Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT
Signal Processing 84 (2004) 561–570
SIGNAL PROCESSING
www.elsevier.com/locate/sigpro

A novel algorithm for computing the 2D split-vector-radix FFT

Hsuan-Yung Huang, Yu-Yun Lee, Pei-Chen Lo*

Department of Electrical and Control Engineering, National Chiao Tung University, 1001 Ta-Hsueh Road, Hsinchu, Taiwan, ROC

Received 21 February 2002; received in revised form 29 October 2003

Abstract

This paper presents a novel two-dimensional split-vector-radix fast-Fourier-transform (2D svr-FFT) algorithm. The modularizing feature of the 2D svr-FFT structure enables us to explore its characteristics by identifying the local structural property. Each local module is designated as a DFT (non-DFT) block if its output corresponds to DFT (non-DFT) values. The block attribute (DFT or non-DFT) directs the algorithm to construct the local module. We will show that the distribution of DFT blocks can be illustrated by the *Sierpinski triangle*—a class of fractals generated by IFS (iterated function system). The finding of the Sierpinski-triangle structural property enables us to actually implement the 2D svr-FFT algorithm. To the best of our knowledge, the 2D svr-FFT has never been realized in software. The computational efficiency of the proposed algorithm is considerably improved in comparison with that provided by Matlab.
© 2003 Elsevier B.V. All rights reserved.

Keywords: DFT; 2D split-vector-radix FFT (2D svr-FFT); Fractal; Sierpinski triangle

1. Introduction

Data analysis based on Fourier transform has been widely used in a large variety of fields. The rapid progress of digital technologies, including both hardware and software methodologies, gives rise to the field of digital signal processing (DSP). One of the major development in the DSP field is the discrete Fourier transform (DFT) that provides a frequency-domain representation of a finite extent sequence. Cooley and Tukey [8] disclosed an efficient algorithm for DFT computation, the radix-2 fast-Fourier-transform (FFT) algorithm. Since then, a variety of the FFT algorithms have been rapidly developed and implemented with software and hardware approaches [1,2,4,6,9,10,13,16–19,22–24,26–32,34]. The 1D split-radix FFT (sr-FFT) algorithm derived by Duhamel and Hollmann [9,10] was shown to have a simple structure with better computational efficiency. Algorithms for 1D implementation of sr-FFT have been well developed [9,10,16,17,23,26,28].

The MD (multi-dimensional) FFT becomes more important as more researches are focused on higher-dimensional signal processing, analysis and interpretation. The most popular 2D FFT algorithm is the row-column decomposition, actually, using 1D FFT algorithm [24]. An alternative approach, the vector-radix FFT, extends the 1D concept of decimation-in-time or decimation-in-frequency directly to the 2D case [12,14]. Nevertheless, both approaches are roughly equivalent in many respects. A few papers reported the mathematical concept,

* Corresponding author. Tel.: +886-3-573-1667; fax: +886-3-571-5998.
E-mail address: pclo@cc.nctu.edu.tw (P.-C. Lo).

0165-1684/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.sigpro.2003.11.018

computational efficiency and flexibility of the 2D svr-FFT structure [5,7,20,25,33]. Nevertheless, to our knowledge none of them brought out the algorithm for implementing the svr-FFT directly in 2D. It is the first attempt to explore the structural property of the 2D svr-FFT. In this paper, we name a local module "a DFT (non-DFT) block" if its output corresponds to the DFT (non-DFT) values. By identifying the attribute of each local module, we disclose the phenomenon that the distribution of the DFT blocks forms a fractal-like structure—the Sierpinski triangle. This structural property enables us to actually implement the svr-FFT algorithm in 2D and fully utilize its advantage of computational efficiency.

In the following section, we introduce the mathematical basis of 2D svr-FFT. In Section 3, we derive the structural geometry revealing the modular attributes. Aspects of the algorithm are discussed in Section 4.

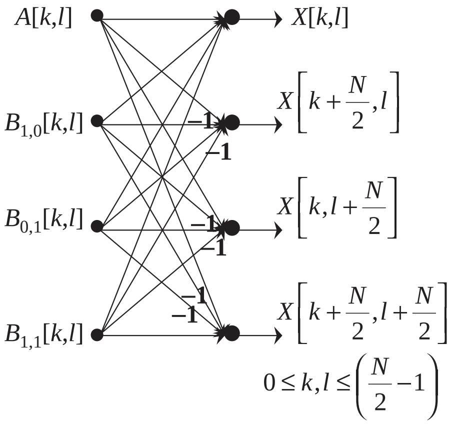
2. The 2D split-vector-radix FFT—mathematical basis

In this paper, we investigate the 2D decimation-in-space (DIS) svr-FFT structure, which can be directly extended to the development of the 2D DIF (decimation-in-frequency) svr-FFT. To optimize the advantage of svr-FFT, size of each dimension is selected to be the integer power of 2. This gives significant reduction in both additions and multiplications [32]. Let $x[m,n]$ be a 2D data array of size $N \times N$, where $N = 2^r$ is an integer power of 2. Its $N \times N$ 2D DFT, $X[k,l]$, is defined as [15]

$$X[k,l] = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x[m,n] W_N^{km} W_N^{ln}, \tag{1}$$

where $W_N = \mathrm{e}^{-\mathrm{j}(2\pi/N)}$. In the vector-radix FFT, an $N \times N$ 2D DFT can be computed by four $(N/2)\times(N/2)$ 2D DFTs. We obtain

$$\begin{aligned} X[k,l] = \sum_{n=0}^{N/2-1} \sum_{m=0}^{N/2-1} \{ & x[2m,2n] W_N^{2km} W_N^{2ln} \\ & + x[2m+1,2n] W_N^{2km+k} W_N^{2ln} \\ & + x[2m,2n+1] W_N^{2km} W_N^{2ln+l} \\ & + x[2m+1,2n+1] W_N^{2km+k} W_N^{2ln+l} \}, \end{aligned} \tag{2}$$



$A[k,l]$
$B_{1,0}[k,l]$
$B_{0,1}[k,l]$
$B_{1,1}[k,l]$
$X[k,l]$
$X\left[k+\frac{N}{2},l\right]$
$X\left[k,l+\frac{N}{2}\right]$
$X\left[k+\frac{N}{2},l+\frac{N}{2}\right]$
$0 \le k,l \le \left(\frac{N}{2}-1\right)$
−1
−1
−1
−1
−1
−1

Fig. 1. Flow graph of the 2D svr-FFT algorithm.

which can be expressed as

$$\begin{aligned} X[k,l] = {} & \mathrm{DFT}_{N/2\times N/2}\{x[2m,2n]\} \\ & + W_N^k \, \mathrm{DFT}_{N/2\times N/2}\{x[2m+1,2n]\} \\ & + W_N^l \, \mathrm{DFT}_{N/2\times N/2}\{x[2m,2n+1]\} \\ & + W_N^{k+l} \, \mathrm{DFT}_{N/2\times N/2}\{x[2m+1,2n+1]\}, \\ & 0 \leqslant n,m \leqslant \left(\frac{N}{2}-1\right). \end{aligned} \tag{3}$$

In the svr-FFT, the last three terms are further decomposed into $(N/4) \times (N/4)$ DFTs. The result can be expressed as

$$\begin{aligned} X[k,l] = {} & \mathrm{DFT}_{N/2\times N/2}(x[2m,2n]) + \mathrm{nonDFT}_{N/2\times N/2,1,0} \\ & + \mathrm{nonDFT}_{N/2\times N/2,0,1} + \mathrm{nonDFT}_{N/2\times N/2,1,1}, \end{aligned} \tag{4}$$

where

$$\begin{aligned} & \mathrm{nonDFT}_{N/2\times N/2,i,j} \\ & \quad = W_N^{ik+jl} \mathrm{DFT}_{N/4\times N/4}\{x[4m+i,4n+j]\} \\ & \qquad + W_N^{(i+2)k+jl} \\ & \qquad \times \mathrm{DFT}_{N/4\times N/4}\{x[4m+(i+2),4n+j]\} \\ & \qquad + W_N^{ik+(j+2)l} \\ & \qquad \times \mathrm{DFT}_{N/4\times N/4}\{x[4m+i,4n+(j+2)]\} \end{aligned}$$

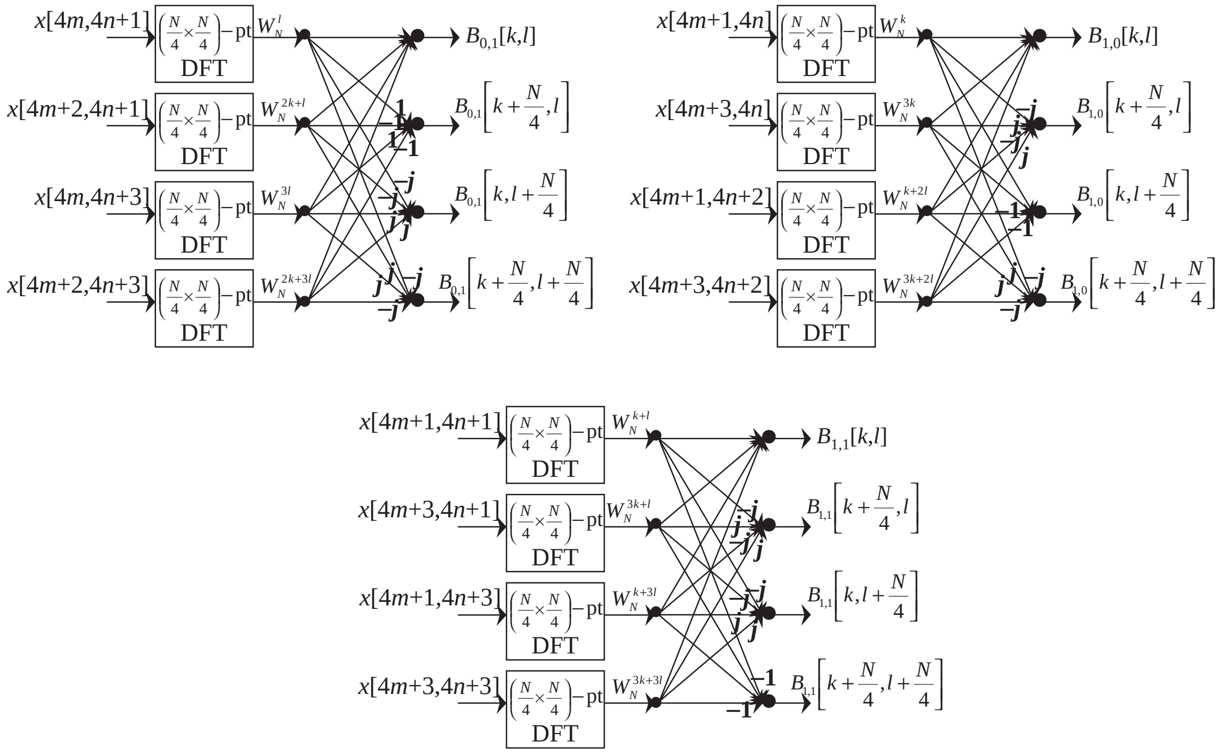


Fig. 2. Flow graph for computing the nonDFT samples from the $(N/4) \times (N/4)$-point DFTs.

$$+ W_N^{(i+2)k+(j+2)l} \times \mathrm{DFT}_{N/4\times N/4}\{x[4m+(i+2), 4n+(j+2)]\}. \tag{5}$$

Derivation of (5) is straightforward. For example, to derive an expression for the second term in (4), we further decompose the second term in (2) and obtain

$$\begin{aligned}
&\mathrm{nonDFT}_{N/2\times N/2,1,0} \\
&= \sum_{n=0}^{N/2-1}\sum_{m=0}^{N/2-1} x[2m+1,2n]W_N^{k(2m+1)}W_N^{l(2n)} \\
&= \sum_{n=0}^{N/4-1}\sum_{m=0}^{N/4-1} \{x[4m+1,4n]W_N^{k(4m+1)}W_N^{l(4n)} \\
&\quad + x[4m+3,4n]W_N^{k(4m+3)}W_N^{l(4n)} \\
&\quad + x[4m+1,4n+2]W_N^{k(4m+1)}W_N^{l(4n+2)} \\
&\quad + x[4m+3,4n+2]W_N^{k(4m+3)}W_N^{l(4n+2)}\} \\
&= W_N^{k}\,\mathrm{DFT}_{N/4\times N/4}\{x[4m+1,4n]\} \\
&\quad + W_N^{3k}\,\mathrm{DFT}_{N/4\times N/4}\{x[4m+3,4n]\} \\
&\quad + W_N^{k+2l}\,\mathrm{DFT}_{N/4\times N/4}\{x[4m+1,4n+2]\} \\
&\quad + W_N^{3k+2l}\,\mathrm{DFT}_{N/4\times N/4}\{x[4m+3,4n+2]\}.
\end{aligned} \tag{6}$$

To express the $N\times N$ array $X[k,l]$ by using the $(N/2)\times(N/2)$-point and $(N/4)\times(N/4)$-point DFT values, Eq. (4) is modified and illustrated in Fig. 1. The 2D array $A[*,*]$ represents the $(N/2)\times(N/2)$-point DFT samples. The other 2D arrays, $B_{0,1}[*,*]$, $B_{1,0}[*,*]$, and $B_{1,1}[*,*]$, correspond to the $(N/2)\times(N/2)$-point nonDFT samples. Fig. 2 shows the signal flow graph for obtaining the nonDFT samples from the $(N/4)\times(N/4)$-point DFT samples. Apparently, one DFT block and three nonDFT blocks constitute a DFT block. On the other hand, a nonDFT block is constructed from four DFT blocks.

Fig. 3 displays the distribution of DFT (dark) and nonDFT (light) blocks at the last four stages. The

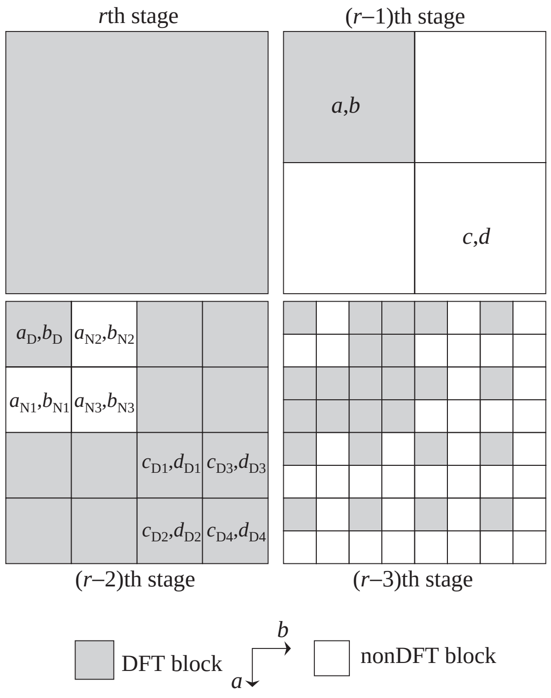


Fig. 3. Distribution of DFT (dark) and nonDFT (light) blocks at the last four stages.

variables a and b denote, respectively, the vertical and horizontal coordinates of a DFT/nonDFT block with respect to the origin which is located at the upper-left corner.

3. Derivation of structural geometry

As mentioned previously, main aim of this paper is to develop the algorithm for implementing the svr-FFT directly in 2D, that has never been realized before. Development of the svr-FFT algorithm based on (4) and (5) requires the knowledge of block attributes (DFT or nonDFT) at each stage. It is the first attempt to explore the structural geometry that characterizes the local block attributes. Let (a,b) denote the spatial position of a local block. The coordinates a and b are expressed in binary form. As illustrated in Fig. 3, the ith stage contains $2^{r-i} \times 2^{r-i}$ splitting blocks. Hence, $(r-i)$ binary digits are required to represent all the a's and b's at the ith stage. For example, at the $(r-3)$th stage, the svr-FFT structure splits into $8 \times 8 = 2^3 \times 2^3$ blocks, that is, $0 \leqslant a, b < 8$. Then coordinates $(a,b) = (110, 011)$ points to the block located at the 6th row and the 3rd column at the $(r-3)$th stage.

The DIS svr-FFT structure reveals some evidence that can be used to develop the criteria for identifying the block attributes at a given stage. Let symbol '$\vee$' represent the binary OR operation between two binary-coded variables. We define the following terms:

$$\mathrm{LSB}_l(\alpha)\text{: the first (lower-significant) } l \text{ bits of } \alpha, \tag{7a}$$

$$\mathrm{HSB}_k(\alpha)\text{: a binary value derived from the last (higher-significant) } k \text{ bits of } \alpha. \tag{7b}$$

That is, $\mathrm{LSB}_3(1011) = 011$ and $\mathrm{HSB}_3(1011) = 101$ (in binary code). The following lists a few relations between the block attributes and the block coordinates a and b.

R1: The last stage involves only one DFT block with $(a,b) = (0,0)$. Thus,

$$a \vee b = 0. \tag{8a}$$

R2: A DFT block (a,b) at the ith stage splits into one DFT block (a_D, b_D) and three nonDFT blocks: (a_{N1}, b_{N1}), (a_{N2}, b_{N2}), and (a_{N3}, b_{N3}), at the $(i-1)$th stage (refer to Fig. 3). It is apparent that

$$\mathrm{HSB}_{r-i}(a_D) = \mathrm{HSB}_{r-i}(a_{N1}) = \mathrm{HSB}_{r-i}(a_{N2}) = \mathrm{HSB}_{r-i}(a_{N3}) = a, \tag{8b}$$

$$\mathrm{HSB}_{r-i}(b_D) = \mathrm{HSB}_{r-i}(b_{N1}) = \mathrm{HSB}_{r-i}(b_{N2}) = \mathrm{HSB}_{r-i}(b_{N3}) = b. \tag{8c}$$

That is, the higher-significant $(r-i)$ bits of the four blocks used to construct the succeeding block (a,b) are the same as the binary-coded coordinates (a,b). And it could be found that

$$\mathrm{LSB}_1(a_D) = \mathrm{LSB}_1(b_D) = 0, \tag{8d}$$

$$\mathrm{LSB}_1(a_{N1}) = 1, \quad \mathrm{LSB}_1(b_{N1}) = 0, \tag{8e}$$

$$\mathrm{LSB}_1(a_{N2}) = 0, \quad \mathrm{LSB}_1(b_{N2}) = 1, \tag{8f}$$

$$\mathrm{LSB}_1(a_{N3}) = 1, \quad \mathrm{LSB}_1(b_{N3}) = 1, \tag{8g}$$

given that (a_{N1}, b_{N1}), (a_{N2}, b_{N2}), and (a_{N3}, b_{N3}) are, respectively, the lower-left, upper-right, and lower-right block (Fig. 3). Results in (8d) to (8g) imply

$$\mathrm{LSB}_1(a_D) \vee \mathrm{LSB}_1(b_D) = \mathrm{LSB}_1(a_D \vee b_D) = 0, \quad (8h)$$

$$\mathrm{LSB}_1(a_{N1} \vee b_{N1}) = \mathrm{LSB}_1(a_{N2} \vee b_{N2}) = \mathrm{LSB}_1(a_{N3} \vee b_{N3}) = 1. \quad (8i)$$

R3: A nonDFT block (c, d) at the ith stage splits into four DFT blocks (c_{Di}, d_{Di}), $i = 1, \ldots, 4$, at the $(i-1)$th stage (Fig. 3). Similarly,

$$\mathrm{HSB}_{r-i}(c_{Di}) = c, \quad (8j)$$

$$\mathrm{HSB}_{r-i}(d_{Di}) = d, \quad (8k)$$

for $i = 1, \ldots, 4$. However, the least-significant bits of c_{Di} and d_{Di} do not correlate with the block attributes since

$$\mathrm{LSB}_1(c_{D1} \vee d_{D1}) = 0, \quad (8l)$$

$$\mathrm{LSB}_1(c_{D2} \vee d_{D2}) = \mathrm{LSB}_1(c_{D3} \vee d_{D3}) = \mathrm{LSB}_1(c_{D4} \vee d_{D4}) = 1. \quad (8m)$$

We cannot obtain straightforward relations by observing only the least-significant bits of the four (DFT) blocks. Yet, the observation in R1, R2, and R3 may be extended. We then obtain two lemmas discussed as follows.

Lemma 1. *Consider a block (a, b) at any stage. If a and b are both even numbers, the block must be a DFT block. It can be expressed as*

$$\{(a,b) \mid \mathrm{LSB}_1(a \vee b) = 0\} \subset \textit{the set of DFT blocks.} \quad (9a)$$

It is because that nonDFT blocks must have coordinates satisfying (8i) according to the relation R2. For the case of $\mathrm{LSB}_1(a \vee b) = 1$, (a, b) may be a DFT or a nonDFT block. Lemmas 2 and 3 provide the criteria of identifying the block attributes for this case.

Lemma 2. *Consider a block $(a0, b0)$ at the ith stage, and $\delta = a0 \vee b0$. Let n*bit1 *be the number of consecutive bit* 1*'s counted from the least-significant (rightmost) bit position of δ. If n*bit1 *is even, $(a0, b0)$ is a DFT block, otherwise, a nonDFT block.*

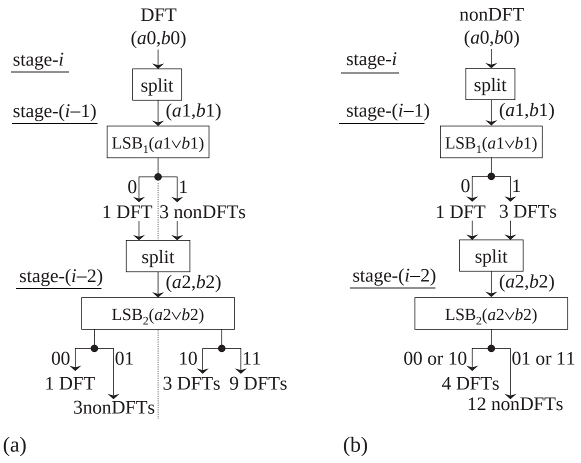


Fig. 4. Evolution of the lower-significant bits after two-stage split from a DFT (a) and a nonDFT (b) block.

Lemma 2 can be explained by observing three consecutive splitting stages, following the concepts introduced in relations R2 and R3. As shown in Fig. 4(a), a DFT block $(a0, b0)$ at the ith stage splits into 16 blocks $(a2, b2)$ at the $(i-2)$th stage with block attributes determined by

$$\mathrm{LSB}_2(a2 \vee b2) = \begin{cases} 00 \text{ or } 10 \text{ or } 11: \text{ DFT}, \\ 01: \text{ nonDFT}. \end{cases} \quad (9b)$$

On the other hand, a nonDFT block $(a0, b0)$ at the ith stage splits into 16 blocks $(a2, b2)$ at the $(i-2)$th stage with block attributes determined by

$$\mathrm{LSB}_2(a2 \vee b2) = \begin{cases} 00 \text{ or } 10: \text{ DFT}, \\ 01 \text{ or } 11: \text{ nonDFT}. \end{cases} \quad (9c)$$

Eqs. (9b) and (9c) support Lemma 1 since $\mathrm{LSB}_1(a2 \vee b2) = 0$ occurs only when $(a2, b2)$ is a DFT block. Note that $(a2, b2)$ must be a nonDFT block given $\mathrm{LSB}_2(a2 \vee b2) = 01$. Moreover, after the two-stage splitting, blocks with $\mathrm{LSB}_2(a2 \vee b2) = 11$ must be the DFT (nonDFT) ones given that $(a2, b2)$ is split from a DFT (nonDFT) block $(a0, b0)$ at the ith stage. The above can be summarized by

(1) $(a0, b0)$: DFT/nonDFT $\rightarrow (a2, b2)$: DFT,

if $\mathrm{LSB}_1(a2 \vee b2) = 0;$ (10a)

(2) $(a0, b0)$: DFT/nonDFT $\rightarrow (a2, b2)$: nonDFT,

if $\mathrm{LSB}_2(a2 \vee b2) = 01;$ (10b)

(3) $(a0, b0)$: DFT $\rightarrow$ $(a2, b2)$: DFT,

$$\text{if } \mathrm{LSB}_2(a2 \vee b2) = 11; \tag{10c}$$

(4) $(a0, b0)$: nonDFT $\rightarrow$ $(a2, b2)$: nonDFT,

$$\text{if } \mathrm{LSB}_2(a2 \vee b2) = 11; \tag{10d}$$

given $(a2, b2)$: a block at the $(i-2)$th stage split from the block $(a0, b0)$ at the ith stage. According to (10a), (10c) and the initial condition given in R1, recursive operation (repeated use) of the signal flow graph in Fig. 4(a) comes to a result that $n$bit1 is *even* for a DFT block. Similarly, we find that $n$bit1 is *odd* for a nonDFT block, according to (10b) and (10d) as well as the recursive operation of the signal-flow graph in Fig. 4(b).

Based on Lemmas 1 and 2, we define two mutually complementary sets of blocks:

$$\mathbf{B}_\mathrm{D}\colon \{(a,b) \mid \mathrm{LSB}_1(a \vee b) = 0 \text{ or } n\mathrm{bit1} \in \text{even integers}\}, \tag{11a}$$

$$\mathbf{B}_\mathrm{N}\colon \{(a,b) \mid n\mathrm{bit1} \in \text{odd integers}\}, \tag{11b}$$

where $n$bit1 represents the number of consecutive bit 1's counted from the least-significant (rightmost) bit position of $(a \vee b)$. Then we come to a conclusion given below

Theorem.

(a,b) is a DFT block if and only if $(a,b) \in \mathbf{B}_\mathrm{D}$;

(a,b) is a nonDFT block if and only if $(a,b) \in \mathbf{B}_\mathrm{N}$.

Note that $\mathbf{B}_\mathrm{D} \cap \mathbf{B}_\mathrm{N} = \phi$, $\mathbf{B}_\mathrm{D} \cup \mathbf{B}_\mathrm{N} = \mathbf{B}$: a set containing all the possible r-bit binary values. The above theorem provides a guideline to direct the 2D svr-FFT algorithm to correctly find and construct each local DFT block when proceeding stage by stage.

The above theorem holds, with a slight modification, for the higher dimensional svr-FFT's. Consider the case of 3D svr-FFT. A local block at a given stage is indexed by (a, b, c). Then, the set $\mathbf{B}_\mathrm{D}$ containing DFT blocks is defined by

$$\mathbf{B}_\mathrm{D}\colon \{(a,b,c) \mid \mathrm{LSB}_1(a \vee b \vee c) = 0 \text{ or } n\mathrm{bit1} \in \text{even integers}\} \tag{12a}$$

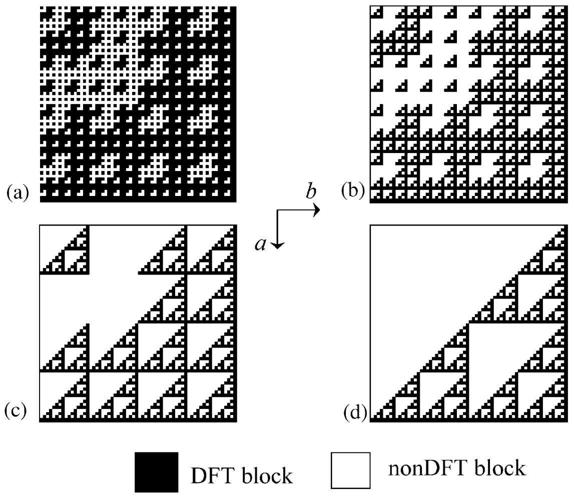


Fig. 5. (a) The complete DFT image ($\boldsymbol{S}$) at the $(r-6)$th stage. Partial DFT images with DFT blocks defined by the set: $\boldsymbol{S}_1 \cup \boldsymbol{S}_2 \cup \boldsymbol{S}_3$ (b), $\boldsymbol{S}_2 \cup \boldsymbol{S}_3$ (c), and $\boldsymbol{S}_3$ (d).

and the set $\mathbf{B}_\mathrm{N}$ containing nonDFT blocks is

$$\mathbf{B}_\mathrm{N}\colon \{(a,b,c) \mid n\mathrm{bit1} \in \text{odd integers}\}. \tag{12b}$$

Based on the theorem derived above, we can easily expand Fig. 3 and obtain an "image" of DFT blocks at any given stage. Fig. 5 plots the images of DFT blocks (the black boxes) at the $(r-6)$th stage. The plot will be called the "DFT image" later. As addressed previously, the $(r-6)$th stage contains 64×64 splitting blocks. The complete DFT image (Fig. 5(a)) displays all the DFT blocks to be constructed at this stage. These blocks locate at (a, b)'s satisfying

$$\begin{aligned} \boldsymbol{S} = \boldsymbol{S}_0\colon & \{(a_0,b_0) \mid a_0 \vee b_0 = \square\square\square\square\square 0\} \\ \cup\, \boldsymbol{S}_1\colon & \{(a_1,b_1) \mid a_1 \vee b_1 = \square\square\square 011\} \\ \cup\, \boldsymbol{S}_2\colon & \{(a_2,b_2) \mid a_2 \vee b_2 = \square 01111\} \\ \cup\, \boldsymbol{S}_3\colon & \{(a_3,b_3) \mid a_3 \vee b_3 = 111111\}. \end{aligned} \tag{13}$$

Note that '$\square$' represents the "don't-care" bit. It is because the $n$bit1 only counts the number of consecutive bit 1's. The set $\boldsymbol{S}_0$ contains 32×32 blocks located at the positions where both a_0 and b_0 are even numbers. In the next section, we will demonstrate that the set $\boldsymbol{S}_1$ ($\boldsymbol{S}_2, \boldsymbol{S}_3$) contains $4\times4\times4\times9$ ($4\times9\times9, 9\times9\times9$) blocks. Fig. 5(b) shows the leftover DFT blocks in $\boldsymbol{S}_1 \cup \boldsymbol{S}_2 \cup \boldsymbol{S}_3$ after constructing the DFT blocks in

$\boldsymbol{S}_0$. Figs. 5(c) and (d) display, respectively, the DFT blocks in $\boldsymbol{S}_2 \cup \boldsymbol{S}_3$ and $\boldsymbol{S}_3$. It is of great interest that the DFT images exhibit fractal structure—the well-known *Sierpinski triangle* [3,21]. This finding is not unique, Peitgen observed that the logistic NAND operation led to a fractal pattern [21].

4. The algorithm

Next, we propose an algorithm to directly implement the 2D svr-FFT based on the theorem derived in Section 3. One important task of the algorithm is to correctly and rapidly find out all the locations of DFT blocks at each stage. We develop a dynamic loop programming strategy of which the number of "for-loop" operations is variable and determined by the stage of svr-FFT structure.

Note that the logistic (binary) OR operation requires significant amount of computer time. According to our test run on the DX100-486PC, it requires approximately 0.17 s to search for all the block locations (a,b)'s satisfying $\{(a,b)\,|\,a \vee b = 1111111111 \text{ (or } n\text{bit1} = 10)\}$ by direct computation. In fact, by exploring some regularity in (a,b), we may greatly reduce the computational effort on performing the task. A scheme that results in a computational saving of at least two-thirds is first introduced as follows.

Let $\boldsymbol{S}_i$ denote a set containing the DFT blocks with block locations (a_i,b_i)'s specified by

$$\boldsymbol{S}_i\text{:}\ \begin{cases} \{(a_0,b_0)\,|\,\mathrm{LSB}_1(a_0 \vee b_0)=0\}, & i=0,\\ \{(a_i,b_i)\,|\,n\mathrm{bit1}=2i\}, & i\neq 0 \end{cases} \tag{14}$$

or defined more precisely as

$$\boldsymbol{S}_0\text{: } \{(a_0,b_0)\,|\,a_0 \vee b_0 = \square\ldots\square 0\}$$

$$\boldsymbol{S}_i\text{: } \{(a_i,b_i)\,|\,a_i \vee b_i = \square\ldots\square 011\ldots11\}, \quad i\neq 0, \tag{15}$$

where the symbol '□' represents the "don't-care" bit, that is, '□' may be either bit 0 or 1. In this case, all the four possible conditions: (0,0), (0,1), (1,0), and (1,1) are allowed at the corresponding bit position of a_i and b_i. The notation "11…11" represents $2i$ consecutive bit 1's. It is obvious that, once the DFT-block set $\boldsymbol{S}_1$: $\{(a_1,b_1)\,|\,n\text{bit1}=2 \text{ (or : } a_1 \vee b_1 = \square\ldots\square 011)\}$ is obtained, one would be able to obtain the sets $\boldsymbol{S}_2, \boldsymbol{S}_3, \ldots$ without exhaustedly evaluating the logistic

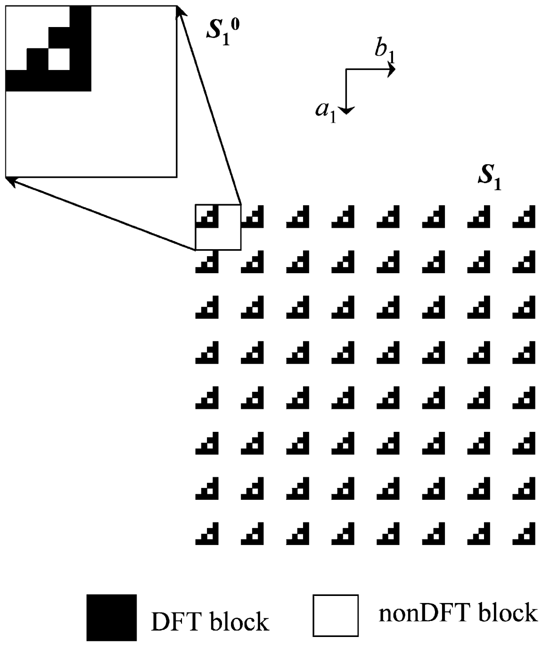


Fig. 6. The complete DFT image of $\boldsymbol{S}_1$ at the $(r-6)$th stage can be obtained by periodical repetition of the fundamental subset $\boldsymbol{S}_1^0$.

OR operations. Two strategies are to be applied in the searching task: (1) given that the fundamental subset of $\boldsymbol{S}_i$ (denoted by $\boldsymbol{S}_i^0$): $\{(a_i^0,b_i^0)\,|\,a_i^0 \vee b_i^0 = 0\ldots0011\ldots11\}$ is determined, the complete set $\boldsymbol{S}_i$ can be obtained by periodic repetition of the fundamental subset, and (2) once the fundamental subset $\boldsymbol{S}_1^0$: $\{(a_1^0,b_1^0)\,|\,a_1^0 \vee b_1^0 = 0\ldots0011\}$ is determined, the others $(\boldsymbol{S}_i^0, i>1)$ can be obtained straightforward. To illustrate these two strategies in details, we consider the problem of finding the DFT-block set $\boldsymbol{S}_2$: $\{(a_2,b_2)\,|\,a_2 \vee b_2 = \square 01111 \text{ (}n\text{bit1} = 4)\}$ at the $(r-6)$th stage. As addressed in Section 3, six bits are required to code the block indexes a_2 and b_2. The complete DFT set at this stage is defined in (13) and plotted in Fig. 5(a). We first determine the fundamental subset $\boldsymbol{S}_1^0$: $\{(a_1^0,b_1^0)\,|\,a_1^0 \vee b_1^0 = 000011\}$. As shown in Fig. 6, $\boldsymbol{S}_1^0$ contains 9 elements:

$$\{(000000,000011)\ (000001,000010)$$
$$(000001,000011)\ (000010,000001)$$
$$(000010,000011)\ (000011,000000)$$

$$(000011, 000001)\ (000011, 000010)$$

$$(000011, 000011)\}_{\text{binary}}$$

$$= \{(0,3)\ (1,2)\ (1,3)\ (2,1)\ (2,3)$$

$$(3,0)\ (3,1)\ (3,2)\ (3,3)\}_{\text{decimal}}. \quad (16)$$

According to the first strategy, the complete set $\boldsymbol{S}_1$ can be obtained by periodically reproducing the 8×8 fundamental pattern all over the 64×64 region. That is

$$\boldsymbol{S}_1\text{: } \{(a_1, b_1) \mid a_1 = 8i + a_1^0, b_1 = 8j + b_1^0\}, \quad (17)$$

where $0 \leqslant i, j \leqslant 7$ and $(a_1^0, b_1^0) \in \boldsymbol{S}_1^0$. Fig. 6 shows that the resulting DFT image includes 64 periods. Next, the second strategy allows us to deduce the fundamental subset $\boldsymbol{S}_i^0$ $(i > 1)$ from $\boldsymbol{S}_1^0$ without explicitly evaluating the logistic OR operation. It is based on the fact that "001111" can be viewed as "001100 + 000011". Moreover, the code "001100" is obtained by rotating "000011" leftwards by two bit positions (or, by multiplying the number 3×4). Let ξ_k be the set of nine (α, β)'s satisfying the condition "$\alpha \vee \beta = 0\ldots0110\ldots0$", where the code "$0\ldots0110\ldots0$" is obtained by rotating "$0\ldots011$" leftwards by $2k$ bit positions. Thus, a set ξ_1 with nine elements (α, β)'s satisfying the condition "$\alpha \vee \beta = 001100$" is ready to be obtained from (16) as

$$\xi_1\text{: } \{(0,12)\ (4,8)\ (4,12)\ (8,4)\ (8,12)\ (12,0)$$

$$(12,4)\ (12,8)\ (12,12)\}_{\text{decimal}}. \quad (18)$$

Hence, the set $\boldsymbol{S}_2^0$ can be derived straightforward from $\boldsymbol{S}_1^0$ as

$$\boldsymbol{S}_2^0\text{: } \{(a_2^0, b_2^0) \mid a_2^0 \vee b_2^0 = 001111\}$$

$$= \{(a_2^0, b_2^0) \mid a_2^0 = \alpha + a_1^0, b_2^0 = \beta + b_1^0\}, \quad (19)$$

where $(a_1^0, b_1^0) \in \boldsymbol{S}_1^0$ and $(\alpha, \beta) \in \xi_1$. Apparently, $\boldsymbol{S}_2^0$ contains 9×9 elements. The size of the fundamental period is 32×32 since the lower-significant five bits are used to address a_2^0 and b_2^0. Then, the complete set $\boldsymbol{S}_2$: $\{(a_2, b_2) \mid a_2 \vee b_2 = \square 01111\}$ can be obtained following the first strategy as below

$$\boldsymbol{S}_2\text{: } \{(a_2, b_2) \mid a_2 = 32i + a_2^0, b_2 = 32j + b_2^0\}, \quad (20)$$

where $0 \leqslant i, j \leqslant 1$ and $(a_2^0, b_2^0) \in \boldsymbol{S}_2^0$. Accordingly, $\boldsymbol{S}_2$ contains $4 \times 9 \times 9$ elements (DFT blocks).

Table 1
Computational time required by the proposed algorithm and Matlab for different FFT sizes

FFT size	2D svr-FFT	Matlab (fft2)
32×32	1.3 ms	3.8 ms
64×64	7.6 ms	16 ms
128×128	28 ms	75 ms
256×256	0.11 s	0.32 s
512×512	0.77 s	1.27 s
1024×1024	2.86 s	5.11 s
2048×2048	12.9 s	—

According to the above strategies, dynamic-loop programming is implemented to determine the locations of DFT blocks [11]. For example, two (three) for-loop's are required to determine $\boldsymbol{S}_2^0$ ($\boldsymbol{S}_3^0$). In the algorithm, the "for-loop" is implemented by the "if-then-else" and "go to" instruction. The pseudo-code program (Appendix) was developed according to the "dynamic-loop programming" code and the svr-FFT structure. In addition to the common 2D-DFT subroutine, the core of the computer program includes the dynamic-loop programming structure and the sr_table[][] tabulating the sequence of accomplishing DFT blocks in consideration of saving of time.

For comparison of computational efficiency, we experimented with the proposed 2D DIS svr-FFT algorithm run on a Pentium MMX-233 PC and compared with the Matlab. Note that the 2D-FFT function (fft2) in Matlab is based on the row-column decomposition scheme and 1D-FFT algorithm. The fft2 function in Matlab 5 offers a moderate and competitive reference for demonstrating the computational efficiency since, to the best of our knowledge, it provides the most efficient computation for 2D-DFT especially for an N that can be factorized as a product of small primes. As listed in Table 1, the computational time spent on the same task is significantly reduced in comparison with the time required by the Matlab 5. Apparently, computational time improves more for a smaller array dimension N since structural complexity increases with N, which requires extra computational loading. A speedup of approximately 1.75 times is obtained for a 1024×1024 2D-DFT.

5. Conclusion

In this paper, we have explored the realizable and programmable structure of 2D split-vector-radix FFT. A theorem was derived to identify the DFT blocks at a given stage. The distribution of DFT blocks (DFT image) was shown to exhibit fractal structure—the well-known *Sierpinski triangle*. The result enables us to develop an efficient algorithm that actually implements the concept of 2D svr-FFT. As addressed previously, the mathematical aspects and computational complexity (number of arithmetic operations) of 2D svr-FFT have been discussed previously [5,7,20,25,33]. Nonetheless, it is the first attempt, to the best of our knowledge, to realize and implement the svr-FFT computational structure actually in 2D to thoroughly utilize the arithmetic efficiency of 2D svr-FFT algorithm. The resulting 2D-FFT program provides a significant reduction in computational time in comparison with that required by the well-developed function provided by Matlab.

Acknowledgements

This work has been supported by the National Science Council of Taiwan (grant #: NSC90-2213-E-009-084) and the MOE Program for Promoting Academic Excellent of Universities (grant #: 91-E-FA06-4-4).

Appendix

Pseudo Code of the 2D dis-svr-FFT Algorithm

```
struct sr_table
    {
    int a;    int b;
    };   // define 2D array s[.][.] as sr_table type
    ⋮
// initialize sr_table[k][0...8]: tabulate the $\xi_k$ array
s[0][0].a = 0; s[0][0].b = 3;
s[0][1].a = 1; s[0][1].b = 2;
    ⋮
s[0][8].a = 3; s[0][8].b = 3;
s[1][0].a = 0; s[1][0].b = 12;
    ⋮
sr_FFT(......)
    {
    ⋮
    // bit reversal
    ⋮
    size = 2;
  for (stage = 1; stage < = r;stage++)   // 2D DFT
  size: N × N, where N = $2^r$
      {
      grid_size = size * 2;
      // implement $\{(a,b) \mid a \vee b = \square \ldots \square 0\}$
      DFT(0, 0, grid_size, size);
      grid_size * =4;
      loop_limit = (int)((r − stage)/2);
      for(loop = 0; loop < loop_limit; loop + +)
          {
          // dynamic loop programming
          k = loop;
          index[k] = 0;
          block_position[k] = sr_table[k][0];
   O1:
          if(index[k] > = 9) goto L1;
          // calculate block position
          if(k = =loop) block_position[k]=
    sr_table[k][index[k]];
                else block_position[k] = block_
                position[k+1]+sr_table[k][index[k]];
          if(k > 0)
                {
                index[- -k] = 0;
                goto O1;
                }
          DFT(block_position[0].a << stage,
          block_position[0].b << stage, grid_size,
                size);
          index[0] + +;
                goto O1;
      L1:
                index[k + +] = 0;
                if(k < = loop)
                    {
                    index[k] + +;
                    goto O1;
                    }
                grid_size * =4;
                }
        size * =2;
        }
}
```

```
void DFT(int sx, int sy, int grid_size, int size)
    {
    int a, b;
    for(a = sx; a < N; a + =grid_size)
      {
      for(b = sy; b < N; b + =grid_size)
          {
          // implement the butterfly structure for
          DFT computation
          // (a,b): origin; size : DFT block size
          ⋮
          }
      }
    }
```

References

[1] B. Arambepola, Fast computation of multidimensional discrete Fourier transforms, Proc. Inst. Electron. Eng. F. 127 (1) (1980) 49–52.

[2] S. Barash, Y. Ritov, Logarithmic pruning of FFT frequencies, IEEE Trans. Signal Process. 41 (3) (1993) 1398–1400.

[3] M. Barnsley, Fractals Everywhere, Academic Press, Inc., San Diego, CA, 1988 (Chapter 4).

[4] G. Bi, New split-radix algorithm for the discrete Hartley transform, IEEE Trans. Signal Process. 45 (2) (1997) 297–302.

[5] G. Bi, Y. Chen, Split-radix algorithm for 2D DFT, Electron. Lett. 33 (3) (1997) 203–205.

[6] V. Boriakoff, FFT computation with systolic arrays, A new architecture, IEEE Trans. Circuits Syst.—II: Analog Digital Signal Process. 41 (4) (1994) 278–284.

[7] S.C. Chan, K.L. Ho, Split vector-radix fast Fourier transform, IEEE Trans. Signal Process. 40 (8) (1992) 2029–2039.

[8] J.W. Cooley, J.W. Tukey, An algorithm for machine computation of complex Fourier series, Math. Comput. 19 (1965) 297–301.

[9] P. Duhamel, Implementation of 'split-radix' FFT algorithms for complex, real and real-symmetric data, IEEE Trans. Acoust. Speech Signal Process. ASSP-34 (1986) 285–295.

[10] P. Duhamel, H. Hollmann, Split radix FFT algorithm, Electron. Lett. 20 (1) (1984) 14–16.

[11] C.N. Fischer, R.J. LeBlanc, Crafting a Compiler with C, The Benjamin/Cummings Publishing Co., Redwood City, CA, 1991.

[12] D.B. Harris, J.H. McClellan, Vector-radix fast Fourier transform, in: Proceedings of the IEEE International Conference on Acoustics, Speech, Signal Processing, Hartford, CT, 1977, pp. 548–551.

[13] S. He, M. Torkelson, Computing partial DFT for comb spectrum evaluation, IEEE Signal Process. Lett. 3 (6) (1996) 173–175.

[14] R. Kumaresan, P.K. Gupta, Vector-radix algorithm for a 2-D discrete Fourier transform, Proc. IEEE 74 (1986) 755–757.

[15] J.S. Lim, Two-Dimensional Signal and Image Processing, Prentice-Hall, Englewood Cliffs, NJ, 1990 (Chapter 3).

[16] P.-C. Lo, Y.-Y. Lee, Real-time implementation of the split-radix FFT—an algorithm to efficiently construct local butterfly modules, Signal Processing 71 (1998) 291–299.

[17] P.-C. Lo, Y.-Y. Lee, Real-time implementation of the moving FFT algorithm, Signal Processing 79 (1999) 251–259.

[18] A.V. Oppenheim, R.W. Schafer, Discrete-Time Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1989 (Chapter 9).

[19] D.E. Paneras, R. Mani, S.H. Nawab, STFT computation using pruned FFT algorithms, IEEE Signal Process. Lett. 1 (4) (1994) 61–63.

[20] S.C. Pei, J.L. Wu, Split vector-radix 2D fast Fourier transform, in: Proceedings of the ICASSP'87, Dallas, TX, 1987, pp. 1987–1990.

[21] H.O. Peitgen, Fractals for the Classroom: Strategic Activities, Springer, Berlin, 1991 (Chapter 3).

[22] W. Philips, On computing the FFT of digital images in quadtree format, IEEE Trans. Signal Process. 47 (7) (1999) 2059–2060.

[23] C. Roche, A split-radix partial input/output fast Fourier transform algorithm, IEEE Trans. Signal Process. 40 (5) (1992) 1273–1276.

[24] I.W. Selesnick, C.S. Burrus, Automatic generation of prime length FFT programs, IEEE Trans. Signal Process. 44 (1) (1996) 14–24.

[25] D. Šević, On computing the 2-D FFT, IEEE Trans. Signal Process. 47 (5) (1999) 1428–1431.

[26] A.N. Skodras, A.G. Constantinides, Efficient computation of the split-radix FFT, IEE Proc. F 139 (1) (1992) 56–60.

[27] H.V. Sorensen, C.S. Burrus, Efficient computation of the DFT with only a subset of input or output points, IEEE Trans. Signal Process. 41 (3) (1993) 1184–1200.

[28] H.V. Sorensen, M.T. Heideman, C.S. Burrus, On computing the split-radix FFT, IEEE Trans. Acoust. Speech Signal Process. ASSP-34 (1) (1986) 152–156.

[29] T.V. Sreenivas, P.V.S. Rao, High resolution narrow-band spectra by FFT pruning, IEEE Trans. Acoust. Speech Signal Process. ASSP-28 (2) (1980) 254–257.

[30] D. Sundararajan, M.O. Ahmad, M.N.S. Swamy, Fast computation of the Discrete Fourier Transform of real data, IEEE Trans. Signal Process. 45 (8) (1997) 2010–2022.

[31] P.R. Uniyal, Transforming real-valued sequences: fast Fourier versus fast Hartley transform algorithms, IEEE Trans. Signal Process. 42 (11) (1994) 3249–3254.

[32] M. Vetterli, P. Duhamel, Split-radix algorithms for Length-p^m DFT's, IEEE Trans. Acoust. Speech Signal Process. ASSP-37 (1) (1989) 57–64.

[33] H.R. Wu, F.J. Paoloni, On the two-dimensional vector split-radix FFT algorithm, IEEE Trans. Acoust. Speech Signal Process. 37 (8) (1989) 1302–1304.

[34] J. You, S.S. Wong, Serial-parallel FFT array processor, IEEE Trans. Signal Process. 41 (3) (1993) 1472–1476.