Guaranteed-Stable Sliding DFT Algorithm With Minimal Computational Requirements

Chun-Su Park

***Abstract*—The discrete Fourier transform (DFT) is the most widely used technique for determining the frequency spectra of digital signals. However, in the sliding transform scenario where the transform window is shifted one sample at a time and the transform process is repeated, the use of DFT becomes difficult due to its heavy computational burden. This paper proposes an optimal sliding DFT (oSDFT) algorithm that achieves both the lowest computational requirement and the highest computational accuracy among existing sliding DFT algorithms. The proposed oSDFT algorithm directly computes the DFT bins of the shifted window by simply adding (or subtracting) the bins of a previous window and an updating vector. We show that the updating vector can be efficiently computed with a low complexity in the sliding transform scenario. Our simulations demonstrate that the proposed algorithm outperforms the existing sliding DFT algorithms in terms of computational accuracy and processing time.**

***Index Terms*—Discrete Fourier transform, sliding window, updating vector, updating vector transform.**

I. Introduction

The Discrete Fourier Transform (DFT) is an orthogonal transform which is practically very valuable for representing signals and images. The DFT has been widely used in many real applications due to its good performance. For example, the DFT has been applied in spectral analysis [1], audio processing [2], big data analysis [3], image registration [4], super resolution [5], texture synthesis [6], image denoising [7], and object tracking [8], [9]. It is worthwhile to note that the DFT is also used to efficiently solve partial differential equations and perform other operations such as convolutions or multiplying large integers [10], [11].

However, in some cases, the computation of the DFT is unacceptably slow with respect to the application requirements [12]–[16]. Further, the use of the DFT becomes even more difficult in the sliding transform scenario where the transform window is shifted one sample at a time and the transform process is repeated. In the past years, numerous algorithms have been developed for the fast computation of the sliding DFT.

In [17], the sliding fast Fourier transform (SFFT) algorithm was introduced, which calculates the bins of the shifted window by exploiting the delayed intermediate calculations of the previous window. In [18], the generalized SFFT (GSFFT) algorithm was proposed for an efficient implementation of the hopping FFT. The concept of the sliding DFT (SDFT) was introduced in [19], which computes the DFT bins of the shifted window based on the circular shift property of the DFT. Thereafter, in 2003, Jacobsen and Lyons gave a detailed account of the SDFT in [20]. The SDFT reduces the computational load of the DFT drastically; it requires, for a complex input signal, one complex multiplication and two complex additions to compute each bin of the shifted window. However, in practice, the complex twiddle factor used in sliding transform operation is often represented by a floating-point format with finite precision. The errors generated by the numerical rounding accumulate in the sliding transform process, which can result in an unstable system.

Several algorithms were proposed to guarantee the stability of the DFT in the sliding transform scenario [21]–[24], [24]. A stable SDFT algorithm that uses a simple recursive updating scheme was proposed in [22]. This algorithm is realized using a periodically-time-varying system designed such that the numerical errors introduced by finite precision arithmetic exponentially decay to zero over time. Another stable SDFT, called rSDFT, was proposed in [23]. This algorithm forces the pole to be at a radius of r inside the unit circle by utilizing the damping factor r, thereby guaranteeing stability. However, the output bin values of the rSDFT are different from those of the DFT and the errors accumulate in the resulting outputs. The modulated SDFT (mSDFT) was introduced in [24]. The mSDFT first generates a modulated sequence by multiplying the input signal by the modulation sequence. Then, using the modulated sequence, the mSDFT formulates the recurrence of the DFT bins. By excluding the complex twiddle factor from the feedback of the resonator, the mSDFT has the pole located exactly on the unit circle and is unconditionally stable. However, the computational requirement of the mSDFT is more than double that of the SDFT in [20]. Note that all of the above-mentioned algorithms guarantee the stability at the cost of computational complexity or computational accuracy.

Recently, the generalized SDFT (gSDFT) algorithm which is the generalized version of the hopping DFT (HDFT) [25] was introduced in our previous work [26]. The gSDFT investigates the special relationship between the DFT bins, which can be expressed without using the imprecise twiddle factors. Based on the relationship, the gSDFT derives a recurrence formula which can be implemented with low computational complexity. In a recent study [27], it was reported that the gSDFT algorithm offers the lowest numerical error among the existing sliding transform

Manuscript received March 6, 2017; revised June 8, 2017; accepted June 26, 2017. Date of publication July 13, 2017; date of current version August 8, 2017. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Yuichi Tanaka. This work was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education, Science and Technology (NRF-2016R1C1B1009682).

The author is with the Department of Computer Education, Sungkyunkwan University, Seoul 03063, South Korea (e-mail: cspk@skku.edu).

Digital Object Identifier 10.1109/TSP.2017.2726988

algorithms including the rSDFT and mSDFT algorithms. This means that the omission of the complex exponential in the feedback enhances the computational accuracy significantly. However, it was shown in [26] that the complexity of the gSDFT increases sharply as the size of window increases.

In this paper, we propose an optimal SDFT (oSDFT) algorithm with the lowest computational requirement among the existing stable sliding DFT algorithms. Motivated by our previous work [26], the oSDFT obtains the bins of the shifted window by updating those of a previous window instead of directly computing them. Specifically, the oSDFT calculates the DFT bins of the shifted window by simply adding (or subtracting) the bins of a previous window and an updating vector. We demonstrate that the updating vector can be obtained with low computation complexity in the sliding transform scenario. The theoretical analysis shows that the output of the proposed algorithm is mathematically equivalent to that of the original DFT at all time indexes. Further, the numerical errors of the oSDFT are exactly the same as those of the state-of-the-art gSDFT algorithm.

The rest of this paper is organized as follows. In Section II, we briefly review the conventional SDFT algorithm. Section III derives the recursive relationship between adjacent DFT outputs and introduces the proposed oSDFT algorithm. Section IV presents the fast implementation scheme of the updating vector calculation. In Section V, we illustrate the overall process and analyze the computational requirement of the proposed algorithm. Comparative experimental results of the proposed and conventional algorithms are presented in Section VI. Finally, our conclusions are drawn in Section VII.

II. Exiting SDFT Algorithm

In the sliding transform, the transform is repeatedly performed on a fixed-size window of the signal, which is continuously updated with new samples as the oldest ones are discarded. Let us denote by $x(n)$, $n = 0, 1, 2, \dots$, a complex input signal which will be divided into overlapping windows of size M. Further, let $X_n(k)$, $k = 0, 1, \dots, M-1$, be the k-th bin of the M-point DFT at time index n, which is represented by

$$X_n(k) = \sum_{m=0}^{M-1} x(\hat{n}+m) W_M^{-km} \tag{1}$$

where $\hat{n} = n - M + 1$ and $W_M = e^{j2\pi/M}$. Then, by the circular shift property, we can derive the following relationship between successive DFT outputs:

$$\begin{aligned} X_n(k) &= \sum_{m=0}^{M-1} x(\hat{n}+m) W_M^{-km} \\ &= \sum_{m=0}^{M-1} x(\hat{n}+m-1) W_M^{-k(m-1)} \\ &\quad + x(\hat{n}+M-1) W_M^{-k(M-1)} - x(\hat{n}-1) W_M^{k} \\ &= W_M^{k} \sum_{m=0}^{M-1} x(\hat{n}+m-1) W_M^{-km} \\ &\quad + x(\hat{n}+M-1) W_M^{k} - x(\hat{n}-1) W_M^{k} \\ &= W_M^{k} (X_{n-1}(k) + x(n) - x(n-M)) \end{aligned} \tag{2}$$

where the periodicity property of the complex twiddle factor is exploited ($W_M^{k+M} = W_M^k$). Equation (2) represents the SDFT algorithm introduced in [20]. The SDFT needs to multiply the complex twiddle factor W_M^k to the DFT bin of the previous window to obtain that of the shifted window. As mentioned, the SDFT requires one complex multiplication and two complex additions for computing each bin of the shifted window.

The SDFT has a marginally stable transfer function because its pole lies on the unit circle in the z-domain [20]. In practice, a complex twiddle factor in (2) is often represented by a floating-point format with finite precision. This numerical rounding of the complex twiddle factor might move the pole outside the unit circle and result in an unstable system. Further, the complex twiddle factor included in the recursive formula increases the computational load of the sliding transform process. To address these issues, we design a guaranteed-stable sliding DFT filter with reduced computational complexity by excluding the imprecise twiddle factor from the feedback loop.

III. Proposed oSDFT Algorithm

We derive the general formula between the DFT bins with L-hop distance by extending the relationship between the bins of successive windows. Let $d(n) = x(n) - x(n-M)$, and then the resultant formula is obtained by substituting $X_n(k)$ into $X_{n-1}(k)$ L times in (2):

$$\begin{aligned} X_n(k) &= W_M^{k} (X_{n-1}(k) + d(n)) \\ &= W_M^{2k} (X_{n-2}(k) + d(n-1) + W_M^{-k} d(n)) \\ &\quad \vdots \\ &= W_M^{Lk} (X_{n-L}(k) + d(n-L+1) \\ &\quad + W_M^{-k} d(n-L+2) + \cdots + W_M^{-(L-1)k} d(n)). \end{aligned} \tag{3}$$

Let us define $D_n(k)$ as the kth bin of the L-point updating vector transform (UVT), which is represented by

$$D_n(k) = \sum_{m=0}^{L-1} d(\tilde{n}+m) W_M^{-mk} \tag{4}$$

where $\tilde{n} = n - L + 1$ and $0 \le k < M$. Then, using this notation, (3) is simplified as

$$X_n(k) = W_M^{Lk} (X_{n-L}(k) + D_n(k)). \tag{5}$$

This leads to the result that the DFT outputs at time index n can be directly computed from those at time index $(n-L)$ by exploiting $D_n(k)$. In the next section, we describe how to efficiently implement the UVT in (4).

Let us focus on the twiddle factor W_M^{Lk} which is multiplied to the delayed output in the recurrence computation (5). We can observe that the value of the twiddle factor is varied depending on the time hop L. Here, we consider the case where the window size M is a power of two and L is equal to $M/4$ ($L = M/4$). In this case, from Euler's formula, we have

$$W_M^{Mk/4} = (\cos(\pi/2) + j\sin(\pi/2))^k = j^k. \tag{6}$$

TABLE I
RECURSIVE RELATIONSHIP BETWEEN REAL AND ODD PARTS OF $X_n(k)$ AND $X_{n-M/4}(k)$

q	$\text{Re}(X_n(4i+q))$	$\text{Im}(X_n(4i+q))$
0	$\text{Re}(X_{n-M/4}(4i)) + \text{Re}(D_n(4i))$	$\text{Im}(X_{n-M/4}(4i)) + \text{Im}(D_n(4i))$
1	$-[\text{Im}(X_{n-M/4}(4i+1)) + \text{Im}(D_n(4i+1))]$	$\text{Re}(X_{n-M/4}(4i+1)) + \text{Re}(D_n(4i+1))$
2	$-[\text{Re}(X_{n-M/4}(4i+2)) + \text{Re}(D_n(4i+2))]$	$-[\text{Im}(X_{n-M/4}(4i+2)) + \text{Im}(D_n(4i+2))]$
3	$\text{Im}(X_{n-M/4}(4i+3)) + \text{Im}(D_n(4i+3))$	$-[\text{Re}(X_{n-M/4}(4i+3)) + \text{Re}(D_n(4i+3))]$

Accordingly, (5) can be further simplified as

$$X_n(k) = j^k(X_{n-M/4}(k) + D_n(k)). \tag{7}$$

Then, from the property of j ($j^2 = -1$), we obtain the following:

$$\begin{bmatrix} X_n(4i) \\ X_n(4i+1) \\ X_n(4i+2) \\ X_n(4i+3) \end{bmatrix} = \begin{bmatrix} X_{n-M/4}(4i) + D_n(4i) \\ j(X_{n-M/4}(4i+1) + D_n(4i+1)) \\ -(X_{n-M/4}(4i+2) + D_n(4i+2)) \\ -j(X_{n-M/4}(4i+3) + D_n(4i+3)) \end{bmatrix} \tag{8}$$

where $i = 0, 1, \ldots, M/4 - 1$. This forms the basis of an efficient scheme for computing the DFT bins of the shifted window. Given the DFT bins of the previous window at time index $(n - M/4)$, the bins at time index n can be directly computed without performing the multiplication by the twiddle factor. For example, when $k = 4i$, the following relationship holds between $X_n(4i)$ and $X_{n-M/4}(4i)$:

$$\begin{aligned} &\text{Re}(X_n(4i)) + j\text{Im}(X_n(4i)) \\ &\quad = \text{Re}(X_{n-M/4}(4i)) + \text{Re}(D_n(4i)) \\ &\qquad + j[\text{Im}(X_{n-M/4}(4i)) + \text{Im}(D_n(4i))] \end{aligned} \tag{9}$$

where $\text{Re}(\cdot)$ and $\text{Im}(\cdot)$ denote the real and imaginary parts of a complex number, respectively. Then, each part of $X_n(4i)$ is obtained as

$$\begin{cases} \text{Re}(X_n(4i)) = \text{Re}(X_{n-M/4}(4i)) + \text{Re}(D_n(4i)) \\ \text{Im}(X_n(4i)) = \text{Im}(X_{n-M/4}(4i)) + \text{Im}(D_n(4i)). \end{cases} \tag{10}$$

Similarly, the efficient computation for the other cases, $k = 4i+1, 4i+2, 4i+3$, can be derived. Table I summarizes the recursive relationship between real and imaginary parts of $X_n(k)$ and $X_{n-M/4}(k)$. These results demonstrate that only two real additions are required for computing $X_n(k)$ using $X_{n-M/4}(k)$ and $D_n(k)$. Further, since the multiplication by the imprecise twiddle factor is excluded from the recurrence calculation, the numerical errors do not accumulate. Therefore, the proposed oSDFT algorithm is stable.

IV. Sliding Updating Vector Transform

In this section, we describe how to efficiently implement $D_n(k)$ in (4). As shown in (4), since the UVT is similar to the existing DFT, traditional FFT algorithms can be used for its fast implementation [28], [29]. We use the radix-2 decimation-in-time (DIT) algorithm that divides a UVT of size L into two interleaved UVTs of size $L/2$ at each decimation stage. Based on the DIT approach, we express $D_n(k)$ using decimated sequences as follows:

$$\begin{aligned} D_n(k) &= \sum_{m=0}^{L-1} d(\tilde{n}+m)W_M^{-mk} \\ &= \sum_{p=0}^{L/2-1} d(\tilde{n}+2p)W_M^{-(2p)k} \\ &\quad + \sum_{p=0}^{L/2-1} d(\tilde{n}+(2p+1))W_M^{-(2p+1)k} \\ &= D_n^e(k) + W_M^{-k}D_n^o(k) \end{aligned} \tag{11}$$

where $D_n^e(k)$ and $D_n^o(k)$, respectively, denote the even and odd parts of $D_n(k)$:

$$D_n^e(k) = \sum_{p=0}^{L/2-1} d(\tilde{n}+2p)W_M^{-2pk} \tag{12}$$

and

$$D_n^o(k) = \sum_{p=0}^{L/2-1} d(\tilde{n}+(2p+1))W_M^{-2pk}. \tag{13}$$

Equation (11) implies that a length-L UVT bin can be obtained by using two length-$(L/2)$ UVT bins of the decimated sequences, $\{d(\tilde{n}), d(\tilde{n}+2), \ldots, d(\tilde{n}+L-2)\}$ and $\{d(\tilde{n}+1), d(\tilde{n}+3), \ldots, d(\tilde{n}+L-1)\}$. The decimation process is repeated until the resulting sequences are reduced to one-point sequences.

Let l, $l = 0, \ldots, \log_2 L - 1$, be the decimation stage of the UVT. Further, denote by Ψ_l a set of sequential decimation operations which are recursively performed l times from an initial seed as follows:

$$\begin{cases} \Psi_0 = \emptyset \\ \Psi_1 = \{e, o\} \\ \Psi_2 = \{ee, eo, oe, oo\} \\ \quad\vdots \\ \Psi_l = \{[s_{l-1}, e], [s_{l-1}, o] : \forall s_{l-1} \in \Psi_{l-1}\} \end{cases} \tag{14}$$

where s_l specifies a length-l sequential decimation operation and $[\cdot, \cdot]$ denotes concatenation. Using these notations, the recursive relationship in (11) can be generally rewritten as

$$D_n^{s_l}(k) = D_n^{[s_l,e]}(k) + W_M^{-2^l k}D_n^{[s_l,o]}(k) \tag{15}$$

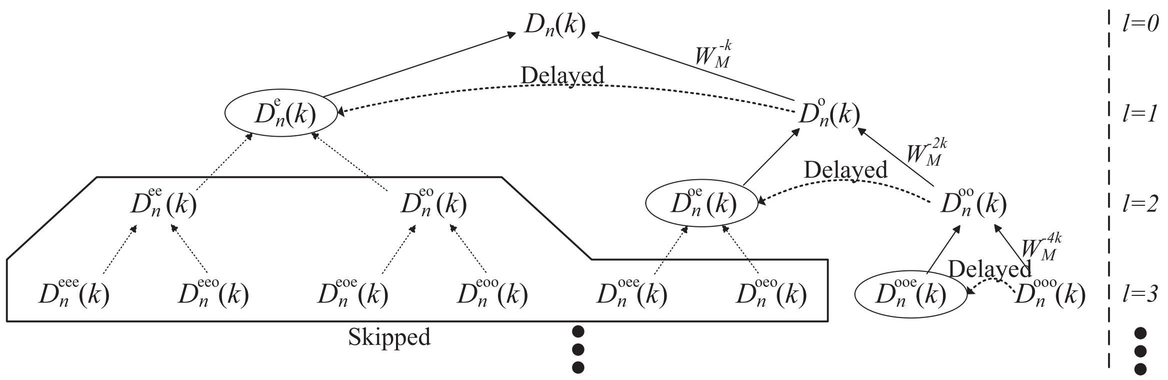


Fig. 1. Decimation process of the single-bin SUVT, where only the solid lines need to be calculated.

where $D_n^{[s_l,e]}(k)$ and $D_n^{[s_l,o]}(k)$ are the even and odd parts of $D_n^{s_l}(k)$, respectively. For example, $D_n^e(k)$ is decimated as

$$D_n^e(k) = D_n^{ee}(k) + W_M^{-2k} D_n^{eo}(k). \tag{16}$$

During the sliding transform process, the UVT needs to be repeatedly computed at each time index. Thus, the computational requirement of the UVT can be reduced further if the UVT bins of the current window can be obtained by exploiting the intermediate calculations of previous windows. Let $\tilde{n}_{s_l}$, $s_l \in \Psi_l$, be the time index of the first element of the decimated sequence used for computing $D_n^{s_l}(k)$. From its definition, we derive that the UVTs between adjacent time indexes have the following relationship:

$$\begin{aligned} D_n^{[s_l,e]}(k) &= \sum_{p=0}^{L/2^l-1} d(\tilde{n}_{s_l} + 2^l \cdot 2p) W_M^{-2^{(l+1)}pk} \\ &= \sum_{p=0}^{L/2^l-1} d(\tilde{n}_{s_l} - 2^l + 2^l(2p+1)) W_M^{-2^{(l+1)}pk} \\ &= D_{n-2^l}^{[s_l,o]}(k). \end{aligned} \tag{17}$$

The above result presents that $D_n^{[s_l,e]}(k)$ is completely identical to $D_{n-2^l}^{[s_l,o]}(k)$. Then, from (15) and (17), the decimation process can be represented by

$$D_n^{s_l}(k) = D_{n-2^l}^{[s_l,o]}(k) + W_M^{-2^l k} D_n^{[s_l,o]}. \tag{18}$$

This is an important fact to be considered because the intermediate calculations of previous windows can be reused for the current window without losing computational accuracy. At each decimation stage, we need to compute only the odd part in a recursive manner and the even part can be simply copied from the precalculated intermediate result of a previous window. On the basis of these findings, we propose a sliding UVT (SUVT) algorithm.

Equation (18) directly leads to the recursive filter structure for computing a single-bin SUVT shown in Fig. 1. At decimation stage l, the proposed SUVT algorithm first computes $D_n^{[s_l,o]}$ using the DIT approach. Then, according to (18), the UVT output $D_n^{s_l}(k)$ is obtained using $D_n^{[s_l,o]}$ and $D_{n-2^l}^{[s_l,o]}(k)$, where $D_{n-2^l}^{[s_l,o]}(k)$ has already been obtained at window position $(n-2^l)$. The price to be paid for this strategy is the additional memory to maintain the necessary intermediate calculations of the previous windows.

The SUVT for computing all bins of the shifted window can be best explained by referring to the butterfly structure shown in Fig. 2. For simplicity of explanation, the size of the sliding window is assumed to be $M = 32$. Assume that all calculations in this structure have been already performed at previous window positions and the results are available at all points of the structure. We see from Fig. 2 that, when the new sample, $d(\tilde{n}+L-1)$, enters and the oldest sample, $d(\tilde{n}-1)$, leaves to update the structure, only the butterflies highlighted by filled circles need to be calculated. The calculations related to the rest of the butterflies are available because they are delayed results of the highlighted butterflies. It is worthwhile to note that the proposed oSDFT algorithm has the same precision as the traditional FFT because the twiddle factors used in the SUVT are identical to those used in the traditional butterfly-based FFT algorithm [29].

We now focus our attention on the fast implementation of the last decimation stage ($l = \log_2 L - 1$). As shown in Fig. 2, four intermediate calculations, I_0, I_1, I_2, and I_3, need to be computed at the last stage regardless of window size. The intermediate calculations are obtained by multiplying the new sample $d(\tilde{n} + L - 1)$ by different twiddle factors as follows:

$$\begin{cases} I_0 = W_M^0 d(\tilde{n}+L-1) = d(\tilde{n}+L-1) \\ I_1 = W_M^{-M/8} d(\tilde{n}+L-1) \\ I_2 = W_M^{-M/4} d(\tilde{n}+L-1) = -jI_0 \\ I_3 = W_M^{-3M/8} d(\tilde{n}+L-1) = -jI_1 \end{cases} \tag{19}$$

where $W_M^0 = 1$, $W_M^{-M/4} = -j$, and $W_M^{-3M/8} = -jW_M^{-M/8}$. Here, we see that I_0 is equal to the input sample $d(\tilde{n}+L-1)$. Further, each part of I_2 can be obtained using I_0 as follows:

$$\begin{cases} \mathrm{Re}(I_2) = \mathrm{Im}(I_0) = \mathrm{Im}(d(\tilde{n}+L-1)) \\ \mathrm{Im}(I_2) = -\mathrm{Re}(I_0) = -\mathrm{Re}(d(\tilde{n}+L-1)) \end{cases} \tag{20}$$

where $I_2 = -jI_0$. Similarly, I_3 can be obtained using I_1. Therefore, we need to compute only I_1 at the last decimation stage. We introduce a fast implementation scheme for the computation of I_1. Since the real and imaginary parts of $W_M^{-M/8}$ are

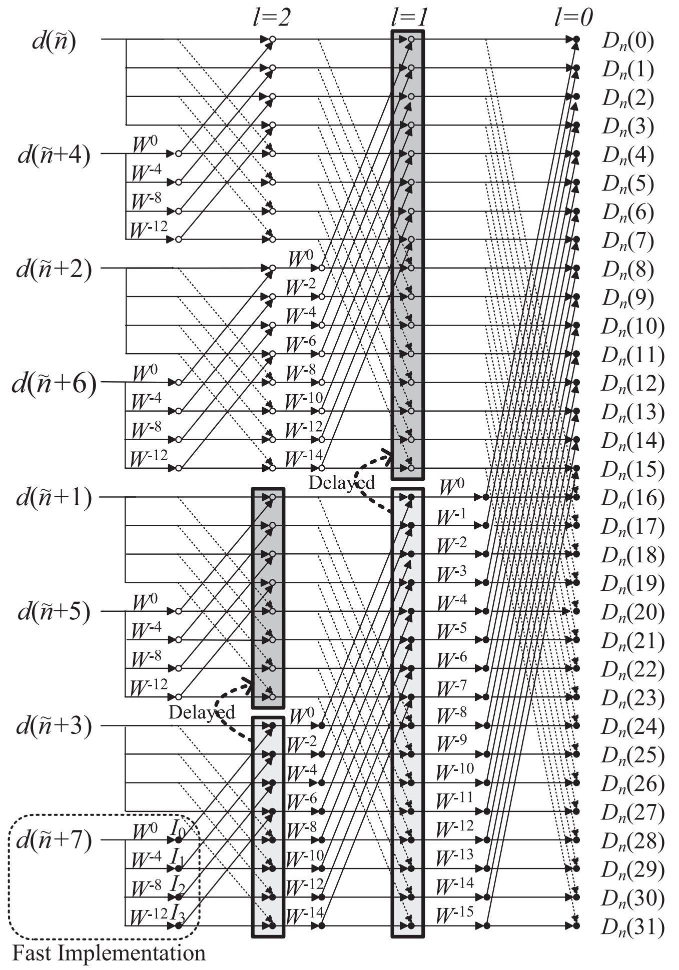


Fig. 2. Graphical explanation of SUVT for $M = 32$ and $L = 8$, where solid and dotted lines indicate plus and minus signs, respectively. Only the butterflies highlighted by filled circles need to be calculated. At the last decimation stage ($l = 2$), four intermediate calculations, I_0, I_1, I_2, and I_3, are computed using the fast implementation scheme.

identical, I_1 can be simply computed as

$$\begin{cases} \mathrm{Re}(I_1) = \alpha[\mathrm{Re}(d(\hat{n}+L-1)) - \mathrm{Im}(d(\hat{n}+L-1))] \\ \mathrm{Im}(I_1) = \alpha[\mathrm{Re}(d(\hat{n}+L-1)) + \mathrm{Im}(d(\hat{n}+L-1))] \end{cases} \quad (21)$$

where $\alpha = \mathrm{Re}(W_M^{-M/8}) = \mathrm{Im}(W_M^{-M/8})$. This means that we can compute I_1 by only two real multiplications and two real additions at the last stage of the decimation.

V. Overall Process and Complexity Analysis

The proposed oSDFT algorithm repeatedly produces the DFT outputs by moving the fixed-size window one sample at a time. For simplicity of explanation, we consider the case where all DFT bins of the shifted window are computed. At the current time index n, the overall algorithm for window size M proceeds as follows:

a) Compute $d(n) = x(n) - x(n-M)$ using the input samples. Only one complex addition is required in this step.

b) Compute the intermediate calculations, I_0, I_1, I_2, and I_3, using the fast implementation schemes in (19), (20), and (21). Using the results, the last decimation stage ($l = \log_2 L - 1$) is performed. Two real multiplications, two real additions, and eight complex additions are required in this step.

c) According to (18), compute $D_n(k)$ using the recursive relationship. At the decimation stage l, $l = 0, 1, \ldots, \log_2 L - 2$, $M/2^{(l+1)}$ complex multiplications and $M/2^l$ complex additions are required. In total, this step requires $(M-8)$ complex multiplications and $(2M-16)$ complex additions.

d) Calculate the DFT $X_n(k)$ using $X_{n-M/4}(k)$ and $D_n(k)$, where $X_{n-M/4}(k)$ is precalculated at the previous window position $(n - M/4)$. This step requires M complex additions. Note that additional memory is required to store the DFT bins of the previous windows.

TABLE II
Computational Requirement for Computing All Bins of the Window of Size M

Algorithm	Operation	Number of operations
FFT	R_M	$2M \log_2 M$
	R_A	$3M \log_2 M$
rSDFT	R_M	$6M$
	R_A	$6M$
mSDFT	R_M	$12M$
	R_A	$10M$
gSDFT	R_M	$M(\log_2 M - 1)$
	R_A	$(M/2)(3\log_2 M + 1) + 2$
oSDFT	R_M	$4M - 30$
	R_A	$8M - 28$

The total computational requirement of the proposed oSDFT is summarized as

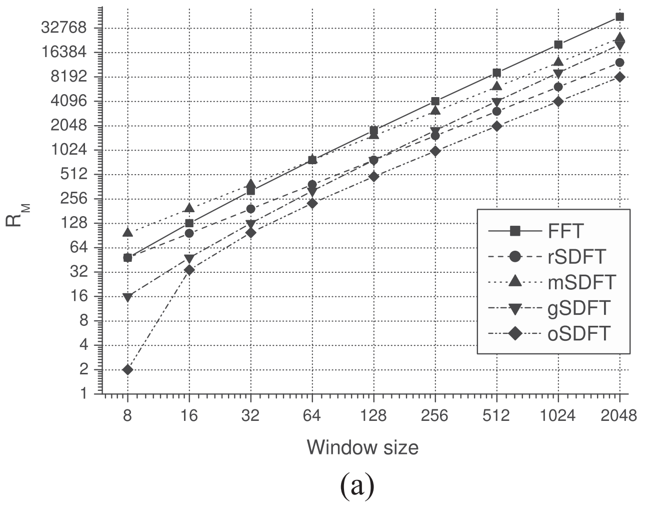
$$R_M = 4M - 30 \quad (22)$$

and

$$R_A = 8M - 28 \quad (23)$$

where R_M and R_A, respectively, denote the numbers of real multiplications and real additions. Here, one complex addition is counted as two real additions and one complex multiplication is counted as four real multiplications and two real additions [30], [31].

A computation requirements of the FFT [29], rSDFT [23], mSDFT [24], gSDFT [26], and oSDFT algorithms are presented in Table II. Further, in order to show the results more clearly, we present R_M and R_A of each algorithms with varying window sizes in Fig. 3. In particular, when the window size is equal to 16, the oSDFT algorithm reduces the number of multiplications by 73.44%, 64.58%, 82.30%, and 29.17% as compared to the FFT, rSDFT, mSDFT, and gSDFT, respectively. As shown in Fig. 3, also the gSDFT shows a relatively good performance in terms of the computation requirement. However, the computational complexity of the gSDFT increases sharply as the size of window increases. Specifically, when the window size is larger than or equal to 256, the number of multiplications of the gSDFT is larger than that of the rSDFT. On the contrary, the oSDFT consistently outperforms the existing algorithms in terms of the computational complexity, i.e., the number of multiplications of



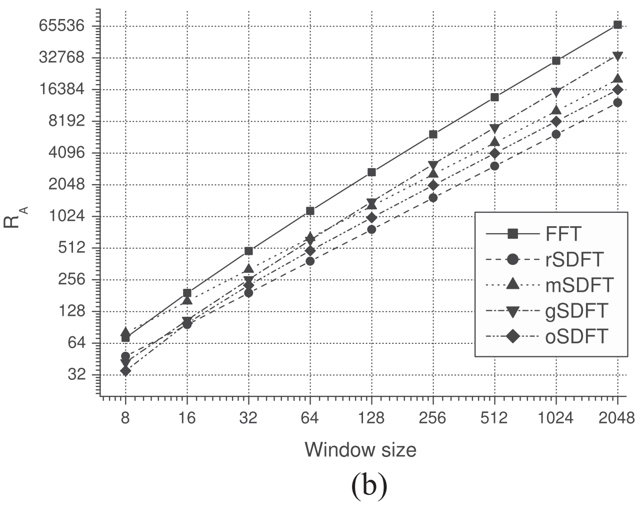


Fig. 3. Computational comparison with varying window sizes. (a) R_M . (b) R_A .

TABLE III
ADDITIONAL MEMORY REQUIREMENT FOR PERFORMING LENGTH-M TRANSFORM

Algorithm	Memory requirement
FFT	$M\log_2 M - M$
rSDFT	$2M$
mSDFT	$2M$
gSDFT	$M\log_2 M + M$
oSDFT	$M\log_2 M + 3M/2 - 8$

the oSDFTis the lowest among the existing algorithms regardless of window sizes. Therefore, for applications that need to perform the sliding DFT, we strongly recommend the use of the oSDFT algorithm.

Further, we present the memory requirements of all the algorithms in Table III. For each algorithm, we examine the amount of memory required for performing the length-M transform repeatedly. For the sake of the clarity, the memory to store the input and output signals is not considered. Table III shows that the FFT, gSDFT, and oSDFT algorithms need a relatively large amount of memory as compared to the rSDFT and mSDFT algorithms. Note that, in the sliding transform scenario, the size of the transform window is usually much smaller than those of the input and output signals. Therefore, in general, the memory overhead of the sliding transform algorithms may not be a big burden for real-world applications.

It is natural that sliding transform algorithms including oSDFT can be adopted only when the output bins of previous windows can be used in the computation process of the current window. For the applications where the precalculated output bins cannot be accessed, traditional fast FFT algorithms [32]–[35] should be used instead of the sliding transform algorithms.

Note that, even when only a single output bin needs to be computed, the gSDFT and oSDFT algorithms should compute all the butterflies related to the particular output bin. On the contrary, the rSDFT and mSDFT that do not exploit the butterfly structure obtain each output independently. Therefore, in this case, the rSDFT and mSDFT are preferred.

VI. Experimental Results

We evaluated the efficiency of the proposed oSDFT by comparing it with existing sliding DFT algorithms including the FFT [29], rSDFT [23], mSDFT [24], and gSDFT [26]. We first investigated the numerical errors generated by the sliding DFT algorithms. Next, we measured the processing times of the algorithms to demonstrate the practical usage of the algorithms. In our simulations, we used a complex test signal which was zero-mean Gaussian noise with a standard deviation equal to one. The simulation was performed in 64-bit double-precision floating-point arithmetic and the window size was set to 16 and 32 [36]–[38]. All algorithms were implemented using a highly efficient ANSI-C code and the performance was evaluated on an Intel i7 3.4GHz CPU with 16 GB RAM.

A. Numerical Error Analysis

We analyze the numerical errors of the algorithms by measuring their accumulated errors generated in the sliding transform process. Similar to the simulations in [26], in order to accumulate the numerical errors, we repeat the sliding transform 10^6 times by moving the transform window one sample at a time. After the error accumulation, we calculate the numerical errors at each time index. The numerical error E_n at time index n is calculated as

$$E_n = \sum_{k=0}^{M-1} |X_n^{FFT}(k) - X_n^{Algorithm}(k)| \tag{24}$$

where $X_n^{FFT}(k)$ represents the k-th bin of the standard FFT [29] and $X_n^{Algorithm}(k)$ is the k-th bin of each algorithm. To present the results clearly, we compute the average numerical error $\bar{E}_n$ over 64 time indexes after the error accumulation process:

$$\bar{E} = \frac{\sum_{n=10^6}^{n=10^6+63} E_n}{64}. \tag{25}$$

Table IV lists the measured results of all algorithms. Table IV shows that the mSDFT, gSDFT, and oSDFT algorithms significantly reduce the numerical errors as compared to the rSDFT,

TABLE IV
NUMERICAL ERRORS ($\bar{E}$'S) OF THE STABLE SDFT ALGORITHMS

Algorithm	Window size (M)	
	16	32
rSDFT	1.05×10^{-7}	6.99×10^{-7}
mSDFT	7.37×10^{-12}	2.29×10^{-11}
gSDFT	4.75×10^{-12}	8.80×10^{-12}
oSDFT	4.75×10^{-12}	8.80×10^{-12}
Average	2.40×10^{-8}	1.55×10^{-7}

TABLE V
PROCESSING TIMES OF THE EXISTING ALGORITHMS FOR THE COMPLEX TEST SIGNAL OF LENGTH 10^6

Algorithm	Window size (M)	
	16	32
FFT	71.54 ms	173.27 ms
rSDFT	39.39 ms	80.53 ms
mSDFT	61.69 ms	123.65 ms
gSDFT	36.63 ms	82.19 ms
oSDFT	32.85 ms	74.48 ms
Average	48.43 ms	106.83 ms

where the damping factor r of the rSDFT is set to 0.9999999999. In our simulations, when the window size is equal to 16, $\bar{E}$'s of the rSDFT, mSDFT, gSDFT, and oSDFT are 1.05×10^{-7}, 7.37×10^{-12}, 4.75×10^{-12}, and 4.75×10^{-12}, respectively. Further, E's of all algorithms increase as the window size increases. For $M = 32$, $\bar{E}$'s of the rSDFT, mSDFT, gSDFT, and oSDFT are 6.99×10^{-7}, 2.29×10^{-11}, 8.80×10^{-12}, and 8.80×10^{-12}, respectively. In all simulations, we observe that the error of the oSDFT is the same as that of the state-of-the-art gSDFT. This is because the oSDFT and gSDFT adopts the same recurrence computation structure which efficiently suppresses the error accumulation. In Table IV, the errors of the oSDFT and gSDFT are consistently smaller than those of the rSDFT and mSDFT. This leads to the result that the oSDFT and gSDFT outperform the other algorithms in terms of computational accuracy.

B. Processing Time Comparison

We next measure the processing time of each algorithm when all DFT bins are computed. The processing times are measured using the complex test signal of length 10^6 and the results are presented in Table V. The proposed algorithm can achieve significant time savings as compared to the FFT, rSDFT, mSDFT, and gSDFT algorithms. We compute the processing time savings of the proposed oSDFT as compared to the existing algorithms as

$$\text{Time saving}(\%) = \frac{T_{algorithm} - T_{oSDFT}}{T_{algorithm}} \times 100 \qquad (26)$$

where T_{oSDFT} denotes the measured processing time of the oSDFT and $T_{algorithm}$ denotes the measured time of each algorithm. For the window size 16, the processing time savings of the oSDFT are 54.08%, 16.60%, 46.75%, and 10.21% as compared to the FFT, rSDFT, mSDFT, and gSDFT algorithms, respectively. In Table V, we can see that the processing time of the oSDFT is even shorter than that of the gSDFT. This means that the proposed fast implementation scheme for the last decimation stage efficiently reduces the computation load of the sliding transform.

In our implementation, when the window size is equal to 32, the processing time of the gSDFT is slight higher than of the rSDFT. On contrary, the oSDFT consistently outperforms the other algorithms in terms of processing time. In this case ($M = 32$), the proposed oSDFT accelerates the sliding transform process by 57.02%, 7.51%, 39.77%, and 9.38% as compared to the FFT, rSDFT, mSDFT, and gSDFT algorithms, respectively. Indeed, the proposed method performs better than the other algorithms consistently. In our simulation, the processing time of the mSDFT is relatively longer than the other algorithms including the rSDFT, gSDFT, and oSDFT. It is observed in Table V that all the sliding SDFT algorithms consistently perform better than the FFT for both the window sizes of 16 and 32.

VII. CONCLUSIONS

A new stable SDFT algorithm with reduced computational complexity was presented for the fast implementation of the DFT on sliding windows. We first analyzed the recursive relationship between the DFT bins of adjacent windows. Then, on the basis of our analysis, we proposed a fast computational algorithm that recursively calculates the DFT bins of the shifted window by simply adding (or subtracting) the bins of a previous window and an updating vector. Through the simulations, we demonstrated that the proposed method consistently outperforms the other algorithms in terms of computational accuracy and processing time.

REFERENCES

[1] J. R. Carvalho, C. A. Duque, M. A. A. Lima, and D. V. Coury, "A novel DFT-based method for spectral analysis under time-varying frequency conditions," *Electr. Power Syst. Res.*, vol. 108, pp. 74–81, 2014.
[2] H. Malvar, "A modulated complex lapped transform and its applications to audio processing," in *Proc. IEEE Int. Conf. Acoust. Speech, Signal Process.*, 1999, pp. 1421–1424.
[3] G. B. Giannakis, F. Bach, R. Cendrillon, M. Mahoney, and J. Nevillel,"Signal processing for big data," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 15–16, Sep. 2014.
[4] G. Tzimiropoulos, V. Argyriou, S. Zafeiriou, and T. Stathaki, "Robust FFT-based scale-invariant image registration with image gradients," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 10, pp. 1899–1906, Oct. 2010.
[5] J. Tian and K.-K. Ma, "A survey on super-resolution imaging," *Signal, Image Video Process.*, vol. 5, no. 3, pp. 329–342, 2011.
[6] B. Abraham, I. C. Octavia, and M. Sznaier, "Dynamic texture with Fourier descriptors," in *Proc. 4th Int. Workshop Texture Anal. Synthesis*, 2008, pp. 53–58.
[7] P.-L. Shui, "Image denoising using 2-D separable oversampled DFT modulated filter banks," *IET Image Process.*, vol. 3, no. 3, pp. 163–173, 2009.
[8] A. Naftel and S. Khalid, "Classifying spatiotemporal object trajectories using unsupervised learning in the coefficient feature space," *Multimedia Syst.*, vol. 12, no. 3, pp. 227–238, 2006.
[9] R. A., Jr, and C. H. Q. Forster, "Experimental comparison of DWT and DFT for trajectory representation," *Lecture Notes Comput. Sci.*, vol. 7435, pp. 670–677, 2012.
[10] C. S. S. Burrus and T. Parks, *DFT/FFT and Convolution Algorithms: Theory and Implementation.* Hoboken, NJ, USA: Wiley, 1991.
[11] Discrete Fourier transform. 2009. [Online]. Available: https://en.wikipedia.org/wiki/Discrete_Fourier_transform

[12] S. Baig and M. J. Mughal, "Performance comparison of DFT, discrete wavelet packet and wavelet transforms, in an OFDM transceiver for multipath fading channel," in *Proc. IEEE Int. Multitopic Conf.*, 2005, pp. 1–5.

[13] C.-S. Park, "Recursive algorithm for sliding Walsh Hadamard transform," *IEEE Trans. Signal Process.*, vol. 62, no. 11, pp. 2827–2836, Jun. 2014.

[14] G. Luzhnica *et al.*, "A sliding window approach to natural hand gesture recognition using a custom data glove," in *Proc. IEEE Symp. 3D User Interfaces*, Mar. 2016, pp. 81–90.

[15] R. K. Vasudevan *et al.*, "Big data in reciprocal space: Sliding fast Fourier transforms for determining periodicity," *Appl. Phys. Lett.*, vol. 106, no. 9, 2015, Art. no. 091601.

[16] K. Y. Byun *et al.*, "Vector radix 2×2 sliding fast Fourier transform," *Math. Problems Eng.*, vol. 2016, 2016, Art. no. 2416286.

[17] B. Farhang-Boroujeny and Y. Lim, "A comment on the computational complexity of sliding FFT," *IEEE Trans. Circuits Syst. II*, vol. 39, no. 12, pp. 875–876, Dec. 1992.

[18] B. Farhang-Boroujeny and S. Gazor, "Generalized sliding FFT and its application to implementation of block LMS adaptive filters," *IEEE Trans. Signal Process.*, vol. 42, no. 3, pp. 532–538, Mar. 1994.

[19] T. Springer, "Sliding FFT computes frequency spectra in real time," *EDN Mag.*, vol. 33, pp. 161–170, Sep. 1988.

[20] E. Jacobsen and R. Lyons, "An update to the sliding DFT," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 110–111, Jan. 2004.

[21] K. Banks, "The Goertzel algorithm," *Embedded Syst. Program. Mag.*, vol. 15, no. 9, pp. 34–42, Sep. 2002.

[22] S. Douglas and J. Soh, "A numerically-stable sliding-window estimator and its application to adaptive filters," in *Proc. 31st Annu. Asilomar Conf. Signals, Systems, Comput.*, Pacific Grove, CA, USA, Nov. 1997, vol. 1, pp. 111–115.

[23] E. Jacobsen and R. Lyons, "The sliding DFT," *IEEE Signal Process. Mag.*, vol. 20, no. 2, pp. 74–80, Mar. 2003.

[24] K. Duda, "Accurate, guaranteed stable, sliding discrete Fourier transform," *IEEE Signal Process. Mag.*, vol. 27, no. 6, pp. 124–127, Nov. 2010.

[25] C. S. Park and S. J. Ko, "The hopping discrete Fourier transform," *IEEE Signal Process. Mag.*, vol. 31, no. 2, pp. 135–139, Mar. 2014.

[26] C. S. Park, "Fast, accurate, and guaranteed stable sliding discrete Fourier transform," *IEEE Signal Process. Mag.*, vol. 32, no. 4, pp. 145–156, Jul. 2015.

[27] A. van der Byl and M. R. Inggs, "Constraining errorA sliding discrete Fourier transform investigation," *Digit. Signal Process.*, vol. 51, pp. 54–61, 2016.

[28] P. Duhamel and M. Vetterli, "Fast Fourier transforms: A tutorial review and a state of the art," *Signal Process.*, vol. 19, no. 4, pp. 259–299, 1990.

[29] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed. Englewood-Cliffs, NJ, USA: Prentice-Hall 1999, p. 635.

[30] C. S. Park, "2D discrete Fourier transform on sliding windows," *IEEE Trans. Signal Process.*, vol. 24, no. 3, pp. 901–907, Mar. 2015.

[31] H. Li, T. Jiang, and Y. Zhou, "A novel subblock linear combination scheme for peak-to-average power ratio reduction in OFDM systems," *IEEE Trans. Broadcast.*, vol. 58, no. 3, pp. 360–369, Sep. 2012.

[32] P. Duhamel, "Implementation of Split-radix FFT algorithms for complex, real, and real-symmetric data," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 34, no. 2, pp. 285–295, Apr. 1986.

[33] J. Wang, C. Xiong, K. Zhang, and J. Wei, "Fixed-point analysis and parameter optimization of the radix-pipelined FFT processor," *IEEE Trans. Signal Process.*, vol. 63, no. 18, pp. 4879–4893, Sep. 2015.

[34] K. Li, W. Zheng, and J. Wei, "A fast algorithm with less operations for length-$N = q \times 2^m$ DFTs," *IEEE Trans. Signal Process.*, vol. 63, no. 3, pp. 673–683, Feb. 2015.

[35] S. G. Johnson and M. Frigo, "A modified split-radix FFT with fewer arithmetic operations," *IEEE Trans. Signal Process.*, vol. 55, no. 1, pp. 111–119, Jan. 2007.

[36] *IEEE Standard for Floating-Point Arithmetic*, ANSI/IEEE Standard 754-2008, Aug. 2008.

[37] N. Shirazi, A. Walters, and P. Athanas, "Quantitative analysis of floating point arithmetic on FPGA based custom computing machines," in *Proc. IEEE Symp. FPGAs Custom Comput. Mach.*, 1995, pp. 155–162.

[38] L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEE single precision floating point addition and multiplication on FPGAs," in *Proc. 83rd IEEE Symp. FPGAs Custom Comput. Mach.*, 1996, pp. 107–116.

Chun-Su Park received the B.S. and Ph.D. degrees in electrical engineering from Korea University, Seoul, South Korea, in 2003 and 2009, respectively. From 2009 to 2010, he was a Visiting Scholar in the Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, USA, and he was a Senior Research Engineer at the Samsung Electronics from 2010 to 2012. From 2012 to 2014, he was an Assistant Professor in the Department of Information & Telecommunications Engineering, Sangmyung University, Seoul, South Korea. From 2014 to 2016, he was an Associate Professor in the Department of Digital Contents, Sejong University, Seoul, South Korea. In 2017, he joined the Department of Computer Education, Sungkyunkwan University, Seoul, South Korea. His research interests include the areas of video signal processing, parallel computing, and multimedia communications.