Minimum Memory Vectorisation of Wavelet Lifting

David Barina and Pavel Zemcik

Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
{ibarina,zemcik}@fit.vutbr.cz

Abstract. With the start of the widespread use of discrete wavelet transform the need for its effective implementation is becoming increasingly more important. This work presents a novel approach to discrete wavelet transform through a new computational scheme of wavelet lifting. The presented approach is compared with two other. The results are obtained on a general purpose processor with 4-fold SIMD instruction set (such as Intel x86-64 processors). Using the frequently exploited CDF 9/7 wavelet, the achieved speedup is about 3× compared to naive implementation.

Keywords: discrete wavelet transform, lifting scheme, parallelization, vectorisation, SIMD

1 Introduction

The discrete wavelet transform (DWT) is mathematical tool which is able to decompose discrete signal into lowpass and highpass frequency components. Such a decomposition can be performed at several scales. DWT is often used as the basis of sophisticated compression algorithms. This is the case of JPEG 2000 and Dirac compression standards in which CDF 9/7 wavelet [4] is employed for lossy compression. Responses of this wavelet can be computed by a convolution with two FIR filters, one with 7 and the other with 9 coefficients. For the DWT computation, the well known Mallat's [8] filtering scheme can be used. Alternatively, one can use usually faster scheme called lifting which was presented by I. Daubechies and W. Sweldens in [5]. Lifting data flow graph consists of regular grid computational scheme suitable for SIMD vectorisation. Both of the algorithms can be performed over some approximation of real numbers. This paper focuses on single-precision floating-point format.

In contemporary personal computers (PCs), the general purpose microprocessor with SIMD instruction set is often found. In case of the x86-64 architecture, the appropriate instruction set used here is SSE (Streaming SIMD Extensions). This 4-fold SIMD set fits exactly the CDF 9/7 lifting data flow graph.

In this work, we discuss vectorisation (parallelization) of 1-D discrete wavelet transform on processors with SIMD extensions. Two of the discussed methods

can be used in memory limited systems. Specifically, we focus on the PC and similar platforms.

The rest of the paper is organized as follows. More traditional approaches to DWT computation are reviewed in Section 2. Section 3 describes opportunities for lifting scheme parallelizations and presents the proposed approach. The vectorisation methods are compared in Section 4. Finally, Section 5 concludes the paper.

2 Related Work

In 2000, the problem of minimum memory implementations of lifting scheme was addressed in [3] by Ch. Chrysafis and A. Ortega. This approach is very general and it is not focused on parallel processing. Anyway, this is essentially the same method as the on-line or pipelined computation mentioned in other papers (although not necessarily using lifting scheme nor 1-D transform). Especially, its variation was presented six year later in [6] which is specifically focused on CDF 9/7 wavelet transform. The work was also later extended to [2] where same authors addressed a problem of minimum memory implementation of 2-D transform.

In [7] R. Kutil *et al.* presented SIMD parallelizations of several frequently used wavelet filters. This vectorisation is applicable only on those filters discussed in their paper. Specifically, vectorisation of CDF 9/7 wavelet computed using lifting scheme is vectorised here by a group of four successive pairs of coefficients. Unlike a general approach proposed in our paper, their 1-D transform vectorisation handles coefficients in blocks. Our vectorisation process pairs of coefficients one by one immediately when available (without packing into groups).

According to the number of arithmetic operations, the lifting scheme [5] is today's most efficient scheme for computing discrete wavelet transforms. Any discrete wavelet transform with finite filters can be factored into a finite sequence of N pairs of predict and update convolution operators P_n and U_n. Each predict operator P_n corresponds to a filter $p_i^{(n)}$ and each update operator U_n to a filter $u_i^{(n)}$. Block diagram of such a system is depicted in Figure 1.

$$P_n(z) = \sum_{i=-l_n}^{g_n} p_i^{(n)} z^{-i} \tag{1}$$

$$U_n(z) = \sum_{i=-m_n}^{f_n} u_i^{(n)} z^{-i} \tag{2}$$

This factorisation is not unique. For symmetric filters, this non-uniqueness can be exploited to maintain symmetry of lifting steps.

Consider the decomposition of the signal of length of L samples. Without loss of generality one can assume only signals with even length L. Possible remaining coefficient can treated separately in the prolog or epilog phases together with border extension. Thus, the transform contains $S = L/2$ pairs of resulting

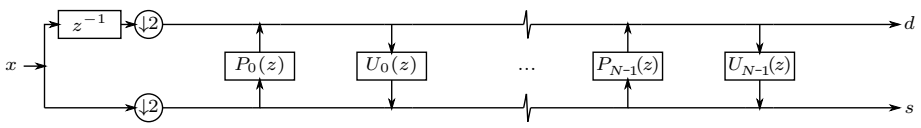

Fig. 1. Block diagram of lifting scheme. The system consists of $2N$ lifting steps. For simplicity, the scaling of the resulting coefficients was ommited.

wavelet coefficients (s, d). The s coefficients represent a smoothed signal. On the contrary, the d coefficients form a difference or detail signal.

In their paper [5], Daubechies and Sweldens demonstrated an example of CDF 9/7 transform factorisation which resulted into four lifting steps ($N = 2$) plus scaling of coefficients. In this example, the individual lifting steps use 2-tap symmetric filters for the prediction as well as the update. This can be graphically described as shown in Figure 2. Here, outer arrows represents 2-tap symmetric filter and inner arrow represents predicted (resp. updated) coefficient. In all figures shown in this paper, the coefficients of these four 2-tap symmetric filter are denoted α, β, γ and δ respectively.

Fig. 2. Elementary lifting operation (P_n or U_n) of the CDF 9/7 wavelet. The flow in the middle is just added into result at the bottom. Side flows are multiplied by a constant first.

When coefficient scaling is omitted, the calculation of a pair of the DWT coefficients at the position l (s_l and d_l) is performed by four such a lifting steps. Intermediate results ($s_l^{(n)}$ and $d_l^{(n)}$) can be appropriately shared between neighbouring pairs of coefficients (s_l and d_l). Finally, the calculation of the complete CDF 9/7 DWT is depicted in Figure 3. This is an in-place implementation, which means the DWT can be calculated without allocating auxiliary memory. Resulting coefficients (s_l and d_l) are interleaved in place of the input signal.

3 Vectorisation

The calculation scheme described in the previous section can be realized in a number of different ways. In this work, three of such ways are described. The main difference between them is in the order of lifting steps evaluation. Alternatively, the data flow graph in Figure 3 can be split into areas that are evaluated sequentially according to their data dependencies.

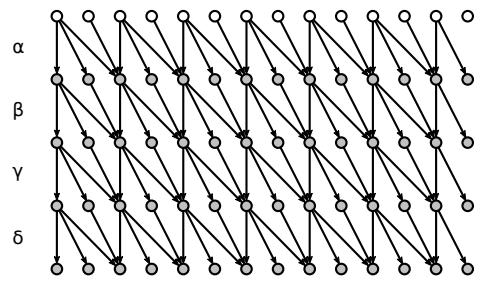

Fig. 3. Complete data flow graph of CDF 9/7 wavelet transform. The input signal is on top, output at the bottom. The graph borders must be treated in a special way using prolog and epilog phases.

3.1 Naive approach

The naive approach of data flow graph evaluation directly follows the lifting steps (n). Thus, all intermediate $s^{(1)}$ and $d^{(1)}$ coefficients are evaluated in the first step. Then, all $s^{(2)}$ and $d^{(2)}$ are evaluated in second step, etc. For a better understanding see the block diagram in Figure 4. Unfortunately, this algorithm requires several reads and writes of the intermediate results $s_l^{(n)}$ and $d_l^{(n)}$. For long signals, these intermediate results will be several times evicted from the CPU cache in favor of other intermediate results. Consequently, many cache misses during such a computation will occur.

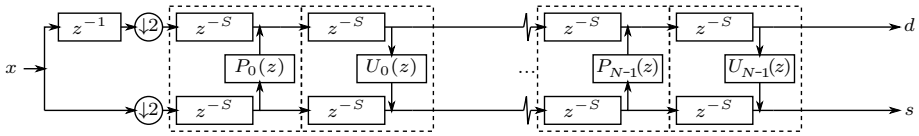

Fig. 4. Block diagram of the horizontal lifting scheme vectorisation. The parts bounded with a dashed line correspond to the areas of parallel computation.

In this paper, this method is called the horizontal vectorisation. This name reflects the fact that the data flow graph is split in horizontal areas as in Figure 5. In each area, elementary calculations are independent and can be computed in parallel. For simplicity, the scaling of coefficients and the prolog and epilog phases were omitted in the referenced figure. An entire signal of $2S$ samples must be loaded into the memory which is not suitable for memory limited systems.

3.2 Vertical vectorisation

Another way of lifting data flow graph evaluation is the double-loop approach [6]. This approach is referred to as the vertical vectorisation. Earlier, it was described in [3] focusing on low memory systems but without vectorisation.

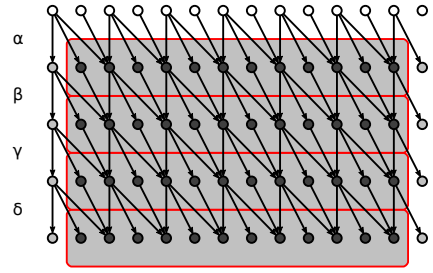

Fig. 5. The horizontal vectorisation of the CDF 9/7 data flow graph. The scaling of coefficients was omitted. The computation within the highlighted areas can be processed in parallel.

The P_n and U_n filters need not be causal. In general, non-causal systems requires storing the whole input signal into memory (as can be seen from Figure 5). This is not suitable for fast or memory limited signal processing as well as for a vectorisation. Therefore, it would be appropriate to convert non-causal lifting steps (P_n and U_n) to causal systems. The key to force these filtering steps to be causal is the introduction of appropriate delays.

$$\mathcal{P}_n(z) = z^{-l_n} P_n(z) = \sum_{i=0}^{g_n+l_n} p^{(n)}_{i-l_n} z^{-i} \tag{3}$$

$$\mathcal{U}_n(z) = z^{-m_n} U_n(z) = \sum_{i=0}^{f_n+m_n} u^{(n)}_{i-m_n} z^{-i} \tag{4}$$

The transition from non-causal to causal system introduce a delay z^{-l_n} on both inputs of the prediction filtering step P_n. In the bottom input s, the delay can be distributed into both branches. This leads to a causal system $\mathcal{P}_n$ as in (3). Analogously, a delay of m_n samples is introduced on both inputs of update step U_n. Again, this delay can distributed into branches of upper input d. The resulting equation is shown in (4). For simplicity, the adjacent delays can combined into single one. Finally in (5), delays of η_n, μ_n and ν_n samples appear around each pair of filtering steps $\mathcal{P}_n$ and $\mathcal{U}_n$. The resulting block diagram is shown in Figure 6.

$$\eta_n = l_n \tag{5a}$$
$$\mu_n = l_n + m_n \tag{5b}$$
$$\nu_n = m_n \tag{5c}$$

In this method, the lifting computation is transformed into one loop instead of multiple loops over all the coefficients. Therefore, one pair of lifting coefficients s_l and d_l is computed in each iteration of such a single loop. However, the

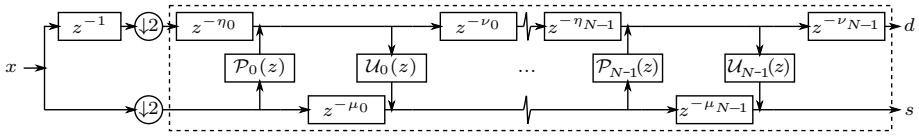

Fig. 6. Block diagram of vertical lifting scheme vectorisation. The part bounded with dashed line correspond to the area of parallel computation.

computations within each of these areas cannot be directly parallelized due to data dependencies. Even so, this procedure is advantageous because the coefficients are read and written only once. Consequently, this prevents unnecessary cache misses. In our 1-D case, the SIMD vectorisation of this method lies in processing of four adjacent areas in parallel like in [7]. The data flow graph is split in vertical areas of width of two coefficients as in Figure 7. Furthermore, this approach is particularly useful for multidimensional (e.g. 2-D) transform on PC platform where several data rows are processed in single loop at once using n-fold SIMD instructions.

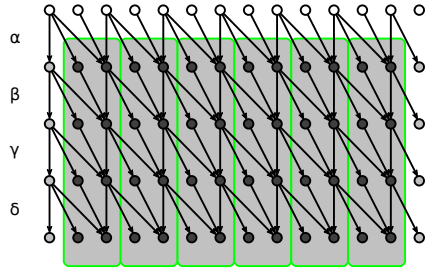

Fig. 7. Vertical vectorisation of the CDF 9/7 data flow graph. The computation within the highlighted areas cannot be processed in parallel due to data dependencies.

3.3 Proposed method

The main contribution of this paper is the following approach. This method is referred to as the diagonal vectorisation here. It is especially useful on limited memory systems because it can start iteration of vectorised loop immediately when a new pair of coefficients is available. Another area of application can be the Intel x86-64 or similar architecture equipped with small CPU cache and SIMD instruction set.

The subsequent lifting operations $\mathcal{P}_n$ and $\mathcal{U}_n$ inside the area of vectorisation above cannot be computed in parallel due to data dependencies. To eliminate these dependencies another delay of one sample is introduced on both lines s and d, see Figure 8.

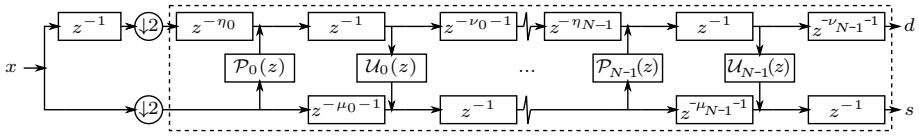

Fig. 8. Block diagram of diagonal lifting scheme vectorisation. In contrast to the vertical vectorisation, a delay of 1 sample is introduced on both lines s and d. This removes immediate data dependencies between subsequent lifting operators $\mathcal{P}_n$ and $\mathcal{U}_n$. Consequently, these lifting operators can be evaluated in parallel.

Similarly to the case of vertical vectorisation, multiple loops of naive approach are transformed into the single loop over all the coefficients. One pair of resulting coefficients s and d is produced in each iteration. Unlike the vertical approach, the elementary lifting operations evaluated in single loop iterations are shifted with respect to each other. This shift removes the data dependency within these loop iteration. Therefore, the elementary operations can be now computed in parallel. This is advantageous especially on the PC platform for processing with SIMD instructions. This approach in called diagonal vectorisation here. Corresponding slices of the data flow graph are depicted in Figure 9.

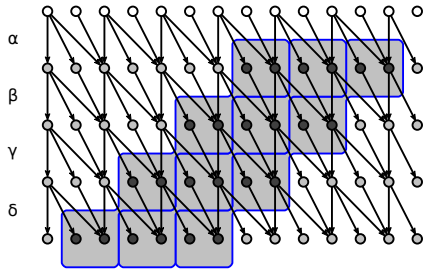

Fig. 9. Diagonal vectorisation of the CDF 9/7 data flow graph. The computation within the highlighted areas can be processed in parallel.

In contrast to the vertical vectorisation, the proposed method does not require buffering of the input samples into groups of width corresponding to the used SIMD instruction set. A pair of resulting coefficients is available immediately after processing a pair of input samples. On the other hand, it is necessary to choose a wavelet with one such lifting factorisation which has the same number of lifting steps (i.e. $2N$) like components of the SIMD set. Depending on the instruction set being used, more shuffling instruction may be needed to implement the proposed diagonal vectorisation (which is the case of Intel's SSE).

Considering the CDF 9/7 wavelet with $N = 2$ pairs of lifting steps, the diagonal vectorisation can be accelerated e.g. using Intel's MMX instruction set with 16-bit integer or fixed-point numbers, SSE set with single-precision floating-

point numbers, SSE2 set with 16-bit integer or fixed-point numbers or AVX set with double-precision floating-point numbers.

4 Evaluation

The implementations of the approaches described in the previous section was evaluated on two x86-64 computers. This comparison was performed on 1-D forward DWT using CDF 9/7 wavelet. All the implementations work over a sequence of single-precision floating point numbers. According to platform performance, a length of the sequence was progressively extended from vector of 32 samples with geometrical step of 1.28 up to 55 millions of samples. The transform was computed including a final coefficient scaling and correct border extensions. The resulting coefficients remain interlaced at their original positions.

The first platform used in this paper is a classical PC with x86-64 CPU. All the results are obtained on Intel Core2 Duo CPU E7600 at 3.06 GHz with 32 kB of Level 1 data cache and 3 MB of Level 2 cache. The Level 1 data cache is 8-way set associative with cache lines of 64 bytes. The processor is equipped with 4-fold SSE instruction set. Thus, the SSE instructions are able to perform simple operation on four 32-bit single-precision floating point numbers in parallel. The evaluated programs ran under 64-bit Linux system and had been compiled by GCC 4.6.3 with `-O2` option. All programs were executed on a lightly loaded system. In addition, the K-best measurement system have been used.

Measurement results were verified on a second PC with x86-64 CPU. In this case, the results are obtained on AMD Athlon 64 X2 4000+ at 2.1 GHz with 64 kB L1 data cache and 512 kB of L2 cache. The L1 data cache is 2-way set associative cache with cache lines of 64 bytes. As well as Intel Core2, the Athlon is equipped with SSE set. Programs ran under 64-bit Linux and had been compiled by GCC 4.7.2 with `-O2` option. Although running at lower frequency compared to Intel, the AMD is in general faster in this task.

The evaluation is shown in Figure 10 and Figure 11. The horizontal vectorisation fails with samples exceeding the CPU cache size due to extensive cache misses. In contrast, the vertical and diagonal vectorisation show stability with increasing input length. In case of the proposed method, the achieved speedup is up to 3.1× on Intel and 3.1× on AMD.

The created implementation of all three algorithms used in this paper can be downloaded from the Internet.[1] The vertical and diagonal vectorisation methods were implemented using SSE intrinsics and inline assembly (no auto-vectorisation of GCC was used). In both cases, aligned memory access instructions was used to access the coefficients. This required merging of several loop iterations (the areas in Figure 7 and Figure 9) into the single one.

Table 1 shows the comparison of the different algorithms in terms of memory consumption. Each of the method require t samples to start iteration of the vectorised loop and b memory cells to store intermediate results. The horizontal

[1] `http://www.fit.vutbr.cz/research/view_product.php?id=211`

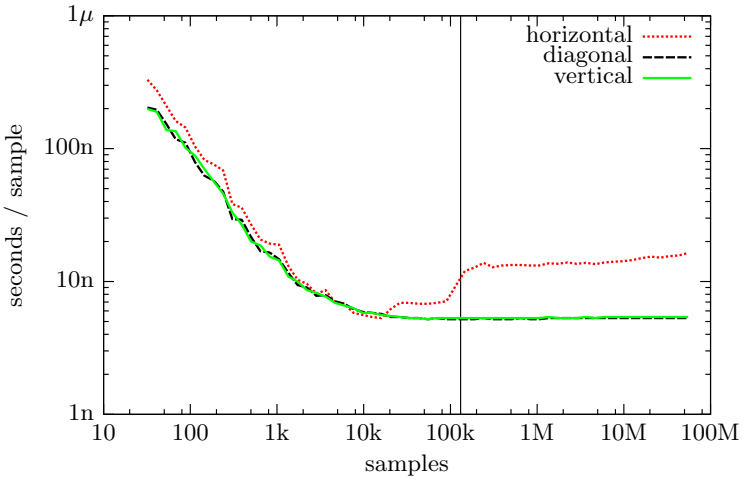

Fig. 10. Comparison of all three vectorisation approaches on the AMD x86-64 platform. The vertical line represents the size of the L2 CPU cache.

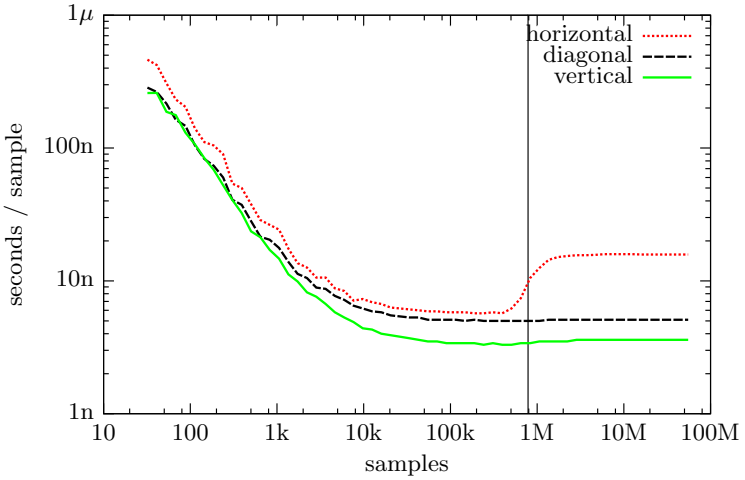

Fig. 11. Comparison performed on the Intel x86-64 platform. The vertical line shows the size of the L2 CPU cache.

vectorisation needs whole signal of $2S$ samples (S pairs of coefficients) to be loaded into memory. On this signal, up to S independent operations can evaluated in parallel. In contrast, the vertical vectorisation needs only $2T$ samples to start iteration of the vectorised loop in that T lifting operators can be evaluated in parallel. In the case of 3-tap P and U operators, this vectorisation needs only $2N$ memory words to store intermediate results between such subsequent iterations. Finally, the diagonal vectorisation requires only 2 new samples for each iteration which evaluate $2N$ lifting operators in parallel.

vectorisation	t samples	b coefficients	q operations
horizontal	$2S$	$2S$	S
vertical	$2T$ (8)	$2N$ (4)	T (4)
diagonal	2 (2)	$6N-2$ (10)	$2N$ (4)

Table 1. Memory consumption of vectorisation methods. Each method needs t samples to start iteration and b memory words to pass intermediate results between them. In each iteration, up to q operations can be evaluated in parallel. The numbers in parentheses are related to SSE implementations of the CDF 9/7 transform.

Table 2 shows execution times of the algorithms measured up to 55 millions of samples. The vertical and diagonal vectorisation were implemented using mostly SSE intrinsics and in specific cases inline assembly (to have more control over the register allocation). Theoretical speedup in both cases is $4\times$ due to using of the SSE instructions. Note, that the speedup in Table 2 is shown relatively to the implementation of the horizontal vectorisation (first row) which is significantly slowed down due to extensive cache misses. This is the reason why there is a value higher than the theoretical speedup $4\times$. Figure 11 and Figure 10 shows that the vertical and diagonal vectorisation exhibit similar properties. In these two cases, the actual speedup approaching the theoretical one. This is specifically apparent for lengths of an input vector bigger than the CPU cache size.

vectorisation	Intel		AMD	
	ns/sample	speedup	ns/sample	speedup
horizontal	15.8	1.0	16.4	1.0
vertical	**3.6**	**4.4**	5.4	3.0
diagonal	5.1	3.1	**5.3**	**3.1**

Table 2. Execution times per sample measured for 55 millions of samples. All times are related to SSE implementations of the CDF 9/7 transform.

5 Conclusion

This paper presented a novel method of minimum memory discrete wavelet transform utilizing SIMD instructions. The proposed method was compared to two other approaches – the naive implementation and similar vectorisation introduced in [7]. The achieved speedup goes asymptotically to 3.1× on the Intel Core2 Duo CPU and 3.1× on the AMD Athlon 64 X2 CPU. This speedup was achieved for CDF 9/7 wavelet using SSE instruction set. The data flow graph of this transform consists of 4 elementary lifting operations which exactly fits into 4-fold SSE registers in the proposed diagonal vectorisation. Moreover, the proposed method requires only two new samples to start iteration of SIMD vectorised loop.

Our next research will focus to an adaptation of the proposed approach to the 2-D wavelet transform. Specifically, we will use the diagonal vectorisation in the single-loop approach proposed by R. Kutil in [6]. The other direction may be an adaptation to other architectures, e.g. ARM processors or FPGA-based systems. It would also be interesting to compare the single-precision floating-point implementation with fixed-point implementations or implementations of reversible integer-to-integer (ITI) wavelet transform [1].

Acknowledgements This work has been supported by the EU FP7-ARTEMIS project IMPART (grant no. 316564) and the national Technology Agency of the Czech Republic project RODOS (no. TE01020155).

References

1. Adams, M.D.: Reversible integer-to-integer wavelet transforms for image coding. Ph.D. thesis, Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada (September 2002)
2. Chrysafis, C., Ortega, A.: Line-based, reduced memory, wavelet image compression. IEEE Transactions on Image Processing 9(3), 378–389 (2000)
3. Chrysafis, C., Ortega, A.: Minimum memory implementations of the lifting scheme. In: Proceedings of SPIE, Wavelet Applications in Signal and Image Processing VIII. SPIE, vol. 4119, pp. 313–324 (2000)
4. Cohen, A., Daubechies, I., Feauveau, J.C.: Biorthogonal bases of compactly supported wavelets. Communications on Pure and Applied Mathematics 45(5), 485–560 (1992)
5. Daubechies, I., Sweldens, W.: Factoring wavelet transforms into lifting steps. Journal of Fourier Analysis and Applications 4(3), 247–269 (1998)
6. Kutil, R.: 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). pp. 413–420 (2006)
7. Kutil, R., Eder, P., Watzl, M.: SIMD parallelization of common wavelet filters. In: Parallel Numerics '05. pp. 141–149 (2005)
8. Mallat, S.: A Wavelet Tour of Signal Processing: The Sparse Way. With contributions from Gabriel Peyré. Academic Press, 3 edn. (2009)