Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 2416286, 8 pages
http://dx.doi.org/10.1155/2016/2416286

Research Article

Vector Radix 2 × 2 Sliding Fast Fourier Transform

Keun-Yung Byun,[1] Chun-Su Park,[2] Jee-Young Sun,[1] and Sung-Jea Ko[1]

[1]*School of Electrical Engineering Department, Korea University, 145 Anam-ro, Sungbuk-gu, Seoul 02841, Republic of Korea*
[2]*Department of Digital Contents, Sejong University, 209 Neungdong-ro, Gwangjin-gu, Seoul 05006, Republic of Korea*

Correspondence should be addressed to Sung-Jea Ko; sjko@korea.ac.kr

Received 11 August 2015; Revised 16 December 2015; Accepted 20 December 2015

Academic Editor: Lotfi Senhadji

The two-dimensional (2D) discrete Fourier transform (DFT) in the sliding window scenario has been successfully used for numerous applications requiring consecutive spectrum analysis of input signals. However, the results of conventional sliding DFT algorithms are potentially unstable because of the accumulated numerical errors caused by recursive strategy. In this letter, a stable 2D sliding fast Fourier transform (FFT) algorithm based on the vector radix (VR) 2 × 2 FFT is presented. In the VR-2 × 2 FFT algorithm, each 2D DFT bin is hierarchically decomposed into four sub-DFT bins until the size of the sub-DFT bins is reduced to 2 × 2; the output DFT bins are calculated using the linear combination of the sub-DFT bins. Because the sub-DFT bins for the overlapped input signals between the previous and current window are the same, the proposed algorithm reduces the computational complexity of the VR-2 × 2 FFT algorithm by reusing previously calculated sub-DFT bins in the sliding window scenario. Moreover, because the resultant DFT bins are identical to those of the VR-2 × 2 FFT algorithm, numerical errors do not arise; therefore, unconditional stability is guaranteed. Theoretical analysis shows that the proposed algorithm has the lowest computational requirements among the existing stable sliding DFT algorithms.

1. Introduction

The two-dimensional (2D) discrete Fourier transform (DFT) has been widely used for spectrum analysis of 2D input signals in the field of signal processing. The vector radix (VR) 2 × 2 FFT [1] is one of the most practical approaches to performing the 2D DFT. The VR-2 × 2 FFT algorithm hierarchically decomposes each DFT bin into sub-DFT bins until the size of the DFT bins becomes 2 × 2. Because the DFT bins can be efficiently obtained from the sub-DFT bins using a butterfly structure, the computational cost of the DFT can be significantly reduced. Additionally, various VR FFT algorithms including VR-4 × 4, split VR, and VR-$2^2 \times 2^2$ have been introduced to further enhance the efficiency of the VR-2 × 2 FFT by more finely decomposing the DFT bins [2–5].

The existing VR-based 2D FFT algorithms are widely applied in various fields of signal processing with satisfactory computing speed. However, these algorithms have computational redundancies when the transform window is shifted to the next sample because the input signals in the previous and current windows overlap. To reduce the redundancies in the sliding window scenario, numerous algorithms have been introduced over the past several years. The sliding DFT (SDFT) algorithm significantly reduces the computational load of the 1D DFT for the shifted window using the circular shift property [6]. The hopping DFT (HDFT) applies the SDFT to the hopping window scenario; HDFT can reduce the number of complex multiplications and additions in intermediate calculations [7]. Recently, the 2D SDFT proposed in [8] further reduced the number of computations of the SDFT using recursive strategy for a 2D input signal. The moving FFT (MFFT) algorithm proposed in [9] introduced a fast implementation of the 2D FFT by updating the previously calculated FFT bins when the current window is shifted. It is noteworthy that the recursive strategy adopted in the aforementioned sliding algorithms significantly reduces the number of complex multiplications and additions. However, the recursive implementation causes error propagation. Because, in practice, the complex numbers used in the DFT are represented in floating-point format with finite precision, the output DFT bins of the sliding algorithm and those of the traditional algorithm are not the same, although

they are mathematically equivalent. Furthermore, the errors accumulate and, thus, can increase in the worst case.

To address this issue, several stable DFT algorithms have been proposed. The stable SDFT proposed in [10], called rSDFT, reduces the numerical error by multiplying the twiddle factor by the damping factor r, where $0 \ll r < 1$. In the modulated SDFT (mSDFT) [11], the unstable twiddle factor is multiplied recursively by the previous output DFT bin, which is considered to be the cause of numerical errors. Therefore, the mSDFT removes the twiddle factor from the feedback loop, thereby guaranteeing unconditional stability. Recently, the guaranteed-stable SDFT (gSDFT) was proposed for fast, stable DFT [12]. The gSDFT not only guarantees the stability by removing the twiddle factor from the feedback of the resonator but also reduces the computational requirements by adopting the butterfly structure of the traditional FFT algorithm to obtain the bins of the updating vector transform. Nevertheless, the output DFT bins still contain the numerical error because the recursive strategy is used. On the other hand, the sliding FFT (SFFT) algorithm introduced by Farhang-Borojueny and Gazor in [13] does not adopt the recursive strategy. The SFFT calculates the bins of the shifted window by exploiting delayed intermediate calculations of the previous window. Therefore, the SFFT has the advantage that numerical errors do not occur as the transform window slides. However, because the SFFT is designed to handle 1D input data, its computational complexity can be further reduced for the 2D input data.

In this letter, we propose a novel stable, fast 2D sliding FFT algorithm. Because the 2D butterfly structure used in the VR-2 × 2 FFT algorithm is comprised of the 1D butterfly structure, the proposed VR-2 × 2 SFFT adopts the concept of the SFFT to guarantee the stability. In addition, we consider the 2D sliding window to move in only one direction, that is, column-wise or row-wise, to reduce the computational complexity. The rest of this letter is organized as follows. In Section 2, we first analyze the computational relationship between the sub-DFT bins and the DFT bins of the VR-2 × 2 FFT. Then, we explain how the proposed VR-2 × 2 SFFT algorithm reduces the amount of computation in the sliding window scenario while guaranteeing the computational stability. Section 3 presents the performance of the VR-2 × 2 SFFT by analyzing its arithmetic complexity and stability, making a comparison with those of the conventional algorithms. Finally, conclusions are given in Section 4.

2. Proposed VR-2 × 2 SFFT Algorithm

Because the proposed VR-2 × 2 SFFT algorithm reduces the computational complexity of the VR-2 × 2 FFT in the sliding window situation, we first explain the structure of the VR-2 × 2 FFT. The VR-2 × 2 FFT can be derived only for an $N \times N$ data sequence, where N is an integer power of two. Let $x(c, r)$ and $X(k, l)$ denote the (c, r)th bin of the $N \times N$ input data and the (k, l)th bin of the $N \times N$ output DFT bins, respectively. Then, the $N \times N$-point DFT is defined as

$$X(k,l) = \sum_{c=0}^{N-1} \sum_{r=0}^{N-1} x(c,r) W_N^{ck+rl}, \tag{1}$$

where $k, l = 0, 1, \ldots, N-1$, and W_N is a twiddle factor equal to $e^{-j2\pi/N}$.

The VR-2 × 2 FFT algorithm first decomposes (1) into 2 × 2 partial sums as follows:

$$\begin{aligned} X(k,l) &= \sum_{c=0}^{N-1} \sum_{r=0}^{N-1} x(c,r) W_N^{ck+rl} \\ &= \sum_{i=0}^{1} \sum_{j=0}^{1} \sum_{m=0}^{N/2-1} \sum_{n=0}^{N/2-1} x(2m+i, 2n+j) W_N^{(2m+i)k+(2n+j)l} \\ &= \sum_{i=0}^{1} \sum_{j=0}^{1} W_N^{ik+jl} \\ &\quad \cdot \left\{ \sum_{m=0}^{N/2-1} \sum_{n=0}^{N/2-1} x(2m+i, 2n+j) W_{N/2}^{mk+nl} \right\} \\ &= \sum_{i=0}^{1} \sum_{j=0}^{1} W_N^{ik+jl} S_{ij}(k,l). \end{aligned} \tag{2}$$

Note that each partial sum denoted by $S_{ij}(k, l)$ is a 2D DFT of size $(N/2) \times (N/2)$.

$S_{ij}(k, l)$ has a period of $N/2$ along both k and l; that is,

$$\begin{aligned} S_{ij}(k,l) &= S_{ij}\left(k + \frac{N}{2}, l\right) = S_{ij}\left(k, l + \frac{N}{2}\right) \\ &= S_{ij}\left(k + \frac{N}{2}, l + \frac{N}{2}\right). \end{aligned} \tag{3}$$

Combining (2) and (3), the following relationship can be obtained:

$$\begin{aligned} &\begin{bmatrix} X(k,l) \\ X\left(k + \frac{N}{2}, l\right) \\ X\left(k, l + \frac{N}{2}\right) \\ X\left(k + \frac{N}{2}, l + \frac{N}{2}\right) \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} S_{00}(k,l) \\ W_N^k S_{10}(k,l) \\ W_N^l S_{01}(k,l) \\ W_N^{k+l} S_{11}(k,l) \end{bmatrix}, \end{aligned} \tag{4}$$

where $k, l = 0, 1, \ldots, (N/2) - 1$, and $W_N^{k+N/2} = -W_N^k$. The matrix relationship shows that the sub-DFT bins are shared to calculate multiple DFT bins, resulting in the VR-2 × 2 FFT algorithm having less computational complexity than 2D DFT. This relationship has generally been illustrated with the butterfly diagram, as shown in Figure 1.

It is obvious that the conventional butterfly diagram represents the computational relations between DFT bins and sub-DFT bins. However, because the butterfly diagram has been developed to illustrate the data flow of the 1D FFT, it is difficult to present both the 2D spatial position of the input

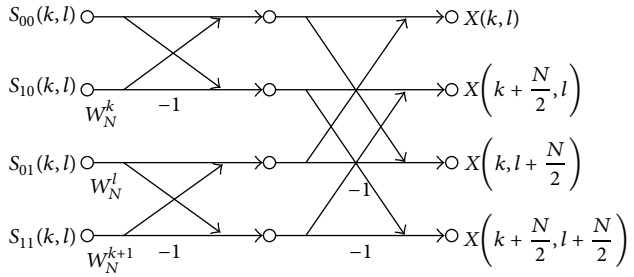

Figure 1: 2D butterfly diagram for the VR-2 × 2 FFT algorithm.

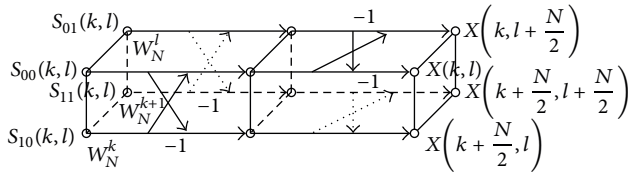

Figure 2: 3D butterfly diagram for the VR-2 × 2 FFT algorithm.

and output data and the spatial relations between DFT bins and sub-DFT bins. Although the index of each sub-DFT bin can indicate its 2D spatial location, the diagram becomes complicated as the amount of data increases.

For visual comprehension of the VR-2 × 2 FFT algorithm, we introduce a new 3D diagram, as shown in Figure 2.

Unlike for the conventional 2D butterfly diagram, the 2D spatial location of the input sub-DFT bins and the output DFT bins can be clearly illustrated, as in Figure 2. In addition, because the X-shaped pairs of arrows do not overlap with each other, it is easier to discriminate the relations between input and output DFT bins. However, the diagram still becomes complicated as the amount of data increases. To make the diagram simpler, let us omit the X-shaped arrows and twiddle factors W^l, W^k, and W^{k+l} from Figure 2. Then, the 3D diagram of the VR-2 × 2 FFT can be simplified as shown in Figure 3.

The decimation procedure repeats $\log_2 N$ times until the size of S_{ij} is reduced to 2 × 2. Let D_n denote the Nth decimation stage, where n is an integer in the range of $[1, \log_2 N]$, and the input data bins appear at stage D_1. The sizes of S_{ij} and X at stage D_n are $2^{n-1} \times 2^{n-1}$ and $2^n \times 2^n$, respectively. The twiddle factors multiplied with $S_{00}(k,l)$, $S_{01}(k,l)$, $S_{10}(k,l)$, and $S_{11}(k,l)$, $k,l = 0, 1, \ldots, 2^{n-1}$, at D_n are 1, W^l, W^k, and W^{k+l}, respectively. An example of the twiddle factors required for the 8 × 8-point VR-2 × 2 FFT is shown in Figure 4.

Now, let us consider the computational relationship between the input data and the values of $S_{ij}(k,l)$. As in (2), the VR-2 × 2 FFT algorithm computes the DFT bins using the decomposed 2 × 2 S_{ij}, which means that the input data give hierarchical dependencies to the output DFT bins through all decimation stages.

This technique can best be explained via an example. Figure 5 shows a flow graph of the 8 × 8-point VR-2 × 2 FFT algorithm. For the 8 × 8-point VR-2 × 2 FFT, three decimation

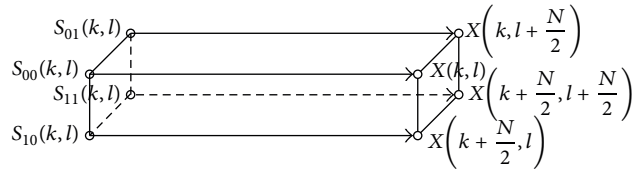

Figure 3: Simplified version of the 3D butterfly diagram.

stages, that is, D_1, D_2, and D_3, are required, and, thus, there are four data layers, as shown in Figure 5. Let L_i, $i = 1, 2, 3, 4$, denote the ith data layer, where L_1 and L_4 consist of the input data and the output DFT bins, respectively. Note that the input data are arranged in bit-reverse order. First, the 2 × 2 input data indicated by black circles in L_1 are linearly combined at stage D_1 to obtain the corresponding 2 × 2 DFT bins indicated by gray circles in L_2. Then, 4 × 4 data including the resultant 2 × 2 DFT bins in L_2 are used at stage D_2 to calculate 4 × 4 DFT bins indicated by gray circles in L_3. Finally, the 8 × 8 DFT bins in L_4 are obtained at stage D_3 using all of the data in L_3. Here, it is noteworthy that the data indicated by white circles in L_1 and L_2 are not involved in calculating the sub-DFT bins indicated by gray circles in L_2 and L_3.

Based on this observation, we derive the proposed algorithm, which can reduce the computational complexity of the VR-2 × 2 FFT in the sliding window scenario. Assume that all calculations in the structure shown in Figure 5 have already been performed for the previous data and that the results were stored. Furthermore, assume that the 8 × 8 window on the input data shifts by one column and that the same process must be performed at the current state.

Figure 6 shows 2D flow graphs of the VR-2 × 2 SFFT at both the previous and the current state by projecting the 3D flow graph in Figure 5 onto the column-layer plane. Denote a set of data of the jth column in the ith layer by l_i^j. Then, the number of data corresponding to l_i is 2^{i-1}. Figure 6(a) shows that the 8 × 8-point VR-2 × 2 FFT is performed using the input data from the 0th to the 7th column from the previous state. All of the calculated data in L_2 and L_3 represented by gray circles are stored. At the current state, the input data of the 0th column are removed from the window and those of the 8th column are newly included. After the input data of the current window are arranged in bit-reverse order, they are transformed using the 8 × 8-point VR-2 × 2 FFT.

Here, note that the data in {1st, 5th}, {3rd, 7th}, and {2nd, 6th} columns are paired, becoming the input of D_1 at the current state, as they were at the previous state. Then, the values of $\{l_2^1, l_2^5, l_2^3, l_2^7, l_2^2, l_2^6\}$ in L_2 and $\{l_3^1, l_3^5, l_3^3, l_3^7\}$ in L_3 at the previous state are the same as those at the current state, and, thus, they can be reused. As a result, only the data indicated by white circles in Figure 6(b) need to be calculated in the sliding scenario.

In addition, the computations for the data indicated by white circles can be further reduced, based on the fact that the 2D window is shifted column-wise. The structure of the 2D butterfly is composed of two stages, where each stage has two 1D butterflies, as shown in Figure 7.

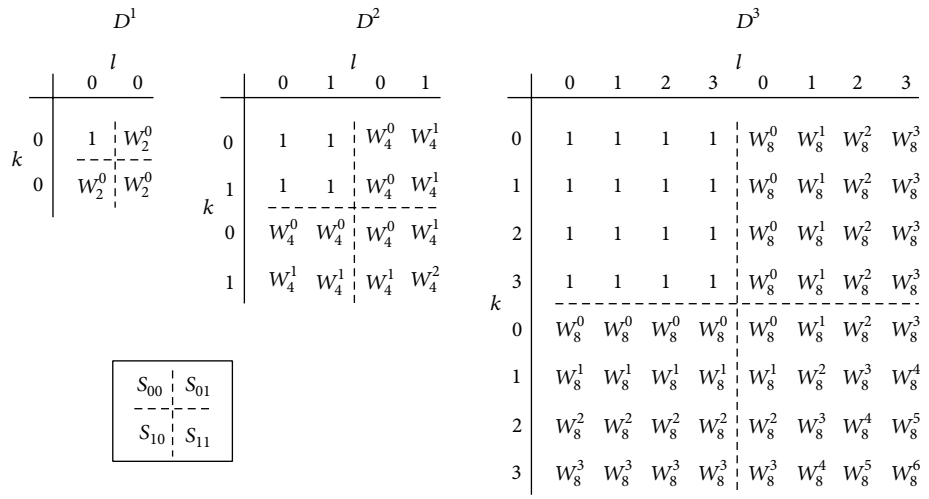

Figure 4: Twiddle factors at D_1, D_2, and D_3 for the 8 × 8-point VR-2 × 2 FFT algorithm.

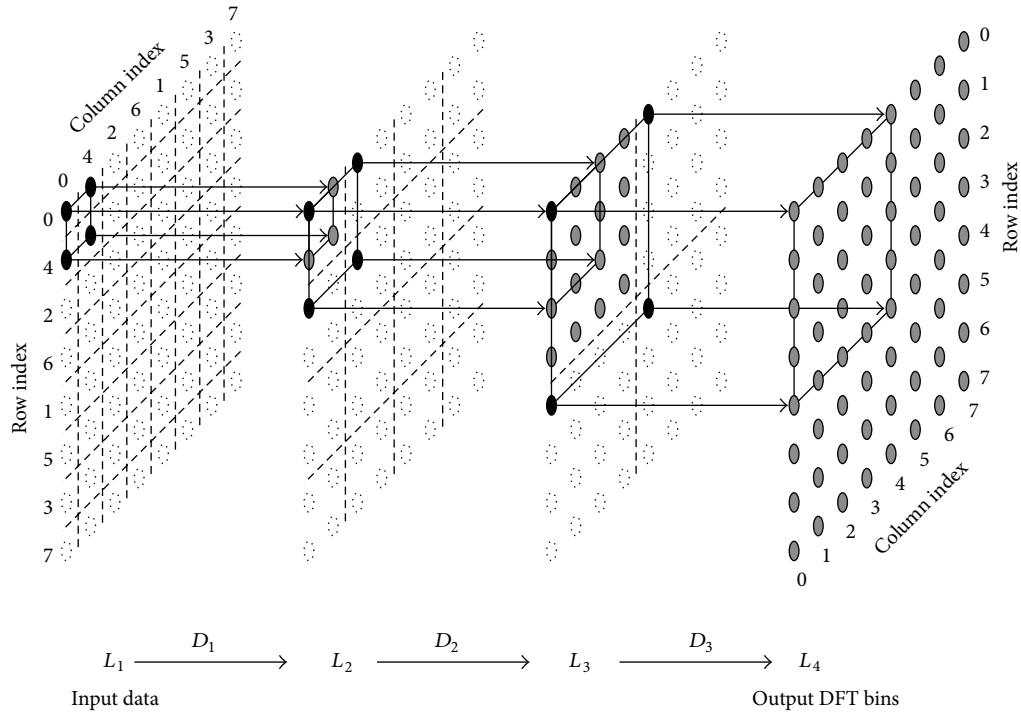

Figure 5: Flow graph of the 8 × 8-point VR-2 × 2 FFT algorithm.

In the first stage, two sets of the data $\{a, b\}$ and $\{c, d\}$ are the inputs of the single butterfly. The resultant $\{a', c'\}$ and $\{b', d'\}$ are used as the input to the butterflies in the second stage. Here, it can be observed that the data set $\{a, b\}$ is not used to calculate $\{c', d'\}$. This means that if the data $\{a', b'\}$ have already been calculated, they do not need to be recalculated. Then, the complexity of one complex multiplication (C_M) and two complex additions (C_A's) can be reduced.

In Figure 6(b), the input data of the 4th and 8th columns are used together as the input to stage D_1. Because the calculations for the data in the 4th column were already performed for the previous state, the results can be reused for the current state. Next, l_2^2 and l_2^6 are paired with l_2^4 and l_2^8, respectively, as the input at stage D_2, and the 2D butterfly is performed for each pair. Because l_2^2 and l_2^6 have also been calculated, the number of computations for the 2D butterflies can be reduced. The complexity reductions can be achieved at stage D_3 in the same manner.

Consequently, each 2D butterfly of the proposed algorithm requires only two C_M's and six C_A's, whereas that of the VR-2 × 2 FFT algorithm needs three C_M's and eight C_A's.

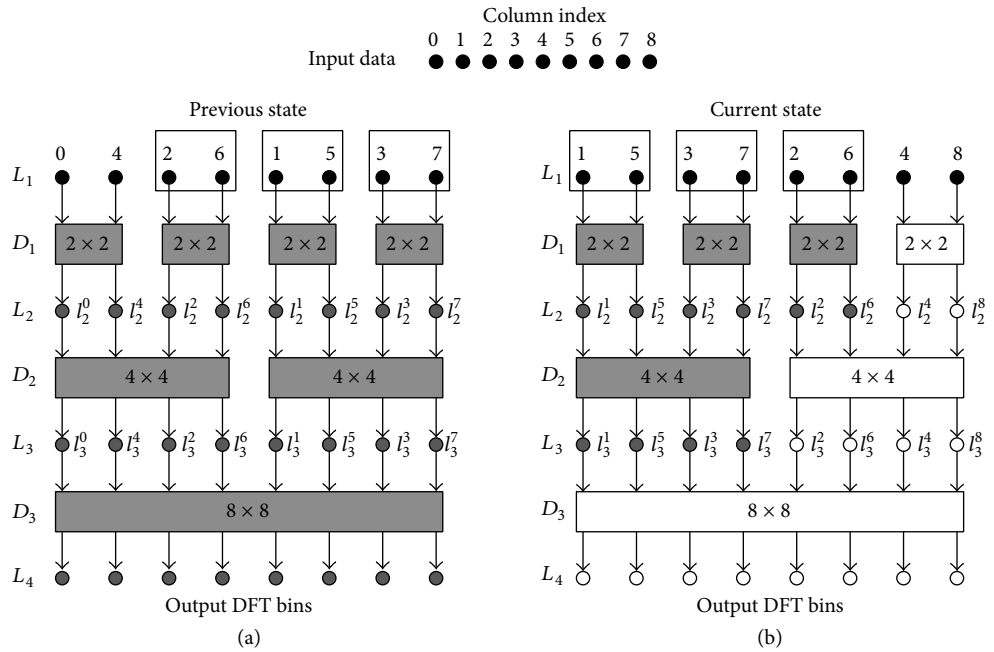

Figure 6: 2D flow graph of the 8 × 8-point VR-2 × 2 FFT algorithm at the previous and current state.

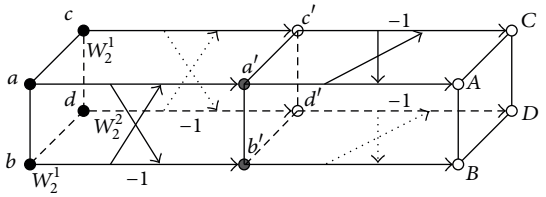

Figure 7: 2 × 2 butterfly diagram composed of two stages where each stage has two 1D butterflies.

Moreover, because the output DFT bins of the proposed algorithm are identical to those of the VR-2 × 2 FFT algorithm, numerical errors do not accumulate. Therefore, the proposed VR-2 × 2 SFFT algorithm can guarantee stability. Next, we analyze the computational complexity of the proposed VR-2 × 2 SFFT and compare it to that of the conventional algorithms.

3. Complexity and Stability Analysis of the VR-2 × 2 SFFT Algorithm

3.1. Complexity Analysis. For $N \times N$ input data, the VR-2 × 2 FFT is comprised of $\log_2 N$ decimation stages. Each decimation stage has $N^2/4$ 2 × 2 butterflies, and each 2 × 2 butterfly needs three C_M's and eight C_A's. Hence, for the $N \times N$ input data, the VR-2 × 2 FFT requires $3(N^2/4)\log_2 N$ C_M's and $8(N^2/4)\log_2 N$ C_A's. Among the butterflies in the structure of the VR-2 × 2 FFT, the 2 × 2 butterflies related to only the newly imported input data must be calculated in the VR-2 × 2 SFFT. For the $N \times N$ input data, the number of 2 × 2 butterflies calculated at D_n is $N \cdot 2^n/4$. Then, the total number of 2 × 2 butterflies required by the VR-2 × 2 SFFT is

$$\sum_{n=1}^{\log_2 N} N \cdot 2^{n-2} = \frac{N}{4} \sum_{n=1}^{\log_2 N} 2^n = \frac{N}{4} \cdot 2\,(N-1) = \frac{N\,(N-1)}{2}. \tag{5}$$

As mentioned in Section 2, a 2 × 2 butterfly needs two C_M's and six C_A's, and the VR-2 × 2 SFFT requires $N^2 - N$ C_M's and $3(N^2 - N)$ C_A's for $N \times N$ input data. Because the proposed VR-2 × 2 SFFT computes only some of the butterflies among those of the VR-2 × 2 FFT structure, the computational requirements of the proposed algorithm are lower than those of the VR-2 × 2 FFT algorithm.

A computational comparison of various window sizes is shown in Table 1, where one C_M is counted as four real multiplications (R_M's) and two real additions (R_A's) and one C_A is counted as two R_A's. The 2D MFFT [9], SFFT [13], and 2D SDFT [8] are chosen for the existing fast DFT/FFT algorithms. In addition, the rSDFT [10], mSDFT [11], and gSDFT [12] are chosen for the existing stable DFT algorithms. Considering that the SFFT, rSDFT, mSDFT, and gSDFT are proposed for the 1D input signal, we assume that each 1D transform is horizontally performed on the 2D input signal, and, then, the 1D FFT is vertically applied to the results. All the algorithms listed in Table 1 were implemented using the ANSI-C code and the performance was evaluated on a 3.3-GHz CPU with 8 GB of RAM. In our simulation, the size of transform window was set to 16 × 16 and we measured the processing time required for performing the sliding transform process 10^6 times. For each algorithm, we

Table 1: Computational requirements of the 2D DFT/FFT algorithms for $N \times N$ input data in the sliding window scenario.

Algorithm	Operation	Window size					
		4×4	8×8	16×16	32×32	64×64	$N \times N$
1D DFT ×2	R_M	512	4,096	32,768	262,144	2,097,152	$8N^3$
	R_A	448	3,840	31,744	258,048	2,080,768	$8N^3 - 4N^2$
1D FFT ×2	R_M	128	768	4,096	20,480	98,304	$4N^2\log_2 N$
	R_A	192	1,152	6,144	30,720	147,456	$6N^2\log_2 N$
1D SFFT + 1-D FFT	R_M	112	608	3,008	14,208	65,280	$2N^2\log_2 N + 4N^2 - 4N$
	R_A	168	912	4,512	21,312	97,920	$3N^2\log_2 N + 6N^2 - 6N$
VR-2 × 2 FFT	R_M	96	576	3,072	15,360	73,728	$3N^2\log_2 N$
	R_A	176	1,056	5,632	28,160	135,168	$(11/2)\, N^2\log_2 N$
2D MFFT	R_M	80	304	1,152	4,416	17,152	$4N^2 + 2N\log_2 N$
	R_A	92	336	1,232	4,608	17,600	$4N^2 + 3N\log_2 N + N$
2D SDFT	R_M	80	288	1,088	4,224	16,640	$4\left(N^2 + N\right)$
	R_A	78	282	1,074	4,194	16,578	$4N^2 + 3N + 2$
1D rSDFT + 1-D FFT	R_M	160	768	3,584	16,384	73,728	$6N^2 + 2N^2\log_2 N$
	R_A	192	960	4,608	21,504	98,304	$6N^2 + 3N^2\log_2 N$
1D mSDFT + 1-D FFT	R_M	256	1,152	5,120	22,528	98,304	$12N^2 + 2N^2\log_2 N$
	R_A	256	1,216	5,632	25,600	114,688	$10N^2 + 3N^2\log_2 N$
1D gSDFT + 1-D FFT	R_M	64	512	2,816	14,336	69,632	$3N^2\log_2 N - N^2$
	R_A	134	912	4,768	23,616	112,768	$(9/2)\, N^2\log_2 N + N^2/2 + 2N$
VR-2 × 2 SFFT	R_M	48	224	960	3,968	16,128	$4\left(N^2 - N\right)$
	R_A	96	448	1,920	7,936	32,256	$8\left(N^2 - N\right)$

measured the processing time 10 times and averaged the measured values. The resultant values are listed in Table 2. The complexity of the proposed VR-2 × 2 SFFT is higher than that of the 2D SDFT and 2D MFFT. However, the proposed algorithm achieves $O(N^2)$ complexity in both R_M and R_A, which is the lowest among the existing stable DFT algorithms.

Further, we present the memory requirements of all the algorithms in Table 2. For each algorithm, we examined the amount of required memory for performing the $N \times N$ sliding transform. For the sake of the clarity, the memory to store the input and output signals is not considered. In Table 2, we see that the memory requirements of the algorithms vary depending on the window size. In general, 2D SDFT and 2D MFFT need a relatively small amount of memory as compared to the other algorithms. Note that the size of the transform window is usually much smaller than those of the input and output signals. Therefore, in general, the amount of memory required to performing the sliding transform may be not a big burden for real-world applications.

3.2. Stability Analysis. We investigated the stability of the proposed VR-2 × 2 SFFT algorithm using a complex test signal, which was zero-mean Gaussian noise with a standard deviation of one. The simulation was performed in 64-bit double-precision arithmetic; N was set to 16. In our simulation, the numerical errors are generated by the recursive strategy of the sliding DFT/FFT algorithms. Therefore, we evaluate the stability of the algorithm by computing the differences between the output DFT bins of the sliding algorithm and those of the reference algorithm. The reference algorithm denotes the original algorithm from which the sliding algorithm was derived, for example, the reference algorithm of the VR-2 × 2 SFFT is the VR-2 × 2 FFT. All algorithms were tested over 10^6 iterations. The error E_n at time index n is calculated as

$$E_n = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} \left\| X_n^{\text{Reference}}(k,l) - X_n^{\text{Algorithm}}(k,l) \right\|, \quad (6)$$

where $X_n^{\text{Reference}}(k,l)$ represents the (k,l)th DFT bin of the reference algorithm and $X_n^{\text{Algorithm}}(k,l)$ is the bin of the test algorithm.

Then, it is observed that the errors in the 2D MFFT and the 1D rSDFT + 1D FFT are accumulated as n increases, as shown in Figure 8. For the rSDFT, the damping factor r is set to $1-10^{-12}$. In our simulation, the average E_n of the 2D MFFT is 4.41×10^{-9} and that of the 1D rSDFT + 1D FFT is 3.53×10^{-9}. In terms of the average increasing ratio of E_n, the 2D MFFT and 1D rSDFT + 1D FFT are 6.36×10^{-15} and 5.14×10^{-15}, respectively.

The measured E_n values of the 1D mSDFT + 1D FFT, 1D gSDFT + 1D FFT, and VR-2 × 2 SFFT are presented in Figure 9. The errors of the 1D mSDFT + 1D FFT and the 1D gSDFT + 1D FFT did not accumulate; however, they fluctuated in the range of $[1.08 \times 10^{-11}, 1.75 \times 10^{-11}]$ and $[1.11\times10^{-11}, 1.80\times10^{-11}]$, respectively. The average E_n values of the 1D mSDFT + 1D FFT and 1D gSDFT + 1D FFT are 1.425×10^{-11} and 1.418×10^{-11}, respectively. On the other hand, as mentioned in Section 2, the outputs of the proposed

Table 2: Comparison of the processing time and the additional memory requirements.

Algorithm	Processing time over 10^6 iterations for 16×16 size window (ms)	Ratio versus 1D DFT ×2 (%)	Additional memory requirement for $N \times N$-points transform
1D DFT ×2	23,908.47	100.00	$N^2 + 1$
1D FFT ×2	4,711.57	19.71	$N^2 + 1$
1D SFFT + 1-D FFT	3,582.43	14.98	$N^2 \log_2 N + N + 1$
VR-2 × 2 FFT	1,563.67	6.54	$2N^2 + 3$
2D MFFT	678.89	2.84	$N + 1$
2D SDFT	532.51	2.23	$2N + 1$
1D rSDFT + 1-D FFT	3,388.33	14.17	$N^2 + N + 1$
1D mSDFT + 1-D FFT	4,213.01	17.62	$N^2 + N + 2$
1D gSDFT + 1-D FFT	3,812.58	15.95	$(5/4)\, N^2 + N + 1$
VR-2 × 2 SFFT	876.87	3.67	$N^2 (2\log_2 N - 1) + 1$

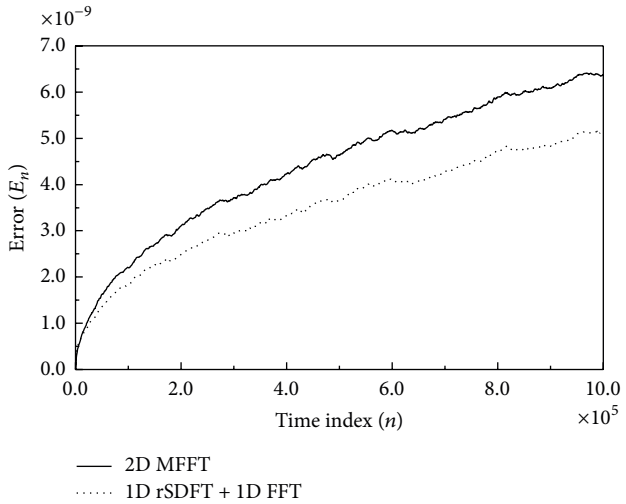

Figure 8: Numerical errors of the 2D MFFT and the 1D rSDFT + 1D FFT algorithm which accumulate over 10^6 iterations.

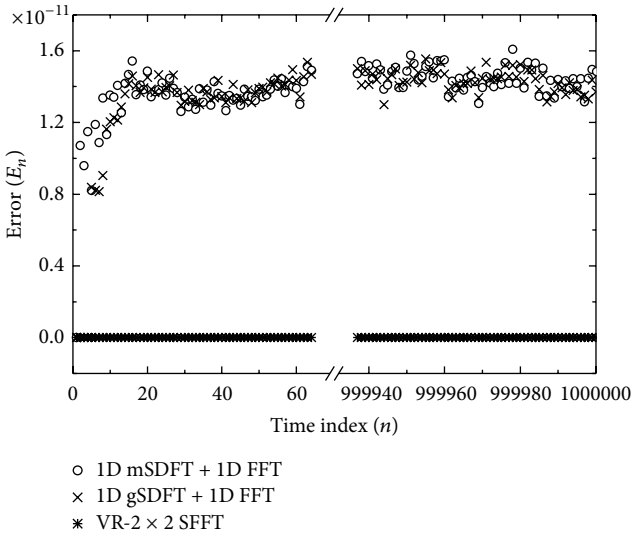

Figure 9: Numerical errors of the 1D mSDFT + 1D FFT and the VR-2 × 2 SFFT algorithm for 10^6 iterations.

algorithm are exactly the same as those of the VR-2 × 2 FFT, and errors do not accumulate. Therefore, the proposed VR-2 × 2 SFFT outperforms the other algorithms in terms of stability.

4. Conclusion

In this letter, a new stable SFFT based on the VR-2 × 2 FFT algorithm was presented for 2D input data. We first analyzed the computational relationship between the sub-DFT bins of the structure of the VR-2 × 2 FFT. Then, we adopt the concept of the 1D SFFT, which calculates the bins of the shifted window by exploiting the delayed intermediate calculations of the previous window. The proposed VR-2 × 2 SFFT algorithm achieves $O(N^2)$ complexity, which is the lowest among the existing stable DFT algorithms. Because the output DFT bins of the proposed method are exactly the same as those of the VR-2 × 2 FFT algorithm, the numerical errors do not accumulate in the sliding transform process.

Conflict of Interests

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgment

This work was supported by Institute for Information & Communications Technology Promotion (IITP) Grant funded by the Korean Government (MSIP) (no. B0101-15-0525; Development of Global Multi-Target Tracking and Event Prediction Techniques Based on Real-Time Large-Scale Video Analysis) and by Business for Cooperative R&D between Industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2015 (Grant no. C0276386).

References

[1] K. R. Rao, D. N. Kim, and J. J. Hwang, *Fast Fourier Transform—Algorithms and Applications*, Springer, 2011.

[2] H. R. Wu and F. J. Paoloni, "Structure of vector radix fast Fourier transforms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 9, pp. 1415–1424, 1989.

[3] H. R. Wu and F. J. Paoloni, "On the two-dimensional vector split-radix FFT algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 8, pp. 1302–1304, 1989.

[4] S. C. Chan and K. L. Ho, "Split vector-radix fast Fourier transform," *IEEE Transactions on Signal Processing*, vol. 40, no. 8, pp. 2029–2039, 1992.

[5] M. T. Hamood and S. Boussakta, "Vector-radix-$2^2 \times 2^2$ fast Fourier transform algorithm," in *Proceedings of the 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS '10)*, pp. 734–737, IEEE, Athens, Greece, December 2010.

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

[7] C.-S. Park and S.-J. Ko, "The hopping discrete fourier transform," *IEEE Signal Processing Magazine*, vol. 31, no. 2, pp. 135–139, 2014.

[8] C.-S. Park, "2D discrete Fourier transform on sliding windows," *IEEE Transactions on Image Processing*, vol. 24, no. 3, pp. 901–907, 2015.

[9] B. G. Sherlock and D. M. Monro, "Moving discrete Fourier transform," *IEE Proceedings, Part F: Radar and Signal Processing*, vol. 139, no. 4, pp. 279–282, 1992.

[10] E. Jacobsen and R. Lyons, "The sliding DFT," *IEEE Signal Processing Magazine*, vol. 20, no. 2, pp. 74–80, 2003.

[11] K. Duda, "Accurate, guaranteed stable, sliding discrete fourier transform," *IEEE Signal Processing Magazine*, vol. 27, no. 6, pp. 124–127, 2010.

[12] C. Park, "Fast, accurate, and guaranteed stable sliding discrete Fourier transform," *IEEE Signal Processing Magazine*, vol. 32, no. 4, pp. 145–156, 2015.

[13] B. Farhang-Borojueny and S. Gazor, "Generalized sliding FFT and its application to implementation of block LMS adaptive filters," *IEEE Transactions on Signal Processing*, vol. 42, no. 3, pp. 532–537, 1994.