Optimization of Sliding-DCT Based Gaussian Filtering for Hardware Accelerator

Tomoki Otsuka*, Norishige Fukushima*, Yoshihiro Maeda†,
Kenjiro Sugimoto‡, and Sei-ichiro Kamata‡

*Nagoya Institute of Technology, Japan, †Tokyo University of Science, Japan, ‡Waseda University, Japan

***Abstract*—Gaussian filtering is a smoothing filter used in various tasks. The main disadvantage is the dependence of the processing time on its kernel radius. One solution is using a sliding-discreet cosine transform (DCT), a constant-time algorithm for the kernel radius, and it provides the best performance in terms of both speed and accuracy. However, the speed and accuracy differ according to the type of DCT used. We can also accelerate the sliding-DCT based Gaussian filter by hardware accelerators, but the acceleration requires modification of the algorithms. In this paper, we focus on the fused multiply-add (FMA) instruction of hardware accelerators in modern computer architectures. The FMA instruction simultaneously performs multiplication and addition, i.e.,$ax+b$. We proposed an acceleration method of the sliding-DCT based Gaussian filtering for the FMA instruction. Moreover, we evaluate the performance of it in terms of computational time and approximation accuracy.**

***Index Terms*—Gaussian filter, sliding DCT, FMA, constant-time Gaussian filter**

I. Introduction

Gaussian filtering (GF) is essential in various image processing applications. For example, pre-filtering for downsampling; feature description, e.g., SIFT [1]; saliency map [2]; internal processing of edge-preserving filters, such as accelerated bilateral filtering [3]–[6] and guided image filtering [7]; high-dimensional Gaussian filtering [8], image quality evaluation indices, such as SSIM [9], [10]; blur removal processing [11]. Hence, accelerating GF is significant in various tasks.

The computational order of GF's straightforward implementation is $O(R^2)$ per pixel, where $R \in \mathbb{N}$ is its kernel radius. The order is the same as the general 2D finite impulse response (FIR) filters. Separable filtering and frequency-domain filtering can accelerate it. The separable filtering reduces the order into $O(R)$. The frequency-domain filtering also reduces the order into $O(\log N)$ per pixel, where $N \in \mathbb{N}$ is the number of image pixels.

We can further accelerate GF by using approximation. Infinite impulse response (IIR) filtering for GF is faster than the classical techniques. Typical IIR GFs are addition [12] and multiplication forms [13]. Each order is $O(K)$, where $K \in \mathbb{N}$ is the approximation order and $K < R$, usually. Also, they have a constant-time property for the kernel radius.

This work was supported by JSPS KAKENHI JP17H01764, 18K18076, 18K19813, 19K24368.

978-1-7281-8068-7/20/$31.00 ©2020 IEEE

TABLE I: Number of operations of naïve implemented GF.

Algorithm	Add/Sub	Mul	# of scans
IIR (add)	$8K-2$	$8K$	4
IIR (mul)	$4K$	$4K+4$	4
Sliding-DCT-1	$4K+5$	$3K+3$	2
Sliding-DCT-3	$3K+3$	$3K+3$	2
Sliding-DCT-5	$3K+5$	$3K+3$	2
Sliding-DCT-7	$3K+5$	$3K+3$	2

As an FIR approximation for GF, the integral image is the representative, e.g., stacked integral image and cosine integral image [14], [15]. These have a recursive form for specific FIR filters. The order is also $O(K)$. Sliding-DCT-based GF is the state-of-the-arts in the integral-image-based methods [16]–[20]. This method approximates a Gaussian kernel by summation of products of cosine kernels and accelerates it by the sliding transform [21], [22]. The sliding transform can be realized by various DCT definitions, i.e., DCT-1 to DCT-8. These speed and accuracy depend on the type of DCT.

The IIR and FIR approximations have a constant-time property; however, these computational times are not the same because the number of operations and filter-passes are different (Tab. I). Recursive FIR based methods are usually faster because the number of filter-passes is smaller, improving cache efficiency.

In this paper, we aim to accelerate sliding-DCT based GF by using a hardware accelerator. We focus on the fused multiply-add (FMA) instruction, which is a hardware accelerator in modern computer architectures. FMA performs multiplication (MUL) and addition (ADD) simultaneously, i.e., $ax+b$. Thus, the number of instructions, the computational time, can be half, theoretically. However, applying FMA to the sliding-DCT based GF requires a modification of the algorithms. The contributions of this study are as follows:

- We optimize the computation of each DCT for FMA to reduce the number of operations;
- Fair comparison is performed by combining specific optimization techniques for each DCT, such as for DCT-3 [16] and DCT-5 [18].
- This study discovers that DCT-5 is the fastest. The previous work [19] states that DCT-3 was the fastest;
- This study also discovers that DCT-1 and -5 are preferable than DCT-3 and -7 in the higher approximation order case in 32-bit precision. Direct current (DC) components of

DCT-1 and -5 stabilize the numerical computation.

DCT is essential; thus, it is optimized for various application such as coding [23] and denoising [24].

II. Sliding-DCT Based Gaussian Filtering

A. DCT-Based Definition

Let g_n be a one-dimensional Gaussian kernel, , where the window length is $2R+1$. A multi-dimensional GF can be decomposed into multiple one-dimensional GFs due to its separable property. The definition is expressed as:

$$g_n = \eta^{-1}\exp(-\frac{n^2}{2\sigma^2}), \quad \eta = \sum_{n=-R}^{R}\exp(-\frac{n^2}{2\sigma^2}), \tag{1}$$

where $\sigma^2 \in \mathbb{R}$ is the variance of the Gaussian distribution. The computational order is $O(R)$.

The FIR convolution is defined as:

$$(f*g)_x = \sum_{n=-R}^{R} f_{x+n}g_n, \tag{2}$$

where f_x ($f_x \in \mathbb{R}, x \in \mathbb{N}$) is input signals, and g_n ($n = -R, ..., R$) is Gaussian weights.

Using DCT, g_n can be re-defined by:

$$g_n = \sum_{k=0}^{R} G_k C_n^{(k)}, \quad C_n^{(k)} = \cos(\phi(k+k_0)(n+n_0)), \tag{3}$$

where G_k (k=0, 1, …, R) is the DCT transformed coefficient from g_n, and $\phi = \frac{2\pi}{T}$. Eq. (3) is the inverse DCT expression of the kernel. The parameters T, k_0, and n_0 depend on the type of DCT that are summarized in Tab. II. Note that DCT-1 and -5 have DC components because $k_0 = 0$. Based on (1), G_k can be approximated as:

$$G_k \simeq \frac{c_k}{T}e^{-\frac{1}{2}\sigma^2\phi^2(k+k_0)^2}, \qquad c_k = \begin{cases} 1 & (k=0) \\ 2 & (otherwise) \end{cases} \tag{4}$$

Substituting (3) into (2) yields:

$$(f*g)_x = \sum_{n=-R}^{R}\sum_{k=0}^{R} f_{x+n}G_k C_n^{(k)} = \sum_{k=0}^{R} G_k F_k^{(x)}, \tag{5}$$

where $F_k^{(x)} = \sum_{n=-R}^{R} f_{x+n}C_n^{(k)}$ is a DCT coefficient of the partial signals extracted from the window at the coordinate x of input pixel value. Because (1) and (4) are even functions, only DCT-1, -3, -5, and -7 are symmetric at $n=0$, i.e., $n_0 = 0$.

G_k is exponential function, which is monotonous decreasing property, (5) can be truncated, such as K less than R:

$$(f*g)_x = \sum_{k=0}^{R} G_k F_k^{(x)} \simeq \sum_{k=0}^{K} G_k F_k^{(x)}. \tag{6}$$

The computational order of $F_k^{(x)}$ is $O(R)$ and total computational order of (6) is $O(KR)$, which is larger than the naïve 1D GF. Thus, we utilize the sliding transform to calculate $F_k^{(x)}$ in $O(1)$ order that reduce $O(KR)$ to $O(K)$.

TABLE II: Parameters of all DCTs

DCT-type	T	k_0	n_0
DCT-1	$2R$	0	0
DCT-3	$2R+2$	$\frac{1}{2}$	0
DCT-5	$2R+1$	0	0
DCT-7	$2R+1$	$\frac{1}{2}$	0

TABLE III: Values of $\Delta_x^{(k)}$ in Eq. (7)

DCT-Type	$\Delta_x^{(k)}$
DCT-1	$(-1)^k\{f_{x-R-1} + f_{x+R+1} - C_1^{(k)}(f_{x-R} + f_{x+R})\}$
DCT-3	$C_R^{(k)}(f_{x-R-1} + f_{x+R+1})$
DCT-5	$C_R^{(k)}(f_{x-R-1} + f_{x+R+1} - f_{x-R} - f_{x+R})$
DCT-7	$C_R^{(k)}(f_{x-R-1} + f_{x+R+1} + f_{x-R} + f_{x+R})$

B. Sliding-Transform of DCT

The second-order shift property is a relational expression involving three short-time transform coefficients:

$$F_k^{(x-1)} + F_k^{(x+1)} = 2C_{1-n_0}^{(k)}F_k^{(x)} + \Delta_x^{(k)}, \tag{7}$$

$$\Delta_x^{(k)} = C_{-R}^{(k)}f_{x-R-1} + C_R^{(k)}f_{x+R+1} - C_{-R-1}^{(k)}f_{x-R} - C_{R+1}^{(k)}f_{x+R}.$$

$F_k^{(x)}$ can be calculated in $O(1)$ by recursive computing of (7).

Focusing on the cosine phase in $\Delta_x^{(k)}$, the number of multiplications, $C_n^{(k)}$, can be reduced because of the periodicity of cosine. Table III shows the optimized $\Delta_x^{(k)}$ of all the DCTs.

We can further reduce operations in (6) by separating $k=0$ and $1 \le k \le K$:

$$(f*g)_x = G_0\{F_0^{(x)} + \sum_{k=1}^{K} Z_k^{(x)}\}, \quad Z_k^{(x)} = \gamma_k F_k^{(x)} \tag{8}$$

where $\gamma_k = \frac{G_k}{G_0} = c_k e^{-\frac{1}{2}\sigma^2\phi^2(k+k_0)^2}$. Based on (7), $Z_k^{(x)}$ is also represented as:

$$Z_k^{(x-1)} + Z_k^{(x+1)} = 2C_{1-n_0}^{(k)}Z_k^{(x)} + \gamma_k\Delta_x^{(k)}. \tag{9}$$

Eq. (8) requires less multiplications than (6). The reduction is shown in [18] for the DCT-5 case. This study extends this approach to the other DCTs. Also, we further reduce the number of operations by using FMA.

III. Reducing the number of operations

We demonstrate how to reduce the number of operations for each DCT. The FMA instructions can compute $\mathrm{FMA(a, x, b)} = \pm\mathrm{ax} \pm \mathrm{b}$ in one cycle. In recent computers, each latency and throughput of ADD, SUB, MUL, and FMA instructions are equal, e.g., the latency is 4 cycles per instruction and throughput is a 0.5 cycle per instruction Intel Skylake architecture. Therefore, we can accelerate processing by merging ADD, SUB, and MUL operations by FMA.

The sliding-DCT computation is divided into three parts: 1) Δ-computing; 2) operations at $k=0$, named 0-th operation; and 3) operation at $k \ge 1$, named k-th operation. We will introduce the results for each process in Tabs. IV and V.

We first describe the reduced number of operations at the common parts of DCTs by FMA. The 0-th operation in (8) requires 1 MUL for G_0. The k-th operation in (8) requires K

TABLE IV: The number of non-optimized operations for each DCT in terms of k-th, 0-th and Δ.

	k-th		0-th		Δ		Total
	MUL	ADD/SUB	MUL	ADD/SUB	MUL	ADD/SUB	
DCT-1	$2K$	$3K$	3	2	$K+1$	$K+3$	$7K+9$
DCT-3	$2K$	$3K$	3	2	0	1	$5K+6$
DCT-5	$2K$	$3K$	3	2	0	3	$5K+8$
DCT-7	$2K$	$3K$	3	2	0	3	$5K+8$

TABLE V: The number of optimized operations for each DCT in terms of k-th, 0-th and Δ.

	k-th			0-th			Δ			Total
	MUL	ADD/SUB	FMA	MUL	ADD/SUB	FMA	MUL	ADD/SUB	FMA	
DCT-1	0	K	$2K$	1	2	0	0	2	K	$4K+5$
DCT-3	0	K	$2K$	1	0	2	0	1	0	$3K+4$
DCT-5	0	K	$2K$	1	2	0	0	1	0	$3K+4$
DCT-7	0	K	$2K$	1	0	2	0	2	0	$3K+5$

TABLE VI: Buffer sizes and number of pixel references.

DCT-types	Buffer size					# of pixel references
	$Z_k^{(x+1)}$	γ_k/γ'_k	C_n^k	Δ'_x	Total	
DCT-1	$2K+1$	K	$2K$	0	$5K+1$	4
DCT-3	$2(K+1)$	$K+1$	$K+1$	0	$4K+4$	2
DCT-5	$2K+1$	K	K	1	$4K+2$	2
DCT-7	$2(K+1)$	$K+1$	$K+1$	1	$4K+5$	2

ADDs. The operations in (9) can be computed by $2K$ FMAs instead of two ADDs/MULs[1] (See Tab. V.). Δ-computing needs one MUL for each $C^{(k)}$. Thus, in total, K MULs and three, one, three, and three times ADDs/SUBs are needed for DCT-1, -3, -5, and -7, respectively. Furthermore, we can specialize in each DCT to reduce the number of operations.

A. DCT-1

DCT-1 has a DC component at $k = 0$; thus, the 0-th operation can be reduced. $Z_0^{(x)}$ ($F_0^{(x)}$) can be calculated as:

$$F_0^{(x+1)} = F_0^{(x)} + f_{x+R+1} - f_{x-R}. \tag{10}$$

The calculation has two ADDs/SUBs. For $k \geq 1$, (9) is used to compute $Z_k^{(x)}$. For Δ-calculation, the terms $f_{x-R-1}+f_{x+R+1}$ and $f_{x-R}+f_{x+R}$ are calculated once; thus, using two ADDs. In calculating $\Delta_x^{(k)}$, we can save the number of operations K times because the calculation can be reduced by FMA [2]. The bit sign, $(-1)^k$ (see Tab. III), can be removed by alternating ADDs/SUBs of Δ. In total, DCT-1 is reduced from $7K+9$ to $4K+5$ operations.

B. DCT-3

DCT-3 has no DC component; thus, we compute (9) at $k = 0$ by two FMAs. For $\Delta_x^{(k)}$-calculation, we can remove MULs of $C^{(k)}$ by the pre-multiplication γ_k and $C_R^{(k)}$. The new value $\gamma'_k = \gamma_k C_R^{(k)}$ is constant for each k; thus, the value can be pre-computed. Furthermore, we define the new $\Delta'_x = \Delta^{(k)}/C_R^{(k)}$, which is also constant for each k; therefore, the Δ-computation is likewise constant for every k. Note that one ADD is required when computing Δ'_x. Totally, DCT-3 is reduced from $5K+6$ to $3K+4$ operations.

[1] $Z_k^{(x+1)} = \mathrm{FMA}(2\mathrm{C}_{1-\mathrm{n}_0}^{(\mathrm{k})}, \mathrm{Z}_{\mathrm{k}}^{(\mathrm{x})}, \mathrm{FMA}(\gamma_{\mathrm{k}}, \Delta_{\mathrm{x}}^{(\mathrm{k})}, -\mathrm{Z}_{\mathrm{k}}^{(\mathrm{x}-1)}))$.
[2] $\Delta_x^{(k)} = \mathrm{FMA}(-\mathrm{C}_1^{(\mathrm{k})}, \mathrm{f}_{\mathrm{x-R}} + \mathrm{f}_{\mathrm{x+R}}, \mathrm{f}_{\mathrm{x-R-1}} + \mathrm{f}_{\mathrm{x+R+1}})$

C. DCT-5

DCT-5 also has a DC component. $Z_0^{(x)}$ ($or F_0^{(x)}$) is calculated by (10), and subsequently two ADDs/SUBs are required. For Δ-calculation, we can reduce the k MULs by the technique shown in DCT-3; thus, we can reduce K ADDs/SUBs. The term $f_{x+R+1} - f_{x-R}$ in the Δ-calculation are the same as the ones in (10). Thus, we can reduce the operations by storing and reusing this value. Furthermore, the terms $f_{x+R+1} - f_{x-R}$ at x are the same as the terms $-f_{x+R} + f_{x-R-1}$ at $x+1$, in terms of their absolute values. Thus, we can omit the operation. Finally, only one SUB is required for Δ-calculation. In total, DCT-5 is reduced from $5K+8$ to $3K+4$ operations.

D. DCT-7

The 0-th operation in (9) can compute two FMAs in the same manner as DCT-3. For Δ-calculation, one ADD can be reduced. Furthermore, the terms $f_{x+R+1} + f_{x-R}$ at x are the same as the terms $f_{x+R} + f_{x-R-1}$ at $x+1$. Thus, two ADDs are required for the terms f in Δ'_x. Totally, DCT-7 is reduced from $5K+8$ to $3K+5$ operations.

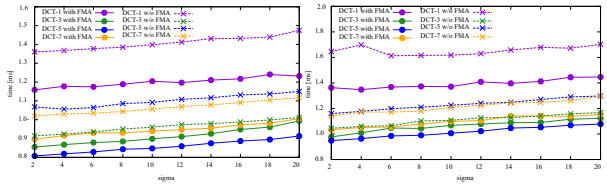
IV. Buffer size and Number of Pixel references

The load/store operations are also essential. In particular, our GF has minimal computational complexities; thus, the time for loading and storing pixels tends to be dominant.

We consider the size of buffers and the number of pixel references for the next pixel of $Z_k^{(x+1)}$ to minimize the number of load/store operations as small as possible.

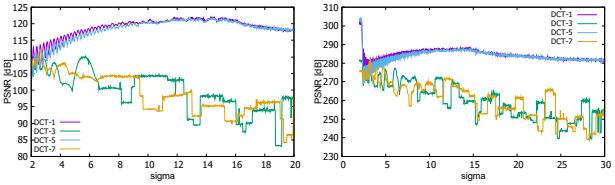
Table VI shows the buffer sizes and the number of pixel references. For $Z_k^{(x+1)}$, DCT-1 and -5 require $2K$ buffers for $Z_k^{(x)}$ and $Z_k^{(x-1)}$ ($k \geq 1$) and one buffer for $Z_0^{(x)}$. In contrast, DCT-3 and -7 require $2(K+1)$ buffers ($k \geq 0$). For γ_k or γ'_k, DCT-1 and -5 require K buffers ($k \geq 1$) and DCT-3 and -7 require $K+1$ buffers ($k \geq 0$). For $C_n^{(k)}$, DCT-1 requires $2K$ buffers ($2C_{1-n_0}^{(k)}$ in Eq. (9) and $C_1^{(k)}$ in $\Delta_x^{(k)}$); DCT-5 requires K buffers and DCT-3 and -7 require $K+1$ buffers. For Δ'_x calculation, DCT-5 and -7 require one buffers by reusing $\pm f_{x+R+1} \pm f_{x-R}$. Totally, DCT-1, -3, -5, and -7 require $5K+1$, $4K+4$, $4K+2$, and $4K+5$ buffers, respectively.

DCT-1 requires four pixels and DCT-3 requires two pixels in terms of the number of pixel references. There are four



(a) $K = 3$. (b) $K = 4$.

Fig. 1: Computational time vs. σ of the sliding-DCT based GF with/without FMA instruction.



(a) $K = 5$ using 32-bits precision. (b) $K = 17$ using 64-bits precision.

Fig. 2: The stability of each DCT in 32 or 64-bit precision.

pixels in Δ'_x for DCT-5 and -7; however, $\pm f_{x+R+1} \pm f_{x-R}$ can be reused. Thus, they require only two pixel-references.

V. Performance Evaluation

We demonstrate the effectiveness of the proposed method. Moreover, we show the accuracy for 32-bit and 64-bit precision to evaluate their performances of constant-time GFs. For the accuracy evaluation, we use the peak signal-to-noise ratio (PSNR) as a metric and the 64-bit precision output image of *cv::GaussianBlur* in OpenCV as the answer image. Note that we used $r = 8.0\sigma$ for generating the answer. The test image was Lenna (512×512) with grayscale values. The tested CPU was Intel Core i9–9900K 3.60GHz 8 cores/16 threads. The code was written in C++ and vectorized by AVX/AVX2. Note that the code is not parallelized.

Figure 1 shows the computational time vs. σ for each GF. In Fig. 1, the proposed method, which is accelerated by FMA, is around 30% faster than the conventional methods.

Figure 2 shows 32-bit and 64-bit precision results at $K = 5$ and $K = 17$. DCT-1 and -5 are stable, and that DCT-3 and -7 are unstable. The reason for this is that DCT-1 and -5 have DC components, unlike the other DCTs.

VI. Conclusion

In this study, we proposed a method to reduce the operational complexity of the sliding-DCT based GF using FMA instructions. Moreover, we evaluated the stability of the sliding-DCT based GF. The experimental results show that the proposed method is around 30% faster than conventional methods. In comparing computational time and accuracy for each DCT, DCT-3 and -5 are faster than the other DCTs; however, DCT-3 requires larger buffer sizes compared with DCT-5. Moreover, in terms of the stability of accuracy, DCT-1 and -5 are the optimal methods. Overall, DCT-5 is the most optimal method among the DCTs.

References

[1] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60(2), pp. 91–110, 2004.

[2] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on Pattern Analysis Machine Intelligence*, vol. 20, pp. 1254–1259, 1998.

[3] K. N. Chaudhury, "Acceleration of the shiftable o(1) algorithm for bilateral filtering and nonlocal means," *IEEE Transactions on Image Processing*, vol. 22, pp. 1291–1300, 2013.

[4] K. Sugimoto and S. Kamata, "Compressive bilateral filtering," *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 3357–3369, 2015.

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

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

[7] N. Fukushima, K. Sugimoto, and S. Kamata, "Guided image filtering with arbitrary window function," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

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

[9] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[10] T. Sasaki, N. Fukushima, Y. Maeda, K. Sugimoto, and S. Kamata, "Constant-time gaussian filtering for acceleration of structure similarity," in *Proc. International Conference on Image Processing and Robotics (ICIPRoB)*, 2020.

[11] M. Irani and S. Peleg, "Improving resolution by image registration," *Graphical models and image processing*, vol. 53(3), pp. 231–239, 1991.

[12] R. Deriche, "Fast algorithms for low-level vision," *IEEE Transactions on Pattern Analysis Machine Intelligence*, vol. 12, pp. 78–87, 1990.

[13] L. J. vanVliet, I. T. Young, and P. W. Verbeek, "Recursive gaussian derivative filters," in *Proc. International Conference on Pattern Recognition (ICPR)*, 1998.

[14] M. Hussein, F. Porikli, and L. Davis, "Kernel integral images: A framework for fast non-uniform filtering," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[15] E. Elboher and M. Werman, "Cosine integral images for fast spatial and range filtering," in *Proc. IEEE International Conference on Image Processing (ICIP)*, 2011.

[16] D. Charalampidis, "Recursive implementation of the gaussian filter using truncated cosine functions," *IEEE Transactions on Signal Processing*, vol. 64, no. 14, pp. 3554–3565, 2016.

[17] K. Sugimoto and S. Kamata, "Fast gaussian filter with second-order shift property of dct-5," in *Proc. IEEE International Conference on Image Processing (ICIP)*, 2013.

[18] ——, "Efficient constant-time gaussian filtering with sliding dct/dst-5 and dual-domain error minimization," *ITE Transactions on Media Technology and Applications*, vol. 3, pp. 12–21, 2015.

[19] K. Sugimoto, S. Kyochi, and S. Kamata, "Universal approach for dct-based constant-time gaussian filter with moment preservation," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[20] T. Yano, K. Sugimoto, Y. Kuroki, and S. Kamata, "Acceleration of gaussian filter with short window length using dct-1," in *Proc. APSIPA Annual Summit and Conference*, 2018.

[21] V. Kober, "Fast algorithms for the computation of sliding discrete sinusoidal transforms," *IEEE Transactions on Signal Processing*, vol. 52, pp. 1704–1710, 2004.

[22] N. Fukushima, Y. Maeda, Y. Kawasaki, M. Nakamura, T. Tsumura, K. Sugimoto, and S. Kamata, "Efficient computational scheduling of box and gaussian fir filtering for cpu microarchitecture," in *Proc. APSIPA Annual Summit and Conference*, 2018.

[23] P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. Lim, and C. Yeo, "Efficient integer dct architectures for hevc," *IEEE Transactions on Circuits and systems for Video Technology*, vol. 24, no. 1, pp. 168–178, 2013.

[24] N. Fukushima, Y. Kawasaki, and Y. Maeda, "Accelerating redundant dct filtering for deblurring and denoising," in *IEEE International Conference on Image Processing (ICIP)*, 2019.