Lifting Scheme Cores for Wavelet Transform
Jádra schématu lifting pro vlnkovou transformaci

PhD Thesis summary / zkrácená verze PhD Thesis

Ing. David Bařina

branch of study / studijní obor
Computer Science and Engineering / Výpočetní technika a informatika

supervised by / školitel
prof. Dr. Ing. Pavel Zemčík

reviewed by / oponenti
prof. Heikki Kälviäinen
doc. Dr. Ing. Eduard Sojka

date of defense / datum obhajoby
May 17, 2016 / 17. května 2016

Keywords

discrete wavelet transform, lifting scheme, Cohen-Daubechies-Feauveau wavelet, SIMD, CPU cache, parallelization, JPEG 2000

Klíčová slova

diskrétní vlnková transformace, schéma lifting, vlnka Cohen-Daubechies-Feauveau, SIMD, cache CPU, paralelizace, JPEG 2000

Location of the printed thesis

Faculty of Information Technology
Brno University of Technology
Božetěchova 1/2
612 66 Brno, Czech Republic

ISBN 978-80-214-5378-4
ISSN 1213-4198

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 a 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 exists 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). These issues are discussed below.

In the 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.

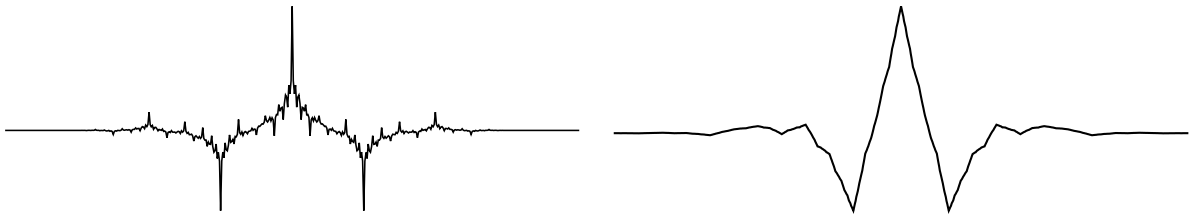

Figure 1.1: Shape of CDF 5/3 and CDF 9/7 wavelets. CDF 5/3 situated on the left, while CDF 9/7 on the right.

As indicated above, for high-resolution data decomposed into several scales by a typical separable transform, 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. The following paragraph particularly 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. For illustration, the CDF 5/3 and 9/7 wavelets are plotted in Figure 1.1.

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.

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. Earliery, 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 decomposition. He showed us 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. The approximation is calculated through two conjugated quadrature filters often referred to as h, g. The relation between the wavelet and these filters is

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

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

where $\phi \in L^2(\mathbb{R})$ is a scaling function, which 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).

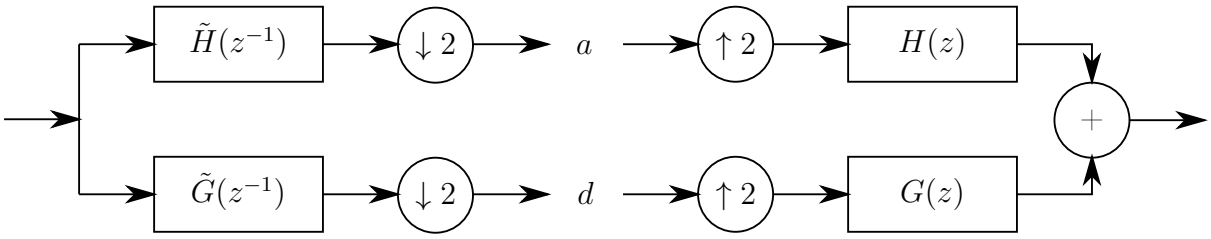

Figure 2.1: Analysis and synthesis part of DWT using FIR filters.

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 a decomposition and analysis is shown in Figure 2.1. Readers not very familiar with 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

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

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 analytical part of DWT (forward transform). 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 $\begin{bmatrix} d & a \end{bmatrix}^T = \tilde{P}(z^{-1})^T \begin{bmatrix} d & a \end{bmatrix}^T$ resulting in a, d subbands. The system is illustrated in Figure 2.2.

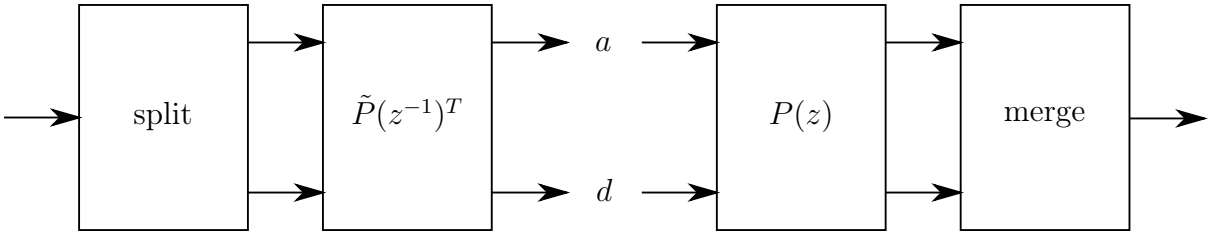

Figure 2.2: Analysis and synthesis part of DWT using lifting schemes.

Focusing on the CDF 9/7 wavelet as an example, the forward transform can be expressed [8] by the dual polyphase matrix

$$\begin{aligned} \tilde{P}(z) = & \begin{bmatrix} 1 & \alpha\,(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \beta\,(1+z) & 1 \end{bmatrix} \\ & \begin{bmatrix} 1 & \gamma\,(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \delta\,(1+z) & 1 \end{bmatrix} \begin{bmatrix} \zeta & 0 \\ 0 & 1/\zeta \end{bmatrix}, \end{aligned} \tag{2.7}$$

where ζ is called the scaling constant. The $\alpha, \beta, \gamma, \delta, \zeta$ are real constants [8] 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. It should be noted 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 a 2-D pyramid leads to a quadruple of wavelet coefficients (a, h, v, d). 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 [8], 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. It should be noted that 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 [13] could perform these separable steps joined together which could even lead 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 an 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 an 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).

A type of the CPU cache is present in all modern platforms. An excellent introduction to this topic can be found in [14]. 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, typically 64-bytes long. 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 the cache lines corresponding to the arbitrary memory location at the same time. In detail, the cache lines are divided into several sets according to an associativity of the cache (e.g. four sets for typical 4-way associativity). The cache associativity indicates the number of lines from a particular set which can be held 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 [15] by Ch. Chrysafis and A. Ortega. Their approach is very general and 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 the lifting scheme nor the 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 – the 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 from doing 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 are likely mapped into the same cache set, especially considering the power-of-two [16, 13] data sizes. In order to solve the last of the mentioned issues, several authors [16, 13, 17] suggested adding 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 be mapped into different sets with a high probability. In particular, the odd or prime strides are often used.

Since only one of the coefficients settled in each cache line is used in a vertical pass, many authors [16, 18, 19] discovered a technique leading to a better utilization of cache lines. This technique is referred to as the aggregation, strip-mine, or loop tiling. Using such a technique, several adjacent columns are filtered concurrently, likely using all the coefficients in each cache line.

So far, the input as well as the output data were stored using a linear-memory layout (particularly, the row-major layout). Several authors investigated the influence of more complicated, possibly non-linear memory layouts (4-D, Morton). 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 [20, 21]. The "mallat" layout utilized in [19, 18] uses an auxiliary matrix in order 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 the memory. This is especially useful for multi-scale decomposition, where the resulting subbands are transformed once more. This layout was employed in [19, 18].

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 [22] used the block-based interleaving with a non-linear 4-D memory layout. Moreover, in [22, 23, 24], the line-based interleaving was used (at least two lines are needed). The most sophisticated techniques were investigated by R. Kutil in [13], which 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 a 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 is 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 (eight in total). 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 variants, if even/odd signal lengths are not considered).

Chapter 3

Lifting Cores

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 for understanding 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 a 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. Although the thesis has focused on image processing, the one-dimensional transform will be discussed first. To simplify the relations, the inequality $0 \leq n_j < N_j$ holds for all $0 \leq j \leq J$. The multi-scale discrete wavelet transform decomposes the input signal $\left(a^0_{n_0}\right)$ of size $N_0 = N$ samples into $J > 0$ scales giving rise to the resulting wavelet bands $\left(d^j_{n_j}\right)$ the temporary bands $\left(a^j_{n_j}\right)$ at scales $0 < j < J$, and the residual signal $\left(a^J_{n_J}\right)$ at the topmost scale J.

In order 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 [VI]. 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. It should be noted that it is also possible to run these cores parallel – this possibility is discussed at the end of the chapter. The corresponding output samples are produced with 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. 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. 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.

The core transforms the fragment $\mathrm{I}_n^j = \begin{pmatrix} a^j_{\Theta(n)} & a^j_{\Theta(n+1)} \end{pmatrix}^T$ of an input signal onto the fragment $\mathrm{O}_n^j = \begin{pmatrix} a^{j+1}_{\Omega(n)} & d^{j+1}_{\Omega(n+1)} \end{pmatrix}^T$ of an input signal, while updating the auxiliary buffer. Finally, operations performed inside the core can be described using a matrix C as the relationship

$$\boldsymbol{y} = C\,\boldsymbol{x} \tag{3.1}$$

from the input vector $\boldsymbol{x} = \mathrm{I}_n^j \parallel \mathrm{B}^j$ onto the output vector $\boldsymbol{y} = \mathrm{O}_n^j \parallel \mathrm{B}^j$, where $\parallel$ denotes the concatenation operator. The (3.1) 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).

3.1 Treatment of Signal Boundaries

In order 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 $\left(a^j_n, a^j_{n+1}\right)$ per fragments of two samples. After performing the calculations, the $\left(d^{j+1}_{\Omega(n)}, a^{j+1}_{\Omega(n+1)}\right)$ 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 a single loop. The naive way [VIII] 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 end of the signal. This buffering breaks the ability of simple stream processing, especially considering the multi-scale decomposition. All the approaches described in Chapter 2 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

$$\boldsymbol{y} = S_{\beta,\Theta(n)}\, T_{\alpha,\Theta(n)}\, \boldsymbol{x}, \tag{3.2}$$

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 when $S_{\beta,\Theta(n+2)}$ runs. It is essential that the coefficients a^b, d^b are initially set to zero. The output signal is generated with a lag $F = 1$ 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 transform is defined using the α, β constants. As a result, the signal is transformed without buffering, possibly on a multi-scale basis.

3.2 Multi-Dimensional Cores

The presented core approach can be naturally extended to multiple dimensions. The key ideas of this section were presented in [VI, VIII, II]. This section 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 $\left(a^0_{m_0,n_0}\right)$ of size $M_0 \times N_0$ pixels into $J > 0$ scales giving rise to the temporary subbands $\left(a^j_{m_j,n_j}\right)$, the resulting wavelet subbands $\left(h^j_{m_j,n_j}\right)$, $\left(v^j_{m_j,n_j}\right)$, $\left(d^j_{m_j,n_j}\right)$, at scales $0 < j < J$, and the residual signal $\left(a^J_{m_J,n_J}\right)$ at the topmost scale J. Such a decomposition is performed using the 2×2 core with lag F samples in both directions. This idea was also proposed in [VIII]. For each scale $0 \leq j < J$, the core requires an access to two auxiliary buffers $\left({}^M B^j_{m_j}\right)_{0 \leq m_j < M_j}$, $\left({}^N B^j_{n_j}\right)_{0 \leq n_j < N_j}$. 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 in 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. The function $\Theta(m, n) = (m + F, n + F)$ maps core output coordinates onto core input coordinates with a lag F. The function $\Omega(m, n) = (\lceil m/2 \rceil, \lceil n/2 \rceil)$ maps the coordinates at the scale j onto coordinates at the scale $j + 1$. It should be noted 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

$$\mathrm{I}^j_{m,n} = \begin{pmatrix} a^j_{\Theta(m,n)} & a^j_{\Theta(m+1,n)} & a^j_{\Theta(m,n+1)} & a^j_{\Theta(m+1,n+1)} \end{pmatrix}^T, \tag{3.3}$$

$$\mathrm{O}^j_{m,n} = \begin{pmatrix} a^{j+1}_{\Omega(m,n)} & h^{j+1}_{\Omega(m+1,n)} & v^{j+1}_{\Omega(m,n+1)} & d^{j+1}_{\Omega(m+1,n+1)} \end{pmatrix}^T, \tag{3.4}$$

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

$$\boldsymbol{y} = C\,\boldsymbol{x} \tag{3.5}$$

from the input vector $\boldsymbol{x} = \mathrm{I}^j_{m,n} \parallel {}^M\mathrm{B}^j_m \parallel {}^M\mathrm{B}^j_{m+1} \parallel {}^N\mathrm{B}^j_n \parallel {}^N\mathrm{B}^j_{n+1}$ onto the output vector $\boldsymbol{y} = \mathrm{O}^j_{m,n} \parallel {}^M\mathrm{B}^j_m \parallel {}^M\mathrm{B}^j_{m+1} \parallel {}^N\mathrm{B}^j_n \parallel {}^N\mathrm{B}^j_{n+1}$, where $\parallel$ denotes the concatenation operator. One needs to 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.

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. On the contrary, many scan orders are now possible. In should be noted that the original single-loop approach from [25] does not have this ability. The above-described degree of freedom allows us 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 [II]. When associated with the capabilities explained in the previous paragraph, these codeblocks can be generated in parallel. This experiment is further evaluated below.

3.3 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 a 2×2 core, the matrix C in the relationship $\boldsymbol{y} = C\boldsymbol{x}$ can be factored into

$$\boldsymbol{y} = {}^N S_\beta \; {}^N T_\alpha \; {}^M S_\beta \; {}^M T_\alpha \, \boldsymbol{x}, \tag{3.6}$$

where the M superscripts refer to the horizontal direction, whereas N refers 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 only strictly fixed but also completely unconstrained. The implementation has the latency of four lifting steps, plus scaling. In total, 16 non-trivial operations (four in each stage) are needed to calculate this core (the scaling is omitted).

In [26], 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. Thanks to the parallel processing of v and h samples, this implementation has a total latency of 3 steps. Practical implementations with a 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

$$\boldsymbol{y} = A_{\beta}\, T_{\alpha,\beta}\, D_{\alpha}\, \boldsymbol{x}, \tag{3.7}$$

where D_{α} operator computes the d coefficient, $T_{\alpha,\beta}$ computes h and v, and A_{β} finally computes a coefficient. Excluding diagonals, the matrices $A_{\beta}, T_{\alpha,\beta}, D_{\alpha}$ have a total of 24 non-zero entries implying 24 non-trivial MAC operations.

Using the core approach presented in this thesis, it is possible to go further. The total number of arithmetic operations in the non-separable scheme [26] can be reduced. The key idea here is not to calculate the wavelet coefficients all at once. Instead, the calculation of these coefficients is subdivided into separate parts. The sum of these parts gives us the desired result. The starting point of the solution is the non-separable lifting scheme of CDF 5/3 transform as described in [26]. After some rewriting of expressions, a new scheme has appeared. 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. It should be emphasized that this newly formed scheme cannot be derived using instruments in [26]. This is caused by the fact that the authors of [26] do not specify a sequence of the operations. The implementation is always a trade-off between latency and the number of operations. In matrix notation, the transform can be written as the product

$$\boldsymbol{y} = A_{\beta}\, D_{\alpha}\, \boldsymbol{x}. \tag{3.8}$$

The steps are graphically illustrated in Figure 3.1.

Table 3.1 provides a summarized 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. 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

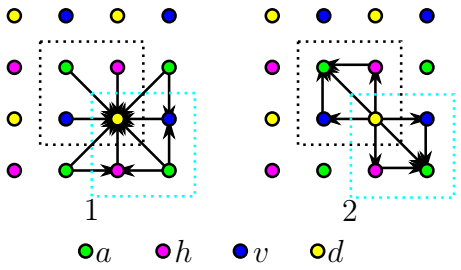

Figure 3.1: Proposed non-separable lifting core of CDF 5/3 with two stages. The input coefficients of the active core are in the bright box. The output ones are in the dark one.

core	latency	buffer	operations	max. operands in step
separable [6]	4	8	16	$2+1$
non-separable [26]	3	10	24	$8+1$
proposed	2	8	22	$8+1$

Table 3.1: Comparison of the 2-D single-loop cores. The operands are given in format non-trivial plus trivial. The scaling is omitted.

the number of stages (steps), the number of coefficients accessed in the auxiliary buffers, and the total number of non-trivial operations.

3.4 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 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]. This section is still focused on the CDF 5/3 transform. The generalization is straightforward.

Iwahashi *et al.* [27, 26] recently 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 $\boldsymbol{x} = \begin{bmatrix} a & h & v & d \end{bmatrix}$, $\boldsymbol{y} = \begin{bmatrix} a & h & v & d \end{bmatrix}$. These transformations can formally be compressed into the matrix $C_{\alpha,\beta}$ in

$$\boldsymbol{y} = C_{\alpha,\beta}\,\boldsymbol{x} = A_\beta\,T_{\alpha,\beta}\,D_\alpha\,\boldsymbol{x}. \tag{3.9}$$

The scheme is graphically illustrated in Figure 3.2b (referred to as *Iwahashi2007*). Simi-

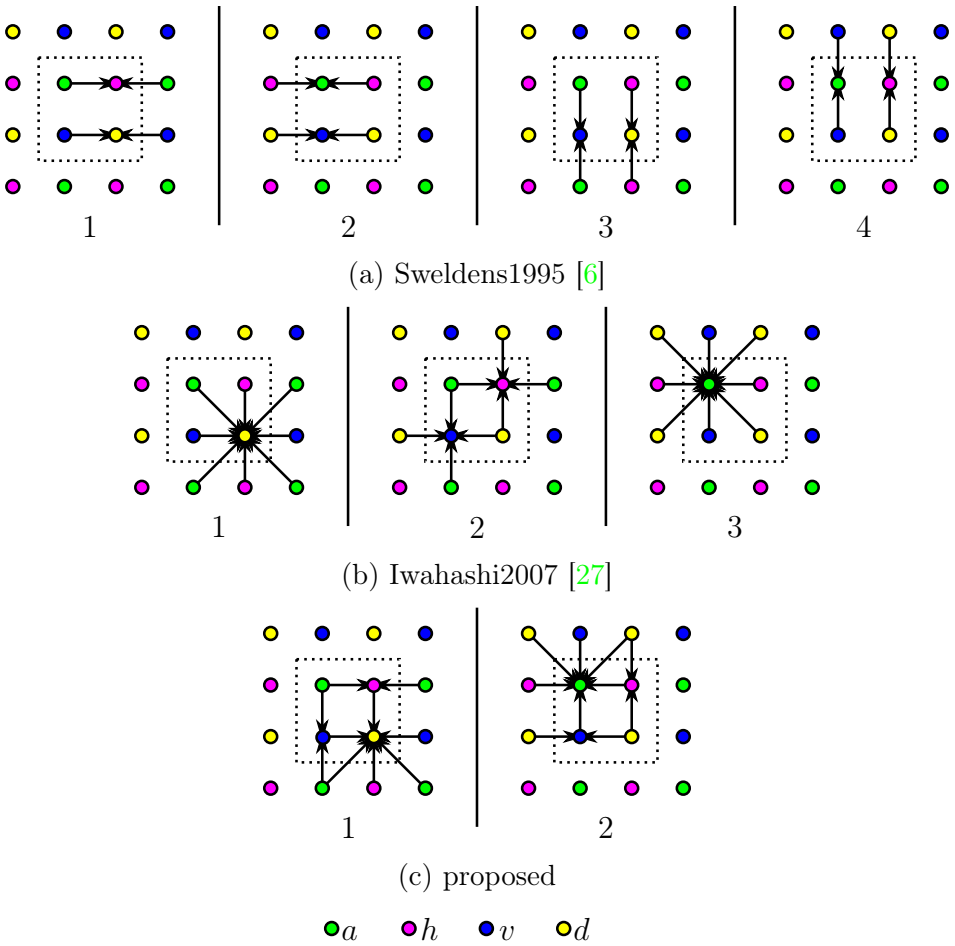

Figure 3.2: 2-D data-flow graphs of the parallel cores. The order of the lifting steps is determined by the bottom numbers. The vertical lines indicate the necessary memory barriers.

larly in the original scheme, a memory barrier must be inserted between each of the two steps. As a result, the scheme consists of 24 non-trivial arithmetic operations in three lifting steps separated by two explicit memory barriers. For the sake of comparison, the baseline separable scheme is illustrated in Figure 3.2a (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.* [26], 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. 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. 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 $h^{(1)}, v^{(1)}$. Their initial as well as final values are unimportant. The scheme

$$\boldsymbol{y} = S_\beta^4 \, S_\beta^3 \, S_\alpha^2 \, S_\alpha^1 \, \boldsymbol{x} \tag{3.10}$$

describes the relation between input $\boldsymbol{x} = \begin{bmatrix} a & h & v & d & h^{(1)} & v^{(1)} \end{bmatrix}$ and output $\boldsymbol{y} = \begin{bmatrix} a & h & v & d & h^{(1)} & v^{(1)} \end{bmatrix}$ vectors. It should be noted that in practical realizations, each single computing unit (e.g. thread) can be responsible of such a vector. In addition, the operations are graphically illustrated in Figure 3.2c (referred to as *proposed*). Compared with [26], the total number of arithmetic operations has been reduced from 24 to 20 for the CDF 5/3 wavelet. The calculation of the CDF 9/7 transform consists of two such connected transforms (the first with α, β, the second with γ, δ) and 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 3.2. 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.

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 only 50 %.

scheme	steps	operations	max. operands	memory cells
Sweldens1995 [6]	4	16	3	4
Iwahashi2007 [27]	3	24	9	4
proposed	**2**	20	9	6

Table 3.2: 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).

Chapter 4

Evaluation

This chapter provides deep performance evaluation of the presented core approach. Several experiments evaluating the performance of different methods were conducted on general-purpose processors. Several 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 the single-loop methods to be 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. Furthermore, considering the 2-D transform, the core implementation of the single-loop approach is discussed in detail. The cores disclose several degrees of freedom. This advantage is especially useful when considering the multi-dimensional transform. Namely, a 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 the JPEG 2000 encoder. Moreover, many possible uses of SIMD extension became available in the case of multi-dimensional core. Particularly, 4×4 vectorized core is the best performing one on the 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, no synchronizations are even required in between threads considering the horizontal adjacency of parallel blocks. The cores incorporated into the JPEG 2000 compression chain have proven 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 the 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 a smaller amount of flip-flops.

The cores may also be advantageously used on massively-parallel architectures. This option was demonstrated using the 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 which was proven to be the fastest way to transform the 2-D data.

4.1 General-Purpose Processors

The following discussion shows the effect of coarse-grained parallelization of the above discussed approaches. The naive approach that uses the horizontal and vertical 1-D transform was parallelized using multiple threads. The same was done with vectorized core single-loop approach. 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. In order to produce correct results, each thread must process a segment (several rows) of an 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. A summarized comparison of parallelizations is shown in Table 4.1. The measurements were performed on a 58-megapixel image. The single-threaded algorithm is used as a reference one. The core approach scales almost linearly with the number of threads. In order to also confirm the performance in a multi-scale scenario, the presented single-loop cores have been incorporated into the JPEG 2000 encoding chain.

Efficient realization of the JPEG 2000 transform was outlined by D. Taubman in [28]. The author 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 the 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 [28], their implementation does not process the data in a single loop. However, I assumed at the time that their implementation would do so. This strategy is still fundamentally different from the architecture described in this section which generates individual blocks sequentially while all the time reusing the same memory area for output coefficients. The above-described line-based processing does not fit the JPEG 2000 codeblocks, does not allow for the parallel codeblock processing or for the reuse of the memory for h, v, and d subbands. The motivation behind my work is to overcome these issues.

threads	1		2		4	
algorithm	time	speedup	time	speedup	time	speedup
naive	46.9	1.0	24.0	2.0	12.1	3.9
core	**4.3**	**11.0**	**2.3**	**20.5**	**1.2**	**39.3**

Table 4.1: Performance evaluation using threads on AMD Opteron. The time is given in nanoseconds per pixel. The speedups are shown compared to the naive algorithm.

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 core is 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 passed straight to the subsequent codeblock processing unit. The horizontal buffer will be used by a strip of codeblocks lying below. The transform of this subsequent unit is not started earlier than the EBCOT [29] on the current unit has been finished. This allows for reusing the memory for h, v, and d subbands.

The described procedure is in effect friendly to the cache hierarchy. The processing engine uses several memory regions for 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 the vertical buffer. However, they are used only for a 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.

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 do not need to exchange any data). In a 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.

The performance of the test implementation was evaluated. The input image is consumed gradually using strips with a height of 2×2^{c_m} lines. No more input data are required to be placed in the physical memory at the same time. 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. It should be noted 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 $3 \times 2^{c_n} \times M_{j+1}$. It must be noted that 3 instead of 2 codeblock strips are needed due to the periodic symmetric extension on the

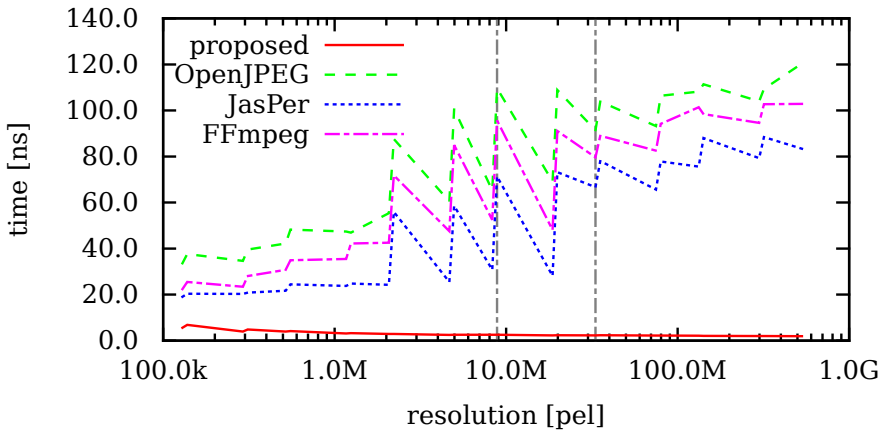

Figure 4.1: Performance comparison of JPEG 2000 libraries. Time per pixel for the transform stage only. DCI 4K and 8K UHD resolutions indicated by the vertical lines.

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 open-source C/C++ libraries listed on the official JPEG committee web pages – OpenJPEG, FFmpeg, and JasPer. The transform stage was extracted from the libraries in order to get accurate results. This stage was then subjected to measurement. The results are shown in Figure 4.1. As observed also in [13], the single-loop processing has stable performance regardless of the input resolution. The proposed implementation was measured using four threads and SSE extensions.

4.2 Graphics Processing Units

Two parallel lifting scheme schedules for GPGPUs were designed and implemented. These schedules are 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. The achieved memory throughput performance is shown in Figure 4.2. Both of the core methods overcome the state-of-the-art method. The proposed non-separable core performs slightly better compared to the separable one. This behavior corresponds to the reduced number of the memory barriers.

4.3 Field-Programmable Gate Arrays

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

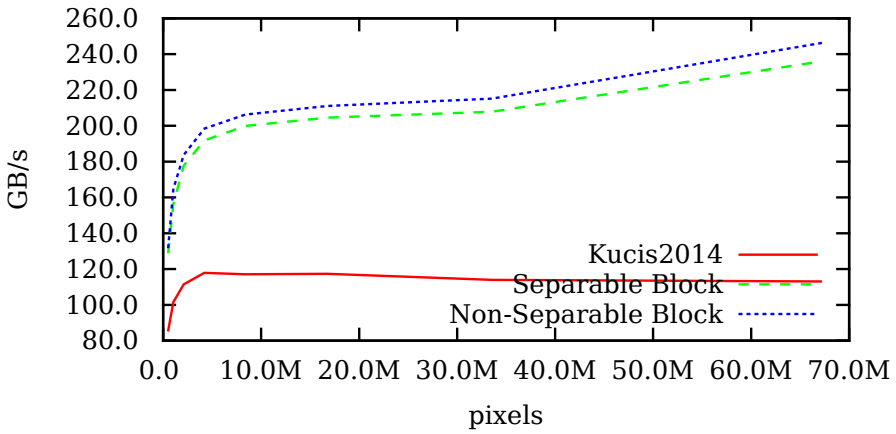

Figure 4.2: NVIDIA TitanX. Throughput performance of parallel methods. *Kucis2014* denotes the reference method presented in [IX].

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 that merely differ in the BRAM size, only to allow comparison with other papers, and also to show that the core is able to process Full HD video (1080p, 60 Hz) faster than in real-time. 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 especially like to highlight the ability to process the video stream without the need of using external memory for intermediate results. The design includes mirroring on the image edges which is not performed by the wavelet core itself, but by the engine, which encapsulates the core. The engine itself then represents an independent block, which can be used in a 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 4.2.

architecture	device	BRAM [bits]	clocks/pel	time [ms]
Dillen [30]	VirtexE1000-8	50K	0.50	1.20
Seo [31]	Altera Stratix	128K	2.64	6.02
Zhang [32]	Virtex-II Pro XC2VP30	6 × 18K	0.50	0.97
proposed	Zynq XC7Z045	1 × 36K	**0.26**	**0.27**

Table 4.2: Comparison of various FPGA implementations. Tiles of size 512 × 512. The processing time and clocks per pixel were projected to the uniform image size.

Chapter 5

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 a 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 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 a 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.

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," *Comm. 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," *Comm. 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 Trans. on Patt. Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693, 1989.

[5] ——, "Multifrequency channel decompositions of images and wavelet models," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 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] R. Kutil, "A single-loop approach to SIMD parallelization of 2-D wavelet lifting," in *Euromicro Intl. Conf. on Par., Distrib., and Network-Based Proc. (PDP)*, 2006, pp. 413–420.

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

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

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

[17] A. Shahbahrami, B. Juurlink, and S. Vassiliadis, "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.

[18] D. Chaver, C. Tenllado, L. Pinuel, M. Prieto, and F. Tirado, "Vectorization of the 2D wavelet lifting transform using SIMD extensions," in *Intl. Par. Dist. Proc. Symp. (IPDPS)*, Apr. 2003.

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

[20] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi, "Nonlinear array layouts for hierarchical memory systems," in *International Conference on Supercomputing (ICS)*. New York, NY, USA: ACM, 1999, pp. 444–453.

[21] D. Chaver, M. Prieto, L. Pinuel, and F. Tirado, "Parallel wavelet transform for large scale image processing," in *Intl. Par. Dist. Processing Symposium (IPDPS)*, Apr. 2002.

[22] D. Chaver, C. Tenllado, L. Pinuel, M. Prieto, and F. Tirado, "2-D wavelet transform enhancement on general-purpose microprocessors: Memory hierarchy and SIMD parallelism exploitation," in *High Perf. Computing (HiPC)*, ser. LNCS, S. Sahni, V. K. Prasanna, and U. Shukla, Eds. Springer, 2002, vol. 2552, pp. 9–21.

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

[24] 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.

[25] R. Kutil, P. Eder, and M. Watzl, "SIMD parallelization of common wavelet filters," in *Parallel Numerics '05*, 2005, pp. 141–149.

[26] 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.

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

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

[29] 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.

[30] 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.

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

[32] 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.

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

Selected Papers

[I] Barina, D.; Klima, O.; Zemcik, P.: Single-Loop Architecture for JPEG 2000. In *Intl. Conf. on Image and Signal Processing (ICISP)*, *LNCS*, vol. 9680, Springer, 2016, pp. 346–355.

[II] Barina, D.; Klima, O.; Zemcik, P.: Single-Loop Software Architecture for JPEG 2000. In *Data Compression Conference (DCC)*, 2016, pp. 582–582.

[III] Barina, D.; Musil, M.; Musil, P.; et al.: Single-Loop Approach to 2-D Wavelet Lifting with JPEG 2000 Compatibility. In *Wksp. Appl. MC Arch. (WAMCA)*, IEEE, 2015, pp. 31–36.

[IV] Barina, D.; Zemcik, P.: Minimum Memory Vectorisation of Wavelet Lifting. In *Advanced Concepts for Intel. Vision Systems (ACIVS)*, *LNCS*, vol. 8192, Springer, 2013, pp. 91–101.

[V] Barina, D.; Zemcik, P.: Wavelet Lifting on Application Specific Vector Processor. In *GraphiCon*, GraphiCon Scientific Society, 2013, pp. 83–86.

[VI] Barina, D.; Zemcik, P.: Diagonal Vectorisation of 2-D Wavelet Lifting. In *Intl. Conf. on Image Processing (ICIP)*, IEEE, 2014, pp. 2978–2982.

[VII] Barina, D.; Zemcik, P.: Real-Time 3-D Wavelet Lifting. In *Intl. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, 2015, pp. 15–23.

[VIII] Barina, D.; Zemcik, P.: Vectorization and parallelization of 2-D wavelet lifting. *Journal of Real-Time Image Processing (JRTIP)*, in press, ISSN 1861-8200.

[IX] Kucis, M.; Barina, D.; Kula, M.; et al.: 2-D Discrete Wavelet Transform Using GPU. In *Workshop on Application for Multi-Core Architectures (WAMCA)*, IEEE, 2014, pp. 1–6.

[X] Kula, M.; Barina, D.; Zemcik, P.: Block-based Approach to 2-D Wavelet Transform on GPUs. In *Information Technology: New Generations (ITNG)*, *Advances in Intelligent Systems and Computing*, vol. 448, Springer, 2016, pp. 643–653.

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 a 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 proven to be fundamentally faster than widely used implementations.

Abstrakt

Práce se zaměřuje na efektivní výpočet dvourozměrné diskrétní vlnkové transformace. Současné metody jsou v práci rozšířeny v několika směrech a to tak, aby spočetly tuto transformaci v jediném průchodu, a to případně víceúrovňově, použitím kompaktního jádra. Tohle jádro dále může být vhodně přeorganizováno za účelem minimalizace užití některých prostředků. Představený přístup krásně zapadá do běžně používaných rozšíření SIMD, využívá hierarchii cache pamětí moderních procesorů a je vhodný k paralelnímu výpočtu. Prezentovaný přístup je nakonec začleněn do kompresního řetězce formátu JPEG 2000, ve kterém se ukázal být zásadně rychlejší než široce používané implementace.