# Basic Study on Domain Specific Description of Convolution with Sliding DFT

Yamato Kanetaka[1], Yoshihiro Maeda[2], Norishige Fukushima[1]
1 Nagoya Institute of Technology
Gokisho-cho, Showa-ku, Nagoya, Aichi 466-0061, Japan
2 Tokyo University of Science
3-1, Shinjuku 6-chome, Katsushika-ku, Tokyo 125-8585, Japan
E-mail: kntkc0de@gmail.com

**Abstract** Constant-time filters often approximate image filtering used in various applications to reduce computational complexity since the computational complexity depends on the window length of the kernel.
This paper focuses on convolution filtering with sliding frequency transforms, one of the constant-time filters. Although this filter is fast and accurate, implementing a program with parallelism and locality in mind results in very complex and redundant code since it is based on recursive processing. Therefore, we propose a domain-specific language for constant-time filtering with the sliding transform.

**Keywords:** sliding-DFT, Halide, RecFilter, DSL, sliding frequency transform

## 1. Introduction

Filtering is a fundamental tool in image processing and computer vision; thus, speeding up the filtering itself plays an important role. One problem with common image filtering, such as Gaussian filtering, is that the computational complexity depends on the window length of the filter. One solution to this problem is a convolution filter using a sliding frequency transform, one of the constant-time filterings with high accuracy and speed. However, the sliding filtering programs are hard to optimize: parallelization, vectorization, and cache blocking. The complexity is based on the recursion processing in sliding filtering requires redundant code.

This paper proposes a domain-specific language that can efficiently describe convolution with Sliding DFT.

## 2. Sliding DFT

On FIR convolution

$$(f * g)_x = \sum_{n=-R}^{R} f_{x+n}\, g_n \quad (1)$$

where $g_n$ is a kernel weight and $R$ is a window radius of the kernel, $g_n$ can be approximated by the discrete Fourier transform (DFT) and described by the following equation.

$$(f * g)_x \simeq \sum_{n=-R}^{R} \sum_{k=0}^{K} f_{x+n}\, G_k Q_n^{(k)+} = \sum_{k=0}^{K} G_k\, F_k^{(x)} \quad (2)$$

where $F_k^{(x)} = \sum_{n=-R}^{R} f_{x+n}\, Q_n^{(k)+}$ and $Q_n^{(k)+} = e^{i\omega(k+k0)(n+n0)}$. A computational complexity of Equation (2) is $O(KR)$. For now, we introduce the second-order shift property [1], which is a relational expression involving three sequential $F_k$, defined by the following equation.

$$F_k^{(x-1)} + F_k^{(x+1)} = 2C_1^{(k)} F_k^{(x)} + \Delta_x^{(k)} \quad (3)$$

where $C_n^{(k)} = \cos\left(\frac{2\pi}{T}kn\right)$ and $\Delta_x^{(k)} = Q_R^{(k)-}(f_{x-R-1} - f_{x+R}) + Q_R^{(k)+}(f_{x+R+1} - f_{x-R})$ (In DFT-V). $F_k$ can be calculated recursively in $O(1)$.

Convolution with sliding DFT, which uses the second-order shift recursively, can be described in the following equation.

$$Z_k^{(x-1)} + Z_k^{(x+1)} = 2C_{1-n_0}^{(k)} Z_k^{(x)} + \Delta_x^{F(k)} G_k^F \quad (4)$$

where $Z_k^{(x)} = G_k F_k^{(x)}$. Equation (2) can be filtered in $O(K)$, which does not depend on the window radius $R$. This means finite impulse response (FIR) convolution (1) can be approximated in a constant-time algorithm.

## 3. RecFilter

RecFilter [3] is a domain-specific language (DSL) for infinite impulse response (IIR) filtering. RecFilter internally uses Halide [2] and makes it simpler to write high-performance IIR filtering. RecFilter can be used for writing the code of convolution with Sliding DFT since sliding DFT is a recursive processing.

## 4. Proposed Method

Convolution with sliding DFT can filter in constant-time O(K) but implementing the sliding DFT program with high parallelism and locality results in very complex and redundant code since it has recursive processing. Therefore, We propose a DSL for convolution with sliding DFT by wrapping RecFilter.

Listing. 1. proposed DSL code for Gaussian filter.

```
Var x("x"), y("y"), c("c");
SlidingConv gf("gf", w, h, d);

gf(x, y, c) = input(x, y, c);

// algorithm part
gf
    .set_kernel(kernel)
    .set_order(3)
    .set_radius(10)
    .set_optimizeCoeff(true)
    .set_algorithm(DFT-5);

// schedule part
gf.cpu_auto_schedule();

gf.realize(output);
```

Listing 1 shows the proposed DSL code for Gaussian filtering. SlidingConv is a function body for convolution with sliding DFT. After specifying the input and parameters for the sliding DFT algorithm, scheduling can be specified.

SlidingConv can simply and separately describe convolution with sliding DFT code in the algorithm and scheduling parts. Usually, Halide has the advantage that the algorithm parts and scheduling parts can be described separately; however, when convolution with sliding DFT is written directly in Halide, it is necessary to write scheduling-aware codes in the algorithm part due to the use of RDom in the definition of recursive processing. Therefore, SlidingConv fixed this problem and can separately describe convolution with Sliding DFT code in the algorithm and scheduling parts. In addition, the scheduling of the sliding DFT convolution can be changed on SlidingConv, and the parallelization and vectorization loops can be adjusted to suit each environment.

## 5. Experimental Results

We compared the implementations of the sliding-DCT Gaussian filter of manually optimized for CPUs using SIMD's vector arithmetic instructions with SlidingConv. Fig 1 shows the results of running the Gaussian filtering with 512 × 512 and 2048×2048 images, varying the $\sigma$ of the Gaussian filter from 3 to 10.

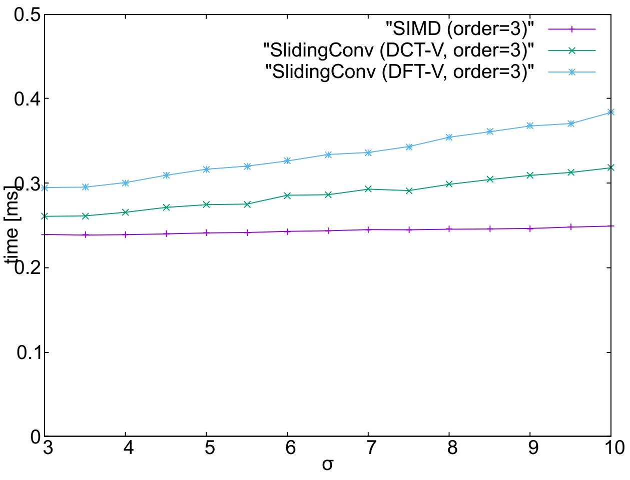
The result of 512×512 shows that SlidingConv is more affected by $\sigma$ than SIMD implementation. The result for 2048 × 2048 shows that SlidingConv is more affected by $\sigma$ than SIMD, but SlidingConv (DCT) is as fast or faster than SIMD up to $\sigma = 7$.

The reason why SlidingConv is more affected by $\sigma$ than SIMD is that SlidingConv is parallelized by tiles, and the number of convolutions required to start a sliding transformation increase compared to SIMD. In a result, a larger $\sigma$ has a greater impact than SIMD when the computational complexity of the convolution increases due to an increase in the window radius of convolution.

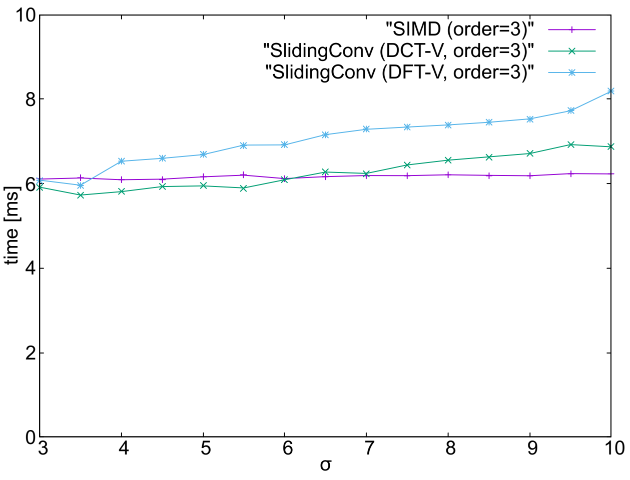
The reason why SlidingConv is faster than SIMD when the image size is increased is that when SlidingConv is parallelized with the tile. For small image sizes, SIMD is faster than SlidingConv because the increase in the number of convolutions required to start the sliding transform is larger than the increase in speed due to parallelization. However, for large image sizes, the opposite effect occurs, making SlidingConv faster than SIMD. As shown above, SlidingConv is significantly affected by $\sigma$ due to parallelization, so it is slower than SIMD for small image sizes but faster than SIMD for large image sizes and medium $\sigma$. The speed-up due to parallelization depends on CPU performance, so the speed-up due to parallelization of SlidingConv is expected to be stronger when a CPU that is more capable of parallel processing is used.

## 6. Conclusion

We proposed a DSL, which makes it simpler to write convolution with sliding DFT to resolve the problem that implementing a program for convolution with sliding DFT with parallelism and locality results in very complex and redundant code. SlidingConv can separately describe the convolution with sliding DFT code in the algorithm and scheduling parts and simply specify and change a kernel for filtering, parameters for the algorithm and schedules. In addition, the execution speed is faster than or comparable to that of SIMD for large image sizes and medium $\sigma$, which means that a proposed DSL has a simple description and sufficient high-speed processing.



(a) 512×512



(b) 2048×2048

Fig. 1. Results of Gaussian Filter (σ)

**Aknowledgements**

This research was supported by Grants-in-Aid for Scientific Research (21H03465, 21K17768).

## References

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

[2] J. Ragan-Kelley, et al., ACM TOG, July. 2012.

[3] Chaurasia, et al., High-Performance Graphics, 2015.