FACULTY OF INFORMATION TECHNOLOGY department of computer graphics and multimedia

Lifting Scheme Cores for Wavelet Transform

Ph.D. thesis summary

Ing. David Barina

supervised by
prof. Dr. Ing. Pavel Zemcik

Brno, 2015

Abstract

The thesis focuses on efficient computation of the two-dimensional discrete wavelet transform. The state-of-the-art methods are extended in several ways to perform the transform in a single loop, possibly in multi-scale fashion, using a compact streaming core. This core can further be appropriately reorganized to target the minimization of certain platform resources. The approach presented here nicely fits into common SIMD extensions, exploits the cache hierarchy of modern general-purpose processors, and is suitable for parallel evaluation. Finally, the approach presented is incorporated into the JPEG 2000 compression chain, in which it has proved to be fundamentally faster than widely used implementations.

Contents

Chapter 1

Introduction

Information contained in many different physical phenomena (e.g., sounds, images) can be described using signals. Manipulation with these signals using computers is the subject of the signal processing field, which uses a variety of mathematical tools to analyse, process, and synthesize them. The wavelet transform is one of these tools, allowing for the time–frequency signal analysis. In other words, one can view the information associated with a particular time and frequency.

The thesis focuses on methods for computing the discrete wavelet transform. Specifically, it extends existing single-loop methods to enable dealing with two-dimensional multi-scale decomposition and to efficiently utilize features of modern CPUs.

The discrete wavelet transform (DWT) is a signal-processing tool suitable to decompose an analysed signal into several scales. For each such scale, the resulting coefficients are formed in several subbands. In the one-dimensional case, the subbands correspond to low-pass (a) and high-pass (d) filtered subsampled variants of the original signal. Plenty of applications are built over the discrete wavelet transform. One of them, nevertheless, stands out quite markedly. The transform is often used as a basis for sophisticated compression algorithms. Particularly, JPEG 2000 is an image-coding system based on such a wavelet compression technique. Unfortunately, there exist several major issues with effective implementation of the discrete wavelet transform. This holds true in particular for images with high resolution (4K, 8K, aerial imagery) decomposed into a number of scales (e.g., 8 scales). The issues are discussed below.

In case of the two-dimensional transform, one level of the transform can be realized using the separable decomposition scheme. In this scheme, the coefficients are evaluated by successive horizontal and vertical 1-D filtering, resulting in four disjoint groups (a, h, v, and d subbands). A naive algorithm of 2-D DWT computation directly follows the horizontal and vertical filtering loops. Unfortunately, this approach is encumbered with several accesses to intermediate results. State-of-the-art algorithms fuse the horizontal and vertical loops into a single one, which results in the single-loop approach.

One level of the just described 1-D transform can be computed utilizing a convolution with two complementary filters. However, on most architectures there exists a more efficient scheme to calculate the transforms coefficients. This scheme is called lifting and, in contrast to convolution, it benefits from sharing intermediate results.

As indicated above, for high-resolution data decomposed into several scales by a typical separable transform, immensely many CPU cache misses occur. These cache misses significantly slow down the overall calculation. Moreover, in real implementations, a large image block often needs to be buffered, which makes the transform memory-demanding. The motivation behind this work is to overcome these issues.

The thesis contributes to the state of the art of discrete wavelet transform computation methods. Particularly, the following paragraph outlines the issues that are not solved satisfactorily when using the existing methods.

The state-of-the-art approaches treat signal boundaries in a complicated and inflexible way. When we take these approaches into consideration, we find that parallelization, SIMD vectorization, and the cache hierarchy exploitation are not handled well. This is especially true in conjunction with multi-scale decomposition. Furthermore, the transform fragments cannot be computed according to arbitrary application requirements. For example, a particular transform block at a particular scale cannot be obtained with minimal or no unnecessary calculations. Finally, these approaches do not address the problem of scheme reorganization aimed at minimizing some of the platform's resources at the expense of others.

The thesis focuses on the CDF (Cohen-Daubechies-Feauveau) 5/3 and 9/7 wavelets, which are often used for image compression (e.g. the JPEG 2000 or Dirac standards). However, the methods are general and they are not limited to any specific type of transform.

Chapter 2

State of the Art

This chapter reviews the wavelet theory and the state of the art of the efficient computation of the two-dimensional discrete wavelet transform.

The discrete wavelet transform can be understood as a method suitable for the decomposition of a signal into low-pass and high-pass frequency components through so-called wavelets. This section introduces wavelet theory in a level of detail necessary to understand the thesis.

Wavelets are functions generated from one basic function by dilations and translations. Many constructions of wavelets have been introduced in the literature in the past three decades; for example [1]. As a key advance, I. Daubechies [2] constructed orthonormal bases of compactly supported wavelets in 1988. Subsequently, in 1992, Cohen–Daubechies–Feauveau (CDF) [3] biorthogonal wavelets provided several families of symmetric (linear phase) biorthogonal wavelet bases. In the meantime, in 1989, S. Mallat [4, 5] demonstrated the orthogonal wavelet representation of images. It was computed with a pyramidal algorithm based on convolutions with quadrature mirror filters (QMF). In the mid-1990s, W. Sweldens [6, 7, 8] presented the lifting scheme which sped up such decomposition. He showed how any discrete wavelet transform can be decomposed into a sequence of simple filtering steps (lifting steps).

For a description of the filters, the well known z-transform notation is employed. The transfer function of the one-dimensional FIR filter $h(k)$ is defined as

$$H(z) = \sum_k h(k)\, z^{-k}. \tag{2.1}$$

For a better insight, the discrete wavelet transform can be understood as the approximation of a continuous signal by superposition of the individual wavelets. Generally, the wavelets $\psi \in L^2(\mathbb{R})$ are continuous functions from the Hilbert space of finite energy functions localized in both time and frequency. However, if we limit ourselves to the discrete wavelet transform, the wavelets are further constrained by the following equations. For illustration, two wavelets

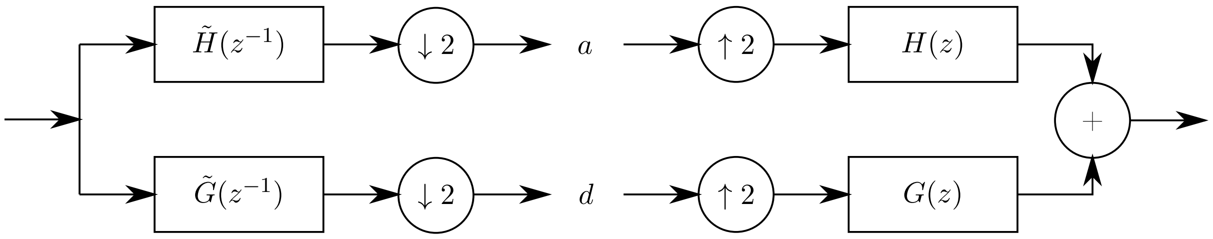


Figure 2.1: Analysis and synthesis part of DWT using FIR filters.

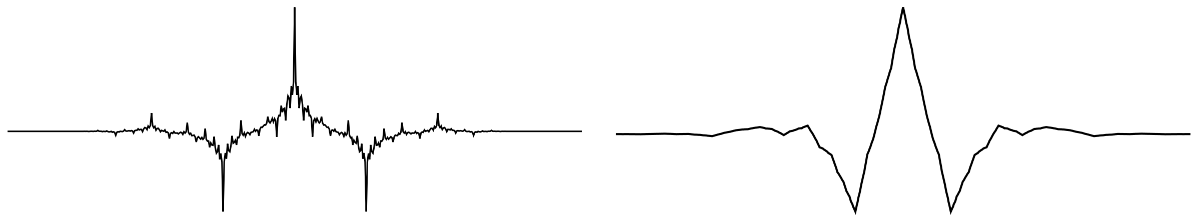


Figure 2.2: Shape of CDF 5/3 and CDF 9/7 wavelets. CDF 5/3 situated on the left, while CDF 9/7 on the right.

frequently used for DWT are plotted in Figure 2.2. The approximation is calculated through two conjugated quadrature filters often referred to as h, g. The relation between the wavelet and these filters

$$\phi(t) = \sqrt{2} \sum_n h(n)\,\phi(2t-n), \tag{2.2}$$

$$\psi(t) = \sqrt{2} \sum_n g(n)\,\phi(2t-n), \tag{2.3}$$

where $\phi \in L^2(\mathbb{R})$ is a scaling function, was formulated [4, 9] by S. Mallat. As a consequence of these equations, the multi-scale DWT can be computed by passing the signal through a filter bank comprising the $\tilde{h}$, $\tilde{g}$ filters followed by subsampling. One level of the decomposition linked with the synthesis is shown in Figure 2.1. The method is also referred to as the multiresolution analysis (MRA).

2.1 Lifting Scheme

DWT decomposes the signal into low-pass (a) and high-pass (d) frequency components using two analysis filters – $\tilde{h}$ (low-pass) and $\tilde{g}$ (high-pass) – followed by subsampling. The inverse transform first upsamples the a and d components and then uses two synthesis filters h (low-pass) and g (high-pass). The signal-processing view of such decomposition and analysis is shown in Figure 2.1. Readers not closely familiar to DWT are referred to the excellent book [10] by S. Mallat. For details about the lifting scheme, see [8, 7].

The polyphase representation [11, 8] is a convenient tool to express the $h, g, \tilde{h}, \tilde{g}$ filters as a sum of shorter filters formed by even (e) and odd (o) coefficients of the original ones.

Having these filters, the assembled polyphase matrix

$$P(z) = \begin{bmatrix} H_e(z) & G_e(z) \\ H_o(z) & G_o(z) \end{bmatrix} \tag{2.4}$$

expresses the inverse transform. Such a polyphase matrix can be factorized, e.g. using the Euclidean algorithm [12], so that

$$P(z) = \prod_{i=0}^{I-1} \left\{ \begin{bmatrix} 1 & S_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ T_i(z) & 1 \end{bmatrix} \right\} \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix}, \tag{2.5}$$

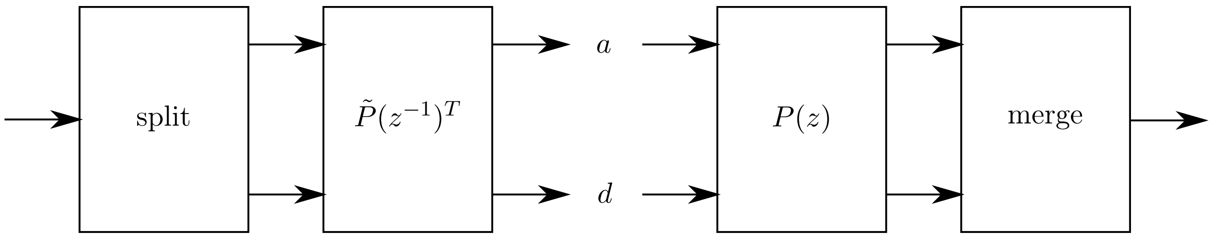


Figure 2.3: Analysis and synthesis part of DWT using lifting schemes.

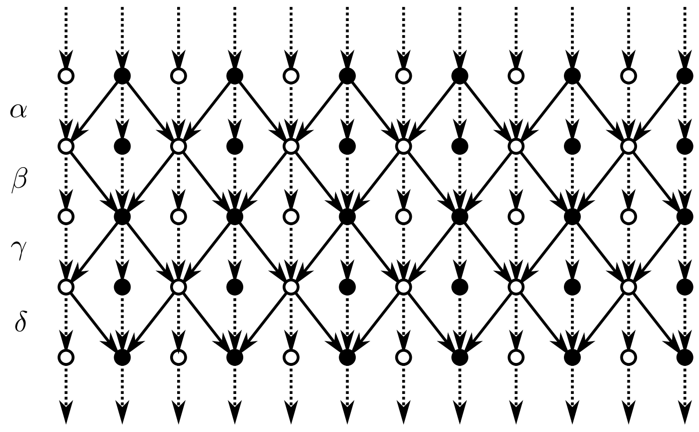


Figure 2.4: Data-flow diagram of CDF 9/7 lifting scheme. The blank bullets represent d coefficients, whereas the solid ones a coefficients. The solid arrows denote multiply operations. The dotted arrows just forward the value. The arrows are accumulated into the bullets.

where K is a non-zero constant, and polynomials $S_i(z), T_i(z)$ for $0 \leq i \leq I-1$ represent the individual lifting steps. Since, the DWT has the perfect reconstruction property $P(z)\,\tilde{P}(z^{-1})^T = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix and $\cdot^T$ denotes the transposition, the dual polyphase matrix

$$\tilde{P}(z) = \prod_{i=0}^{I-1} \left\{ \begin{bmatrix} 1 & 0 \\ -S_i(z^{-1}) & 1 \end{bmatrix} \begin{bmatrix} 1 & -T_i(z^{-1}) \\ 0 & 1 \end{bmatrix} \right\} \begin{bmatrix} 1/K & 0 \\ 0 & K \end{bmatrix} \tag{2.6}$$

describes the analysis part of DWT (forward transform). The system is illustrated in Figure 2.3. The $-T_i(z^{-1})$ is called the predict, whereas $-S_i(z^{-1})$ is called the update.

Before the decomposition, the input signal is split into two disjoint groups a, d (using even/odd sample indices). Then, the individual lifting steps are performed

$$\begin{bmatrix} d & a \end{bmatrix}^T = \tilde{P}(z^{-1})^T \begin{bmatrix} d & a \end{bmatrix}^T \tag{2.7}$$

resulting into a, d subbands.

Focusing on the CDF 9/7 wavelet as an example, the forward transform in can be expressed

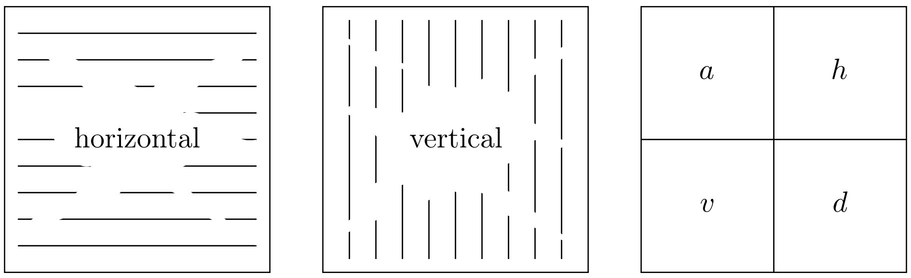


Figure 2.5: Illustrative 2-D decomposition using double sequence of 1-D transforms.

[8] by the dual polyphase matrix

$$\tilde{P}(z) = \begin{bmatrix} 1 & \alpha\left(1+z^{-1}\right) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \beta\left(1+z\right) & 1 \end{bmatrix} \begin{bmatrix} 1 & \gamma\left(1+z^{-1}\right) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \delta\left(1+z\right) & 1 \end{bmatrix} \begin{bmatrix} \zeta & 0 \\ 0 & 1/\zeta \end{bmatrix}. \tag{2.8}$$

Where ζ is called the scaling constant. The $\alpha, \beta, \gamma, \delta, \zeta$ are real constants specific to the CDF 9/7 transform. The forms $(1+z^{-1})$ and $(1+z)$ of the polynomials indicate symmetry of the lifting steps as well as the original filters. The corresponding data-flow diagram is shown in Figure 2.4 (scaling is omitted for simplicity). Please note that this particular wavelet is widely used in image processing, for example in JPEG 2000 compression standard.

2.2 2-D Decomposition

S. Mallat [4] demonstrated the wavelet representation of two-dimensional signals computed with a pyramidal algorithm based on convolutions with quadrature mirror filters. One level of such 2-D pyramid leads to a quadruple of wavelet coefficients (a, h, v, d) as outlined in Figure 2.5. The transform is defined as the tensor product of 1-D transforms. In this case, the two-dimensional transform consists of horizontal and vertical filtering steps. Considering the lifting scheme, the order of these steps has some constraints, but it is not strictly fixed (the horizontal and vertical steps can be interleaved). The decomposition is repeatedly applied on a subband which leads to the pyramidal decomposition. Note a naive algorithm implementing this 2-D scheme could perform a series of 1-D transforms horizontally, followed by a series of 1-D transforms vertically (or vice versa). The above mentioned 1-D transform could be implemented through the filter bank (convolution) or the lifting scheme.

The following discussion considers the situation in the context of a naive implementation. It does not matter whether the convolution or the lifting scheme is used. In both cases, the data coefficients are accessed at least twice (firstly for horizontal, secondly for vertical pass). Thus, the approach is inherently burdened with several accesses to intermediate results. More sophisticated algorithms could perform these separable steps joined together which could lead even into a single loop transform. In any case, the decomposition is further applied to a subband in order to get multi-scale representation. As in the previous case, individual scales of the decomposition can be performed in interleaved manner. Performing the multi-scale decomposition in this way is described as the multi-scale single-loop approach.

Images can be understood as finite two-dimensional arrays (matrices), where the values of individual elements represent image pixels. As these matrices are finite, a problem with appropriate treatment of transform margins arises.

2.3 Computation Schedules

This section discusses existing methods of computing the 2-D discrete wavelet transform on various platforms, especially contemporary general-purpose processors (GPPs).

Some type of the CPU cache is present in all modern platforms. Excellent introduction to this topic can be found in [13]. The cache is usually organized as a hierarchy of more cache levels. In the cache hierarchy, the individual coefficients of the transform are stored inside larger and integral blocks – cache lines. A hardware prefetcher attempts to speculatively load these lines in advance, before they are actually required. Moreover, due to a limited cache associativity, it is also impossible to hold in cache the lines corresponding to arbitrary memory location at the same time. In detail, the cache lines are divided into several sets according to a associativity of the cache. The cache associativity indicates the number of lines from a particular set which can be hold in the cache at one time. When more lines from this set are accessed, the older lines are evicted in favour of the new ones.

Originally, the problem of efficient implementation of the 1-D lifting scheme was addressed in [14] by Ch. Chrysafis and A. Ortega. Their approach is very general and it is not focused on parallel processing. Nonetheless, this is essentially the same method as the on-line or pipelined method mentioned in other papers (although not necessarily using lifting scheme nor 1-D transform). The key idea is to make the lifting scheme causal, so that it may be evaluated as a running scheme without buffering of the whole signal.

Many authors have tried to find an efficient schedule for calculating the 2-D lifting scheme. Having the input 2-D image in the main memory, different strategies of 2-D transform implementation can be used. These strategies can be divided into three groups – row-column (fully separable), block-based, and line-based methods. The groups will be discussed with the individual techniques below. Aside from these basic strategies, several techniques were independently presented in several papers. All of them led to performance improvements.

The separable implementation of the 2-D transform is performed by two passes of the 1-D transform – horizontal and vertical pass. The horizontal pass densely visits the coefficients likely prefetched in the cache. Usually, there is no bottleneck in the horizontal pass. However, the vertical pass accesses the coefficients using a stride that prevents the hardware prefetcher to do its job well. Moreover, usually only one coefficient from each cache line is accessed; the rest remains unused. Finally, considering the vertical access pattern, the coefficients lying in a particular column likely map into the same cache set, especially considering power-of-two [15, 16] data sizes.

To solve the last of the mentioned issues, several authors [15, 16, 17, 18, 19] suggested to add a padding after each data row (or the resulting subband row in some cases). This row extension causes the coefficients in a particular column to map into different sets with high probability. In particular, the odd or prime strides are often used.

Since only one of the coefficients settled in the cache lines is used in vertical pass, many authors [15, 20, 21] discovered a technique leading to better utilization of cache lines. The technique is referred to as the aggregation, strip-mine, or loop tiling. Using it, several adjacent columns are filtered concurrently, likely using all the coefficients in the cache line.

So far, the input as well as the output data were stored using a linear-memory layout (particularly, the row-major layout). Some authors investigated the influence of more complicated, possibly non-linear memory layouts. The 4-D, Morton layouts are internally organized into blocks and thus imply the block-based strategy mentioned above. The working set for each block can now fit into the cache. The performance of these layouts was investigated in [22, 23, 24]. The "mallat" layout utilized in [21, 20, 25] uses an auxiliary matrix to store the results of the horizontal filtering. As a result, no rearrangement stage is needed after the transform, since the coefficients can be directly stored at arbitrary locations in the original memory area. Using the recursive layout, each subband is laid out contiguously in memory. This is especially useful for multi-scale decomposition, where the resulting subbands are transformed once more. This layout was employed in [21, 20].

Among all these disclosed techniques, probably the most important one is to interleave processing of the vertical and horizontal loop. This 2-D technique is often referred to as the pipelined, line-based, or single-loop transform. Some granularity (e.g., several input lines, large blocks) is used for interleaving of the loops. For instance, D. Chaver [26] used the block-based interleaving with non-linear 4-D memory layout. Moreover, in [26, 27, 28, 29, 30], the line-based interleaving was used (at least two lines are needed). The most sophisticated techniques were investigated by R. Kutil in [16], focused on CDF 9/7 wavelet and SIMD vectorization. In Kutil's work, one step of the lifting processing requires two values (a pair) to perform an loop iteration. Thus, the algorithm needs to perform two horizontal filterings (on two consecutive rows) at once. For each row, a low-pass and a high-pass coefficient are produced, which makes 2×2 values in total. The algorithm passes four values from one iteration to the other in the horizontal direction for each row. In the vertical direction, the algorithm needs to pass four rows between iterations. This algorithm can be vectorized by handling the coefficients in blocks. Special prolog and epilog parts are needed (at least nine, if even/odd signal lengths are not considered).

2.4 Lifting Vectorization

This section should serve as a bridge between the above-described methods and my own work, presented in the following chapters. Let me now focus on the data-flow diagrams of signal-processing algorithms. Vectorization is the process of evaluating these diagrams using operations that are applied to whole vectors instead of individual coefficients. Many GPPs have vector instruction sets which apply the same operation simultaneously to several coefficients in such a

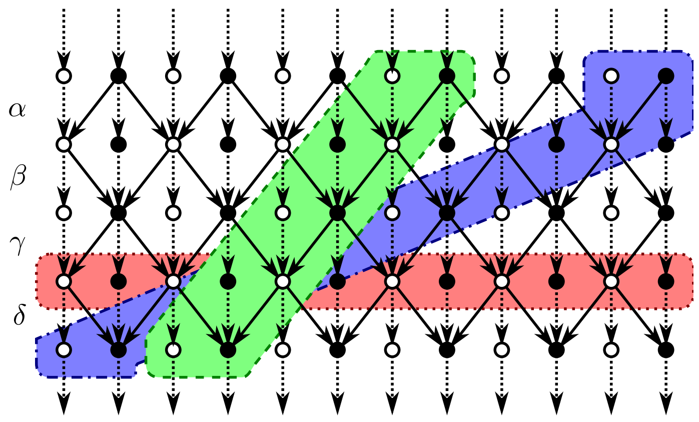


Figure 2.6: Vectorizations of CDF 9/7 lifting scheme. In all cases, the highlighted areas are evaluated in a single iteration of the loop.

vector. Exploiting these instruction sets can be a particular reason of the vectorization. The vector instruction sets are also called SIMD sets or SIMD extensions. Intel's SSE or AVX are frequent examples of such sets. Even if no vector instructions can be utilized, dividing the diagram into the vectors may be useful thanks to a data locality in such vector.

One particular example of this technique can be found in [31] where the author considered the vectorization of FIR filtering (i.e. convolution). Basically, he identified three methods of the vectorization in the convolution

$$y(n) = \sum_k x(n-k)\,h(k) \tag{2.9}$$

data-flow diagram. In the equation above, two loops can be seen – the inner one for k and the outer for n variable. The causality of the scheme and dependencies of these loop iterations in principle allow three vectorization methods. In [31], these are denoted as A, B, and C. In method A, several consecutive inner loop iterations are combined into one vectorized iteration. In this iteration, a sample of the signal is associated with a sample of the filter coefficient. Unfortunately, dependencies between iterations break the possibility to utilize a parallel evaluation. In method C, a sample of the signal is associated with a vector of several distinct filter coefficients. This allows for parallel processing. However, several shuffle operations are required to implement this method. In method B, several input samples are associated with a vector of the same filter coefficient. Also this method allows for parallel processing.

Now, consider the data-flow diagram of the lifting scheme for CDF 9/7 wavelet. Three analogous methods can be identified here as well. For their understanding, please refer to the Figure 2.6. The terminology will be outlined with respect to this figure. The dashed green area corresponds to so-called vertical vectorization. This method was employed under different names in many papers, e.g. in [32]. The method can be seen as an analogy to the method A from the previous paragraph. The dash–dot blue area corresponds to the diagonal vectorization which was proposed in [III]. It can be seen as an analogy to the method C. Finally, similarly to method B, the dotted red area depicts the horizontal vectorization which was investigated in [IV].

Chapter 3

Lifting Core

The main contribution of the thesis is presented in this chapter. The contribution is a formulation of a computation unit built over the lifting scheme technique. The direct consequence of this formulation is the possibility of reorganizing operations in order to minimize the requirements for certain resources. Moreover, several other possibilities arise – for example, an elegant treatment of signal boundaries, or, in the case of multi-dimensional signals, a variety of allowed processing orders. The presented unit is further referred to as *the core*. In this chapter, the core is formally specified.

In this paragraph, some terminology necessary to understand the following text is clarified. Lag F describes a delay of the output samples with respect to the input samples. The stage is used in the sense of the scheme step, usually the lifting step. In linear algebra, such stage can be described by the linear operator (a matrix) mapping the input vector onto the output vector. In this context, the operation denotes the multiply–accumulate (MAC) operation. Considering the output coefficient, the most demanding operation is identified as the operation having the highest number of operands. Please note that the notation is slightly different in some parts of this chapter – using a subscript for indexing the signals.

The following part of the chapter leads to the formulation of the core. Although the thesis has focused on image processing, the one-dimensional transform will be discussed first. The multi-scale discrete wavelet transform decomposes the input signal

$$\left(a^0_{n_0}\right)_{0 \le n_0 < N_0} \tag{3.1}$$

of size $N_0 = N$ samples into $J > 0$ scales giving rise to the resulting wavelet bands

$$\left(d^j_{n_j}\right)_{0 \le n_j < N_j}, \tag{3.2}$$

the temporary bands

$$\left(a^j_{n_j}\right)_{0 \le n_j < N_j}, \tag{3.3}$$

at scales $0 < j < J$, and the residual signal

$$\left(a^J_{n_J}\right)_{0 \le n_J < N_J}, \tag{3.4}$$

at the topmost scale J.

To solve the issues summarized at the beginning of this thesis, a unit which continuously consumes the input signal a^j and produces the output a^{j+1}, d^{j+1} subbands is proposed. This unit was also presented in [V]. As mentioned above, this unit is referred to as the "core" in this thesis. As a consequence of the DWT nature, the core has to consume pairs of input samples. The input signal is processed progressively from the beginning to the end, therefore in a single loop. Note that it is also possible to run these cores in parallel – this possibility is discussed at the end of the chapter. The corresponding output samples are produced with a lag F samples depending on the underlying computation scheme. For each scale $0 \leq j < J$, the core requires access to an auxiliary buffer

$$B^j. \tag{3.5}$$

These buffers hold intermediate results of the underlying computation scheme. At each scale, the size of the buffer can be expressed as κ coefficients, where κ is the number of values that have to be passed between adjacent cores.

Considering the single-loop approach, the vertical and diagonal vectorizations formulated in the previous chapter can be understood as baseline examples of the underlying computation scheme. However, the possibilities are much larger, as disclosed below in the thesis.

To simplify relations, two functions will be introduced.The function $\Theta(n) = n + F$ maps core output coordinates onto core input coordinates with the lag F. The function $\Omega(n) = \lceil n/2 \rceil$ maps the coordinates at the scale j onto coordinates at the scale $j+1$ with respect to the chosen coordinate system. Note that the $\Omega(n)$ can be defined in many ways.

The core transforms the fragment I_n^j of an input signal onto the fragment O_n^j of an input signal

$$\mathrm{I}_n^j = \begin{pmatrix} a_{\Theta(n)}^j & a_{\Theta(n+1)}^j \end{pmatrix}^T, \tag{3.6}$$

$$\mathrm{O}_n^j = \begin{pmatrix} a_{\Omega(n)}^{j+1} & d_{\Omega(n+1)}^{j+1} \end{pmatrix}^T, \tag{3.7}$$

while updating the auxiliary buffer.

Finally, operations performed inside the core can be described using a matrix C as the relationship

$$\boldsymbol{y} = C\,\boldsymbol{x} \tag{3.8}$$

from the input vector

$$\boldsymbol{x} = \mathrm{I}_n^j \parallel \mathrm{B}^j \tag{3.9}$$

onto the output vector

$$\boldsymbol{y} = \mathrm{O}_n^j \parallel \mathrm{B}^j, \tag{3.10}$$

where $\parallel$ denotes the concatenation operator. The (3.8) is the most fundamental equation of this thesis. In this linear mapping, the matrix C defines the core.

The meaning and the number of individual coefficients in B^j is not firmly given. The choice of the matrix C is a degree of freedom of the presented framework. Particularly, this matrix can be reorganized in order to minimize some of the resources (e.g., memory cells, operations, latency). An illustrative example of such a reorganization is demonstrated in the next section.

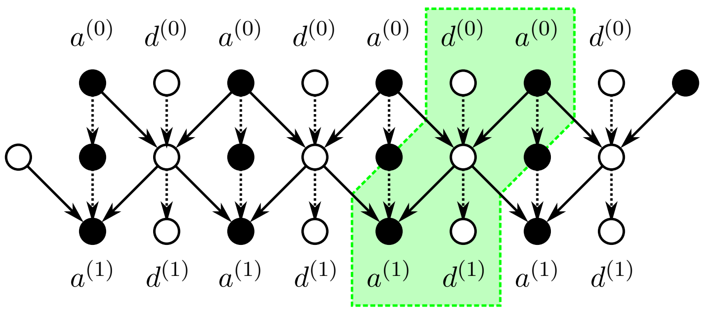


Figure 3.1: Implementation of CDF 5/3 transform. The core is the highlighted region. Circles represent the results of operations.

3.1 Core Reorganization

The following discussion is focused on a single pair of lifting steps of the vertically vectorized core. Since such a core consists of two lifting steps (predict and update), two data dependent blocks of operations have to be evaluated. This section demonstrates how to reduce the depth of the calculation per the output coefficients. These ideas were also presented in [II]. The basic principles of this idea is disclosed discussing the CDF 5/3 transform.

At the beginning, the CDF 5/3 lifting scheme is described. This description is related to Figure 3.1. As in [8], CDF 5/3 lifting scheme is defined using α and β constants. The resulting core has 2 independent stages suitable for hardware pipelining. The core consists of 5 operations in total. The number of operands of the most demanding expression is 3 in both stages. The lag $F = 2$ or $F = 1$ can be obtained by a trivial reorganization of coefficients stored in the auxiliary buffer. The following discussion focuses on the latter case.

In more detail, the core transforms the vector $\boldsymbol{x}$ into $\boldsymbol{y}$. These two vectors are composed of two samples of input signal and some specific intermediate results.

The individual stages correspond to predict T_α and update S_β steps, respectively. The core can be then described in matrix notation as

$$\boldsymbol{y} = C_{\alpha,\beta}\, \boldsymbol{x} = S_\beta\, T_\alpha\, \boldsymbol{x}. \tag{3.11}$$

The computation inside the $S_\beta T_\alpha$ may be implemented as follows. For a better understanding, the slice of computation is depicted in Figure 3.1.

$$d_n^{(1)} = d_n^{(0)} + \alpha\left(a_{n-1}^{(0)} + a_n^{(0)}\right) \tag{3.12}$$

$$a_{n-1}^{(1)} = a_{n-1}^{(0)} + \beta\left(d_{n-1}^{(1)} + d_n^{(1)}\right) \tag{3.13}$$

The reference implementation above has latency of 2 lifting steps. In the same single-loop framework, one can reduce this latency to just one step. Such a core still uses the auxiliary buffer of $\kappa = 2$ coefficients, has the lag of one sample and produces numerically exactly the same results. However, the operations inside the core are appropriately reorganized. The key idea is a direct calculation of a coefficient from one intermediate coefficient and two current input samples. The intermediate coefficient is forwarded by previous core iteration through the

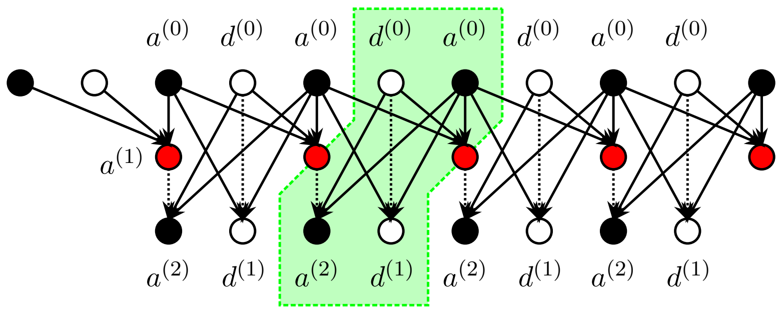


Figure 3.2: The 1-D implementation of CDF 5/3 filter with reduced latency. The circles in different color correspond to $a^{(1)}$ coefficients from (3.15). The core is highlighted.

core	lag F	buffer κ	latency	max. operands	total operands
vertical lag-2	2	2	2	5	6
vertical lag-1	1	2	2	5	6
reorganized	1	2	1	3	9

Table 3.1: Attributes of CDF 5/3 cores. Only the cores based on the vertical vectorization are shown. The latency is the number of subsequent steps.

auxiliary buffer. As a result, all the operations inside can be computed in parallel. The auxiliary buffer covers the values of $(a^{(0)}, a^{(1)})$. The resulting dataflow graph is shown in Figure 3.2. For clarity, a new constant $\gamma = 1 + 2\alpha\beta$ was introduced. The core implementation is as follows.

$$d_n^{(1)} = d_n^{(0)} + \alpha\left(a_{n-1}^{(0)} + a_n^{(0)}\right) \tag{3.14}$$

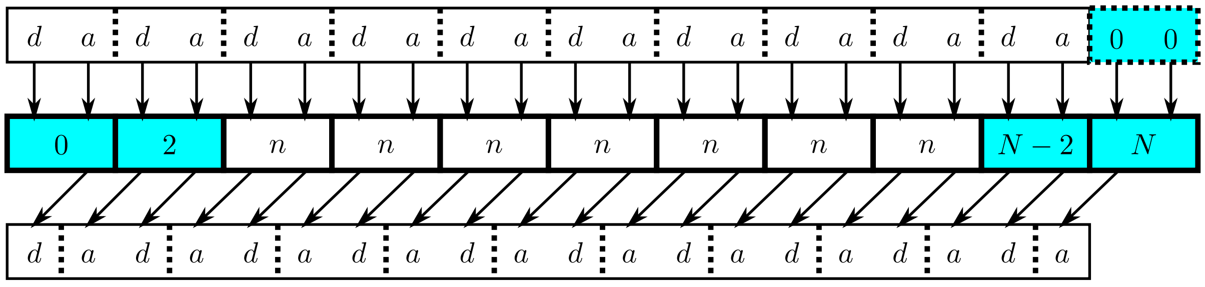
$$a_n^{(1)} = \gamma a_n^{(0)} + \alpha\beta a_{n-1}^{(0)} + \beta d_n^{(0)} \tag{3.15}$$

$$a_{n-1}^{(2)} = a_{n-1}^{(1)} + \beta d_n^{(0)} + \alpha\beta a_n^{(0)} \tag{3.16}$$

This new scheme is fundamentally different from the original one described earlier. In the original scheme, the a coefficient was calculated based on the value of the d coefficient. Unfortunately, the d coefficient had first to be calculated from the input samples. This process implied a delay of 2 steps. However, the newly formed core has a latency of 1 step. Note that the new scheme cannot be evaluated using the horizontal vectorization due to the formation of the new intermediate results $a^{(1)}$. On the other hand, the core consists of 9 operations in total. See Table 3.1 for the comparison of all three mentioned cores. The total depth of the entire calculation is smaller (as the number of subsequent stages was halved) than in the original case. In matrix notation, the reorganized core can be expressed as

$$\boldsymbol{y} = C_{\alpha,\beta}\,\boldsymbol{x}. \tag{3.17}$$

The core reorganization takes new depths with the increasing number of dimensions. Before switching to multiple dimensions, further in the text, a graceful signal border treatment is demonstrated.



d a d a d a d a d a d a d a d a d a d a 0 0
0 2 n n n n n n n N − 2 N
d a d a d a d a d a d a d a d a d a d a

Figure 3.3: Signal processing using the mutable core. The input position on the original position n is shown. The first and last two cores (highlighted) differ from the others.

3.2 Treatment of Signal Boundaries

To keep the total number of wavelet coefficients equal to the number of input samples, symmetric border extension is widely used. A particular variant of this extension is employed in JPEG 2000 standard. This section describes the core calculating the CDF 5/3 transform. The core described in the previous section consumes the input signal (a_n^j, a_{n+1}^j) per fragments of two samples. After performing the calculations, the $(d_{\Omega(n)}^{j+1}, a_{\Omega(n+1)}^{j+1})$ is produced with a lag F. For the purposes of the following discussion, only even length signals are considered. The core consists of two stages suitable for hardware pipelining.

As mentioned earlier, the core processes the signal in the single loop. The naive way of border handling is described first. Due to the symmetric border extension, the core begins the processing at a certain position before the start of the actual signal. Analogously, the processing stops at a certain position after the end of the signal. The samples outside the actual signal are mirrored into the valid signal area. This processing introduces the need for buffering of the input at least at the beginning and the end of the signal. This buffering breaks the ability of simple stream processing, especially considering the multi-scale decomposition. All the stage-of-the-art approaches also suffer from this issue.

The situation can be neatly resolved changing the core near the signal border. In more detail, the "mutable" core performs 5 different calculations depending on the position of the input signal. Therefore, the core comprises 2×5 slightly different steps (stages) in total. As in the previous section, each of them is implemented by a linear transformation operating with four-component vectors. This can be written in matrix notation as

$$\boldsymbol{y} = S_{\beta,\Theta(n)} \, T_{\alpha,\Theta(n)} \, \boldsymbol{x}, \tag{3.18}$$

where $T_{\alpha,\Theta(n)}, S_{\beta,\Theta(n)}$ are the linear transformations of the predict and update stages performed at the subsampled output position $\Theta(n)$. Moreover, $\boldsymbol{x} = \begin{bmatrix} a^b & d^b & a_n & d_n \end{bmatrix}^T$ and $\boldsymbol{y} = \begin{bmatrix} a^b & d^b & a_{n-1} & d_{n-1} \end{bmatrix}^T$ are the input and output vectors, respectively. Here, the b superscript denotes the content of the auxiliary buffer. These coefficients are generated in $T_{\alpha,\Theta(n)}$ so that these can be used by $T_{\alpha,\Theta(n+2)}$ at the same time at which $S_{\beta,\Theta(n+2)}$ runs. It is essential that the coefficients a^b, d^b are initially set to zero. The output signal is generated with a lag

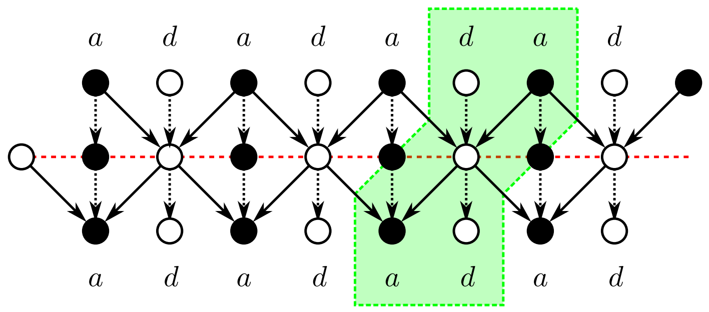


Figure 3.4: Parallel implementation of CDF 5/3 transform. The core is highlighted. The dashed horizontal line identifies the memory barrier. This line intersects the d coefficients (blank bullets), which are, therefore, subject of the synchronization.

$F = 1$ sample with respect to the input signal. The input a samples outside of the input signal are treated as zeros. Similarly, the output a, d coefficients outside of the output signal are discarded. The following table describes the individual $T_{\alpha,\Theta(n)}, S_{\beta,\Theta(n)}$ transformations. The transform is defined using the α, β constants. In addition, Figure 3.3 illustrates theirs usage. As a result, the signal is transformed without buffering, possibly on a multi-scale basis.

3.3 Parallel Cores

This section considers conditions under which the cores can run simultaneously. The methods presented here were used earlier in several papers, e.g. [33]. Their work was further examined and improved in [VIII, IX].

The single-loop cores presented in this chapter require the auxiliary buffer to pass the intermediate results into subsequent iteration. Such a strategy is however not suitable for all platforms, especially for the massively-parallel architectures. Considering such platforms, it would be preferable to run all the cores in parallel. Indeed, such approaches were already presented in literature, e.g. [33]. However, they were not referred to as the "cores" so far. I will refer these to as parallel cores in this section.

The single-loop cores can be notionally modified to exchange the intermediate results directly without the auxiliary buffers. In parallel environment, it causes the need of synchronization points. These points are denoted as the memory barriers. Particular example of such a barrier is shown in Figure 3.4. In the referenced figure, the dashed horizontal line indicates the point where the processing units have to be synchronized. The initial values can be loaded without the synchronization. However, the intermediate results after the first lifting steps can be loaded no sooner than after the synchronization point. Note that it is acceptable to pass the intermediate results in either direction – to the left and to the right.

In conclusion, the chapter presented a new formulation of the lifting scheme. The formulation has the ability to retain certain intermediate results through the introduced auxiliary buffers. This was not possible in the original scheme. As a consequence of the presented formalism, the computation can be modified in such a way to meet different requirements on various platforms.

Chapter 4

Multi-Dimensional Cores

The presented core approach can be naturally extended to multiple dimensions. The key ideas of this section were presented in [V, VII, I].

This chapter is particularly focused on two-dimensional cores. However, the same principles also apply to more dimensions. Several benefits of the implementation arise by extending the core into two dimensions. Thanks to the linear nature of DWT, the horizontal and vertical steps can be interleaved or even merged. Merging of the final coefficient scaling is a useful involvement of this property. The extension into two dimensions follows.

To simplify the relations, the inequality $(0, 0) \leq (m_j, n_j) < (M_j, N_j)$ holds for all $0 \leq j \leq J$. The 2-D transform decomposes the input raster

$$\left(a^0_{m_0,n_0}\right) \tag{4.1}$$

of size $M_0 \times N_0$ pixels into $J > 0$ scales giving rise to the temporary subbands

$$\left(a^j_{m_j,n_j}\right), \tag{4.2}$$

the resulting wavelet subbands

$$\left(h^j_{m_j,n_j}\right), \left(v^j_{m_j,n_j}\right), \left(d^j_{m_j,n_j}\right), \tag{4.3}$$

at scales $0 < j < J$, and the residual signal

$$\left(a^J_{m_J,n_J}\right) \tag{4.4}$$

at the topmost scale J. Such a decomposition is performed using the 2×2 core with a lag F samples in both directions. This idea was also proposed in [VII]. For each scale $0 \leq j < J$, the core requires an access to two auxiliary buffers

$$\left({}^M B^j_{m_j}\right)_{0 \leq m_j < M_j}, \left({}^N B^j_{n_j}\right)_{0 \leq n_j < N_j}. \tag{4.5}$$

These buffers hold intermediate results of the underlying lifting scheme. The size of the buffers can be expressed as $M \times \kappa$ (horizontal buffer) and $N \times \kappa$ coefficients (vertical buffer), where κ is the number of values that have to be passed between adjacent 1-D cores. Taken together, the 2×2 core needs to access $2 \times \kappa$ values in the horizontal buffer and $2 \times \kappa$ values in the vertical buffer.

Similarly to the 1-D case, the 2-D core consumes a 2×2 fragment of the input signal and immediately produces a four-tuple of coefficients (a, h, v, d). The produced coefficients have a delay of F samples in the vertical as well as the horizontal direction with respect to the input coordinate system. To simplify relations, two functions will be introduced once again

$$\Theta(m,n) = (m+F, n+F), \text{ and } \Omega(m,n) = (\lceil m/2 \rceil, \lceil n/2 \rceil). \tag{4.6}$$

The function $\Theta(m,n)$ maps core output coordinates onto core input coordinates with a lag F. The function $\Omega(m,n)$ maps the coordinates at the scale j onto coordinates at the scale $j+1$. Note that $\Omega(m,n)$ can be defined in many ways. However, this particular example fits into the JPEG 2000 coordinate system. The 2×2 core transforms the fragment $\mathrm{I}^j_{m,n}$ of the input signal onto the fragment $\mathrm{O}^j_{m,n}$ of the output signal

$$\mathrm{I}^j_{m,n} = \begin{pmatrix} a^j_{\Theta(m,n)} & a^j_{\Theta(m+1,n)} & a^j_{\Theta(m,n+1)} & a^j_{\Theta(m+1,n+1)} \end{pmatrix}^T, \tag{4.7}$$

$$\mathrm{O}^j_{m,n} = \begin{pmatrix} a^{j+1}_{\Omega(m,n)} & h^{j+1}_{\Omega(m+1,n)} & v^{j+1}_{\Omega(m,n+1)} & d^{j+1}_{\Omega(m+1,n+1)} \end{pmatrix}^T, \tag{4.8}$$

while updating the two auxiliary buffers. Finally, operations performed inside the core can be described using a matrix C as a relationship

$$\boldsymbol{y} = C\,\boldsymbol{x} \tag{4.9}$$

from the input vector

$$\boldsymbol{x} = \mathrm{I}^j_{m,n} \parallel {}^M\mathrm{B}^j_m \parallel {}^M\mathrm{B}^j_{m+1} \parallel {}^N\mathrm{B}^j_n \parallel {}^N\mathrm{B}^j_{n+1} \tag{4.10}$$

onto the output vector

$$\boldsymbol{y} = \mathrm{O}^j_{m,n} \parallel {}^M\mathrm{B}^j_m \parallel {}^M\mathrm{B}^j_{m+1} \parallel {}^N\mathrm{B}^j_n \parallel {}^N\mathrm{B}^j_{n+1}, \tag{4.11}$$

where $\parallel$ denotes the concatenation operator. Recall that the choice of the C matrix and the consequent arrangement and the size κ of elements in the buffers is the subject of this thesis.

Considering the SIMD extensions, the two-dimensional core can nicely exploit capabilities of modern GPPs. Moreover, as the 2-D data occupy one order of magnitude more memory compared to the 1-D signals, the processing can be divided across multiple independent processing units. Combined with the single-loop approach [16], the cache hierarchy can also be properly utilized. The influence of all these possibilities was investigated in [VII].

So far, the main ability of the 2-D extension remains undisclosed. The single loop over the data does not have a strictly fixed order. Quite the contrary, many scan orders are now possible. Note that the original single-loop approach from [34] does not have this ability.

The above-described degree of freedom allows to adapt the processing to specific needs of the application. For instance, it turned out that the 2-D core approach can be adapted to JPEG 2000 coding units (so-called codeblocks) in [I]. When associated with the capabilities explained in the previous paragraph, these codeblocks can be generated in parallel. This experiment is further evaluated below.

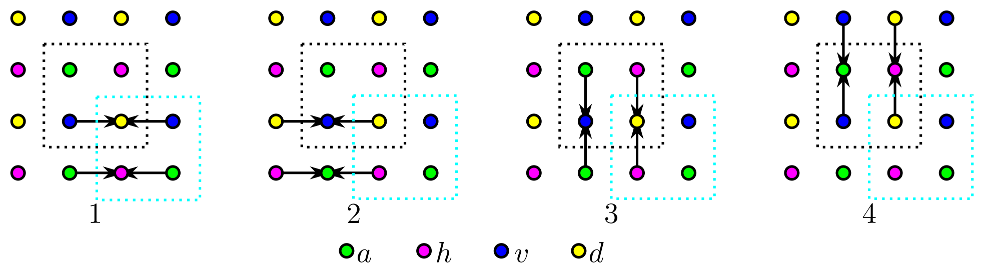


Figure 4.1: The separable implementation of CDF 5/3 core with 4 stages. The output is highlighted in dark, the input in bright. The arrows correspond to the multiply operations which are accumulated into the bullets.

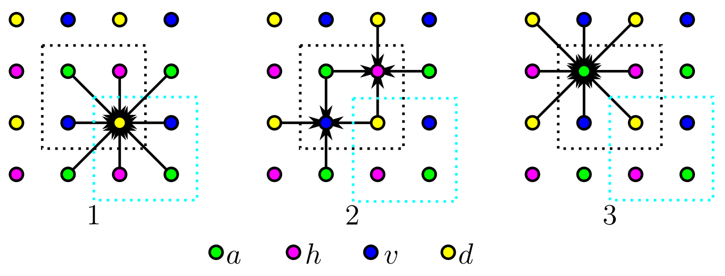


Figure 4.2: The non-separable 3-stage implementation of CDF 5/3 core. The input (in bright) and the output (in dark) of the core are highlighted.

4.1 2-D Core Reorganization

For purposes of illustration, the following text is focused on two-dimensional CDF 5/3 transform.

Considering the baseline separable extension into two dimensions resulting into 2×2 core, the matrix C in the relationship $\boldsymbol{y} = C\boldsymbol{x}$ can be factored into

$$\boldsymbol{y} = {}^{N}S_{\beta}\ {}^{N}T_{\alpha}\ {}^{M}S_{\beta}\ {}^{M}T_{\alpha}\,\boldsymbol{x}, \tag{4.12}$$

where the M superscripts refer to the horizontal direction, whereas N to the vertical one. Taken together, ${}^{M}T_{\alpha}$ performs two horizontal predicts, ${}^{M}S_{\beta}$ two horizontal updates, etc. The order of these steps (or stages) is not strictly fixed but also not completely unconstrained. The implementation has the latency of four lifting steps, plus scaling. The scheme is graphically illustrated in Figure 4.1. In total, 16 non-trivial operations (four in each stage) are needed to calculate this core (the scaling is omitted).

In [35], the authors derived a non-separable 2-D lifting scheme for CDF 5/3 DWT. One can easily identify a suitable core in their construction. The same core can be obtained by proper reorganization of operations inside the separable 2×2 core. The result is shown in Figure 4.2. Thanks to the parallel processing of v and h samples, this implementation has a total latency of 3 steps.

The more detailed description of this non-separable core follows. As initial step, a 2×2 fragment of the input signal is notionally split into the input quadruple $(a, v, h, d)^{(0)}_{m,n}$. Then, the prediction of the detailed coefficient is performed. This is followed by parallel prediction of

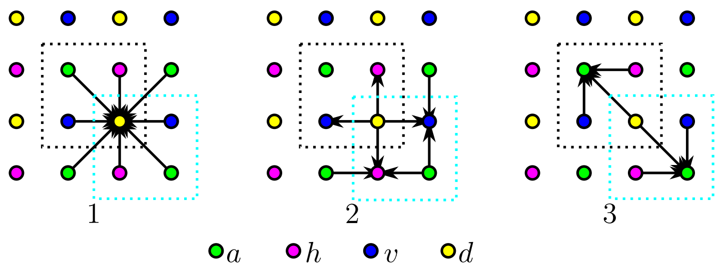


Figure 4.3: Proposed non-separable lifting core of CDF 5/3 with three stages. The input coefficients of active core are in bright box. The output ones are in dark one.

the horizontal and vertical coefficients. In third step, the approximation coefficient is updated. Finally, the four coefficients corresponding to four subbands can be scaled. Ten coefficients need to be forwarded through the auxiliary buffers. However, it is not necessary to construct such a huge buffer. Instead, the buffers from a separable implementation can be used complemented by small 2×2 buffer for the exchange of intermediate results in 2-D. Therefore, practical implementations with horizontal image scanning order would require two rows of coefficients to be buffered. The core has the lag F of one sample in each direction.

Using the matrix notation, the core is described as

$$\boldsymbol{y} = A_\beta \, T_{\alpha,\beta} \, D_\alpha \, \boldsymbol{x}, \tag{4.13}$$

where D_α operator computes the d coefficient, $T_{\alpha,\beta}$ computes h and v, and A_β finally computes a coefficient. It is easy to identify buffers forming borders between the consequent cores. Excluding diagonals, the matrices $A_\beta, T_{\alpha,\beta}, D_\alpha$ have 24 non-zero entries in total implying 24 non-trivial MAC operations.

Using the core approach presented in this thesis, it is possible to go further. For now, ignore the separable core comprising the reorganized cores from the previous chapter. The total number of arithmetic operations in the non-separable scheme [35] can be reduced. The key idea here is not to calculate the wavelet coefficients at once. Instead, the calculation of these coefficients is subdivided into separate parts. The sum of these parts gives the desired result.

I will now explain this approach in detail. The starting point of the solution is the non-separable lifting scheme of CDF 5/3 transform as described in [35]. The operations inside the core were further appropriately reorganized. After some rewriting of expressions, the following scheme has appeared. First, the scheme requires only 8 coefficients to be passed between core iterations. This means that this new scheme has the same memory demands as in the original separable case. Second, the number of the non-trivial arithmetic operations was reduced from 24 to 22 compared to the original non-separable scheme. The trick is that the h, v and a coefficients are not calculated at once. Instead, these are calculated separately, split in two parts each.

For a better understanding, the new scheme is graphically illustrated in Figure 4.3. It should be emphasized that this newly formed scheme cannot be derived using instruments in [35]. This is caused by the fact that the authors of [35] do not specify a sequence of the operations.

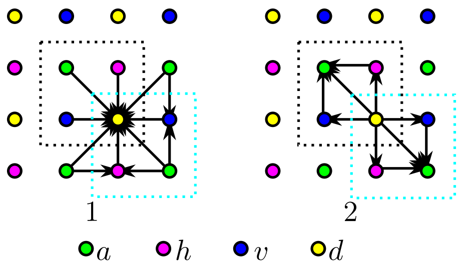


Figure 4.4: Proposed non-separable lifting core of CDF 5/3 with two stages. The input coefficients of active core are in bright box. The output ones are in dark one.

core	latency	buffer	operations	max. operands in step
separable	4	8	16	$2 + 1$
non-separable [35]	3	10	24	$8 + 1$
non-separable new	3	8	22	$8 + 1$
non-separable new	2	8	22	$8 + 1$

Table 4.1: Comparison of the 2-D single-loop cores. The operands are given in format non-trivial plus trivial. The scaling is omitted.

Using the matrix notation, the proposed transform core can be described as

$$\boldsymbol{y} = A_\beta \, T_{\alpha,\beta} \, D_\alpha \, \boldsymbol{x}. \tag{4.14}$$

The implementation is always a trade-off between latency and number of operations. In next example, another non-separable implementation with latency of 2 steps is shown. This implementation has the same buffer requirements. The achieved number of operations is 22.

In matrix notation, the transform can be written as the product

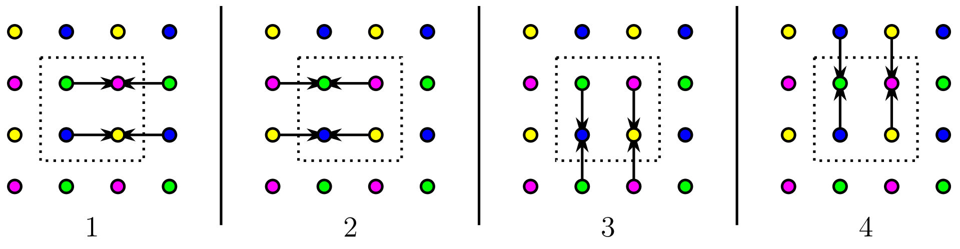
$$\boldsymbol{y} = A_\beta \, D_\alpha \, \boldsymbol{x}. \tag{4.15}$$

The steps are graphically illustrated in Figure 4.4.

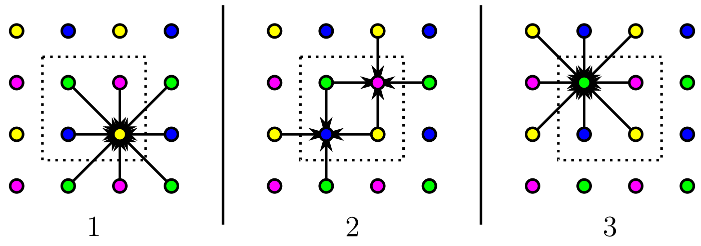
Table 4.1 provides summarizing comparison of the discussed 2-D single-loop cores. The most complicated calculation from all the steps is indicated in the last column. This number is given in the format of the non-trivial operations plus the trivial operations. As before, the scalings were omitted. When the stages of the core are pipelined (run in parallel), the clock latency of the core is directly subordinated by the maximum number of operands. The table further indicates the number of stages (steps), the number of coefficients accessed in the auxiliary buffers, and the total number of non-trivial operations.

4.2 Parallel 2-D Cores

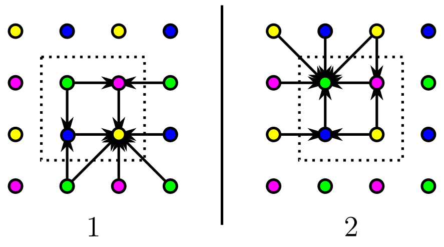
So far, only the single-loop two-dimensional cores were discussed. Considering the parallel environment, the cores can be modified in order to run in parallel. In such a case, the cores



(a) Sweldens1995



(b) Iwahashi2007



(c) proposed

●a ●h ●v ●d

Figure 4.5: 2-D data-flow graphs of the parallel cores. The order of the lifting steps is determined by the bottom numbers. The vertical lines indicate the necessary memory barriers.

have to exchange the intermediate results directly, without buffers. This modification introduces the need for synchronization using the memory barrier. Usually, these barriers form the major bottleneck of the overall computation. Taken together, it is desirable to minimize the number of memory barriers (along with another resources). The cores discussed in this section were proposed in [X, IX]. This section is still focused on CDF 5/3 transform. The generalization is straightforward.

Recently, Iwahashi *et al.* [36, 37, 35] presented the non-separable lifting scheme employing genuine spatial filtering steps. In this scheme, it is no longer possible to distinguish the vertical and horizontal filtering. The transform can be described as linear transformations of the vectors

$$\begin{aligned} \boldsymbol{x} &= \left[\begin{array}{cccc} a & h & v & d \end{array} \right]^{\mathrm{T}}, \\ \boldsymbol{y} &= \left[\begin{array}{cccc} a & h & v & d \end{array} \right]^{\mathrm{T}}. \end{aligned} \tag{4.16}$$

Formally, these transformations can be compressed into the matrix $C_{\alpha,\beta}$ in

$$\boldsymbol{y} = C_{\alpha,\beta}\,\boldsymbol{x} = A_\beta\,T_{\alpha,\beta}\,D_\alpha\,\boldsymbol{x}. \tag{4.17}$$

The scheme is graphically illustrated in Figure 4.5b (referred to as *Iwahashi2007*). Similarly to the original scheme, a memory barrier must be inserted between each two steps. As the result, such a scheme consists of 24 non-trivial arithmetic operations in three lifting steps separated by two explicit memory barriers. The most complex operation is calculated over 9 operands which leads to a performance issue. This is caused by the number of operands being proportional to the data path with the maximum delay. For the sake of comparison, the baseline separable scheme is illustrated in Figure 4.5a (referred to as *Sweldens1995*). Note that the scheme for CDF 9/7 comprises two such connected transforms.

Motivated by the work of Iwahashi *et al.* [35], the elementary lifting filters were reorganized in order to obtain a highly parallelizable scheme. The main purpose of this modification is to minimize the number of memory barriers that slow down the calculation. As a result, several non-separable two-dimensional FIR filters arise.

The scheme consists of two parts between which a memory barrier is placed.

Finally, the new scheme is composed of four operators referred to as S^1 to S^4. Between the second S^2 and third S^3 operator, the memory barrier must be inserted in order to properly exchange intermediate results. Thus, S^1 and S^2 form the first lifting step and S^3 and S^4 form the second one. Note that it is also possible to rewrite the scheme using six operators instead of four. It would be also possible to rewrite the scheme with just two operands, however, it is not possible to capture a retention of intermediate results in such a case. Additionally, the scheme requires the induction of two auxiliary variables (the intermediate results) per each quadruple of coefficients a, h, v, and d. These auxiliary variables are denoted as $h^{(1)}, v^{(1)}$. It does not matter of their initial as well as final values.

The scheme

$$\boldsymbol{y} = S^4_\beta \, S^3_\beta \, S^2_\alpha \, S^1_\alpha \, \boldsymbol{x} \tag{4.18}$$

describes the relation between input $\boldsymbol{x}$ and output $\boldsymbol{y}$ vectors. Note that in practical realizations, each single computing unit (e.g., thread) can be responsible of one such a vector. The vectors are given by the following equations.

$$\begin{aligned} \boldsymbol{x} &= \begin{bmatrix} a & h & v & d & h^{(1)} & v^{(1)} \end{bmatrix}^\mathrm{T} \\ \boldsymbol{y} &= \begin{bmatrix} a & h & v & d & h^{(1)} & v^{(1)} \end{bmatrix}^\mathrm{T} \end{aligned} \tag{4.19}$$

Regarding this notation, the individual steps are defined as follows. In addition, the operations are graphically illustrated in Figure 4.5c (referred to as *proposed*). Note that operators S^1 and S^2 are represented by the first lifting step and operators S^2 and S^3 by the second one.

Compared with [35], the total number of arithmetic operations has been reduced from 24 to 20 for the CDF 5/3 wavelet. The calculation of CDF 9/7 transform consists of two such connected transforms (the first with α, β, the second with γ, δ) between them another barrier is placed. In total, such a calculation contains three explicit memory barriers.

A quantitative comparison for the CDF 5/3 wavelet of all the cores discussed is provided in Table 4.2. For the CDF 9/7 wavelet, the number of lifting steps and thus the number of operations must be doubled. In general, the cores can be used for any lifting factorization with two-tap filters.

scheme	steps	operations	max. operands	memory cells
Sweldens1995	4	16	3	4
Iwahashi2007	3	24	9	4
proposed	**2**	20	9	6

Table 4.2: Parameters of the 2-D parallel cores for CDF 5/3 wavelet. The columns describe: number of lifting steps, number of arithmetic operations, maximum number of operands per the lifting step result (the complexity of steps), number of memory cells per coefficient quadruple (inclusive).

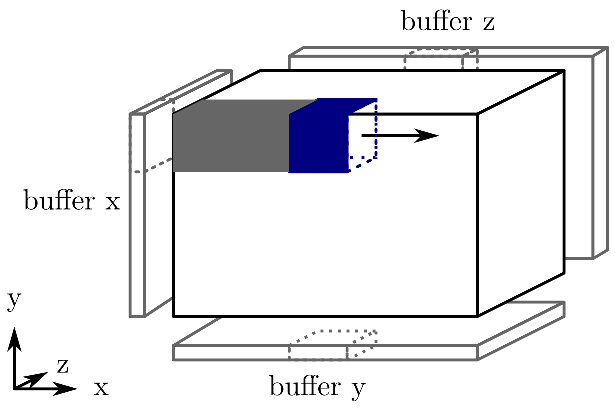


Figure 4.6: Complete processing by the 3-D single-loop core. The auxiliary buffer for each dimension is shown on the sides. The x, y, z notation is used instead of m, n.

The original *Sweldens1995* scheme provides the best choice in terms of arithmetic operands as well as their complexity. However, it requires three explicit synchronization points (memory barriers) for the CDF 5/3 wavelet. This can be an issue for parallel processing. The recently proposed *Iwahashi2007* scheme uses the highest number of operations of all schemes. On the other hand, it requires only two synchronizations for the CDF 5/3 wavelet and does not need any additional memory. In numbers, this scheme reduces the number of lifting steps to 75 %. Finally, the *proposed* scheme provides a trade-off in the number of operations. Moreover, for the CDF 5/3 wavelet, only one barrier is needed for its realization. In comparison to the original scheme, this scheme reduces the number of lifting steps to 50 % only.

4.3 Extension to Multiple Dimensions

Similarly to the previous 2-D extension, the 2^3 cores transforming the 3-D data in the single loop were proposed in [VI]. Access to three 2-D auxiliary buffers is required during the computation. As in the previous cases, the implementation can further be extended to allow SIMD-optimizations.

In the most generic variant, each 2-D auxiliary buffer has the same size as the corresponding volume side. The depth of each auxiliary buffer is κ coefficients. See Figure 4.6. This memory consumption can be reduced using an appropriate processing order. When using the horizontal (raster scan) order, it is not necessary to allocate the full side size buffers. It is sufficient to allocate only the following sizes. Allocate one full side size buffer for the first dimension. Allocate one edge size buffer for the second dimension. Finally, allocate one point size buffer for the third dimension. For instance, considering the 2^3 core, it is sufficient to allocate buffers of total size $\kappa\,(NM + 2N + 4)$ elements, where N, M are sizes of the volume in first two dimensions.

Considering the separable implementation, the core consists of three blocks performing calculations corresponding to the three dimensions. Each block updates the necessary intermediate results in the corresponding auxiliary buffer. This is followed by scaling of the output coefficients.

The similar extension to an arbitrary number of dimensions can be constructed. In the general case, the core has a size of 2^Π samples, where Π is the number of dimensions. Such a core needs to access into Π buffers. Each of them is indexed by $\Pi - 1$ coordinates. Following the previous discussion, it is not necessary to allocate the complete $(\Pi-1)$-dimensional buffers. Instead, only a small fraction of them is actually needed. This fraction is proportional to a single $(\Pi - 1)$-dimensional hypercube.

Formally, for each scale $0 \leq j < J$, the 2^Π core requires an access to Π auxiliary buffers

$$\left({}^{\pi}B^{j}_{{}^{\pi}\lambda_j}\right) \tag{4.20}$$

for $0 \leq \pi < \Pi$ and $0 \leq {}^{\pi}\lambda_j < {}^{\pi}\Lambda_j$, where ${}^{\pi}\Lambda_j$ is the size of the data in π-direction at the scale j. For simplicity, the following description is limited on a particular scale j and the scripts j are thus dropped. Moreover, let λ to be the vector of coordinates $({}^{\pi}\lambda)_{0\leq\pi<\Pi}$, $\mathbf{I}_\lambda$ to be the vector of 2^Π input coefficients, $\mathbf{O}_\lambda$ to be the vector of 2^Π output coefficients, and $\mathbf{B}_\lambda$ to be the vector $({}^{\pi}B_{{}^{\pi}l})_{0\leq\pi<\Pi,{}^{\pi}\lambda\leq{}^{\pi}l<{}^{\pi}\lambda+2}$ of corresponding buffer fragments. Then the core is described as the mapping

$$\boldsymbol{y} = C\,\boldsymbol{x} \tag{4.21}$$

from the input vector

$$\boldsymbol{x} = \mathbf{I}_\lambda \parallel \mathbf{B}_\lambda \tag{4.22}$$

onto the output vector

$$\boldsymbol{y} = \mathbf{O}_\lambda \parallel \mathbf{B}_\lambda. \tag{4.23}$$

Again, the choice of matrix C and the arrangement of the buffers is not fixed and can be subject to optimization with respect to some criterion.

Chapter 5

Evaluation

This chapter provides brief performance evaluation of the presented core approach.

5.1 Image Processing

The following discussing shows the effect of coarse-grained parallelization of the above discussed approaches. The naive approach that uses horizontal and vertical 1-D transform was parallelized using multiple threads. The same was done with vectorized core single-loop approaches (4×4 vertical and 6×2 diagonal, both with merged scaling). In the latter case, the image was split into several rectangular regions assigned to different threads.

The parallelization of the single-loop core approach is not as straightforward as the parallelization of the naive approach. To produce correct results, each thread must process a segment (several rows) of input image before its assigned area. In this segment, no coefficients are written to output. Therefore, this phase can be seen as a prolog. Without the prolog, the threads would produce independent transforms, each with zero border extension.

Finally, a summarizing comparison of parallelizations is shown in Table 5.1. The measurements were performed on a 58-megapixel image. The single-threaded algorithm is used as a reference one. Using the core approach, both the vectorizations scale almost linearly with the number of threads.

5.2 JPEG 2000

Efficient realization of JPEG 2000 transform was outlined by D. Taubman in [38]. However, the author did not provide much implementation details and he did not consider any friendliness to the CPU cache nor the SIMD set. Nevertheless, he expressed the memory requirements for multi-scale DWT as $(4 + I)M$ samples. As the transform coefficients have to be arranged into codeblocks, the total memory requirements for JPEG 2000 codec are $(4 + I + 3 \times 2^{c_n})M$ samples, where 2^{c_n} is the codeblock height. The initial 4 term corresponds to 2 lines per one decomposition scale. This imposes that his implementation generates all codeblocks at the same time, not one after another. According to the description in [38], their implementation does not process the data in the single loop. However, for a moment, I assume that their implementation would do so. Still, this strategy is fundamentally different from the architecture described in this section which generates individual blocks sequentially while all the time reusing the same

Intel Core2 Quad

threads	1		2		4	
algorithm	time	speedup	time	speedup	time	speedup
naive vertical	21.5	1.0	15.8	1.4	9.8	2.2
naive diagonal	19.7	1.1	14.4	1.5	8.9	2.5
core vertical	**4.3**	**5.1**	**2.3**	**9.5**	**1.5**	**14.6**
core diagonal	5.9	3.7	3.1	7.1	2.0	11.0

AMD Opteron

threads	1		2		4	
algorithm	time	speedup	time	speedup	time	speedup
naive vertical	46.9	1.0	24.0	2.0	12.1	3.9
naive diagonal	46.6	1.0	23.6	2.0	11.8	4.0
core vertical	**4.3**	**11.0**	**2.3**	**20.5**	**1.2**	**39.3**
core diagonal	7.1	6.6	3.7	12.7	1.9	24.8

Table 5.1: Performance evaluation using threads. The time is given in nanoseconds per pixel. The speedups are shown compared to the non-parallelized naive vertical algorithm.

memory area for output coefficients. The above-described line-based processing does not fit the JPEG 2000 codeblocks, does not allow the parallel codeblock processing, and does not allow to reuse the memory for h, v, and d subbands. The motivation behind my work is to overcome these issues.

The cores in the previous section consume a fragment of the input signal and immediately produces a four-tuple of coefficients). The produced coefficients have a lag of $F = 4$ samples in the vertical as well as the horizontal direction with respect to the input coordinate system. In the JPEG 2000 coordinate system, such a core consumes the fragment of the input starting on odd (m, n) coordinates. Unfortunately, the original core does not fit the system of codeblocks in JPEG 2000. For explanation, every codeblock starts on even (m, n) coordinates (which corresponds to the a coefficient). On the other hand, the core with the lag of $F = 4$ samples produces a fragment of coefficients starting on odd (m, n) coordinates (which corresponds to the d coefficient). In order to solve this incompatibility, the original core has been modified in a way that the output coefficients are produced with a lag of $F = 3$ samples. Note that any shorter lag is not possible due to the nature of CDF 9/7 lifting scheme.

As the next step, the processing of the codeblocks was encapsulated into monolithic units. These units are evaluated in horizontal "strips" due to the assumed line-oriented processing order. Inside the codeblock unit, the 2×2 core can be used. Moreover, the unit requires access to two auxiliary buffers (one for each direction). The size of the buffer can be expressed as $2^{c_m} \times \kappa$ (for the horizontal buffer) and $2^{c_n} \times \kappa$ (for the vertical buffer), where $\kappa = 4$. As the strip-based processing with a granularity of the codeblock size is used, the vertical buffer is straightly passed to the subsequent codeblock processing unit. The horizontal buffer will be used by a strip of codeblocks lying below. At the beginning of the strip, the vertical buffer contains arbitrary values. The first codeblock unit initializes this buffer and passes it to the subsequent unit in horizontal direction. The transform of this subsequent unit is started not earlier than the EBCOT [39] on the current unit has been finished.

The above-described procedure is in effect friendly to the cache hierarchy. As shown, the processing engine uses several memory regions of a different purpose. (1) The resulting codeblock subbands occupy several KiB of memory likely settled in the top-level cache. (2) The vertical buffer occupies several hundreds of bytes. (3) The fragments of horizontal buffers occupy the same size as the total size of vertical buffer. However, they are used only for short time and then can be evicted from all levels of the cache hierarchy. (4) The input strip can be simply streamed into the same memory region which may be in part mirrored in the cache. (5) The temporary a subbands can be partially mirrored as well. For a smaller resolution, there is a good chance that the entire working set can fit into the cache hierarchy.

The entire process described above can be efficiently parallelized. The key idea is to split the strip processing into several independent regions. Thus, a single thread is responsible for several adjacent codeblocks. Each thread holds its private copy of the vertical buffer and the memory region for the resulting subbands (h, v, d). Therefore, several EBCOT coders can work in parallel. Moreover, the threads are completely synchronization-free (they does not need to exchange any data). In the test implementation, the wavelet decomposition as well as Tier-1 encoding was parallelized. On parallel architectures, it is also possible to encode every single codeblock of the strip in parallel. However, the parallelization of the implementation is constrained by the number of computing units. In the following text, the performance of the test implementation and compare it to the competitive solutions was evaluated.

The input image is consumed gradually using strips with height of 2×2^{c_m} lines. No more input data are required to be placed in physical memory at the same moment. For the output subbands, memory for only $4 \times 2^{c_m+c_n}$ coefficients is allocated (considering all four subbands). This memory is reused by all codeblocks in the transform (or a processing thread). Additionally, two auxiliary buffers of size $M_j \times \kappa$ and $N_j \times \kappa$ coefficients have to be allocated for each decomposition level j. Note that $M_{j+1} = \lceil M_j/2 \rceil_{c_m}$ and $N_{j+1} = \lceil N_j/2 \rceil_{c_n}$, where $\lceil . \rceil_c$ denotes ceiling to the next multiple of 2^c; initially $M_0 = M$ and $N_0 = N$. For each auxiliary a band (excluding the input and the final one), the window of physical memory can be maintained and progressively mapped onto the right place in the virtual memory. The size of such a window is roughly $3 \times 2^{c_n} \times M_{j+1}$. Note that 3 instead of 2 codeblock strips are needed due to the periodic symmetric extension on the image borders, additionally, a lag of $F = 3$ lines from the input to the output of the core. Roughly speaking, the codeblocks of the subsequent scales do not exactly fit each other. Taken together, the presented solution requires $(I + 3 \times 2^{c_n})M$ samples populated into the physical memory.

The presented solution was compared to C/C++ libraries listed on the official JPEG committee web pages – OpenJPEG, FFmpeg, and JasPer. The OpenJPEG, FFmpeg, and JasPer

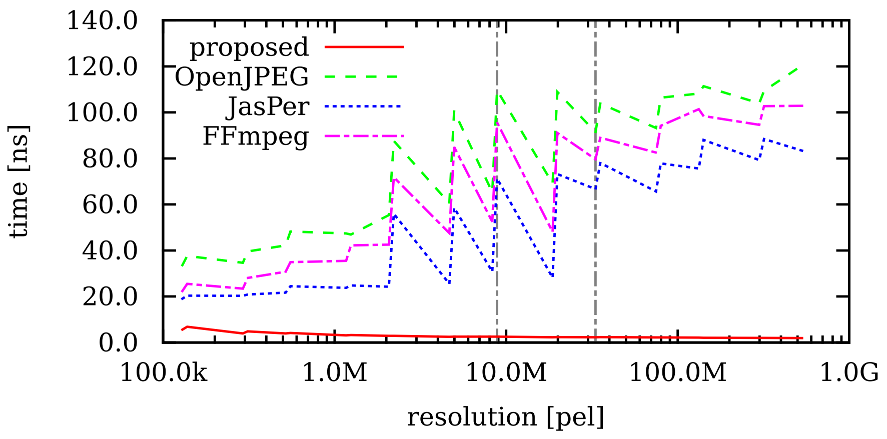


Figure 5.1: Performance comparison of JPEG 2000 libraries. Time per pixel for the transform stage only. DCI 4K and 8K UHD resolutions indicated by the vertical lines.

libraries are distributed under the terms of open-source licences. Thus, these could be analysed through their source code in detail. Note that OpenJPEG and JasPer are approved as reference JPEG 2000 implementations.

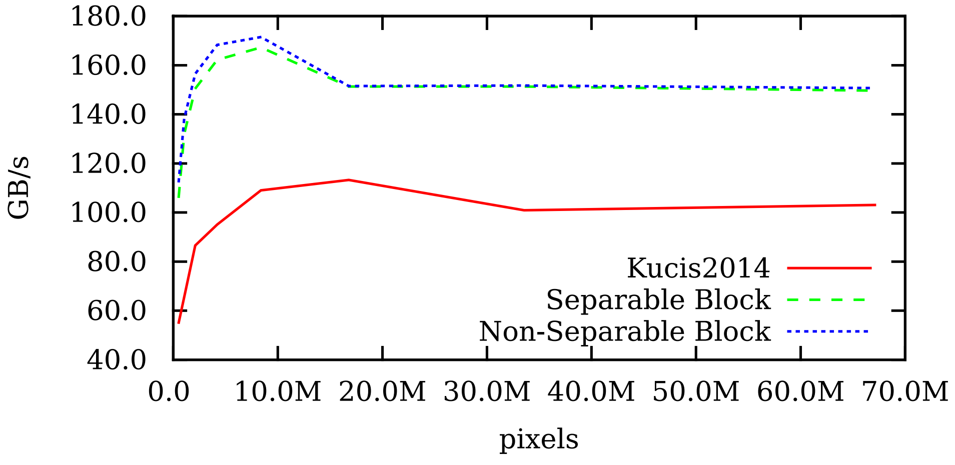
The naive approach refers to processing the entire image at once while keeping the horizontal and vertical passes as well as the transform levels (scales) separated. Furthermore, inside the horizontal and vertical passes, the lifting steps are processed sequentially (the horizontal vectorization). As a consequence, samples of the tile are visited many times while being over and over again evicted from the cache. Unlike the naive approach, the other two approaches use sophisticated technique where the processing of consecutive scales is interleaved. Moreover, in case of the presented strip-based processing, the horizontal and vertical passes were fused into the single loop. Regarding the strip-based processing, the input is consumed using strips, one by one. The subsequent scales are recursively processed as soon as enough data is available.

The transform stage was extracted from the libraries described above in order to get accurate results. This stage was then subjected to measurement. The results are shown in Figure 5.1. As observed also in [16], the single-loop processing has stable performance regardless the input resolution. The proposed implementation was measured using four threads and SSE extensions. However, the SSE or AVX extensions boost the performance by at most 5 %.

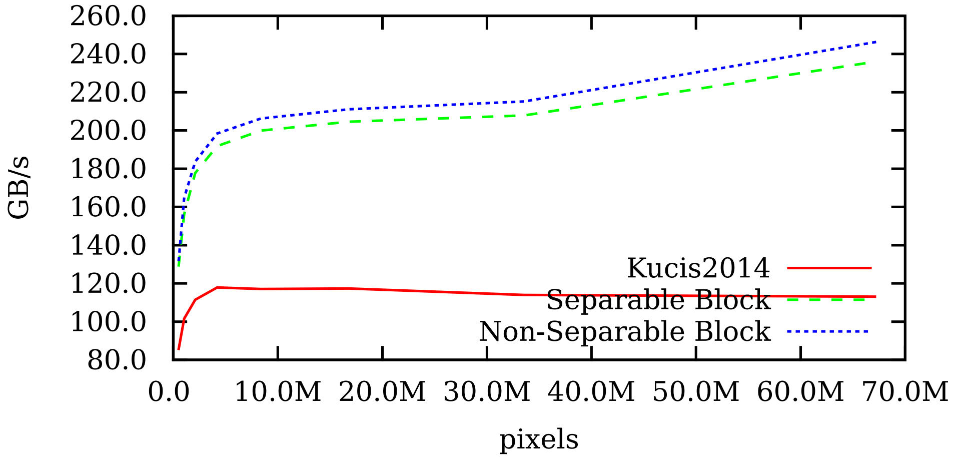
5.3 Parallel Processing on GPU

As described in Section Parallel 2-D Cores, two parallel lifting scheme schedules for GPGPUs were designed and implemented. These schedules are, in fact, based on the separable and non-separable parallel cores presented in the previous chapter. The implementation is based on the OpenCL framework. All of the algorithms are evaluated on CDF 9/7 transform using the most recent GPUs of two biggest vendors, namely, AMD R9 290X and NVIDIA TitanX graphics cards.

The achieved memory throughput performance is shown in Figure 5.2. As it can be seen,



(a) AMD R9 290X



(b) NVIDIA TitanX

Figure 5.2: Throughput performance of parallel methods. *Kucis2014* denotes the reference method presented in [VIII].

both the core methods overcome the reference state-of-the-art method. Moreover, the proposed non-separable core performs slightly better compared to the separable one. As it can be expected, this behavior corresponds to the reduced number of the memory barriers.

5.4 FPGA Devices

In the last substantial experiment, the hardware implementation is evaluated. The implementation is focused on JPEG 2000 system. Particularly, the lossless CDF 5/3 transform was implemented in FPGA.

The wavelet engine was experimentally synthesized in a Xilinx Zynq XC7Z045 FPGA and evaluated on the Xilinx ZC706 board (with DDR3 at 1066 MHz). The engine was synthesized for several image resolutions (as seen in Table 5.2) that merely differ in the BRAM size only

	small tile	Full HD	4K UHD
resolution	512×512	1920×1080	3840×2160
theoretical framerate	3698	477	120
framerate with DRAM	2670	338	84

Table 5.2: The framerates achieved in FPGA implementation. Given for various image resolutions.

architecture	device	BRAM [bits]	clocks/pel	time [ms]
Dillen [40]	VirtexE1000-8	50K	0.50	1.20
Descampe [41]	Virtex-II XC2V6000	N/A	0.60	1.75
Seo [42]	Altera Stratix	128K	2.64	6.02
Zhang [43]	Virtex-II Pro XC2VP30	6×18K	0.50	0.97
the cores	Zynq XC7Z045	1×36K	**0.26**	**0.27**

Table 5.3: Comparison of various FPGA implementations. Tiles of size 512×512. The processing time and clocks per pixel were projected to the uniform image size. The best results are in bold.

to allow comparison with other papers and also to show that the core is able to process Full HD video faster than in real-time. As it could be seen, the core is ready to process image resolution of up to 4K UHD with the outlook to even higher resolutions without need of any fundamental changes. The input expects streamed video frames in the predefined resolution, the output stream is generating interlaced coefficients of wavelet transform that can be easily split into four separate data streams for further multi-scale decomposition. I would like especially to highlight the ability to process the video stream without the need to use external memory for intermediate results. The design includes mirroring on the image edges which is not performed by wavelet core itself, but by the engine, which encapsulates the core. The engine itself then represents an independent block, which can be used in more complex system or which can be easily duplicated and chained to perform more levels of wavelet transform of one image.

The overall comparison with the selected architectures is shown in Table 5.3.

5.5 Discussion

Several experiments evaluating the performance of different methods were conducted on general-purpose processors. The 1-D transform is considered first. Some findings are evident from these experiments. The most important effect occurs as soon as the working set exceed the cache size. The discussed effect causes that the single-loop methods are faster compared with the naive horizontal vectorization.

Considering the 2-D transform, the single-loop methods are faster as compared with the naive separable methods. The single-loop method can be built above the vertical or diagonal vectorization, or above their variants.

Considering 2-D transform, the core implementation of the single-loop approach was discussed in detail. Its performance is similar the the "hardwired" single-loop code.. However, the cores disclose several degrees of freedom. This advantage is especially useful when considering the multi-dimensional transform. Namely, variety of processing orders can be employed during the transform. This is true even in connection with the multi-scale decomposition as shown when integrating into JPEG 2000 encoder. Moreover, many possible uses of SIMD extension became available in the case of multi-dimensional core. Particularly, 4×4 vertically vectorized core is the best performing one on Intel x86 platform with SSE extensions. Moreover, the transform employing the cores allow for easy coarse-grained parallelization. As demonstrated in the JPEG 2000 encoding chain, even no synchronizations are required in between threads considering the horizontal adjacency of parallel blocks. The cores incorporated into JPEG 2000 compression chain have proved to be fundamentally faster than the widely used implementations.

The cores can also be internally reorganized in order to minimize some of the resources. This property was demonstrated on FPGA where the minimization of the core latency has a direct impact on the utilization of flip-flop circuits and look-up tables (LUT). Specifically, the reduced latency core consumes more LUTs and uses smaller amount of flip-flops.

The cores may also be advantageously used on massively-parallel architectures. This option was demonstrated using OpenCL framework and the most recent GPUs of two biggest vendors. Specifically, the transform employing the parallel non-separable core reducing the number of memory barriers proved to be the fastest way to transform the 2-D data.

Chapter 6

Conclusions

The thesis focuses on efficient methods for computing the discrete wavelet transform. The state-of-the-art methods suffer from several ailments. For example, the parallelization, exploitation of SIMD extensions and the cache hierarchy are not handled well. The treatment of signal boundaries is done in a complicated and inflexible way. Additionally, these methods do not address the problem of scheme reorganization in order to minimize some of the resources. The aim of the thesis has been to overcome these issues. This was accomplished with the formation of a compact streaming core which performs the transform in a single loop, possibly in multi-scale fashion. Using this core, transform fragments can be computed according to application requirements.

New features of the approach presented are indicated by numbers. The presented core can (1) efficiently exploit the capabilities of modern CPUs; especially the cache hierarchy, SIMD extensions, and parallel computing. Operations inside the core can be (2) reorganized in order to minimize some of the platform resources (e.g., the number of memory barriers, the number of steps). Since the core itself (3) treats the signal boundaries, no special prolog or epilog phases are needed. Moreover, the cores can be adapted to (4) massively-parallel environments.

The core can be described as a direct mapping from the input coefficients on the output ones while retaining and exploiting some auxiliary intermediate results. This mapping can be seen as a standalone streaming unit, implemented either in software or in hardware. Using the core, the transform fragments can be computed with several (5) new degrees of freedom (the processing order, the interleaving of the multi-scale decomposition). For example, a particular transform block at a particular scale can be obtained with minimal or no unnecessary calculations.

When searching for the best core, I have found that the core can optimize only one criterion at the expense of others. For instance, minimizing the number of arithmetic operations goes against the number of synchronization points (the memory barriers) and the number of scheme steps (the latency). Moreover, a universal core suitable for all cases and environments probably does not exist.

The future work I would like to do comprises the concatenation of the analysis and synthesis cores coupled with some useful algorithm. This can be done on the multi-scale basis. The algorithms can perform, for example, tone-mapping, denoising, compression, etc. Another area of activities can be the generalization to non-linear transforms.

Curriculum vitae

Personal Data

full name	David Bařina
born	December 7, 1984, Hodonín, Czech Republic
residency	Brno
marital status	married
email address	ibarina@fit.vutbr.cz
telephone number	+420 723850574

Education

2007 Bachelor Degree Programme,
Faculty of Information Technology, Brno University of Technology

– Bachelor's thesis: *Lossy Image Compression*

2009 Master Degree Programme,
Faculty of Information Technology, Brno University of Technology

– Master's thesis: *Videocodec – Videosequence Compression*

Experiences

- signal/image/video processing and compression
- C programming language
- Linux

Teaching

Courses, Laboratories, and Projects

- *Multimedia*, 6 lectures, FIT BUT, Brno.
- *Image Processing*, 1 lecture, FIT BUT, Brno.

Supporting Texts

- Barina, D.; Zemcik, P.: *Multimedia*, supporting text, 116 pages, FIT BUT, Brno, 2013.
- Zemcik, P.; Spanel, M.; Beran, V.; Barina, D.; Mlich, J.: *Image Processing*, supporting text, chap. 6 and 10, 92 pages, FIT BUT, Brno, 2011.

Selected Products

- Barina, D.; Zemcik, P.: *A Cross-platform Discrete Wavelet Transform Library (libdwt)*, software, 2010–2015.
- Barina, D.; Bromova, P.: *Software framework for a stellar classification*, software, 2013.
- Barina, D.; Hradis, M.; Reznicek, I.; Zemcik, P.: *Classifier creation framework for diverse classification tasks*, software, 2010.
- Barina, D.; Zahradka, J.; Zemcik, P.: *Framework for Wavelet Representation of Optical System Distortion*, software, 2014.
- Barina, D.; Zemcik, P.: *A small C library for modular arithmetic and primarily testing (libmodexp)*, software, 2014.
- Kula, M.; Barina, D.; Zemcik, P.: *OpenCL Implementations of Discrete Wavelet Transform*, software, 2015.
- Musil, P.; Musil, M.; Barina, D.; Zemcik, P.: *FPGA Cores for Discrete Wavelet Transform*, software, 2015.

Patents

- Seeman, M.; Zemcik, P.; Barina, D.: Method and an Apparatus for Fast Convolution of Signals with a One-Sided Exponential Function. US Patent Application, appl. no. 14/105129, Assignee Brno University of Technology, Filed 2013.

Selected Papers

[I] Barina, D.; Klima, O.; Zemcik, P.: Single-Loop Software Architecture for JPEG 2000. In *Data Compression Conference (DCC)*, 2016, submitted.

[II] Barina, D.; Musil, M.; Musil, P.; et al.: Single-Loop Approach to 2-D Wavelet Lifting with JPEG 2000 Compatibility. In *6th Workshop on Applications for Multi-Core Architectures (WAMCA)*, 2015, ISBN 978-3-598-21500-1.

[III] Barina, D.; Zemcik, P.: Minimum Memory Vectorisation of Wavelet Lifting. In *Advanced Concepts for Intelligent Vision Systems (ACIVS)*, *Lecture Notes in Computer Science*, vol. 8192, Springer, 2013, ISBN 978-3-319-02894-1, pp. 91–101, doi:10.1007/978-3-319-02895-8_9.

[IV] Barina, D.; Zemcik, P.: Wavelet Lifting on Application Specific Vector Processor. In *GraphiCon'2013*, GraphiCon Scientific Society, 2013, ISBN 978-5-8044-1402-4, pp. 83–86.

[V] Barina, D.; Zemcik, P.: Diagonal Vectorisation of 2-D Wavelet Lifting. In *IEEE International Conference on Image Processing (ICIP)*, Paris, France, 2014, pp. 2978–2982.

[VI] Barina, D.; Zemcik, P.: Real-Time 3-D Wavelet Lifting. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, 2015, ISBN 978-80-86943-65-7, pp. 15–23.

[VII] Barina, D.; Zemcik, P.: Vectorization and parallelization of 2-D wavelet lifting. *Journal of Real-Time Image Processing (JRTIP)*, in press, ISSN 1861-8200, doi:10.1007/s11554-015-0486-6.

[VIII] Kucis, M.; Barina, D.; Kula, M.; et al.: 2-D Discrete Wavelet Transform Using GPU. In *5th Workshop on Application for Multi-Core Architectures (WAMCA)*, IEEE Computer Society, 2014, ISBN 978-1-4799-7014-8, pp. 1–6.

[IX] Kula, M.; Barina, D.; Zemcik, P.: Block-based Approach to 2-D Wavelet Transform on GPUs. In *International Conference on Information Technology – New Generations (ITNG)*, 2016, accepted.

[X] Kula, M.; Barina, D.; Zemcik, P.: New Non-Separable Lifting Scheme for Images. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, submitted.

References

[1] I. Daubechies, *Ten Lectures on Wavelets*, ser. CBMS-NSF regional conference series in applied mathematics. Philadelphia, Pennsylvania: Society for Industrial and Applied Mathematics, 1992, vol. 61.

[2] ——, “Orthonormal bases of compactly supported wavelets,” *Communications on Pure and Applied Mathematics*, vol. 41, no. 7, pp. 909–996, 1988.

[3] A. Cohen, I. Daubechies, and J.-C. Feauveau, “Biorthogonal bases of compactly supported wavelets,” *Communications on Pure and Applied Mathematics*, vol. 45, no. 5, pp. 485–560, 1992.

[4] S. G. Mallat, “A theory for multiresolution signal decomposition: The wavelet representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[5] ——, “Multifrequency channel decompositions of images and wavelet models,” *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 12, pp. 2091–2110, Dec. 1989.

[6] W. Sweldens, “The lifting scheme: A new philosophy in biorthogonal wavelet constructions,” in *Wavelet Applications in Signal and Image Processing III*, A. F. Laine and M. Unser, Eds. Proc. SPIE 2569, 1995, pp. 68–79.

[7] ——, “The lifting scheme: A custom-design construction of biorthogonal wavelets,” *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 186–200, 1996.

[8] I. Daubechies and W. Sweldens, “Factoring wavelet transforms into lifting steps,” *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, pp. 247–269, 1998.

[9] S. G. Mallat, “Multiresolution approximations and wavelet orthonormal bases of $L_2(R)$,” *Transactions of the American Mathematical Society*, vol. 315, no. 1, pp. 69–87, 1989.

[10] ——, *A Wavelet Tour of Signal Processing: The Sparse Way. With contributions from Gabriel Peyre.*, 3rd ed. Academic Press, 2009.

[11] G. Strang and T. Nguyen, *Wavelets and Filter Banks.* Wellesley-Cambridge Press, 1996.

[12] R. E. Blahut, *Fast Algorithms for Signal Processing.* Cambridge University Press, 2010.

[13] U. Drepper, “What every programmer should know about memory,” Red Hat, Inc., Tech. Rep., Nov. 2007.

[14] C. Chrysafis and A. Ortega, “Minimum memory implementations of the lifting scheme,” in *Proceedings of SPIE, Wavelet Applications in Signal and Image Processing VIII*, ser. SPIE, vol. 4119, 2000, pp. 313–324.

[15] P. Meerwald, R. Norcen, and A. Uhl, “Cache issues with JPEG2000 wavelet lifting,” in *Visual Communications and Image Processing (VCIP)*, ser. SPIE, vol. 4671, 2002, pp. 626–634.

[16] R. Kutil, “A single-loop approach to SIMD parallelization of 2-D wavelet lifting,” in *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 2006, pp. 413–420.

[17] A. Shahbahrami, B. Juurlink, and S. Vassiliadis, "Improving the memory behavior of vertical filtering in the discrete wavelet transform," in *Proceedings of the 3rd conference on Computing frontiers (CF)*. ACM, 2006, pp. 253–260.

[18] ——, "Implementing the 2-D wavelet transform on SIMD-enhanced general-purpose processors," *IEEE Transactions on Multimedia*, vol. 10, no. 1, pp. 43–51, Jan. 2008.

[19] J. Tao and A. Shahbahrami, "Data locality optimization based on comprehensive knowledge of the cache miss reason: A case study with DWT," in *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, Sep. 2008, pp. 304–311.

[20] D. Chaver, C. Tenllado, L. Pinuel, M. Prieto, and F. Tirado, "Vectorization of the 2D wavelet lifting transform using SIMD extensions," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2003, p. 8.

[21] S. Chatterjee and C. D. Brooks, "Cache-efficient wavelet lifting in JPEG 2000," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, vol. 1, 2002, pp. 797–800.

[22] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi, "Nonlinear array layouts for hierarchical memory systems," in *ICS '99: Proceedings of the 13th international conference on Supercomputing*. New York, NY, USA: ACM, 1999, pp. 444–453.

[23] D. Chaver, C. Tenllado, L. Pinuel, M. Prieto, and F. Tirado, "Wavelet transform for large scale image processing on modern microprocessors," in *High Performance Computing for Computational Science — VECPAR 2002*, ser. Lecture Notes in Computer Science, J. M. L. M. Palma, A. A. Sousa, J. Dongarra, and V. Hernandez, Eds. Springer, 2003, vol. 2565, pp. 549–562.

[24] D. Chaver, M. Prieto, L. Pinuel, and F. Tirado, "Parallel wavelet transform for large scale image processing," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, Apr. 2002.

[25] D. Chaver, C. Tenllado, L. Pinuel, M. Prieto, and F. Tirado, "Vectorization of the 2D wavelet lifting transform using SIMD extensions," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, Apr. 2003.

[26] ——, "2-D wavelet transform enhancement on general-purpose microprocessors: Memory hierarchy and SIMD parallelism exploitation," in *High Performance Computing — HiPC 2002*, ser. Lecture Notes in Computer Science, S. Sahni, V. K. Prasanna, and U. Shukla, Eds. Springer, 2002, vol. 2552, pp. 9–21.

[27] A. Shahbahrami and B. Juurlink, "A comparison of two SIMD implementations of the 2D discrete wavelet transform," in *Proc. 18th Annual Workshop on Circuits, Systems and Signal Processing*, Veldhoven, The Netherlands, Nov. 2007, pp. 169–177.

[28] A. Shahbahrami, "Improving the performance of 2D discrete wavelet transform using data-level parallelism," in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, Jul. 2011, pp. 362–368.

[29] C. Chrysafis and A. Ortega, "Line-based, reduced memory, wavelet image compression," *IEEE Transactions on Image Processing*, vol. 9, no. 3, pp. 378–389, 2000.

[30] J. Oliver, E. Oliver, and M. P. Malumbres, "On the efficient memory usage in the lifting scheme for the two-dimensional wavelet transform computation," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 1, Sep. 2005, pp. I–485–8.

[31] R. Kutil, “Short-vector SIMD parallelization in signal processing,” in *Parallel Computing*, R. Trobec, M. Vajtersic, and P. Zinterhof, Eds. Springer London, 2009, pp. 397–433.

[32] R. Kutil and P. Eder, “Parallelization of wavelet filters using SIMD extensions,” *Parallel Processing Letters*, vol. 16, no. 3, pp. 335–349, 2006.

[33] W. van der Laan, J. B. T. M. Roerdink, and A. Jalba, “Accelerating wavelet-based video coding on graphics hardware using CUDA,” in *Proceedings of 6th International Symposium on Image and Signal Processing and Analysis (ISPA)*, Sep. 2009, pp. 608–613.

[34] R. Kutil, P. Eder, and M. Watzl, “SIMD parallelization of common wavelet filters,” in *Parallel Numerics ’05*, 2005, pp. 141–149.

[35] M. Iwahashi and H. Kiya, “Non separable two dimensional discrete wavelet transform for image signals,” in *Discrete Wavelet Transforms – A Compendium of New Approaches and Recent Applications*. InTech, 2013.

[36] M. Iwahashi, “Four-band decomposition module with minimum rounding operations,” *Electronics Letters*, vol. 43, no. 6, pp. 27–28, 2007.

[37] M. Iwahashi and H. Kiya, “A new lifting structure of non separable 2D DWT with compatibility to JPEG 2000,” in *Acoustics Speech and Signal Processing (ICASSP)*, 2010, pp. 1306–1309.

[38] D. S. Taubman, “Software architectures for JPEG2000,” in *Proceedings of the IEEE International Conference for Digital Signal Processing*, 2002, pp. 197–200.

[39] D. S. Taubman, E. Ordentlich, M. Weinberger, and G. Seroussi, “Embedded block coding in JPEG 2000,” *Signal Processing: Image Communication*, vol. 17, no. 1, pp. 49–72, 2002.

[40] G. Dillen, B. Georis, J.-D. Legat, and O. Cantineau, “Combined line-based architecture for the 5-3 and 9-7 wavelet transform of JPEG2000,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 9, pp. 944–950, Sep. 2003.

[41] A. Descampe, F. Devaux, G. Rouvroy, B. Macq, and J.-D. Legat, “An efficient FPGA implementation of a flexible JPEG2000 decoder for digital cinema,” in *12th European Signal Processing Conference (EUSIPCO)*, 2004.

[42] Y.-H. Seo and D.-W. Kim, “VLSI architecture of line-based lifting wavelet transform for motion JPEG2000,” *IEEE Journal of Solid-State Circuits*, vol. 42, no. 2, pp. 431–440, Feb. 2007.

[43] C. Zhang, C. Wang, and M. O. Ahmad, “A pipeline VLSI architecture for fast computation of the 2-D discrete wavelet transform,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 8, pp. 1775–1785, Aug. 2012.