can be used with a core processor such as a GPP. Such a platform includes both GPPs and dedicated hardware accelerators that have been designed using FPGAs. Such a heterogeneous architecture benefits from the complementarity of processors and hardware accelerator: the flexibility of software execution and the effectiveness of hardware execution.

In our future work, we focus on the combination of the fine-grained reconfigurable architectures such as FPGAs with coarse-grained reconfigurable architectures to increase the performance and to provide flexibility for 2D DWT with different filter banks, filter lengths, and decomposition levels.

References

1. Stollnitz EJ, Derose TD, Salesin DH (1996) Wavelets for computer graphics: theory and applications. Morgan Kaufmann, San Mateo
2. Yusof Y, Khalifa OO (2007) Digital watermarking for digital images using wavelet transform. In: Proc IEEE int conf on telecommunications, May
3. Hsieh MS, Tseng DC, Huang YH (2006) Hiding digital watermarks using multiresolution wavelet transform. IEEE Trans Ind Electron 48(5):875–882
4. Liang KC, Kuo CJ (1997) Progressive image indexing and retrieval based on embedded wavelet coding. In: Proc int conf on image processing, October, pp 572–575
5. Chang T, Kuo CCJ (1993) Texture analysis and classification with tree-structured wavelet transform. IEEE Trans Image Process 2(4):429–441

6. Biswas PK, Chatterji BN (2007) Texture image retrieval using rotated wavelet. Pattern Recognit Lett 28:1240–1249
7. Wang JZ, Wiederhold G, Firschein O, Wei SX (1997) Content-based image indexing and searching using Daubechies' wavelets. Int J Digit Libr, 311–328
8. Mandal MK, Aboulnasr T, Panchanathan S (1996) Image indexing using moments and wavelets. IEEE Trans Consum Electron 3(42):557–565
9. Dettori L, Semler L (2007) A comparison of wavelet, ridgelet, and curvelet-based texture classification algorithms in computed tomography. Comput Biol Med 37(4):486–498
10. Jin Y, Angelini E, Laine A (2005) Wavelets in medical image processing: denoising, segmentation, and registration. In: Handbook of biomedical image analysis volume I: segmentation models part A. Springer, New York, pp 305–358
11. Djebouri D, Djebbari A, Djebbouri M (2005) A new robust GPS satellite signal acquisition using lifting wavelet. Telecommun Radio Eng 65(2–6)
12. Shahbahrami A, Juurlink B, Vassiliadis S (2008) Implementing the 2D wavelet transform on SIMD-enhanced general-purpose processors. IEEE Trans Multimed 10(1):43–51
13. Andreopoulos Y, Masselos K, Schelkens P, Lafruit G, Cornelis J (2002) Cache misses and energy dissipation results for JPEG-2000 filtering. In: Proc 14th IEEE int conf on digital signal processing, pp 201–209
14. Andreopoulos Y, Schelkens P, Cornelis J (2001) Analysis of wavelet transform implementations for image and texture coding applications in programmable platforms. In: Proc IEEE signal processing systems, pp 273–284
15. Andreopoulos Y, Schelkens P, Lafruit G, Masselos K, Cornelis J (2003) High-level cache modeling for 2-D discrete wavelet transform implementations. J VLSI Signal Process 34:209–226
16. Andreopoulos Y, Zervas ND, Lafruit G, Schelkens P, Stouraitis T, Goutis CE, Cornelis J (2001) A local wavelet transform implementation versus an optimal row-column algorithm for the 2D multilevel decomposition. In: Proc IEEE int conf on image processing, vol 3, pp 330–333
17. Chrysafis C, Ortega A (2000) Line-based, reduced memory, wavelet image compression. IEEE Trans Image Process 9(3):378–389
18. Shahbahrami A, Juurlink B, Vassiliadis S (2005) Efficient vectorization of the FIR filter. In: Proc 16th annual workshop on circuits, systems and signal processing (ProRISC2005), November, pp 432–437
19. Kuo SM, Gan WS (2005) Digital signal processors architectures, implementations, and applications. Prentice Hall, New York
20. Trenas MA, Lopez J, Zapata EL, Arguello F (1998) A memory system supporting the efficient SIMD computation of the two dimensional DWT. In: Proc IEEE int conf on acoustics speech and signal processing, May, vol 3, pp 1521–1524
21. Cohen A, Daubechies I, Eauveau JCF (1992) Biorthogonal bases of compactly supported wavelets. Commun Pure Appl Math 45(5):485–560
22. Daubechies I, Sweldens W (1998) Factoring wavelet transforms into lifting steps. J Fourier Anal Appl 4(3):247–269
23. Ferretti M, Rizzo D (2001) A parallel architecture for the 2D discrete wavelet transform with integer lifting scheme. J VLSI Signal Process 28:165–185
24. Shahbahrami A (2008) Avoiding conversion and rearrangement overhead in SIMD architectures. PhD thesis, Delft University of Technology, September
25. Shahbahrami A, Juurlink B, Vassiliadis S (2005) Performance comparison of SIMD implementations of the discrete wavelet transform. In: Proc 16th IEEE int conf on application-specific systems architectures and processors (ASAP), July
26. Andra K, Chakrabarti C, Acharya T (2002) A VLSI architecture for lifting-based forward and inverse wavelet transform. IEEE Trans Signal Process 50(4):966–977
27. Chen CY, Yang ZL, Wang TC, Chen LG (2001) A programmable parallel VLSI architecture for 2D discrete wavelet transform. J VLSI Signal Process 28:151–163
28. Xiong CY, Hou JH, Tian JW, Liu J (2007) Efficient array architectures for multi-dimensional lifting-based discrete wavelet transforms. Signal Process 87(5):1089–1099
29. Weeks M, Bayoumi MA (2002) Three-dimensional discrete wavelet transform architectures. IEEE Trans Signal Process 50(8):2050–2063
30. Liao H, Mandal MK, Cockburn BF (2004) Efficient architectures for 1D and 2D lifting-based wavelet transforms. IEEE Trans Signal Process 52(5):1315–1326
31. Tseng PC, Huang CT, Chen LG (2003) Reconfigurable discrete wavelet transform architecture for advanced multimedia systems. In: Proc IEEE workshop on signal processing systems, August

32. Tseng PC, Huang CT, Chen LG (2005) Reconfigurable discrete wavelet transform processor for heterogeneous reconfigurable multimedia systems. J VLSI Signal Process Syst 41(1):35–47
33. Hyun E, Sima M, McGuire M (2006) Reconfigurable implementation of wavelet transform on an FPGA-augmented NIOS processor. In: Proc IEEE Canadian conf on electrical and computer engineering, May
34. Schelkens P, Decroos F, Cornelis J, Lafruit G, Catthoor F (1999) Implementation of an integer wavelet transform on a parallel TI TMS320C40 platform. In: Proc IEEE workshop on signal processing systems, October
35. Cho JK, Hwang MC, Kim JS, Choi BD, Ko SJ (2004) Fast implementation of wavelet lifting for JPEG2000 on a fixed-point DSP. In: Proc int conf on circuits, systems, computers, and communications, July
36. Chaver D, Prieto M, Pinuel L, Tirado F (2002) Parallel wavelet transform for large scale image processing. In: Proc IEEE int symp on parallel and distributed processing, April, pp 4–9
37. Chaver D, Tenllado C, Pinuel L, Prieto M, Tirado F (2002) 2-D wavelet transform enhancement on general-purpose microprocessors: memory hierarchy and SIMD parallelism exploitation. In: Proc int conf on the high performance computing, December
38. Tenllado C, Setoain J, Prieto M, Pinuel L, Tirado F (2008) Parallel implementation of the 2D discrete wavelet transform on graphics processing units: filter bank versus lifting. IEEE Trans Parallel Distrib Syst 19(3):299–310
39. Gnavi S, Penna B, Grangetto M, Magli E, Olmo G (2002) Wavelet kernels on a DSP: a comparison between lifting and filter banks for image coding. EURASIP J Appl Signal Process 2002(1):981–989
40. Vishwanath M, Owens RM, Irwin MJ (1992) Discrete wavelet transforms in VLSI. In: Proc int conf on application specific array processors, August
41. Denk TC, Parhi KK (1997) VLSI architectures for lattice structure based orthonormal discrete wavelet transform. IEEE Trans Circuits Syst II, Analog Digit Signal Process 44(2):129–132
42. Martina M, Masera G, Piccinini G, Zamboni M (2000) A VLSI architecture for IWT (integer wavelet transform). In: Proc 43rd IEEE midwest symposium on circuits and systems, vol 3, pp 1174–1177
43. Limqueco JC, Bayoumi MA (1998) A VLSI architecture for separable 2D discrete wavelet transform. J VLSI Signal Process Syst 18(2):125–140
44. Weeks M, Bayoumi M (2003) Discrete wavelet transform: architectures, design and performance issues. J VLSI Signal Process 35:155–178
45. Tenllado C, Lario R, Prieto M, Tirado F (2004) The 2D discrete wavelet transform on programmable graphics hardware. In: Proc 4th IASTED int conf on visualization, imaging, and image processing
46. Hopf M, Ertl T (2000) Hardware-accelerated wavelet transformations. In: Proc IEEE TVCG symp on visualization, May
47. Wang J, Wong T, Heng P, Leung C (2004) Discrete wavelet transform on GPU. In: Proc ACM workshop general-purpose computing on graphics processors
48. Mirsky E, DeHon A (1996) MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources. In: Proc IEEE symp on FPGAs for custom computing machines, April, pp 157–166
49. Ebeling C, Cronquist D, Franklin P, Fisher C (1996) RaPiD a configurable computing architecture for compute-intensive applications. Technical report TR-96-11-03, University of Washington Department of Computer Science and Engineering, November
50. Lee RB (1996) Subword parallelism with MAX-2. IEEE MICRO 16(4):51–59
51. Peleg A, Wiljie S, Weiser U (1997) Intel MMX for multimedia PCs. Commun ACM 40(1):24–38
52. Peleg A, Weiser U (1996) MMX technology extension to the intel architecture. IEEE MICRO 16(4):42–50
53. Tremblay M, O'Connor JM, Narayanan V, He L (1996) VIS speeds new media processing. IEEE MICRO 16(4):10–20
54. Bannon P, Saito Y (1997) The alpha 21164PC microprocessor. In: IEEE proc compcon 97, February, pp 20–27
55. Gwennap L (1996) Digital, MIPS add multimedia extensions. Microprocess Rep 10(15):24–28
56. Jennings MD, Conte TM (1998) Subword extensions for video processing on mobile systems. IEEE Concurr 6(3):13–16
57. Lee RB (1997) Multimedia extensions for general-purpose processors. In: Proc IEEE workshop on signal processing systems, November, pp 9–23
58. Advanced Micro Devices Inc (2000) 3DNow technology manual
59. Raman SK, Pentkovski V, Keshava J (2000) Implementing streaming SIMD extensions on the Pentium 3 processor. IEEE MICRO 20(4):47–57

60. Thakkar S, Huff T (1999) The internet streaming SIMD extensions. Intel Technol J, 1–8
61. Semiconductor F (2002) AltiVec technology programming environments manual
62. Diefendorff K, Dubey PK, Hochsprung R, Scales H (2000) AltiVec extension to powerPC accelerates media processing. IEEE MICRO 20(2):85–95
63. Flachs B, Asano S, Dhong SH, Hofstee HP, Kim GGR, Le T, Liu P, Leenstra J, Oh JLBMHJ, Mueller SM, Takahashi O, Watanabe AHY, Yano N, Brokenshire DA, Peyravian M, Vandung T, Iwata E (2006) The microarchitecture of the synergistic processor for a cell processor. IEEE J Solid-State Circuits 41:63–70
64. Hofstee HP (2005) Power efficient processor architecture and the cell processor. In: Proc 11th IEEE int symp on high-performance computer architecture, February, pp 258–262
65. IBM (2007) Synergistic processor unit instruction set architecture, January, version 1.2
66. Asokan R, Nazareth S (2001) Processor architectures for multimedia. In: Proc 14th annual workshop on architecture and system design, November, pp 589–594
67. Hen HY (2002) Programmable digital signal processors: architecture, programming, and applications. Dekker, New York
68. Slingerland NT, Smith AJ (2000) Multimedia instruction sets for general purpose microprocessors: a survey. Technical Report UCB//CSD-00-1124, University of California, December
69. Ferrand F (2003) Optimization and code parallelization for processors with multimedia SIMD instructions. Master's thesis, ENST Bretagne
70. Meerwald P, Norcen R, Uhl A (2002) Cache issues with JPEG2000 wavelet lifting. In: Proc of visual communications and image processing, January
71. Chatterjee S, Brooks CD (2002) Cache efficient wavelet lifting in JPEG 2000. In: Proc IEEE int conf on multimedia, pp 797–800
72. Komi H, Ortega A (2001) Analysis of cache efficiency in 2D wavelet transform. In: Proc IEEE int conf on multimedia and expo, pp 465–468
73. Tao J, Shahbahrami A, Juurlink B, Buchty R, Karl W, Vassiliadis S (2007) Optimizing cache performance of the discrete wavelet transform using a visualization tool. In: Proc 9th IEEE int symp on multimedia, December
74. Shahbahrami A, Juurlink B, Vassiliadis S (2006) Improving the memory behavior of vertical filtering in the discrete wavelet transform. In: Proc 3rd ACM int conf on computing frontiers, May, pp 253–260
75. Chaver D, Tenllado C, Pinuel L, Prieto M, Tirado F (2003) Vectorization of the 2D wavelet lifting transform using SIMD extensions. In: Proc 17th IEEE int symp on parallel and distributed image processing and multimedia
76. Kutil R (2006) A single-loop approach to SIMD parallelization of 2D wavelet lifting. In: Proc 14th Euromicro int conf on parallel, distributed, and network-based processing, pp 413–420
77. Bernabe G, Garcia JM, Gonzales J (2003) Reducing 3D wavelet transform execution time through the streaming SIMD extensions. In: Proc 11th Euromicro conf on parallel distributed and network based processing, February
78. Shahbahrami A, Juurlink B (2007) A comparison of two SIMD implementations of the 2D discrete wavelet transform. In: Proc 18th annual workshop on circuits, systems and signal processing (ProRISC2007), November
79. Shahbahrami A, Juurlink B, Vassiliadis S (2006) Performance impact of misaligned accesses in SIMD extensions. In: Proc 17th annual workshop on circuits, systems and signal processing (ProRISC2006), November, pp 334–342
80. Shahbahrami A (2011) Improving the performance of 2D discrete wavelet transform using data-level parallelism. In: Proc IEEE int conf on high performance computing and simulation, July, pp 362–368
81. Shahbahrami A, Juurlink B (2009) SIMD architectural enhancements to improve the performance of the 2D discrete wavelet transform. In: Proc 12th EUROMICRO conf on digital system design, August, pp 497–504