Vectorized Computing for Edge-Avoiding Wavelet

Yuto Sumiya, Hirokazu Kamei, Kazuya Ishikawa, and Norishige Fukushima

Nagoya Institute of Technology, Nagoya, Japan

ABSTRACT

The discrete wavelet transform (DWT) is essential for image and signal processing. The edge-avoiding wavelet (EAW) is an extension for DWT to have edge-preserving property. EAW constructs a basis based on the edge content of input images; thus, the wavelet contains nonlinear filtering. DWT is computationally efficient processing in the scale-space analysis; however, EAW has a complex loop structure. Therefore, parallel computing for EAW is not an easy task. This paper vectorizes EAW computing using single instruction, multiple data (SIMD) parallelization. Significantly, the lifting-based wavelet allows the in-place operation, i.e., the source and destination array for DWT can be shared, and the in-place operation improves cache efficiency. However, the EAW prevents the operation in the update processing. Moreover, data interleaving for wavelet computing is the bottleneck for SIMD computing. Therefore, we show the suitable data structure for effective SIMD vectorization for EAW. Experimental results show that our effective implementation accelerates EAW. For the WCDF method, we accelerate more than 2 times faster, and for the WRB method, we accelerate about 3 times faster than the simple implementation.

Keywords: Edge-avoiding wavelet, edge-preserving filtering, discrete wavelet, transform, SIMD

1. INTRODUCTION

Manipulating images with multi-scale analysis is a challenging task. For the multi-scale processing, an input image is decomposed into multiple layers by decomposition methods, such as Gaussian and Laplacian pyramids,[1] wavelet transform,[2] and difference of Gaussians (DoG) form scale-space analysis.[3] In the multi-scale processing, coefficients in the pyramid, wavelet transform, or scale-space are manipulated, and then the pyramid is collapsed to obtain an output image.

Classic approaches are not considering image edges, while current methods utilize edge information for multi-scale processing. Instead of using linear filtering, bilateral filtering[4] is used for two-scale processing.[5] Iterative filtering represents multiple layers processing.[6] The Laplacian pyramid[1] is extended to an edge-aware pyramid and is utilized for an edge-preserving filter, named local Laplacian filter (LLF).[7]

For each approach, acceleration is the main problem. For bilateral filtering, various approaches are proposed for accelerating gray[8,9] and color bilateral filtering.[10–12] The multi-scale processing is accelerated[13] by À-Trous wavelet method. The acceleration of the local Laplacian filter is also proposed.[14,15]

Edge-avoiding wavelet (EAW)[16] is an edge-preserving extension of the discrete usual wavelet transform (DWT) for detail manipulation and is one of the fastest edge-preserving filters. EAW constructs a basis based on the edge content of the images; thus, the wavelet becomes nonlinear filtering. EAW has a complex loop structure and data structure; thus, parallel computing is not easy. The lifting-based wavelet allows the in-place operation with the suitable data structure, i.e., the source and destination array for the DWT can be shared. The in-place operation improves cache efficiency; however, EAW prevents the operation in its update processing. Moreover, data interleaving for wavelet computing is the bottleneck for single instruction, multiple data (SIMD) vectorization. For an example of the bilateral filtering case, the vectorization is efficient for extracting the performance.[17,18]

This paper proposed an efficient data structure for computing EAW by SIMD vectorization. The contribution of this paper is as follow:

Further author information: (Send correspondence to N. Fukushima) Web: https://fukushima.web.nitech.ac.jp/en/. This work was supported by JSPS KAKENHI (21H03465).

- *weight sequentialization*: optimizing the weight map structure in EAW for efficient data loading and storing.
- *semi-reordered data structure*: optimizing data swizzling by minimizing the number of instructions of hardware data moving operation.

2. EDGE-AVOIDING WAVELET

EAW is a type of second-generation wavelet, which has a lifting scheme. The lifting scheme splits signals into even and odd parts and predicts the odd signal from the even signal. This paper handle Cohen–Daubechies–Feauveau (CDF) (2,2) wavelet and Red-Black wavelets, which are extended in EAW.[16]

2.1 Weighted Cohen-Daubechies-Feauveau Wavelets

We introduce weighted CBF (WCDF) wavelets. First, we define the prediction in WCDF. Let the prediction operator be $\mathcal{P}$, the input be $a_p^j, p = x, y$, and the weights be $w_n^j[m]$. $\mathcal{P}$ can be defined as follows:

$$\mathcal{P}(a_C^j)[x,y] = \frac{w_{x,y}^j[x-1,y]a_C^j[x-1,y] + w_{x,y}^j[x+1,y]a_C^j[x+1,y]}{w_{x,y}^j[x-1,y] + w_{x,y}^j[x+1,y]}, \quad w_n^j[m] = \exp\left(\frac{-(a^j[n]-a^j[m])^2}{\sigma^2}\right) \tag{1}$$

where $C = \{(x,y)|\, x \in \text{even}\}, F = \{(x,y)|\, x \in \text{odd}\}$, and j denotes the level. It corresponds to the Fig. 1(a). The next-level detail coefficients at the fine points in F are computed by

$$d^{j+1} = a_F^j - \mathcal{P}(a_C^j). \tag{2}$$

Next, we define the update formula. The coarse variable a_C^j is usually not used as an approximation factor for the next level since it corresponds to a naïve subsampling of the original data and suffers from severe aliasing. An update operator $\mathcal{U}$ is utilized to avoid the problem and it is defined as follows:

$$\mathcal{U}(d^{j+1})[x,y] = \frac{w_{x,y}^j[x-1,y]d^{j+1}[x-1,y] + w_{x,y}^j[x+1,y]d^{j+1}[x+1,y]}{2(w_{x,y}^j[x-1,y] + w_{x,y}^j[x+1,y])}, \tag{3}$$

where $(x,y) \in C$. In the update stage, the 4 predicted pixels are referenced: above, below, left, and right. The coefficients of the next-level approximation are computed by:

$$a^{j+1} = a_C^j + \mathcal{U}(d^{j+1}). \tag{4}$$

Note that we apply this predict and update formula in the y-axis direction as well (Fig. 1(b)). Since the form of the equation is almost the same as x version, we omit the description. Also we predict the lower right pixel based on their four diagonally-nearest neighbors (Fig. 1(c)).

$$\mathcal{P}(a_{C'}^j)[x,y] = \frac{\sum_{x',y' \in N_{x,y}} w_{x,y}^j[x',y']a_{C'}^j[x',y']}{2\sum_{x',y' \in N_{x,y}} w_{x,y}^j[x',y']}. \tag{5}$$

Here, $F' = \{(x,y)|\, x{+}y \text{ even, and } x,y \text{ odd}\}$, $C' = \{(x,y)|\, x{+}y \text{ even, and } x,y \text{ even}\}$, and $N_{x,y} = \{(x{+}1,y{+}1),(x{-}1,y{+}1),(x{+}1,y{-}1),(x{-}1,y{-}1)\}$. Based on this prediction, we compute $d^{j+1} = a_{F'}^j - \mathcal{P}(a_{C'}^j)$.

2.2 Weighted Red-Black Wavelets

We explain the weighted Red-Black (WRB) wavelet, which also predicts and updates pixels. The prediction formula is defined as follows:

$$\mathcal{P}_{\text{red}}(a_C^j)[x,y] = \frac{\sum_{x',y' \in N_{x,y}} w_{x,y}^j[x',y']a_C^j[x',y']}{2\sum_{x',y' \in N_{x,y}} w_{x,y}^j[x',y']}, \tag{6}$$

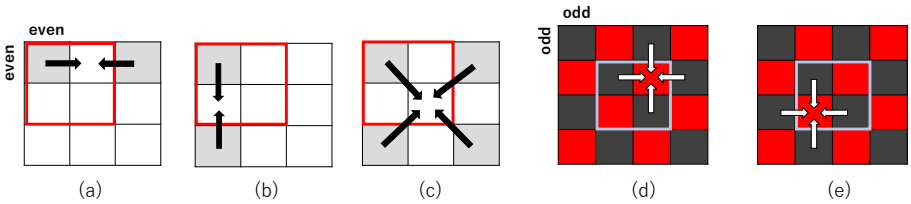


Figure 1. The prediction of WCDF and WRB.

where $C = \{(x,y)|\, x+y \ \in \text{even}\}$(black), $F = \{(x,y)|\, x+y \in \text{odd}\}$(red), and $N_{x,y} = \{(x+1,y),(x-1,y),(x,y-1),(x,y+1)\}$. It corresponds Fig. 1(d), (e). The next-level detail coefficient d^{j+1} is computed by eq. (2).

The update operator is also defined by the four nearest fine points of the coarse point:

$$\mathcal{U}_{\text{red}}(d^{j+1})[x,y] = \frac{\sum_{x',y' \in N_{x,y}} w^j_{x,y}[x',y']d^{j+1}[x',y']}{2\sum_{x',y' \in N_{x,y}} w^j_{x,y}[x',y']}. \tag{7}$$

Here, every $(x,y) \in C$ in eq. (6). The next-level approximation coefficient a^{j+1} is computed by Eq. (4)

The prediction for the lower-right pixel is exactly the same as for WCDF (Fig. 1(c)). It is computed by eq. (5). Finally, update the upper-left black pixel in Fig. 1 (d), (e); $(x,y) \in C' = \{(x,y)|\, x+y$ even, and x, y even$\}$. The update operator $\mathcal{U}_{\text{black}}$ averages at every pixel C' using its four diagonally nearest d^{j+1} in F' by eq. (7), and the next-level approximation coefficient a^{j+1} is computed by eq. (4).

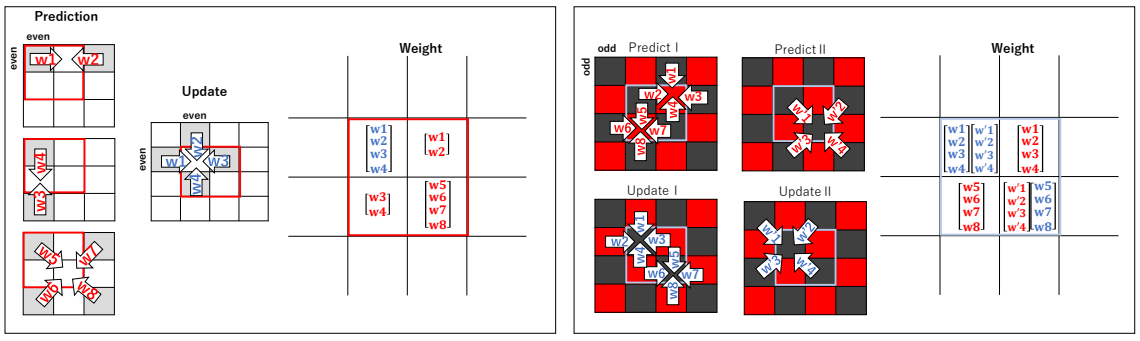
Note that in both methods, by applying these steps in the reverse order and replacing additions with subtractions and vice verse, the perfect-reconstructing inverse transformation is obtained in both methods.

3. VECTORIZATION OF EDGE-AVOIDING WAVLET

In SIMD computing, the packed data in registers is computed parallel with an arithmetic instruction, e.g., 8 elements for AVX. However, wavelets require different arithmetic instructions for even samples and odd samples. Usual operations called vertical arithmetics do not support the instructions. For this case, there are three approaches: 1) use special instructions called horizontal arithmetics to mix instructions for operations between elements, 2) perform the operations for even numbers and odd numbers twice, and mix the results. 3) sort the order of the data into even and odd lanes, and compute the results in each lane.

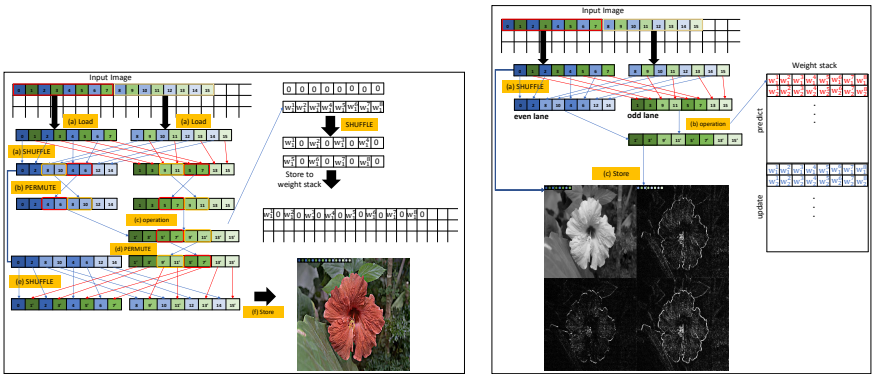
Among these methods, method-1 of horizontal arithmetics is slower than vertical ones. method-2 requires double processing time (but is usually utilized in GPU computing); thus, method-3 has the highest computational efficiency. However, the computation of the sort in method-3 is an offset. In this paper, we propose two methods that minimize the offset time: weight sequentialization and semi-reordered data structure.

First, we introduce weight sequentialization. EAW required weight data for wavelet transformation, and the weight is required at the invert transformation. Therefore, we must store the weight data. The size of



(a) WCDF (b) WRB

Figure 2. Weight data mapping to image data structure.



(a) conventional (b) proposed

Figure 3. Forward-transform flow and its data structure.

the weight map is $3N$ and $6N$ for WCDF and WRB, respectively, where N is the input image size. The weight data is interleaved in naïve C++ implementation for pixel-by-pixel, i.e., mapping to image structure (See Fig. 2); however, the interleaving is cost-consuming for vectorized computing. Fortunately, forward and inverse transformations have the same loop structure; thus, the timing of the requested data is also the same.

Therefore, we map the weight data to computing pipelines. We use the stack data structure for containing the weight sequentially. The computed weight map is pushed in the stack when we compute the forward predicting stage. Next, we compute the forward updating stage; then, the computed weight map is pushed in the stack. The computed weight map is popped in the stack when we compute the invert predicting stage. Next, we compute the forward updating stage; then, the computed weight map is popped in the stack. The flow completely removes the reordering of the weight map, and the data load/store becomes sequential. The sequentialization improves the processing speed.

Next, we introduce a semi-reordered data structure. Here, Fig. 3 shows the forward-transform flow and its data structure of conventional and our proposed method. The reordering in SIMD can be done in any order by using the SET instruction, which reorders the elements one by one and is the simplest way to reorder even or odd-numbered lines. However, this instruction does not have parallelism, and the offset time becomes significant.

There is an instruction category called SWIZZLE that reorder elements in parallel. SWIZZLE is implemented by hardware wiring between SIMD registers, and register elements can be swapped between the wired registers. However, suppose the number of elements is $\mathcal{E}$. In that case, the number of registers is $\mathcal{R}$, in order to replace any element, $\mathcal{E} \times \mathcal{E} \times \mathcal{R} \times \mathcal{R}$ connections are required, and the circuit size is enormous. Therefore, the number of connections is limited, and parallel and high-speed SWIZZLE is realized by combining several instructions.

The most lightweight SWIZZLE instruction is SHUFFLE, which combines two registers and sorts them within some moving range. This can be used to retrieve even and odd lanes, but the ordering is not complete due to the restriction moving. For example, the order of the element numbers of even lane in Fig. 3 is (0, 2, 8, 10, 4, 6, 12, 14). To reorder them in the ascending order, i.e., (0, 2, 4, 6, 8, 10, 12, 14), we need another instruction called PERMUTE, which is less restrictive but slower than SHUFFLE. Note that we do not need the last sort since it does not matter where the intermediate coefficients are stored or not as long as the final output is aligned. The same can be said for the order of the weights. The order of the weights which are pushed differs from Fig. 2, but it also does not affect the output if the weights are popped from the top of the stack and the pixel positions and the weights are correctly matched. In addition, the forward and inverse transform need to be reordered into even and odd lines, but if we do not issue an instruction to return the order, we can further reduce the number of useless instructions and reduce the offset. The straightforward procedure is as follows.

1. Forward-transform/Inverse-transform (same flow)
   (a) Load two registers and SHUFFLE them to split even and odd lanes.
   (b) Sort the even and odd lanes in the ascending order by PERMUTE

(a) Input image (b) Output image (c) CDF wavelet

Figure 4. Experimental images and wavelet coefficient of CDF.

(c) Perform arithmetic operation between registers.
(d) PERMUTE the resulting even and odd lanes to reorder for SHUFFLE.
(e) SHUFFLE the registers to interleave them
(f) Store the registers to the destination image.

The proposed method can remove most processing and storing data structure is changed:

1. Forward-transform
   (a) Load two registers and SHUFFLE the registers to split even and odd lanes.
   (b) Perform arithmetic operation between registers.
   (c) Store resulting data to sub-images.
2. Inverse-transform.
   (a) Load two registers for the sub-images.
   (b) Perform arithmetic operation between registers.
   (c) PERMUTE only the resulting data of DC component and then store to sub-images.

4. EXPERIMENTAL RESULTS

We compared the processing time of our proposed implementation with that of a simple implementation. The code was written in C++ and vectorized by AVX/AVX2. The test CPU was AMD Ryzen Threadripper 3970X (32 cores 3.7GHz). The test image size was 512×512.

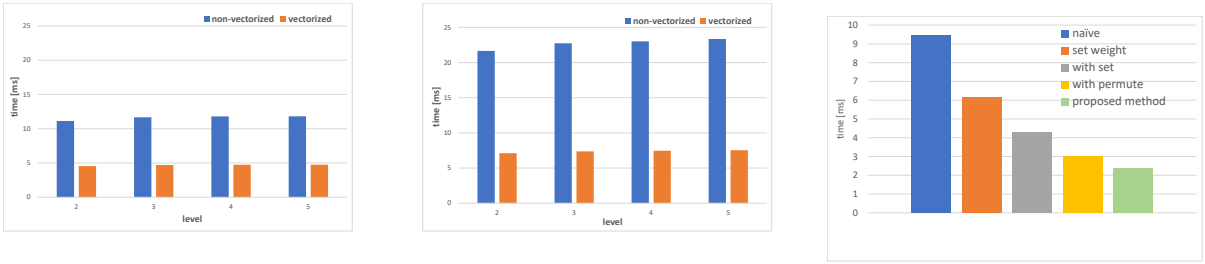
Figure 4 shows the output images and wavelet coefficients of WCDF. Compared with the input image, we can see that the details are enhanced.

Figure 5(a), 5(b) compare the processing time for each level of the WRB and WCDF methods with and without the vectorization implementation. For WCDF, the processing time is accelerated more than two times, and for WRB, it accelerated about three times. The processing time does not change much as the level increases because the processing cost at the next level is only a quarter of the previous level.

Figure 5(c) shows the processing time of each CDF implementation. We compared 5 implementations. They are "naïve" (this is naïve EAW), "set weight" (it uses the set instruction and sorts pixels and weights), "with set" (it uses the set instruction and sorts only the pixels), "with permute" (it uses the shuffle instruction and sorts pixels and weights), and "proposed method" (remove all unnecessary shuffle and permute instructions). The figure shows that the set instruction is computationally more expensive than SWIZZLE. It also shows that the proposed method without extra reordering is faster than the other methods.

5. CONCLUSION

We proposed an efficient data structure for the vectorized implementation of EAW: weight sequentialization and semi-reordered data structure. The experimental results show that WCDF and WRB methods are faster than the simple implementation.



(a) WCDF (b) WRB (c) WCDF (level 5) each imp.

Figure 5. (a) and (b)Processing time of each method and level. (c) one of each CDF implementation.

REFERENCES

[1] Adelson, E. H., Anderson, C. H., Bergen, J. R., Burt, P. J., and Ogden, J. M., "Pyramid methods in image processing," *RCA engineer* **29**(6), 33–41 (1984).

[2] Mallat, S., *A wavelet tour of signal processing*, Elsevier (1999).

[3] Lindeberg, T., *Scale-space theory in computer vision*, vol. 256, Springer Science & Business Media (2013).

[4] Tomasi, C. and Manduchi, R., "Bilateral filtering for gray and color images," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 839–846 (1998).

[5] Durand, F. and Dorsey, J., "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. on Graphics* **21**(3), 257–266 (2002).

[6] Fattal, R., Agrawala, M., and Rusinkiewicz, S., "Multiscale shape and detail enhancement from multi-light image collections.," *ACM Transactions on Graphics* **26**(3), 51 (2007).

[7] Paris, S., Hasinoff, W., and Kautz, J., "Local laplacian filters: Edge-aware image processing with a laplacian pyramid," *ACM Trans. on Graphics* **30**(4) (2011).

[8] Sugimoto, K., Fukushima, N., and Kamata, S., "200 fps constant-time bilateral filter using svd and tiling strategy," in *Proc. IEEE International Conference on Image Processing (ICIP)*, (2019).

[9] Sumiya, Y., Fukushima, N., Sugimoto, K., and Kamata, S., "Extending compressive bilateral filtering for arbitrary range kernel," in *Proc. IEEE International Conference on Image Processing (ICIP)*, (2020).

[10] Fukushima, N., Tsubokawa, T., and Maeda, Y., "Vector addressing for non-sequential sampling in fir image filtering," in *IEEE International Conference on Image Processing (ICIP)*, (2019).

[11] Miyamura, T., Fukushima, N., Waqas, M., Sugimoto, K., and Kamata, S., "Image tiling for clustering to improve stability of constant-time color bilateral filtering," in *Proc. International Conference on Image Processing (ICIP)*, (2020).

[12] Oishi, S. and Fukushima, N., "Clustering-based acceleration for high-dimensional gaussian filtering," in *Proc. Signal Processing and Multimedia Applications (SIGMAP)*, (2021).

[13] Dammertz, H., Sewtz, D., Hanika, J., and Lensch, H. P. A., "Edge-Avoiding A-Trous Wavelet Transform for fast Global Illumination Filtering," in *Proc. ACM SIGGRAPH/EUROGRAPHICS Conference on High Performance Graphics*, The Eurographics Association (2010).

[14] Aubry, M., Paris, S., Kautz, J., and Durand, F., "Fast local laplacian filters: Theory and applications," *ACM Trans. on Graphics* **33**(5) (2014).

[15] Sumiya, Y., Otsuka, T., Maeda, Y., and Fukushima, N., "Gaussian fourier pyramid for local laplacian filter," *IEEE Signal Processing Letters* (2021).

[16] Fattal, R., "Edge-avoiding wavelets and their applications," *ACM Transactions on Graphics* **28**(3), 1–10 (2009).

[17] Maeda, Y., Fukushima, N., and Matsuo, H., "Effective implementation of edge-preserving filtering on cpu microarchitectures," *Applied Sciences* **8**(10) (2018).

[18] Takagi, H. and Fukushima, N., "Domain specific description for randomized image convolution," in *Proc. Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, (2021).