Journal of Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Real-Time Wavelet Transform for Infinite Image Strips

David Barina

Received: date / Accepted: date

Abstract This article presents a single-loop approach to a 2-D discrete wavelet transform that allows processing infinitely high image strip-maps. The paper gradually compares several computational strategies to finally show how to deal with a multi-scale wavelet transform of infinite image streams. Besides, the transform is followed by a bit-plane encoder which also processes data in a single loop. The whole machinery is part of a CCSDS 122.0 image codec which manages to process a single pixel in about 33 nanoseconds on a contemporary desktop computer, without the contribution of any parallel computing or SIMD vectorization.

Keywords Discrete wavelet transforms · Image processing · Image compression · CCSDS 122.0

1 Introduction

Perhaps all existing image formats are based on the processing of image frames (frame-based input data), originated, e.g., by CCD sensors. However, at least one format is based on a different principle. The CCSDS 122.0 is able to process infinite strip-based inputs produced by push-broom type sensors. The resolution of the input image is therefore infinite (as the height is infinite). This requires very specific design of the architecture that will be able to handle such kind of data. Specifically, all computations must be performed in memory-effective fashion and in a single pass through the data. This is in sharp contrast to, e.g., the JPEG 2000 format which can only hold images of finite dimensions and buffering of the entire input and output stream is thus allowed.

This short article gradually compares several computational schemes of the two-dimensional multi-scale discrete wavelet transform (DWT), which is the heart of the CCSDS 122.0 format, and eventually answers the question, "What is the best scheme for processing infinite image data?" In more detail, the individual schemes are described in Section 2 of this paper. The same section also describes the CCSDS 122.0 image compression standard. Subsequent Section 3 evaluates the schemes and then selects the most efficient one, which is able to compress input data with the rate of 33 nanoseconds per pixel on a desktop computer. The implementation is single-threaded and does not exploit any SIMD instructions. Finally, Section 5 concludes the paper.

2 Related Work

The CCSDS 122.0 format [1] can hold images of infinite dimensions. However, at least one of its two dimensions must be finite. For this reason, it seems appropriate to consume input data line by line (line-based consumption). Similarly to the JPEG 2000 format, the format can compress the image either in a lossy or lossless manner. The format can also hold high-bit-depth images (25 bits for lossless compression, 28 bits for lossy one). Note that the implementation used in this paper can handle 16-bit pixels and internally uses 32-bit machine words. The format is based on a three-level discrete wavelet transform (see Fig. 1 for better mental picture). The transform is computed using either real or integer numbers. The real transform is intended for lossy compression. Conversely, the integer transform is intended for lossless compression. A compressed image is divided

Centre of Excellence IT4Innovations
Faculty of Information Technology
Brno University of Technology
Bozetechova 1/2, Brno, Czech Republic
E-mail: ibarina@fit.vutbr.cz

Fig. 1 Illustration of the three-level DWT (on the right) on frequently-used Lenna image (on the left).

into segments and blocks, so the data error is not propagated much. The blocks and segments must be formed on the fly to comply with the single-loop requirement. The format covers only the processing of a single image component. The processing of individual multispectral components is covered by another standard.

In the beginning, the input image is extended so that its dimensions are multiples of eight. Three levels (or scales) of the discrete wavelet transform are then calculated. Interestingly, these three levels produce blocks of 8×8 coefficients with a similar meaning as blocks of 8×8 DCT coefficients in the JPEG format. See Fig. 2 which illustrates the meaning of coefficients within a block. These coefficients are further encoded by bit-planes from the most significant to the least significant one. When all bit-planes of integer wavelet transform are encoded, we got a lossless compression. The CCSDS standard uses the CDF 9/7 wavelet [2] for both—the real DWT intended for lossy compression as well as the integer-to-integer DWT for lossless processing. The JPEG 2000, on the contrary, uses the CDF 5/3 wavelet for lossless compression.

Various computational schemes for 1-D and 2-D DWT can be found in the literature. Considering the 1-D transform, one usually starts with the transform defined by two complementary FIR filters. The corresponding computational scheme is referred to as the

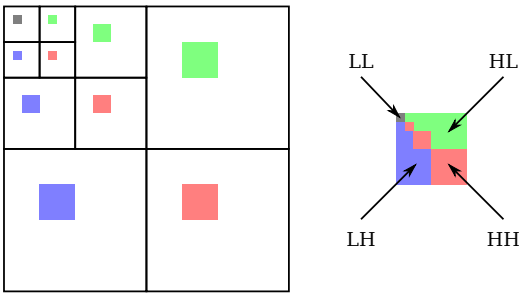

Fig. 2 Illustration of the blocks of 8×8 coefficients: a transformed image on the left, a subset of transform coefficients rearranged into a block on the right. Four wavelet sub-bands are indicated—LL (DC coefficient), HL, LH, and HH.

convolution. In [3], Sweldens and Daubechies showed how the convolution scheme can be decomposed into a sequence of simple filtering steps. These steps are known as the lifting steps and the scheme as the lifting scheme. The lifting scheme asymptotically reduces the number of arithmetic operations by a factor of two. It is also used for the definition of integer-to-integer transforms (lossless compression). Since the lifting scheme comprises a sequence of filtering steps, it is tempting to implement it as a sequence of passes through the input data (multi-loop approach). This has the advantage of easy treatment of the signal boundaries. The downside is the repeated eviction of intermediate data from the CPU cache [4]. This approach is also not compatible with single-loop data processing required in our case. Merging these several filtering steps into a single pass through the input data forms a single-loop algorithm (pipelined computation) [5]. As one might expect, the advantage is friendliness to the CPU cache. However, the disadvantage is shown in the complicated treatment of signal boundaries. Namely, the state-of-the-art algorithms treat such boundaries in a complicated and inflexible way, using special prolog or epilog phases. In [6], these algorithms are extended to perform the treatment using a compact streaming core, possibly in multi-scale fashion. As a result, every input sample is visited only once, while the results are produced immediately, i.e. without buffering. This fits perfectly with our single-loop approach.

A two-dimensional DWT is defined [7] as a tensor-product of two one-dimensional transforms—one for rows and one for columns. Various computational schemes for 2-D DWT can be found in the literature as well. These schemes include the loop fission (splits the vertical loop so it accesses at most as many rows as the cache associativity) [8, 9], aggregation (adjacent columns are filtered concurrently) [10, 11], usage of complicated memory layouts [12–14], interleaving of the vertical and horizontal loop [15–19], SIMD vectorization and parallelization [14, 15, 19], etc. Basically, naive implementations implement this transform using two passes through input data (e.g., OpenJPEG codec). Such a solution is simple but very slow (repeated eviction of intermediate data from the CPU cache), especially for large images. In [19], Kutil presents a single-loop approach to 2-D DWT, i.e. interleaving horizontal and vertical filtering steps. As a result, entire 2-D DWT is computed using the lifting in a single pass through the input data. This approach does not suffer from any cache-related or other issues, except for a very complicated treatment of image boundaries. As a consequence, the author uses nine transform phases for all combinations of horizontal and vertical filterings. This makes the coding arduous

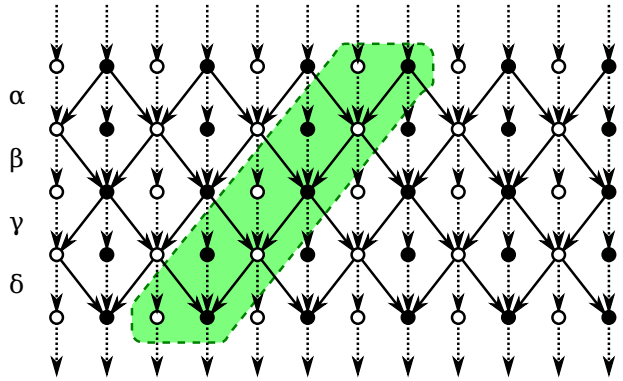

Fig. 3 CDF 9/7 lifting scheme comprising 4 lifting steps (identified as α–δ). The highlighted area is evaluated in a single iteration of the loop (thus a single-loop approach).

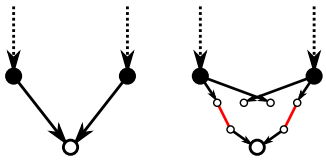

Fig. 4 Detail of the 2-tap FIR filter from the CDF 9/7 data-flow diagram (on the left) and border treatment using the method presented in [6] (on the right). The switches (in red) are set according to the position in the input signal.

and the code very complicated. Combining the Kutil's approach with the approach presented in [6] creates a true single-loop approach, i.e. without any shortcomings. The last unresolved problem regarding the 2-D DWT is the computation of multi-scale decomposition, which is difficult to compute in a single pass due to buffering required to start the next level of transform. This step has not even been done in [19]. However, such single-loop multi-scale processing is necessary for processing infinite images and is, therefore, the subject of the rest of this article.

3 Single-Loop Approach

At the beginning, let us take a closer look at the treatment of signal boundaries presented in [6]. To better understand this algorithm, look at Figure 3, which shows a data-flow diagram of one DWT level with the CDF 9/7 wavelet implemented in a floating-point format. The highlighted area is a core, which consumes the input signal from left to right (in a single loop) and produces output coefficients. The problem with this approach is the need to buffer the input signal, at least at its beginning and end. The data-flow diagram in Figure 3 consists of filtering using 2-tap FIR filters. This FIR filter is shown on the left side of Figure 4. In [6], such a single-loop approach is modified in such a way that the input signal does not need to be buffered. Instead, the

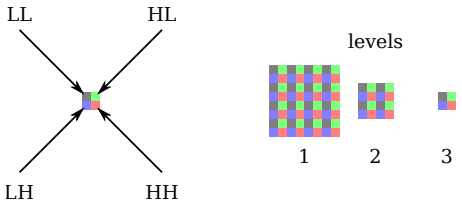

Fig. 5 Quadruple (quad) of LL, HL, LH, and HH coefficients (on the left) and three-level interleaving of subsequent scales using these quads (on the right).

filtering by the 2-tap FIR filter is adaptively modified so that it never accesses an undefined part of the input signal. This is essentially achieved through two switches which are set so that the result of the calculation corresponds to a symmetric extension of the signal. The modification is detailed in the right part of Figure 4. Since this algorithm does not require any input buffering, it can be used directly for multi-scale processing. In this case, the subsequent levels of DWT are simply triggered interlaced, without having to wait for a larger block of input data.

Because we chose line-based consumption of input data and because the CCSDS standard internally divides the image into 8×8 pixel blocks, it makes sense to implement a multi-scale transform either (1a) sequentially (which would prohibit processing infinite image strips), or (1b) by interleaving individual levels of the transform in 8-pixel high strips, or (1c) by interleaving individual levels in blocks 8×8 pixels. Similar possibilities arise in the implementation of a single level of the 2-D DWT. The two-dimensional transform can be implemented either (2a) in a separable fashion, (2b) using line-based processing, or (2c) using the true single-loop approach (referred to herein as quad-based, since the smallest unit is a quadruple of LL, HL, LH, and HH coefficients, see Fig. 5). The underlying one-dimensional transform can be computed using (3a) single-loop convolution, (3b) multi-loop lifting, or (3c) single-loop lifting scheme. Because of the two compression modes (lossy and lossless), all this has to be implemented twice—once in integer arithmetic and a second time using a floating-point format. And finally, all this above has to be implemented both in the encoder and in the decoder. It brings together $3 \times 3 \times 3 \times 2 \times 2$ different implementations, but not all of them make sense. However, we have implemented and evaluated all the meaningful ones (the implementation used in this article is highly configurable). By removing meaningless combinations out of this number, the following options remain (from the most naive one to the most tuned one): (i)

Algorithm 1 (i) separable convolution, sequential

```
for each scale do
    for each row do
        for each sample do
            compute convolution coefficient
        end for
    end for
    for each column do
        for each sample do
            compute convolution coefficient
        end for
    end for
end for
```

Algorithm 2 (ii) separable multi-loop, sequential

```
for each scale do
    for each row do
        for each lifting step do
            for each second sample do
                compute lifting step
            end for
        end for
    end for
    for each column do
        for each lifting step do
            for each second sample do
                compute lifting step
            end for
        end for
    end for
end for
```

Algorithm 3 (iii) separable single-loop, sequential

```
for each scale do
    for each row do
        for each two samples do
            for each lifting step do
                compute lifting step
            end for
        end for
    end for
    for each column do
        for each two samples do
            for each lifting step do
                compute lifting step
            end for
        end for
    end for
end for
```

Algorithm 4 (iv) line-based, sequential

```
for each scale do
    for each two rows do
        compute horizontal transforms
        for each column do
            for each lifting step do
                compute vertical lifting step
            end for
        end for
    end for
end for
```

Algorithm 5 (v) quad-based, sequential

```
for each scale do
    for each 2 × 2 quad in raster scan do
        for each lifting step do
            compute vertical lifting step
        end for
        for each lifting step do
            compute horizontal lifting step
        end for
    end for
end for
```

Algorithm 6 (vi) quad-based, strips interleaved

```
for each horizontal strip do
    for each scale do
        for each 2 × 2 quad do
            for each lifting step do
                compute vertical lifting step
            end for
            for each lifting step do
                compute horizontal lifting step
            end for
        end for
    end for
end for
```

Algorithm 7 (vii) quad-based, blocks interleaved

```
for each block in raster scan do
    for each scale do
        for each 2 × 2 quad do
            for each lifting step do
                compute vertical lifting step
            end for
            for each lifting step do
                compute horizontal lifting step
            end for
        end for
    end for
end for
```

convolution, horizontal/vertical transforms separated, scales sequentially; (ii) multi-loop lifting, transforms separated, scales sequentially; (iii) single-loop lifting, transforms separated, scales sequentially; (iv) line-based two-dimensional lifting, scales sequentially; (v) quad-based two-dimensional lifting, scales sequentially; (vi) quad-based two-dimensional lifting, scales interleaved using strips; and (vii) quad-based two-dimensional lifting, scales interleaved using blocks. The interleaving of subsequent scales in (vi) and (vii) is detailed in Fig. 5. One has to transform 4×4 quads at the first levels. The resulting LL coefficients are then fed into a second-level transform which transforms 2×2 quads. Finally, the four resulting LL coefficients are fed into a third-level transform which computes final results. All these implementations have been implemented using integer as well as floating-point numbers, and also on the encoder as well as decoder side. To allow the reader to compare

the individual schemes (i)–(vii), these are presented by the pseudocode in Algorithms 1–7. The strip means 8 image lines, and the block means 8×8 pixels.

The implementation (1a) entails no additional memory requirements. The implementations (1b) and (1c) require additional buffers proportional to the image width (the finite dimension) and number of scales. Similarly, the case (2a) does not have any extra memory requirements, whereas the (2b) and (2c) require for each scale a buffer of the size proportional to the image width. Finally, the (3a) and (3b) implementations require extra space of the same size as the image width (cannot be computed in-place). The reason why also the (3b) requires the extra space is the usage of a different data type for intermediate results. Finally, the (3c) operates in-place and thus does not require any extra memory.

4 Results

To illustrate the differences, the performance of the three-level floating-point discrete wavelet transform was evaluated on an x86-64 machine. The code does not exploit any SIMD extensions or parallel processing. The results are shown in Fig. 6. The x-axis shows the image resolution (in megapixels), the y-axis is the processing time per pixel (in nanoseconds/pixel). The evaluation was performed in three different scenarios—for 4:3 aspect ratio, for 16:9 ratio, and for 1024-pixel wide strips (infinite strip-based data). This is because these aspect ratios are mapped differently to the CPU cache due to its limited associativity. The results were obtained on the AMD Ryzen Threadripper 2990WX 32-Core Processor (64 MiB L3 cache, 128 GiB DDR4 @ 2933 MHz). The CPU clock was oscillating around 3.8 GHz. One might find that the (v) quad-based single-loop lifting with sequential scale processing shows the highest performance. It achieves asymptotically the time about 12.15 nanoseconds per pixel on image strips. However, sequential scale processing precludes the processing of infinite image strips. Focusing on the strip-based scenario, one can find that the (vi) quad-based single-loop lifting with scales interleaved in strips is the most powerful scheme suitable for processing infinite data. However, its performance is worse than in case (v). This is the cost for completely single-pass processing. The time asymptotically reaches 12.95 nanoseconds per pixels. On 16:9 ratio, the asymptotic performance is even better—11.7 nanoseconds per pixel–which corresponds to over 40 frames per second for Full HD resolution. This is real-time processing. Note also that single-loop approaches do not suffer from cache-related issues and copy the linear time complexity of the transform, whereas separable

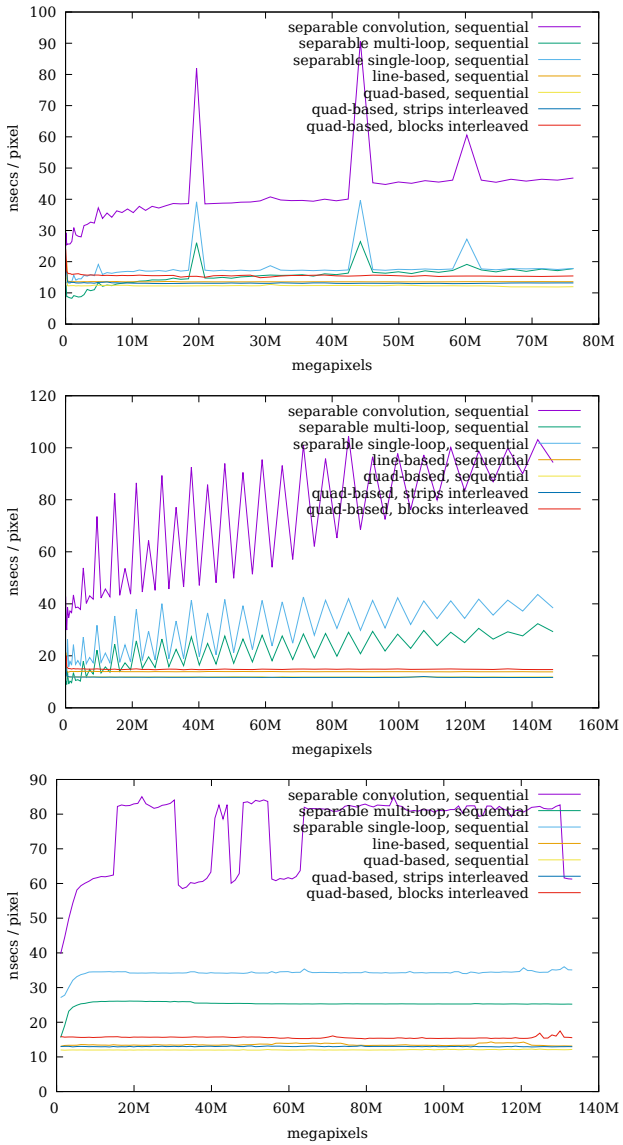

Fig. 6 The performance of the floating-point forward transform. From top: 4:3 aspect ratio, 16:9 ratio, and 1024-pixel wide strips.

horizontal and vertical loops lead to unpleasant performance anomalies. This is especially evident in the 16:9 ratio. The same behavior was observed by Kutil in [19]. Finally, note that the computation of integer transforms is noticeably faster, and decoder-side implementations behave the same way as those on the encoder-side.

Let us also look specifically at the implementation (ii), which is often implemented in open source libraries (e.g., OpenJPEG). This implementation suffers from the performance anomalies that are caused by CPU cache issues. The main issue here is that this implementation repeatedly accesses data that have already been evicted from the cache. Besides, it shows significantly worse per-

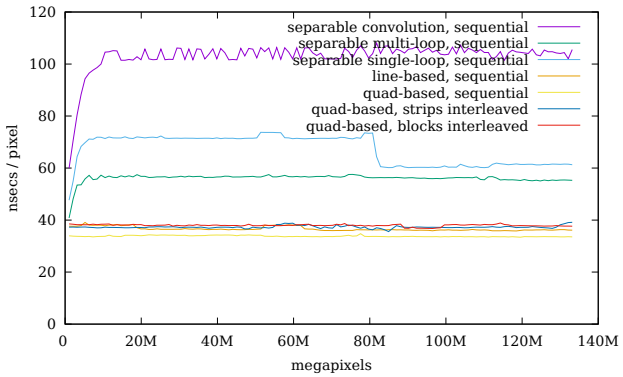

Fig. 7 The performance of the floating-point DWT and BPE chain for 1024-pixel wide strips.

formance than all single-loop approaches and precludes the processing of infinite image data.

The second experiment, shown in Fig. 7, evaluates the performance of the DWT/BPE chain (wavelet transform followed by the bit-plane encoder). The BPE also processes the data in a single loop. However, this processing is more complicated since it handles data in 8×8 blocks, arranges these blocks into segments, and then compresses the segments at once. The evaluation was performed in the same scenarios and the same machine. The results largely copy the results of the previous experiment. This time only the result for 1024-pixel wide strips is shown. Apart from a slowdown corresponding to bit-plane encoding, the findings described in the previous paragraph are still valid. The (vi) quad-based single-loop lifting with scales interleaved using strips manages to process a single pixel in about 33 nanoseconds, which is still enough for real-time processing.

It may be obvious that it is possible to further accelerate the above-evaluated schemes using multi-threading and SIMD vectorization. This step was done for example in [19, 20]. However, such a step only causes shifting the curves in Fig. 6 and 7 down, and does not change the behavior of the schemes.

5 Conclusions

It is possible to compute a multi-scale 2-D discrete wavelet transform of infinite image strips in real time on a contemporary desktop computer without any contribution of parallel processing or SIMD instructions. The key is a single-loop transform approach and simple treatment of image boundaries. The implementation presented in this article manages to transform a single pixel in about 12.95 nanoseconds using floating-point CDF 9/7 transform. The implementation is part of a CCSDS 122.0 image codec which manages to process a single pixel in about 33 nanoseconds, still considering infinite image strips. To allow other developers and scientists to benefit from this work and build on it, the codec used in this paper has been released as open-source software.[1]

I believe that the work presented in this article can find application in other software implementations of the 2-D discrete wavelet transform. It is especially suitable for the implementation of the JPEG 2000 format (e.g., OpenJPEG or FFmpeg).

Acknowledgements This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science (LQ1602) and the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737475.

References

1. CCSDS 1220-B-2 (2007) Image data compression. Recommended standard, Consultative Committee for Space Data Systems
2. Cohen A, Daubechies I, Feauveau JC (1992) Biorthogonal bases of compactly supported wavelets. Communications on Pure and Applied Mathematics 45(5):485–560, DOI 10.1002/cpa.3160450502
3. Daubechies I, Sweldens W (1998) Factoring wavelet transforms into lifting steps. Journal of Fourier Analysis and Applications 4(3):247–269, DOI 10.1007/BF02476026
4. Drepper U (2007) What every programmer should know about memory. Tech. rep., Red Hat
5. Chrysafis C, Ortega A (2000) 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, DOI 10.1117/12.408615
6. Barina D, Zemcik P, Kula M (2016) Simple signal extension method for discrete wavelet transform. In: 2016 IEEE International Conference on Signal and Image Processing (ICSIP), pp 534–538, DOI 10.1109/SIPROCESS.2016.7888319
7. Mallat S (1989) A theory for multiresolution signal decomposition: the wavelet representation. IEEE Transactions on Pattern Analysis and Machine Intelligence 11(7):674–693, DOI 10.1109/34.192463
8. Shahbahrami A, Juurlink B, Vassiliadis S (2006) Improving the memory behavior of vertical filtering in the discrete wavelet transform. In: Proceedings of the 3rd conference on Computing frontiers (CF), ACM, pp 253–260, DOI 10.1145/1128022.1128056

[1] `https://bitbucket.org/ibarina/ccsds/`

9. Tao J, Shahbahrami A (2008) Data locality optimization based on comprehensive knowledge of the cache miss reason: A case study with DWT. In: High Performance Computing and Communications (HPCC), IEEE, pp 304–311, DOI 10.1109/HPCC.2008.7
10. Meerwald P, Norcen R, Uhl A (2002) Cache issues with JPEG2000 wavelet lifting. In: Proceedings of 2002 Visual Communications and Image Processing (VCIP), SPIE, vol 4671, pp 626–634
11. Chaver D, Tenllado C, Pinuel L, Prieto M, Tirado F (2003) Vectorization of the 2D wavelet lifting transform using SIMD extensions. In: Proceedings. International Parallel and Distributed Processing Symposium (IPDPS), DOI 10.1109/IPDPS.2003.1213416
12. Chatterjee S, Jain VV, Lebeck AR, Mundhra S, Thottethodi M (1999) Nonlinear array layouts for hierarchical memory systems. In: Proceedings of the 13th International Conference on Supercomputing (ICS), ACM, pp 444–453, DOI 10.1145/305138.305231
13. Chatterjee S, Brooks CD (2002) Cache-efficient wavelet lifting in JPEG 2000. In: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), IEEE, vol 1, pp 797–800, DOI 10.1109/ICME.2002.1035902
14. Chaver D, Tenllado C, Pinuel L, Prieto M, Tirado F (2003) Wavelet transform for large scale image processing on modern microprocessors. In: Palma JMLM, Sousa AA, Dongarra J, Hernandez V (eds) High Performance Computing for Computational Science (VECPAR), Lecture Notes in Computer Science (LNCS), vol 2565, Springer, pp 549–562, DOI 10.1007/3-540-36569-9_37
15. Chaver D, Tenllado C, Pinuel L, Prieto M, Tirado F (2002) 2-D wavelet transform enhancement on general-purpose microprocessors: Memory hierarchy and SIMD parallelism exploitation. In: Sahni S, Prasanna VK, Shukla U (eds) High Performance Computing (HiPC), Lecture Notes in Computer Science, vol 2552, Springer, pp 9–21, DOI 10.1007/3-540-36265-7_2
16. Shahbahrami A, Juurlink B (2007) A comparison of two SIMD implementations of the 2D discrete wavelet transform. In: Annual Workshop on Circuits, Systems and Signal Processing, pp 169–177
17. Chrysafis C, Ortega A (2000) Line-based, reduced memory, wavelet image compression. Transactions on Image Processing 9(3):378–389, DOI 10.1109/83.826776
18. Oliver J, Oliver E, Malumbres MP (2005) On the efficient memory usage in the lifting scheme for the two-dimensional wavelet transform computation. In: International Conference on Image Processing (ICIP), IEEE, vol 1, pp I–485–8, DOI 10.1109/ICIP.2005.1529793
19. Kutil R (2006) 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, DOI 10.1109/PDP.2006.14
20. Barina D, Zemcik P (2015) Vectorization and parallelization of 2-D wavelet lifting. Journal of Real-Time Image Processing 15(2):349–361, DOI 10.1007/s11554-015-0486-6