A Novel Split-Radix Fast Algorithm for 2-D Discrete Hartley Transform

Longyu Jiang, Huazhong Shu, Jiasong Wu, Lu Wang, Lotfi Senhadji

A Novel Split-Radix Fast Algorithm for 2-D Discrete Hartley Transform

Longyu Jiang, Huazhong Shu, *Senior Member*, *IEEE*, Jiasong Wu, Lu Wang and Lotfi Senhadji, *Senior Member*, *IEEE*

***Abstract*—This paper presents a fast split-radix-(2×2)/(8×8) algorithm for computing the two-dimensional (2-D) discrete Hartley transform (DHT) of length $N\times N$ with $N = q*2^m$, where q is an odd integer. The proposed algorithm decomposes an $N\times N$ DHT into one $N/2\times N/2$ DHT and forty-eight $N/8\times N/8$ DHTs. It achieves an efficient reduction on the number of arithmetic operations, data transfers and twiddle factors compared to the split-radix-(2×2)/(4×4) algorithm. Moreover, the characteristic of expression in simple matrices leads to an easy implementation of the algorithm. If implementing the above two algorithms with fully parallel structure in hardware, it seems that the proposed algorithm can decrease the area complexity compared to the split-radix-(2×2)/(4×4) algorithm, but requires a little more time complexity. An application of the proposed algorithm to 2-D medical image compression is also provided.**

Index Terms—**Two-dimensional (2-D) discrete Hartley transform (DHT), split-radix, fast algorithm**

I. Introduction

The discrete Hartley transform (DHT) is widely used in signal and image processing applications. The advantage of the DHT over the discrete Fourier transform (DFT) is that it can be used to avoid complex operations when the input sequence is real. Moreover, the forward and inverse DHTs differ from each other in their form only in the scaling factor. Owing to these properties, the DHT is now finding an increasing interest in the signal processing community. In the past decades, fast algorithms and implementations of one-dimensional (1-D) DHT and DFT have been extensively investigated [1]-[21]. Meantime, special attention has also been paid on the two-dimensional (2-D) and three-dimensional (3-D) DHT [22]-[39], this is due to the growing interest in applications involving multi-dimensional (M-D) signals. In this paper, fast algorithm means lower computational complexity in terms of the number of arithmetic operations, data transfers and twiddle factors.

The algorithms proposed for fast computing the 2-D DHT can be classified into four categories: i) the row-column method; ii) the vector-radix fast Hartley transform (FHT) algorithms [22]-[24]; iii) the split-radix FHT algorithm [25]-[31]; and iv) the polynomial transform FHT algorithm [32]-[34]. The row-column method computes the 2-D DHT by taking the 1-D FHT sequentially along each dimension of the input data while in the vector-radix algorithm, the 2-D DHT is decomposed into many smaller ones until the trivial sequence length is reached. The vector-radix method reduces the number of arithmetic operations over the row-column algorithm and possesses the desirable properties such as regular structure and low implementation cost. This approach was then extended to 3-D DHT [35]-[37] and M-D DHT [23]. In [39], a vector-radix-3×3 algorithm was developed for computing the 2-D DHT of sequence whose length is $3^m\times 3^m$. The polynomial transform based FHT algorithms for M-D DHT have been reported in [32] and [34], which lead to a great reduction of the arithmetic operations at the expense of very complicated structure. The split-radix 2-D DHT algorithm is more efficient than the vector-radix algorithm in terms of arithmetic complexity and it is easy to implement. All the split-radix algorithms for 2-D DHT reported so far are based on a mixture of radix-2×2 and radix-4×4 index maps.

Huang *et al.* [25] applied a radix-2×2 decomposition to the even-even, even-odd, odd-even indexed samples and a radix-4×4 decomposition to the odd-odd indexed samples. Thus, an $N\times N$ DHT is decomposed into three $N/2\times N/2$ DHTs and four $N/4\times N/4$ DHTs. By using a radix-4×4 decomposition to even-odd, odd-even and odd-odd indexed terms, an improved split-radix algorithm for 2-D DHT was further derived [28], which decomposes an $N\times N$ 2-D DHT into one $N/2\times N/2$ DHT and twelve $N/4\times N/4$ DHTs. The split-radix algorithms for the 2-D DHT have been presented using decimation-in-frequency (DIF) [29] and decimation-in-time (DIT) [30]. It seems that the algorithms reported in [29] and [30] are the most efficient ones among all the existing split-radix algorithms in terms of the arithmetic complexity.

Manuscript received November 23, 2008. This work was supported by the National Natural Science Foundation of China under Grant 60873048, the Program for Changjiang Scholars and Innovative Research Team in University and the Natural Science Foundation of Jiangsu Province of China under Grant BK2008279.

L. Jiang is with the Laboratory of Image Science and Technology, School of Computer Science and Engineering, Southeast University, Nanjing 210096, China (e-mail: jianglongyu01412@yahoo.com.cn).

H. Shu and L. Wang are with the Laboratory of Image Science and Technology, School of Computer Science and Engineering, Southeast University, Nanjing 210096, China, and also with the Centre de Recherche en Information Biomédicale Sino-Français (CRIBs), Nanjing 210096, China (e-mail: shu.list@seu.edu.cn; wanglu@seu.edu.cn).

J. Wu is with the Laboratory of Image Science and Technology, School of Biological Science and Medical Engineering, Southeast University, Nanjing 210096, China, and with the Centre de Recherche en Information Biomédicale Sino-Français (CRIBs), Nanjing 210096, China, and with INSERM, U 642, 35000 Rennes, France, and with the Laboratoire Traitement du Signal et de l'Image (LTSI), Université de Rennes 1, 35000 Rennes, France, and also with the Centre de Recherche en Information Biomédicale Sino–Français (CRIBs), 35000 Rennes, France (e-mail: jswu@seu.edu.cn).

L. Senhadji is with INSERM, U 642, 35000 Rennes, France, and with the Laboratoire Traitement du Signal et de l'Image (LTSI), Université de Rennes 1, 35000 Rennes, France, and also with the Centre de Recherche en Information Biomédicale Sino–Français (CRIBs), 35000 Rennes, France (e-mail: lotfi.senhadji@univ-rennes1.fr).

Moreover, these two algorithms support various sequence lengths. Specifically, the block size can be chosen as $q*2^m \times q*2^m$, where q is an odd integer. In [31], the radix-2/4 approach has been generalized to the M-D DHT. In particular, for the case of 2-D DHT, it has the same arithmetic complexity as that of the algorithms presented in [29] and [30].

Among all the algorithms mentioned above, the split-radix algorithms based on radix-2/4 are the most attractive ones because they provide a good comprise between the arithmetic and structural complexities. Recently, Bouguezel et al. [3] proposed a new split-radix fast algorithm based on a mixture of radix-2 and radix-8 index maps for 1-D DHT of sequences whose length is $q\times 2^m$, where q is an odd integer. This algorithm is more efficient than the conventional split radix-2/4 FHT algorithm in terms of the number of data transfers and twiddle factor evaluations, which also contribute significantly to the execution time of FHT algorithms. Inspired by the algorithm presented in [3], we propose a split-radix-(2×2)/(8×8) algorithm for computing the 2-D DHT of sequences with length-$q*2^m \times q*2^m$, which consists of decomposing an $N\times N$ DHT into one $N/2\times N/2$ DHT and forty-eight $N/8\times N/8$ DHTs. Besides, the split radix-2/8 algorithm has been already used for computing the 2-D DFT [40], [41].

The rest of the paper is organized as follows. Section II presents the derivation of the algorithm. In Section III, the computational complexity and the hardware area and time complexity of the proposed algorithm are analyzed, and the comparison with some existing algorithms is also provided. Section IV presents the result of software implementation of the proposed and some existing algorithms. Section V concludes the work.

II. PROPOSED RADIX-(2×2)/(8×8) ALGORITHM

The 2-D DHT $X(k_1, k_2)$ of real valued sequence, $x(n_1, n_2)$, for $0 \le n_1, n_2 \le N-1$, is defined by

$$X(k_1,k_2) = \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\,\mathrm{cas}\left(\frac{2\pi}{N}\sum_{i=1}^{2} n_i k_i\right),\ 0 \le k_1,k_2 \le N-1, \tag{1}$$

where $\mathrm{cas}(\theta) = \cos(\theta) + \sin(\theta)$. The sequence length N is assumed to be $q\times 2^m$, where q is an odd integer and $m > 0$.

Let us first consider the case when $m = 1$, that is, $N = 2q$.

A. The case m = 1, i.e., N = 2q

In this case, the radix-2×2 algorithm is used to decompose a length-$2q\times 2q$ DHT. The even-even indexed outputs are obtained by

$$X(2k_1,2k_2) = \sum_{n_1=0}^{q-1}\sum_{n_2=0}^{q-1} y_{00}(n_1,n_2)\,\mathrm{cas}\left(\frac{2\pi}{q}\sum_{i=1}^{2} n_i k_i\right),\ 0 \le k_1,k_2 \le q-1. \tag{2}$$

The even-odd, odd-even, and odd-odd indexed outputs can be computed by

$$X(2k_1+p_1q, 2k_2+p_2q) = \sum_{n_1=0}^{q-1}\sum_{n_2=0}^{q-1} (-1)^{(n_1p_1+n_2p_2)}\, y_{p_1p_2}(n_1,n_2)\,\mathrm{cas}\left(\frac{2\pi}{q}\sum_{i=1}^{2} n_i k_i\right), \tag{3}$$

where $p_1, p_2 = 0, 1,\ (p_1, p_2) \ne (0,0),\ 0 \le k_1, k_2 \le q-1$.

The sequences $y_{p_1,p_2}(n_1,n_2)$ for $p_1, p_2 = 0, 1$, in (2) and (3) are obtained from the original input sequence as

$$\begin{aligned}&\left(y_{00}(n_1,n_2), y_{01}(n_1,n_2), y_{10}(n_1,n_2), y_{11}(n_1,n_2)\right)^T \\ &= (H_2 \otimes H_2)\left(x(n_1,n_2), x(n_1,n_2+q), x(n_1+q,n_2), x(n_1+q,n_2+q)\right)^T\end{aligned} \tag{4}$$

where T denotes the transpose, $H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, and "$\otimes$" is the Kronecker product [42]. Fig. 1 shows the implementation of (4).

B. The case m = 2, i.e., N = 4q

When $m = 2$, the decomposition of (1) for the even-even indexed outputs is given by

$$X(2k_1,2k_2) = \sum_{n_1=0}^{2q-1}\sum_{n_2=0}^{2q-1} y_{00}^{2/4}(n_1,n_2)\,\mathrm{cas}\left(\frac{2\pi}{2q}\sum_{i=1}^{2} n_i k_i\right),\ 0 \le k_1,k_2 \le 2q-1, \tag{5}$$

where

$$\begin{aligned}y_{00}^{2/4}(n_1,n_2) &= [x(n_1,n_2) + x(n_1,n_2+2q)] \\ &\quad + [x(n_1+2q,n_2) + x(n_1+2q,n_2+2q)].\end{aligned} \tag{6}$$

The even-odd, odd-even and odd-odd indexed outputs are obtained as follows

$$\begin{aligned}&X(4k_1 \pm p_1q, 4k_2 \pm p_2q) \\ &= \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\,\mathrm{cas}\left(\frac{2\pi}{q}\sum_{i=1}^{2} n_i k_i \pm \frac{\pi}{2}\sum_{i=1}^{2} n_i p_i\right) \\ &= F_{p_1,p_2}^{2/4}(k_1,k_2) \pm G_{p_1,p_2}^{2/4}(k_1,k_2),\quad 0 \le k_1,k_2 \le q-1,\end{aligned} \tag{7}$$

where

$$F_{p_1,p_2}^{2/4}(k_1,k_2) = \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\cos\left(\frac{\pi}{2}\sum_{i=1}^{2} n_i p_i\right)\mathrm{cas}\left(\frac{2\pi}{q}\sum_{i=1}^{2} n_i k_i\right), \tag{8}$$

$$G_{p_1,p_2}^{2/4}(k_1,k_2) = \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\sin\left(\frac{\pi}{2}\sum_{i=1}^{2} n_i p_i\right)\mathrm{cas}\left(-\frac{2\pi}{q}\sum_{i=1}^{2} n_i k_i\right). \tag{9}$$

Using the matrix representation, (7) can be expressed as

$$\begin{aligned}&\begin{bmatrix} X((4k_1+p_1q) \bmod 4q, (4k_2+p_2q) \bmod 4q) \\ X((N+4k_1-p_1q) \bmod 4q, (N+4k_2-p_2q) \bmod 4q) \end{bmatrix} \\ &= H_2 \begin{bmatrix} F_{p_1,p_2}^{2/4}(k_1,k_2) \\ G_{p_1,p_2}^{2/4}(k_1,k_2) \end{bmatrix}.\end{aligned} \tag{10}$$

1) Even-odd output terms ($p_1 = 0, 2; p_2 = 1$)

$F^{2/4}_{p_1,p_2}(k_1,k_2)$ and $G^{2/4}_{p_1,p_2}(k_1,k_2)$ defined by (8) and (9) can be further decomposed as follows.

$$\begin{aligned}
&F^{2/4}_{p_1,p_2}(k_1,k_2)\\
&=\sum_{n_1=0}^{2q-1}\sum_{n_2=0}^{2q-1}\{[x(n_1,n_2)+x(n_1+2q,n_2)]\\
&-[x(n_1,n_2+2q)+x(n_1+2q,n_2+2q)]\}\cos\left(\frac{\pi}{2}\sum_{i=1}^{2}n_ip_i\right)\\
&\quad\times\operatorname{cas}\left(\frac{2\pi}{q}\sum_{i=1}^{2}n_ik_i\right) \qquad (11)\\
&=\sum_{n_1=0}^{2q-1}\sum_{n_2=0}^{2q-1}y^{2/4}_{01}(n_1,n_2)\cos\left(\frac{\pi}{2}\sum_{i=1}^{2}n_ip_i\right)\operatorname{cas}\left(\frac{2\pi}{q}\sum_{i=1}^{2}n_ik_i\right)\\
&=\sum_{n_1=0}^{q-1}\sum_{n_2=0}^{q-1}f^{2/4}_{p_1,p_2}(n_1,n_2)\operatorname{cas}\left(\frac{2\pi}{q}\sum_{i=1}^{2}n_ik_i\right),
\end{aligned}$$

where

$$\begin{aligned}
y^{2/4}_{01}(n_1,n_2)&=[x(n_1,n_2)-x(n_1,n_2+2q)]\\
&+[x(n_1+2q,n_2)-x(n_1+2q,n_2+2q)], \qquad (12)\\
f^{2/4}_{p_1,p_2}(n_1,n_2)&=\left[y^{2/4}_{01}(n_1,n_2)+(-1)^{p_1/2}y^{2/4}_{01}(n_1+q,n_2)\right]\\
&\quad\times\cos\left(\frac{\pi}{2}\sum_{i=1}^{2}n_ip_i\right)+(-1)^{(q+1)/2}[y^{2/4}_{01}(n_1,n_2+q) \qquad (13)\\
&\quad+(-1)^{p_1/2}y^{2/4}_{01}(n_1+q,n_2+q)]\sin\left(\frac{\pi}{2}\sum_{i=1}^{2}n_ip_i\right).
\end{aligned}$$

The decomposition of $G^{2/4}_{p_1,p_2}(k_1,k_2)$ ($p_1 = 0, 2; p_2 = 1$) can be done in a similar way.

$$\begin{aligned}
&G^{2/4}_{p_1,p_2}(k_1,k_2)\\
&=\sum_{n_1=0}^{2q-1}\sum_{n_2=0}^{2q-1}y^{2/4}_{01}(n_1,n_2)\sin\left(\frac{\pi}{2}\sum_{i=1}^{2}n_ip_i\right)\operatorname{cas}\left(-\frac{2\pi}{q}\sum_{i=1}^{2}n_ik_i\right) \qquad (14)\\
&=\sum_{n_1=0}^{q-1}\sum_{n_2=0}^{q-1}g^{2/4}_{p_1,p_2}(n_1,n_2)\operatorname{cas}\left(-\frac{2\pi}{q}\sum_{i=1}^{2}n_ik_i\right)
\end{aligned}$$

where

$$\begin{aligned}
g^{2/4}_{p_1,p_2}(n_1,n_2)&=\left[y^{2/4}_{01}(n_1,n_2)+(-1)^{p_1/2}y^{2/4}_{01}(n_1+q,n_2)\right]\\
&\quad\times\sin\left(\frac{\pi}{2}\sum_{i=1}^{2}n_ip_i\right)+(-1)^{(q-1)/2}\left[y^{2/4}_{01}(n_1,n_2+q)\right. \qquad (15)\\
&\quad\left.+(-1)^{p_1/2}y^{2/4}_{01}(n_1+q,n_2+q)\right]\cos\left(\frac{\pi}{2}\sum_{i=1}^{2}n_ip_i\right)
\end{aligned}$$

$f^{2/4}_{p_1,p_2}(n_1,n_2)$ and $g^{2/4}_{p_1,p_2}(n_1,n_2)$ ($p_1 = 0, 2; p_2 = 1$) defined by (13) and (15) can be expressed in matrix form as

$$\begin{aligned}
&\begin{bmatrix} f^{2/4}_{0,1}(n_1,n_2)\\ f^{2/4}_{2,1}(n_1,n_2)\\ g^{2/4}_{0,1}(n_1,n_2)\\ g^{2/4}_{2,1}(n_1,n_2)\end{bmatrix}\\
&=\begin{bmatrix} \cos\frac{n_2\pi}{2} & 0 & -\sin\frac{n_2\pi}{2} & 0\\ 0 & (-1)^{n_1}\cos\frac{n_2\pi}{2} & 0 & (-1)^{n_1+1}\sin\frac{n_2\pi}{2}\\ \sin\frac{n_2\pi}{2} & 0 & \cos\frac{n_2\pi}{2} & 0\\ 0 & (-1)^{n_1}\sin\frac{n_2\pi}{2} & 0 & (-1)^{n_1}\cos\frac{n_2\pi}{2}\end{bmatrix}\\
&\times\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&(-1)^{(q-1)/2}&0\\0&0&0&(-1)^{(q-1)/2}\end{bmatrix}\begin{bmatrix}1&0&1&0\\1&0&-1&0\\0&1&0&1\\0&1&0&-1\end{bmatrix}\begin{bmatrix} y^{2/4}_{01}(n_1,n_2)\\ y^{2/4}_{01}(n_1,n_2+q)\\ y^{2/4}_{01}(n_1+q,n_2)\\ y^{2/4}_{01}(n_1+q,n_2+q)\end{bmatrix}
\end{aligned} \qquad (16)$$

The above equation can be rewritten as

$$\begin{aligned}
\begin{bmatrix}\mathbf{f}^{2/4}_{p_1,p_2}\\ \mathbf{g}^{2/4}_{p_1,p_2}\end{bmatrix}&=\begin{bmatrix}\mathbf{C}^{2/4}_{eo}&-\mathbf{S}^{2/4}_{eo}\\ \mathbf{S}^{2/4}_{eo}&\mathbf{C}^{2/4}_{eo}\end{bmatrix}\begin{bmatrix}\mathbf{I}_2&\mathbf{0}\\ \mathbf{0}&(-1)^{(q-1)/2}\mathbf{I}_2\end{bmatrix}\\
&\times\begin{bmatrix}\mathbf{J}_1&\mathbf{R}^{2/4}_1\mathbf{J}_1\\ \mathbf{R}^{2/4}_1\mathbf{J}_2&\mathbf{J}_2\end{bmatrix}\mathbf{y}^{2/4}_{01},
\end{aligned} \qquad (17)$$

where $\mathbf{I}_2$ is the identity matrix, and

$$\mathbf{f}^{2/4}_{p_1,p_2}=\begin{bmatrix} f^{2/4}_{0,1}(n_1,n_2)\\ f^{2/4}_{2,1}(n_1,n_2)\end{bmatrix},\quad \mathbf{g}^{2/4}_{p_1,p_2}=\begin{bmatrix} g^{2/4}_{0,1}(n_1,n_2)\\ g^{2/4}_{2,1}(n_1,n_2)\end{bmatrix}, \qquad (18)$$

$$\begin{aligned}
\mathbf{C}^{2/4}_{eo}&=\begin{bmatrix}\cos\alpha_{01}&0\\0&(-1)^{n_1}\cos\alpha_{01}\end{bmatrix},\\
\mathbf{S}^{2/4}_{eo}&=\begin{bmatrix}\sin\alpha_{01}&0\\0&(-1)^{n_1}\sin\alpha_{01}\end{bmatrix},\quad \alpha_{01}=(n_2\pi)/2,
\end{aligned} \qquad (19)$$

$$\mathbf{J}_1=\begin{bmatrix}1&0\\1&0\end{bmatrix},\ \mathbf{J}_2=\begin{bmatrix}0&1\\0&-1\end{bmatrix},\ \mathbf{R}^{2/4}_1=\operatorname{diag}(1,-1), \qquad (20)$$

$$\begin{aligned}
\mathbf{y}^{2/4}_{01}=(&y^{2/4}_{01}(n_1,n_2)\quad y^{2/4}_{01}(n_1,n_2+q)\\
&y^{2/4}_{01}(n_1+q,n_2)\quad y^{2/4}_{01}(n_1+q,n_2+q))^T.
\end{aligned} \qquad (21)$$

Fig. 2 shows the implementation of (17).

2) Odd-even output terms ($p_1 = 1; p_2 = 0, 2$)

As for the previous case, the odd-even output terms can be obtained as

$$\begin{bmatrix}\mathbf{f}^{2/4}_{p_1,p_2}\\ \mathbf{g}^{2/4}_{p_1,p_2}\end{bmatrix}=\begin{bmatrix}\mathbf{C}^{2/4}_{oe}&-\mathbf{S}^{2/4}_{oe}\\ \mathbf{S}^{2/4}_{oe}&\mathbf{C}^{2/4}_{oe}\end{bmatrix}\begin{bmatrix}\mathbf{I}_2&\mathbf{0}\\ \mathbf{0}&(-1)^{(q-1)/2}\mathbf{I}_2\end{bmatrix}\begin{bmatrix}\mathbf{H}_2&\mathbf{0}\\ \mathbf{0}&\mathbf{H}_2\end{bmatrix}\mathbf{y}^{2/4}_{10}, \qquad (22)$$

where

$$\mathbf{f}^{2/4}_{p_1,p_2}=\begin{bmatrix} f^{2/4}_{1,0}(n_1,n_2)\\ f^{2/4}_{1,2}(n_1,n_2)\end{bmatrix},\ \mathbf{g}^{2/4}_{p_1,p_2}=\begin{bmatrix} g^{2/4}_{1,0}(n_1,n_2)\\ g^{2/4}_{1,2}(n_1,n_2)\end{bmatrix}, \qquad (23)$$

$$\mathbf{C}_{oe}^{2/4}=\begin{bmatrix}\cos\alpha_{10} & 0\\ 0 & (-1)^{n_2}\cos\alpha_{10}\end{bmatrix},$$
$$\mathbf{S}_{oe}^{2/4}=\begin{bmatrix}\sin\alpha_{10} & 0\\ 0 & (-1)^{n_2}\sin\alpha_{10}\end{bmatrix},\quad \alpha_{10}=(n_1\pi)/2, \tag{24}$$

$$\mathbf{y}_{10}^{2/4}=(y_{10}^{2/4}(n_1,n_2)\quad y_{10}^{2/4}(n_1,n_2+q)\quad y_{10}^{2/4}(n_1+q,n_2)\quad y_{10}^{2/4}(n_1+q,n_2+q))^T, \tag{25}$$

$$y_{10}^{2/4}(n_1,n_2)=[x(n_1,n_2)+x(n_1,n_2+2q)]-[x(n_1+2q,n_2)+x(n_1+2q,n_2+2q)]. \tag{26}$$

3) Odd-odd output terms ($p_1 = 1, -1$; $p_2 = 1$)

We have the following decomposition for the odd-odd output terms

$$\begin{bmatrix}\mathbf{f}_{p_1,p_2}^{2/4}\\ \mathbf{g}_{p_1,p_2}^{2/4}\end{bmatrix}=\begin{bmatrix}\mathbf{C}_{oo}^{2/4} & -\mathbf{S}_{oo}^{2/4}\\ \mathbf{S}_{oo}^{2/4} & \mathbf{C}_{oo}^{2/4}\end{bmatrix}\begin{bmatrix}\mathbf{I}_2 & \mathbf{0}\\ \mathbf{0} & (-1)^{(q-1)/2}\mathbf{I}_2\end{bmatrix}\begin{bmatrix}\mathbf{J}_1 & \mathbf{J}_2\\ \mathbf{R}_1^{2/4}\mathbf{J}_2 & \mathbf{R}_2^{2/4}\mathbf{J}_1\end{bmatrix}\mathbf{y}_{11}^{2/4} \tag{27}$$

where

$$\mathbf{f}_{p_1,p_2}^{2/4}=\begin{bmatrix}f_{-1,1}^{2/4}(n_1,n_2)\\ f_{1,1}^{2/4}(n_1,n_2)\end{bmatrix},\ \mathbf{g}_{p_1,p_2}^{2/4}=\begin{bmatrix}g_{-1,1}^{2/4}(n_1,n_2)\\ g_{1,1}^{2/4}(n_1,n_2)\end{bmatrix}, \tag{28}$$

$$\mathbf{R}_2^{2/4}=\mathrm{diag}(-1,1),$$

$$\mathbf{C}_{oo}^{2/4}=\begin{bmatrix}\cos\alpha_{11} & 0\\ 0 & \cos\alpha'_{11}\end{bmatrix},\ \mathbf{S}_{oo}^{2/4}=\begin{bmatrix}\sin\alpha_{11} & 0\\ 0 & \sin\alpha'_{11}\end{bmatrix}, \tag{29}$$

$$\alpha_{11}=[(n_2-n_1)\pi]/2,\ \alpha'_{11}=[(n_1+n_2)\pi]/2,$$

$$\mathbf{y}_{11}^{2/4}=(y_{11}^{2/4}(n_1,n_2)\quad y_{11}^{2/4}(n_1,n_2+q)\quad y_{11}^{2/4}(n_1+q,n_2)\quad y_{11}^{2/4}(n_1+q,n_2+q))^T, \tag{30}$$

$$y_{11}^{2/4}(n_1,n_2)=[x(n_1,n_2)-x(n_1,n_2+2q)]-[x(n_1+2q,n_2)-x(n_1+2q,n_2+2q)]. \tag{31}$$

C. The case $m \geq 3$

By introducing a mixture of radix-2×2 and radix-8×8 index maps, we propose a novel decomposition of (1). The even-even output terms can be computed by

$$X(2k_1,2k_2)=\sum_{n_1=0}^{N/2-1}\sum_{n_2=0}^{N/2-1}y_{00}^{2/8}(n_1,n_2)\mathrm{cas}\left(\frac{2\pi}{N/2}\sum_{i=1}^{2}n_ik_i\right),\ 0\le k_1,k_2\le N/2-1, \tag{32}$$

where

$$y_{00}^{2/8}(n_1,n_2)=[x(n_1,n_2)+x(n_1,n_2+N/2)]+[x(n_1+N/2,n_2)+x(n_1+N/2,n_2+N/2)]. \tag{33}$$

The even-odd, odd-even, and odd-odd output terms can be derived as follows.

$$X(8k_1\pm p_1q,8k_2\pm p_2q)=\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}x(n_1,n_2)\mathrm{cas}\left(\frac{2\pi}{N/8}\sum_{i=1}^{2}n_ik_i\pm\frac{2\pi}{N/q}\sum_{i=1}^{2}n_ip_i\right) \tag{34}$$
$$=F_{p_1,p_2}^{2/8}(k_1,k_2)\pm G_{p_1,p_2}^{2/8}(k_1,k_2),\ 0\le k_1,k_2\le N/8-1,$$

where

$$F_{p_1,p_2}^{2/8}(k_1,k_2)=\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}x(n_1,n_2)\cos\left(\frac{2\pi}{N/q}\sum_{i=1}^{2}n_ip_i\right)\mathrm{cas}\left(\frac{2\pi}{N/8}\sum_{i=1}^{2}n_ik_i\right), \tag{35}$$

$$G_{p_1,p_2}^{2/8}(k_1,k_2)=\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}x(n_1,n_2)\sin\left(\frac{2\pi}{N/q}\sum_{i=1}^{2}n_ip_i\right)\mathrm{cas}\left(-\frac{2\pi}{N/8}\sum_{i=1}^{2}n_ik_i\right). \tag{36}$$

Equation (34) can be written in matrix form as

$$\begin{bmatrix}X((8k_1+p_1q)\bmod N,(8k_2+p_2q)\bmod N)\\ X((N+8k_1-p_1q)\bmod N,(N+8k_2-p_2q)\bmod N)\end{bmatrix}=H_2\begin{bmatrix}F_{p_1,p_2}^{2/8}(k_1,k_2)\\ G_{p_1,p_2}^{2/8}(k_1,k_2)\end{bmatrix} \tag{37}$$

The input data sequences $F_{p_1,p_2}^{2/8}(k_1,k_2)$ and $G_{p_1,p_2}^{2/8}(k_1,k_2)$ are determined as follows.

1) Even-odd output terms ($p_1 = 0, 2, 4, 6$; $p_2 = 1, 3$)

Equation (35) can be decomposed as

$$F_{p_1,p_2}^{2/8}(k_1,k_2)=\sum_{n_1=0}^{N/2-1}\sum_{n_2=0}^{N/2-1}y_{01}^{2/8}(n_1,n_2)\cos\left(\frac{2\pi}{N/q}\sum_{i=1}^{2}n_ip_i\right)\mathrm{cas}\left(\frac{2\pi}{N/8}\sum_{i=1}^{2}n_ik_i\right) \tag{38}$$
$$=\sum_{n_1=0}^{N/8-1}\sum_{n_2=0}^{N/8-1}f_{p_1,p_2}^{2/8}(n_1,n_2)\mathrm{cas}\left(\frac{2\pi}{N/8}\sum_{i=1}^{2}n_ik_i\right),$$

where

$$y_{01}^{2/8}(n_1,n_2)=\left[x(n_1,n_2)-x(n_1,n_2+N/2)\right]+\left[x(n_1+N/2,n_2)-x(n_1+N/2,n_2+N/2)\right], \tag{39}$$

$$f_{p_1,p_2}^{2/8}(n_1,n_2)=\sum_{l_1=0}^{3}\sum_{l_2=0}^{3}y_{01}^{2/8}\left(n_1+\frac{l_1N}{8},n_2+\frac{l_2N}{8}\right)\times\cos\left(\frac{2\pi}{N/q}\sum_{i=1}^{2}n_ip_i+\frac{\pi}{4}q(p_1l_1+p_2l_2)\right). \tag{40}$$

Equation (36) can be decomposed in a similar manner as

$$G_{p_1,p_2}^{2/8}(k_1,k_2)=\sum_{n_1=0}^{N/8-1}\sum_{n_2=0}^{N/8-1}g_{p_1,p_2}^{2/8}(n_1,n_2)\mathrm{cas}\left(-\frac{2\pi}{N/8}\sum_{i=1}^{2}n_ik_i\right) \tag{41}$$

where

$$g_{p_1,p_2}^{2/8}(n_1,n_2)=\sum_{l_1=0}^{3}\sum_{l_2=0}^{3}y_{01}^{2/8}\left(n_1+\frac{l_1N}{8},n_2+\frac{l_2N}{8}\right)\times\sin\left(\frac{2\pi}{N/q}\sum_{i=1}^{2}n_ip_i+\frac{\pi}{4}q(p_1l_1+p_2l_2)\right). \tag{42}$$

We need to use the following lemma, which was stated in [3].

Lemma 1: Let $\cos(\beta(\pi/4))=\sqrt{2}/2\,c_q$ and $\sin(\beta(\pi/4))=\sqrt{2}/2\,s_q$, where β is an odd integer. Then the following is true

i) For $\beta = q$, $c_q s_q = (-1)^{(q-1)/2}$.

ii) For $\beta = 3q$, $c_{3q} = -c_q$ and $s_{3q} = s_q$.

Letting

$$\gamma_{01}=\frac{2\pi}{N/q}\sum_{i=1}^{2}n_ip_i \tag{43}$$

and using Lemma 1, the twiddle factors $\cos\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)$ and $\sin\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)$ appeared in (40) and (42) can be simplified as

a) $(p_1l_1+p_2l_2)\bmod 4=0$.

$$\cos\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)=(-1)^{(p_1l_1+p_2l_2)/4}\cos(\gamma_{01}), \quad (44a)$$

$$\sin\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)=(-1)^{(p_1l_1+p_2l_2)/4}\sin(\gamma_{01}). \quad (44b)$$

b) $(p_1l_1+p_2l_2)\bmod 4=1$

$$\cos\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor}\cos\left(\gamma_{01}+\frac{q\pi}{4}\right) \quad (45a)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor}\frac{c_q}{\sqrt{2}}\left[\cos(\gamma_{01})-(-1)^{(q-1)/2}\sin(\gamma_{01})\right],$$

$$\sin\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor}\sin\left(\gamma_{01}+\frac{q\pi}{4}\right) \quad (45b)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor}\frac{c_q}{\sqrt{2}}\left[\sin(\gamma_{01})+(-1)^{(q-1)/2}\cos(\gamma_{01})\right],$$

where $\lfloor x\rfloor$ denotes the integer part of x.

c) $(p_1l_1+p_2l_2)\bmod 4=2$

$$\cos\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor+(q-1)/2}\sin(\gamma_{01}), \quad (46a)$$

$$\sin\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor+(q-1)/2}\cos(\gamma_{01}). \quad (46b)$$

d) $(p_1l_1+p_2l_2)\bmod 4=3$

$$\cos\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor}\cos\left(\gamma_{01}+\frac{3q\pi}{4}\right) \quad (47a)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor+1}\frac{c_q}{\sqrt{2}}\left[\cos(\gamma_{01})+(-1)^{(q-1)/2}\sin(\gamma_{01})\right],$$

$$\sin\left(\gamma_{01}+\frac{q\pi}{4}(p_1l_1+p_2l_2)\right)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor}\sin\left(\gamma_{01}+\frac{3q\pi}{4}\right) \quad (47b)$$
$$=(-1)^{\lfloor (p_1l_1+p_2l_2)/4\rfloor+1}\frac{c_q}{\sqrt{2}}\left[\sin(\gamma_{01})+(-1)^{(q+1)/2}\cos(\gamma_{01})\right].$$

Using the above results, $f^{2/8}_{p_1,p_2}(n_1,n_2)$ and $g^{2/8}_{p_1,p_2}(n_1,n_2)$ defined by (40) and (42) can be expressed in matrix form as

$$\begin{bmatrix}\mathbf{f}^{2/8}_{p_1,p_2}\\ \mathbf{g}^{2/8}_{p_1,p_2}\end{bmatrix}=\begin{bmatrix}\mathbf{C}^{2/8}_{eo} & -\mathbf{S}^{2/8}_{eo}\\ \mathbf{S}^{2/8}_{eo} & \mathbf{C}^{2/8}_{eo}\end{bmatrix}\begin{bmatrix}\mathbf{I}_8 & 0\\ 0 & (-1)^{(q-1)/2}\mathbf{I}_8\end{bmatrix}$$
$$\times\begin{bmatrix}\mathbf{A}^{2/8}_{eo} & \mathbf{R}_1\mathbf{A}^{2/8}_{eo}\\ \mathbf{B}^{2/8}_{eo} & \mathbf{R}_1\mathbf{B}^{2/8}_{eo}\end{bmatrix}(\mathbf{I}_2\otimes\mathbf{Q}_1)\mathbf{y}^{2/8}_{01}, \quad (48)$$

where $\mathbf{I}_L$ is an identity matrix of order L, the pth component of the vectors $\mathbf{f}^{2/8}_{p_1,p_2}$ and $\mathbf{g}^{2/8}_{p_1,p_2}$ is related to the input sequences of (40) and (42) by

$$f^{2/8}_{p_1,p_2}(p)=f^{2/8}_{p_1,p_2}(n_1,n_2),$$
$$g^{2/8}_{p_1,p_2}(p)=g^{2/8}_{p_1,p_2}(n_1,n_2),\quad p=p_1+(p_2-1)/2. \quad (49)$$

The matrices $\mathbf{C}^{2/8}_{eo}$ and $\mathbf{S}^{2/8}_{eo}$ are composed by twiddle factors whose components are given by

$$\mathbf{C}^{2/8}_{eo}(p,p)=\cos(\gamma_{01}),\ \mathbf{S}^{2/8}_{eo}(p,p)=\sin(\gamma_{01}). \quad (50)$$

The new input sequences $\mathbf{y}^{2/8}_{01}$ is related to the original sequences as

$$y^{2/8}_{01}(r)=y^{2/8}_{01}(n_1+r_1N/8,n_2+r_2N/8),$$
$$r=0,1,...,15,\ r_1=\lfloor r/4\rfloor,\ r_2=r\bmod 4, \quad (51)$$

$$\mathbf{A}^{2/8}_{eo}=\begin{bmatrix}\mathbf{A}^{01}_{eo} & \mathbf{A}^{10}_{eo}\\ \mathbf{A}^{01}_{eo} & -\mathbf{A}^{10}_{eo}\end{bmatrix}\mathbf{A}^{01}_{eo}=\begin{bmatrix}1&1&0&-1\\1&-1&0&1\\1&1&0&-1\\1&-1&0&1\end{bmatrix}\mathbf{A}^{10}_{eo}=\begin{bmatrix}1&1&0&-1\\1&-1&0&1\\0&-1&-1&-1\\0&-1&1&-1\end{bmatrix} \quad (52)$$

$$\mathbf{B}^{2/8}_{eo}=\begin{bmatrix}\mathbf{B}^{01}_{eo} & \mathbf{B}^{10}_{eo}\\ \mathbf{B}^{01}_{eo} & -\mathbf{B}^{10}_{eo}\end{bmatrix}\mathbf{B}^{01}_{eo}=\begin{bmatrix}0&1&1&1\\0&1&-1&1\\0&1&1&1\\0&1&-1&1\end{bmatrix}\mathbf{B}^{10}_{eo}=\begin{bmatrix}0&1&1&1\\0&1&-1&1\\1&1&0&-1\\1&-1&0&1\end{bmatrix} \quad (53)$$

$$\mathbf{R}_1=\mathrm{diag}(1,1,-1,-1,1,1,-1,-1), \quad (54)$$

$$\mathbf{Q}_1=diag\left(1,\frac{\sqrt{2}}{2}c_q,1,\frac{\sqrt{2}}{2}c_q,1,\frac{\sqrt{2}}{2}c_q,1,\frac{\sqrt{2}}{2}c_q\right). \quad (55)$$

Fig. 3 shows the implementation of (48).

2) Odd-even output terms (p_1 = 1, 3; p_2 = 0, 2, 4, 6)

We have

$$\begin{bmatrix}\mathbf{f}^{2/8}_{p_1,p_2}\\ \mathbf{g}^{2/8}_{p_1,p_2}\end{bmatrix}=\begin{bmatrix}\mathbf{C}^{2/8}_{oe} & -\mathbf{S}^{2/8}_{oe}\\ \mathbf{S}^{2/8}_{oe} & \mathbf{C}^{2/8}_{oe}\end{bmatrix}\begin{bmatrix}\mathbf{I}_8 & 0\\ 0 & (-1)^{(q-1)/2}\mathbf{I}_8\end{bmatrix}$$
$$\times\begin{bmatrix}\mathbf{A}^{2/8}_{oe} & \mathbf{R}_3\mathbf{B}^{2/8}_{oe}\\ \mathbf{B}^{2/8}_{oe} & \mathbf{R}_2\mathbf{A}^{2/8}_{oe}\end{bmatrix}(\mathbf{I}_2\otimes\mathbf{Q}_2)\mathbf{y}^{2/8}_{10} \quad (56)$$

where the pth component of the vectors $\mathbf{f}^{2/8}_{p_1,p_2}$ and $\mathbf{g}^{2/8}_{p_1,p_2}$ is related to the input sequences of (40) and (42) by

$$f^{2/8}_{p_1,p_2}(p)=f^{2/8}_{p_1,p_2}(n_1,n_2),$$
$$g^{2/8}_{p_1,p_2}(p)=g^{2/8}_{p_1,p_2}(n_1,n_2),\ \ p=p_2+(p_1-1)/2. \quad (57)$$

The elements of the matrices $\mathbf{C}^{2/8}_{oe}$ and $\mathbf{S}^{2/8}_{oe}$ are given by

$$\mathbf{C}_{oe}(p,p)=\cos(\gamma_{01}),\ \mathbf{S}_{oe}(p,p)=\sin(\gamma_{01}). \quad (58)$$

The new input sequences $\mathbf{y}^{2/8}_{10}$ is related to the original sequences as

$$y_{10}^{2/8}(r) = y_{10}^{2/8}(n_1 + r_1 N/8, n_2 + r_2 N/8), \quad r = 0,1,...,15, \quad r_1 = \lfloor r/4 \rfloor, \quad r_2 = r \bmod 4, \tag{59}$$

$$y_{10}^{2/8}(n_1, n_2) = [x(n_1, n_2) + x(n_1, n_2 + N/2)] - [x(n_1 + N/2, n_2) + x(n_1 + N/2, n_2 + N/2)], \tag{60}$$

$$\mathbf{A}_{oe}^{2/8} = \begin{bmatrix} \mathbf{A}_{oe}^{01} & \mathbf{A}_{oe}^{10} \\ \mathbf{A}_{oe}^{01} & \mathbf{A}_{oe}^{11} \end{bmatrix}, \qquad \mathbf{B}_{oe}^{2/8} = \begin{bmatrix} \mathbf{B}_{oe}^{01} & \mathbf{B}_{oe}^{10} \\ \mathbf{B}_{oe}^{01} & \mathbf{B}_{oe}^{11} \end{bmatrix}, \tag{61}$$

$$\mathbf{A}_{oe}^{01} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & -1 & 0 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix} \mathbf{A}_{oe}^{10} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \mathbf{A}_{oe}^{11} = \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix} \tag{62}$$

$$\mathbf{B}_{oe}^{01} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \mathbf{B}_{oe}^{10} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \mathbf{B}_{oe}^{11} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \tag{63}$$

$$\mathbf{R}_2 = \text{diag}(1, 1, 1, 1, -1, -1, -1, -1), \quad \mathbf{R}_3 = \text{diag}(-1, -1, -1, -1, 1, 1, 1, 1), \tag{64}$$

$$\mathbf{Q}_2 = diag\left(1,\ 1,\ 1,\ 1,\ \frac{\sqrt{2}}{2}c_q, \frac{\sqrt{2}}{2}c_q, \frac{\sqrt{2}}{2}c_q, \frac{\sqrt{2}}{2}c_q\right). \tag{65}$$

3) Odd-odd output terms ($p_1 = -3, -1, 1, 3$; $p_2 = 1, 3$)

We have

$$\begin{bmatrix} \mathbf{f}_{p_1,p_2}^{2/8} \\ \mathbf{g}_{p_1,p_2}^{2/8} \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{oo}^{2/8} & -\mathbf{S}_{oo}^{2/8} \\ \mathbf{S}_{oo}^{2/8} & \mathbf{C}_{oo}^{2/8} \end{bmatrix} \begin{bmatrix} \mathbf{I}_8 & 0 \\ 0 & (-1)^{(q-1)/2}\mathbf{I}_8 \end{bmatrix} \times \begin{bmatrix} \mathbf{A}_{oo}^{2/8} & \mathbf{R}_4\mathbf{B}_{oo}^{2/8} \\ \mathbf{B}_{oo}^{2/8} & \mathbf{R}_1\mathbf{A}_{oo}^{2/8} \end{bmatrix} (\mathbf{I}_2 \otimes \mathbf{Q}_3)\mathbf{y}_{11}^{2/8} \tag{66}$$

where the pth component of the vectors $\mathbf{f}_{p_1,p_2}^{2/8}$ and $\mathbf{g}_{p_1,p_2}^{2/8}$ is related to the input sequences of (40) and (42) by

$$f_{p_1,p_2}^{2/8}(p) = f_{p_1,p_2}^{2/8}(n_1, n_2), \quad g_{p_1,p_2}^{2/8}(p) = g_{p_1,p_2}^{2/8}(n_1, n_2),\ p = (p_1 + 3) + (p_2 - 1)/2. \tag{67}$$

The (p, p)th components of the matrices $\mathbf{C}_{oo}^{2/8}$ and $\mathbf{S}_{oo}^{2/8}$ are respectively given by

$$\mathbf{C}_{oo}(p, p) = \cos(\gamma_{01}), \quad \mathbf{S}_{oo}(p, p) = \sin(\gamma_{01}). \tag{68}$$

The new input sequences $\mathbf{y}_{11}^{2/8}$ is defined as

$$y_{11}^{2/8}(r) = y_{11}^{2/8}(n_1 + r_1 N/8, n_2 + r_2 N/8), \quad r = 0,1,...,15,\ r_1 = \lfloor r/4 \rfloor,\ r_2 = r \bmod 4, \tag{69}$$

$$y_{11}^{2/8}(n_1, n_2) = \left[x(n_1, n_2) - x\left(n_1, n_2 + N/2\right)\right] - \left[x\left(n_1 + N/2, n_2\right) - x\left(n_1 + N/2, n_2 + N/2\right)\right], \tag{70}$$

$$\mathbf{A}_{oo}^{2/8} = \begin{bmatrix} \mathbf{A}_{oo}^{01} & \mathbf{A}_{oo}^{10} \\ \mathbf{A}_{oo}^{01} & -\mathbf{A}_{oo}^{10} \end{bmatrix} \mathbf{A}_{oo}^{01} = \begin{bmatrix} 1 & 1 & 0 & -1 \\ 1 & -1 & 0 & 1 \\ 1 & 1 & 0 & -1 \\ 1 & -1 & 0 & 1 \end{bmatrix} \mathbf{A}_{oo}^{10} = \begin{bmatrix} -1 & 0 & 1 & 1 \\ -1 & 1 & -1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & -1 & 1 \end{bmatrix} \tag{71}$$

$$\mathbf{B}_{oo}^{2/8} = \begin{bmatrix} \mathbf{B}_{oo}^{01} & -\mathbf{B}_{oo}^{10} \\ \mathbf{B}_{oo}^{01} & \mathbf{B}_{oo}^{10} \end{bmatrix} \mathbf{B}_{oo}^{01} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & -1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & -1 & 1 \end{bmatrix} \mathbf{B}_{oo}^{10} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & -1 & 1 \\ 1 & 0 & -1 & -1 \\ 1 & -1 & 1 & 0 \end{bmatrix} \tag{72}$$

$$\mathbf{R}_4 = \text{diag}(-1, -1, 1, 1, -1, -1, 1, 1), \tag{73}$$

$$\mathbf{Q}_3 = diag\left(1,\ \frac{\sqrt{2}}{2}c_q, 1, \frac{\sqrt{2}}{2}c_q,\ \frac{\sqrt{2}}{2}c_q, 1, \frac{\sqrt{2}}{2}c_q, 1\right). \tag{74}$$

III. Computational Complexity and Hardware Area and Time Analysis

In this section, we analyze the performance of the proposed 2-D split-radix-(2×2)/(8×8) algorithm and compare it with some existing algorithms. The analysis and comparison will not only include the arithmetic operations, but also the operations such as data transfers and twiddle factor evaluations since they contribute significantly to the execution time of the algorithm. The analysis of the area and time complexities is also provided.

A. Arithmetic complexity

It is assumed that the butterfly computations are implemented by four multiplications and two additions.

1) When $N = 2q$, from (2) and (3), the number of multiplications and additions is given by

$$M_{2q\times 2q} = 4M_{q\times q}, \ A_{2q\times 2q} = 4A_{q\times q} + 8q^2. \tag{75}$$

2) When $N = 4q$, the twiddle factors in (17), (22) and (27) become trivial. Therefore

$$M_{4q\times 4q} = M_{2q\times 2q} + 12M_{q\times q}, \ A_{4q\times 4q} = A_{2q\times 2q} + 12A_{q\times q} + 56q^2. \tag{76}$$

3) When $N \geq 8q$

a) The computation of the input data sequences $y_{00}^{2/8}(n_1, n_2)$, $y_{01}^{2/8}(n_1, n_2)$, $y_{10}^{2/8}(n_1, n_2)$ and $y_{11}^{2/8}(n_1, n_2)$ defined by (33), (39), (60) and (70) requires $2N^2$ additions.

b) In (47), for each given pair (n_1, n_2), the matrix $\begin{bmatrix} \mathbf{A}_{eo}^{2/8} & \mathbf{R}_1\mathbf{A}_{eo}^{2/8} \\ \mathbf{B}_{eo}^{2/8} & \mathbf{R}_1\mathbf{B}_{eo}^{2/8} \end{bmatrix}$ requires 56 additions since the elements of the matrices are either 1 or –1, so that $7N^2/8$ additions are needed for $0 \leq n_1, n_2 \leq N/8–1$.

c) In equation (48), the computation of the matrix $\begin{bmatrix} \mathbf{C}_{eo}^{2/8} & -\mathbf{S}_{eo}^{2/8} \\ \mathbf{S}_{eo}^{2/8} & \mathbf{C}_{eo}^{2/8} \end{bmatrix}$, which is composed by twiddle factors, requires $N^2/4–A_s$ additions and $N^2/2–M_s$ multiplications, where A_s and M_s are the savings from the special cases of twiddle factors such as 0, ±1, $\pm\sqrt{2}/2$, $\cos(\pi/8)$, $\sin(\pi/8)$, $\cos(3\pi/8)$ and $\sin(3\pi/8)$. Specifically, we can obtain the number of additions and multiplications saved from the special cases of twiddle factors as follows: When $p_1 = 0$ and $p_2 = 1, 3$ for a given value of n_1 and $0 \leq n_2 \leq N/8–1$, this case can be taken as an 1-D DHT for the special twiddle factors. The saved number of additions and multiplications can be derived in a way similar to the one presented in [3], they are respectively $6q$ and $10q$. So that the total saved number of additions and multiplications in the case of $p_1 = 0, p_2 = 1, 3$, for $0 \leq n_1, n_2 \leq N/8–1$ is $3qN/4$ and $5qN/4$. Thus, for all the combination of (p_1, p_2) in (48), we can obtain $A_s = 3qN$ and $M_s = 5qN$.

d) The computation of the matrix $\mathbf{I}_2 \otimes \mathbf{Q}_1$ requires $N^2/8$ multiplications.

e) The analysis described from (b) to (d) shows that the computation of $F^{2/8}_{p_1,p_2}(k_1,k_2)$ and $G^{2/8}_{p_1,p_2}(k_1,k_2)$ for even-odd output terms needs $9N^2/8–3qN$ additions and $5N^2/8–5qN$ multiplications. The same number of arithmetic operation is required for odd-even and odd-odd cases.

f) The computation of equation (34) requires $3N^2/4$ additions for even-odd, odd-even and odd-odd output terms.

From the above discussion, it can be seen that the total number of additions and multiplications involved in the proposed algorithm for $N > 8q$ is as follows

$$\begin{aligned} A_{N\times N} &= A_{N/2\times N/2} + 48A_{N/8\times N/8} + 49N^2/8 - 9qN, \\ M_{N\times N} &= M_{N/2\times N/2} + 48M_{N/8\times N/8} + 15N^2/8 - 15qN. \end{aligned} \tag{77}$$

For $N = 8q$, the twiddle factors are given by $\cos\left(\frac{2\pi}{N/q}\sum_{i=1}^{2} n_i p_i\right) = \cos\left(\frac{\pi}{4}\sum_{i=1}^{2} n_i p_i\right)$, $0 \le n_1, n_2 \le q-1$. In this case, only $(3/8)\times(8q)^2 = 24q^2$ multiplications are needed in the computation of even-odd, odd-even and odd-odd output terms. Thus, the arithmetic complexity when $N = 8q$, is given by

$$\begin{aligned} A_{8q\times 8q} &= A_{4q\times 4q} + 48A_{q\times q} + 344q^2, \\ M_{8q\times 8q} &= M_{4q\times 4q} + 48M_{q\times q} + 24q^2. \end{aligned} \tag{78}$$

The initial values for $q = 1$ are given by

$$\begin{aligned} &A_{1\times 1} = 0, A_{2\times 2} = 8, A_{4\times 4} = 64, A_{8\times 8} = 384, \\ &M_{1\times 1} = 0, M_{2\times 2} = 0, M_{4\times 4} = 0, M_{8\times 8} = 24. \end{aligned} \tag{79}$$

Similarly, for $q = 3$

$$\begin{aligned} &A_{3\times 3} = 47, A_{6\times 6} = 260, A_{12\times 12} = 1328, A_{24\times 24} = 6680, \\ &M_{3\times 3} = 4, M_{6\times 6} = 16, \ M_{12\times 12} = 64, \ M_{24\times 24} = 472. \end{aligned} \tag{80}$$

The flowgraph of length-3×3 DHT is shown in Fig. 4.

Tables I and II show respectively the arithmetic complexities for $q = 1$ and $q = 3$ of the proposed algorithm, the radix-(2×2)/(4×4) algorithms in [29] and [30], and the row-column method based on the 1-D algorithm in [3]. It can be seen from these tables that the proposed algorithm can save almost 10% multiplications and has lower total number of additions and multiplications than that of the algorithms in [29] and [30], and saves about 60% multiplications and 40% additions compared to the row-column method.

B. Data transfers

Based on the fact that the on-chip memory can be accessed faster than external memory (off-chip memory), an appropriate use of the internal registers (on-chip memory) is becoming an important strategy. It is assumed that sufficient registers are available in the processor without using any intermediate transfer operation. The implementation scheme of the proposed algorithm is shown in Fig. 5. The implementation of the butterfly for a given value of n_1, n_2, consists of reading two points from the external memory of the processor and performing the operations of addition and subtraction using these two points. The result of addition is returned to the external memory whereas that of the subtraction is kept in an internal register. The points kept in the processor are grouped to form $\mathbf{y}^{2/8}_{01}$, $\mathbf{y}^{2/8}_{10}$ and $\mathbf{y}^{2/8}_{11}$ in (48), (56), (66) and to compute the outputs of (48), (56), (66), which are the inputs of the $N/8\times N/8$ DHT in (40) and (42). The number of data transfers is analyzed as follows:

1) Reading all the input terms $x(n_1, n_2)$, $x(n_1, n_2+N/2)$, $x(n_1+N/2, n_2)$, and $x(n_1+N/2, n_2+N/2)$ for $0 \le n_1, n_2 \le N/2-1$ from external memory, which requires $N^2/2$ data transfers.

2) Writing $y^{2/8}_{00}(n_1,n_2)$ for $0 \le n_1, n_2 \le N/2-1$, into external memory to form the input sequences of (32). It needs $N^2/8$ data transfers.

3) Writing $y^{2/8}_{01}(n_1,n_2)$, $y^{2/8}_{10}(n_1,n_2)$ and $y^{2/8}_{11}(n_1,n_2)$ for $0 \le n_1, n_2 \le N/2-1$, into the external memory to form the input sequences of forty-eight $N/8\times N/8$ DHTs in (48), (56) and (66). It requires $3N^2/8$ data transfers.

4) Computation of (37) needs $3N^2/4$ data transfers.

Thus, the data transfers of the proposed algorithm are given by

$$D^{2/8}_{N\times N} = D^{2/8}_{N/2\times N/2} + 48D^{2/8}_{N/8\times N/8} + 7N^2/4, \quad N > 8, \tag{81}$$

with

$$\begin{aligned} &D^{2/8}_{1\times 1} = 0, \quad D^{2/8}_{2\times 2} = 4, \quad D^{2/8}_{4\times 4} = 20, \\ &D^{2/8}_{8\times 8} = N^2 + D^{2/8}_{4\times 4} + 48D^{2/8}_{1\times 1} = 84. \end{aligned} \tag{82}$$

Similarly, the data transfers of the radix-(2×2)/(4×4) algorithm in [29] and [30] are given by

$$D^{2/4}_{N\times N} = D^{2/4}_{N/2\times N/2} + 12D^{2/4}_{N/4\times N/4} + 7N^2/4, \quad N \ge 8, \tag{83}$$

with

$$D^{2/4}_{1\times 1} = 0, \quad D^{2/4}_{2\times 2} = 4, \quad D^{2/4}_{4\times 4} = 20. \tag{84}$$

Tables III and IV show respectively the number of data transfers for $q = 1$ and $q = 3$ of the different methods for certain value of N. The proposed algorithm leads to a reduction of data transfers over 20% compared to radix-(2×2)/(4×4) algorithm in [29] and [30] and approximately 60% compared to the row-column algorithm.

C. Twiddle factors

It is assumed that the coefficients required by the special butterflies, such as $\sqrt{2}/2$, $\cos(\pi/8)$ and $\sin(\pi/8)$ are initialized and kept in the internal registers of the processor during the processing time. Firstly, equations (48), (56) and (66) require $3\times 16\times (N/8)\times(N/8) = 3N^2/4$ twiddle factors. Secondly, we can obtain the number of twiddle factors for the special cases as follows: When $p_1 = 0$ and $p_2 = 1, 3$, for a given value of n_1 and $0 \le n_2 \le N/8-1$, the number of the twiddle factors required in this case can be derived in a way similar to the one presented in [3], it is $8q$. So that the total number of the twiddle factors for the case where $p_1 = 0$, $p_2 = 1, 3$, for $0 \le n_1, n_2 \le N/8-1$ is qN. Thus, for all the combinations of (p_1, p_2) in (48), (56) and (66), the number of the twiddle factors is $12qN$. Therefore, the twiddle factors of the proposed algorithm are given by

$$TF^{2/8}_{N\times N} = TF^{2/8}_{N/2\times N/2} + 48TF^{2/8}_{N/8\times N/8} + 3N^2/4 - 12qN, \ N > 8q. \tag{85}$$

For $q = 1$, we have

$$TF^{2/8}_{1\times 1} = 0 \quad TF^{2/8}_{2\times 2} = 0 \quad TF^{2/8}_{4\times 4} = 0 \quad TF^{2/8}_{8\times 8} = 0. \tag{86}$$

For $q = 3$, we have

$$TF^{2/8}_{3\times 3} = 0 \quad TF^{2/8}_{6\times 6} = 0 \quad TF^{2/8}_{12\times 12} = 0 \quad TF^{2/8}_{24\times 24} = 0. \tag{87}$$

Analyzing the twiddle factors required in the radix-(2×2)/(4×4) algorithm in [29] in a similar way, we have

$$TF^{2/4}_{N\times N} = TF^{2/4}_{N/2\times N/2} + 12TF^{2/4}_{N/4\times N/4} + 3N^2/4 - 6qN, \ N > 4q. \tag{88}$$

For $q = 1$

$$TF_{1\times1}^{2/8} = 0 \qquad TF_{2\times2}^{2/8} = 0 \qquad TF_{4\times4}^{2/8} = 0. \tag{89}$$

For $q = 3$

$$TF_{3\times3}^{2/8} = 0 \qquad TF_{6\times6}^{2/8} = 0 \qquad TF_{12\times12}^{2/8} = 0. \tag{90}$$

Tables V and VI show respectively the comparison of twiddle factors for $q = 1$ and $q = 3$ for different methods. It can be seen that, in most cases, our algorithm saves approximately 35% compared to the algorithm in [29] and [30], and approximately 40% compared to the row-column method.

D. Area complexity and time complexity analysis

In this subsection, we compare the area complexity and time complexity of the proposed split-radix-(2×2)/(8×8) algorithm with the split-radix-(2×2)/(4×4) algorithm presented in [29] and the row-column method using [3] based on single multipliers, multiplier/accumulators and butterfly processors. The algorithm presented in [30] has the same area and time complexity as that of [29].

1) Systems Using Multiplier or Multiplier/Accumulator Primitives

As described in [9], in systems using software in conjunction with a hardware adder to accomplish multiplications, such as general-purpose microcomputers without coprocessors, the computation time of the algorithm is determined primarily by the number of multiplications. In systems using a single hardware multiplier, such as DSP microcomputers, both multiplies and additions contribute heavily in determining the run time. In both cases, the area complexity (the area of one multiplier or multiplier/accumulator) of three algorithms is the same. Therefore, the area-time complexity is determined by the computational time. As can be seen from Tables I and II, the proposed split-radix-(2×2)/(8×8) is clearly preferable to the split-radix-(2×2)/(4×4) presented in [29] and row-column method based on [3] in terms of computational time.

2) Multiprocessor Implementations Based on Butterflies

In this subsection, for simplicity, we implement strictly the algorithms according to the flowgraph. That is to say, we dedicate one multiplier (or one adder) to implement one multiplicative (or additive) operation. Let T_M and T_A be respectively the computational time of one multiplication and one addition. The designed modules of the three algorithms are described as follows.

a) Implementation of the split-radix-(2×2)/(8×8) algorithm with 5 modules

The first module is used to implement (33), (39), (60) and (70) to obtain $y_{00}^{2/8}(n_1,n_2)$, $y_{01}^{2/8}(n_1,n_2)$, $y_{10}^{2/8}(n_1,n_2)$ and $y_{11}^{2/8}(n_1,n_2)$ for $0 \le n_1,\ n_2 \le N/2-1$. We design the butterfly shown in Fig. 1 as type-I butterfly, which consists of four radix-2 butterflies. Totally, $(N\times N)/4$ type-I butterflies are required. The computational time of the first module is $2T_A$.

The second module is designed to obtain the even-even output terms, that is, to implement one $(N/2)\times(N/2)$ DHT. The computational time of the second module is $T_{N/2\times N/2}^{2/8}$.

The third module is used to obtain the even-odd output terms, including (48) and 16 parallel $(N/8)\times(N/8)$ DHTs and one third data processing of (37). We divide further this module into 3 smaller modules. The module 3-1 is used to implement (48). We design the butterfly shown in Fig. 3 as the type-II butterfly, which can be decomposed into five stages. The first stage consists of four radix-2 butterflies and four modified multiplier-adder butterflies. The second stage, the third stage and the fourth stage consist of six, six and eight radix-2 butterflies, respectively. The last stage consists of eight multiplier-adder butterflies. Note that for the last stage, we assume that some special twiddle factors, such as $\sqrt{2}/2$, $\cos(\pi/8)$ and $\sin(\pi/8)$, are implemented by the special butterflies. Therefore, $(N/8)\times(N/8)$ type-III butterflies are required. The computational time is $2T_M+5T_A$. The module 3-2 is used to implement 16 parallel $(N/8)\times(N/8)$ DHTs. The computational time is $T_{N/8\times N/8}^{2/8}$. The module 3-3 is used to implement one third data of (37). This module is implemented by $8\times(N/8)\times(N/8)$ radix-2 butterflies. The computational time is T_A. Totally, The computational time of the third module is $2T_M + 5T_A + T_{N/2\times N/2}^{2/8} + T_A$.

The fourth module is used to obtain the odd-even output terms, including (56) and 16 parallel $(N/8)\times(N/8)$ DHTs and one third data processing of (37).

The fifth module is used to obtain the odd-odd output terms, including (66) and 16 parallel $(N/8)\times(N/8)$ DHTs and one third data processing of (37).

The design of the fourth and the fifth module is similar to the third one. We assume that when the first module is finished, the second module, the third module, the fourth module and the fifth module are working in parallel. Under this assumption, the total computational time for the proposed algorithm is given by

$$T_{N\times N}^{2/8} = 2T_A + \max\left\{T_{N/2\times N/2}^{2/8}, 2T_M + 6T_A + T_{N/8\times N/8}^{2/8}\right\}. \tag{91}$$

For $q = 1$, the initial values of (91) are

$$\begin{aligned} T_{2\times2}^{2/8} &= 2T_A, \\ T_{4\times4}^{2/8} &= 2T_A + T_{2\times2}^{2/8} = 4T_A. \end{aligned} \tag{92}$$

For $q = 3$, as can be seen in Fig. 4, the initial values of (91) are

$$\begin{aligned} T_{3\times3}^{2/8} &= T_M + 9T_A, \\ T_{6\times6}^{2/8} &= 2T_A + T_{3\times3}^{2/8} = T_M + 11T_A \\ T_{12\times12}^{2/8} &= T_M + 5T_A + T_{3\times3}^{2/8} = 2T_M + 14T_A. \end{aligned} \tag{93}$$

Substituting the above initial values into $T_{N/2\times N/2}^{2/8}$ and $2T_M + 6T_A + T_{N/8\times N/8}^{2/8}$, we find that the former is always smaller than the latter. Thus, (91) becomes

$$T_{N\times N}^{2/8} = 2T_M + 8T_A + T_{N/8\times N/8}^{2/8}. \tag{94}$$

Since the multiplication by (1/2) in Fig. 4 is simply a right-shift operation, hence, the computational time is not taken into account in this analysis.

b) Implementation of the split-radix-(2×2)/(4×4) algorithm with 5 modules

The first module is used to implement (6), (12), (26) and (31) to obtain $y_{00}^{2/4}(n_1,n_2)$, $y_{01}^{2/4}(n_1,n_2)$, $y_{10}^{2/4}(n_1,n_2)$, and $y_{11}^{2/4}(n_1,n_2)$ for $0 \le n_1, n_2 \le N/2-1$. The computational time is $2T_A$.

The second module is used to obtain the even-even output terms. The computational time is $T_{N/2\times N/2}^{2/4}$.

The third module is used to obtain the even-odd output terms, including (17), 4 parallel $(N/4)\times(N/4)$ DHTs and one third data processing of (10). The implementation is similar to that of the split-radix-(2×2)/(8×8) algorithm. The computational time of the third module is $T_M + 2T_A + T_{N/2\times N/2}^{2/4} + T_A$.

The fourth module and the fifth module are used to obtain the odd-even and odd-odd output terms, respectively. Their design is similar to the third module.

The total computational time for the split-radix-(2×2)/(4×4) algorithm in [29] is given by

$$\begin{aligned} T_{N\times N}^{2/4} &= 2T_A + \max\left\{T_{N/2\times N/2}^{2/4}, T_M + 3T_A + T_{N/4\times N/4}^{2/4}\right\} \\ &= T_M + 5T_A + T_{N/4\times N/4}^{2/4} \end{aligned} \tag{95}$$

The initial values of (95) for q = 1 and q = 3 are the same as those of (92) and (93).

c) Implementation of the row-column method

Using the similar implemental scheme as the aforementioned two algorithms, we can easily obtain the computational time for the 1-D split-radix-2/8 DHT algorithm [3] as follows:

$$T_N^{2/8} = 2T_A + \max\left\{T_{N/2}^{2/8}, 2T_M + 3T_A + T_{N/8}^{2/8}\right\} \tag{96}$$

For q = 1, the initial values of (96) are

$$\begin{aligned} T_2^{2/8} &= T_A, \\ T_4^{2/8} &= T_A + T_2^{2/8} = 2T_A. \end{aligned} \tag{97}$$

For q = 3, the initial values of (96) are

$$\begin{aligned} T_3^{2/8} &= T_M + 2T_A, \\ T_6^{2/8} &= T_A + T_3^{2/8} = T_M + 3T_A, \\ T_{12}^{2/8} &= T_M + 3T_A + T_3^{2/8} = 2T_M + 5T_A. \end{aligned} \tag{98}$$

Therefore, the total computational time for the row-column method is given by:

$$\begin{aligned} T_N^{RC} &= 2T_A + 2T_N^{2/8} \\ &= 6T_A + 2\max\left\{T_{N/2}^{2/8}, 2T_M + 3T_A + T_{N/8}^{2/8}\right\} \end{aligned} \tag{99}$$

The initial values of (99) for q = 1 and q = 3 are the same as those of (97) and (98).

Table VII shows the comparison of computational time for q = 1 and q = 3. As can be seen from this table, the proposed algorithm requires less computational time than row-column method based on [3] but a little more computational time than that of the algorithm in [29] and [30]. The additional time complexity will be discussed in the following.

When using the parallel implementation structure described above, the required multipliers and adders are the same as the number of multiplications and additions given in Tables I and II. Therefore, the area complexity can be directly evaluated from these two tables. It can be seen that the proposed algorithm requires less area complexity than that of the algorithm presented in [29] and [30] and the row-column method based on [3].

As a conclusion of this section, we explain why the proposed algorithm achieves the above attractive results (reductions in arithmetic complexity, data transfers and twiddle factors) compared to the algorithms in [29] and [30]. There are mainly three reasons. Firstly, the pair of special angles ($\pi/8$) and ($3\pi/8$), just like the 1-D split-radix-2/8 algorithm in [3], is taken into consideration in the proposed algorithm to reduce both the arithmetic complexity and the twiddle factors. However, these cases have not been considered in [29] and [30]. Secondly, the proposed approach, decomposing an $N\times N$ DHT into one $N/2\times N/2$ DHT and forty-eight $N/8\times N/8$ DHTs, can save the data transfer. Meanwhile, the new scheme decreases the number of multiplications at the cost of a little more additions, as can be seen in Tables I and II. Finally, the computation process is recursive, the savings in arithmetic complexity, data transfers and twiddle factors of initial values (or relative smaller transform length) are accumulated with the increases of the value of transform length N. However, for the hardware time complexity analysis, the additional time complexity of the proposed algorithm is mainly caused by the spread of cosine and sine functions in (40) and (42). This can be observed from the first stage of Fig. 3. When implementing the proposed algorithm, we have to dedicate an additional group of multipliers compared to the algorithm in [29] and [30].

IV. Software Implementation of the Proposed and Some Existing 2-D DHT Algorithms

In this section, just like [43], we compare the proposed algorithm with some existing algorithms for the 2-D DHT in terms of computer run times, which include fetch instruction time, decoding time and write back time. These algorithms have been implemented with "C" programming language and carried out on a PC machine, which has an Intel Core2 Duo CPU with speed of 2200MHz and 3072 MB RAM. The run-time of these algorithms has been calculated using Visual C++ (VC++) Version (9).

A. Comparison of the proposed algorithm with some existing 2-D DHT algorithms in terms of computer run times

We compare the proposed algorithm with the algorithms presented in [29] and [30] and the row-column method based on [3] in terms of computer run-times. Tables VIII and IX show respectively the run times required in these algorithms for q = 1 and q = 3. The times in Table VIII and IX represent the average obtained by repeating the execution of the algorithm. As it can be seen from these tables, the proposed algorithm approximately saves average 11% compared to the algorithms in [29] and [30] and 60% compared to the row-column method based on [3]. Since we use the recursive structure to implement the algorithms, the C codes are still far from optimal and there is much room for performance improvement.

B. Comparison of the proposed algorithm with some existing 2-D DHT algorithms in terms of the image compression

As stated in [44] and [45], the DHT outperforms the discrete cosine transform (DCT) in terms of the compression performance when applying to the magnetic resonance (MR) images and positron emission tomography (PET) images. Therefore, we have designed a compression scheme to evaluate the computer run time of the above noted algorithms on MR image compression and decompression. The encoder consists of applying the 2-D DHT to an MR image, and then using the set partitioning in hierarchical trees (SPIHT) algorithm [46] to encode the DHT coefficients to obtain the binary output. The decoder executes the inverse process: decoding the binary code using the inverse SPIHT algorithm, and then applying the inverse 2-D DHT, rounding the decompressed pixel values into integer. The steps of the scheme are shown in Fig. 6, where the 2-D DHT and IDHT have been calculated by the proposed algorithm and the algorithms presented in [3], [29] and [30], respectively. Fig. 7 shows an example of a 512×512 MR image compression using the aforementioned scheme. The related errors between the original image and the decompressed images, subtracted by 64 in order to be visible, are shown in the last row of Fig. 7. For this example, the compression ratio is restricted to 16:1, 32:1 and 64:1, and the computer run times and the peak signal to noise ratio (PSNR) values have been calculated. The results are shown in Table X. It can be seen that, to obtain the same PSNR values, the proposed algorithm requires less computer run time than that of the algorithms in [3], [29] and [30].

V. Conclusion

In this paper, we have proposed a split radix-(2×2)/(8×8) algorithm for 2-D DHT. Compared to the existing best algorithm presented in [29] and [30], the proposed algorithm not only preserves the good properties such as providing a wider choice on sequence lengths, having a regular computational structure and in-place computation, but also has a lower arithmetic complexity and reduces around 30% data transfers and 35% twiddle factors, which contribute significantly to the execution time of FHT algorithms. The algorithm is expressed in a simple matrix form, which facilitates the implementation of the algorithm in both software and hardware systems.

Acknowledgment

The authors are grateful to the anonymous reviewers for their constructive comments and suggestions to greatly improve both of the quality and the clarity of this paper.

References

[1] G. Bi and Y. Q. Chen, "Fast DHT algorithms for length $N= q*2^m$," *IEEE Trans. Signal Process.*, vol. 47, no. 3, pp. 900-903, Mar. 1999.

[2] A. M. Grigoryan, "A novel algorithm for computing the 1-D discrete Hartley transform," *IEEE Signal Process. Lett.*, vol. 11, no. 2, pp. 156-159, Feb. 2004.

[3] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "A new split-radix FHT algorithm for length-$q*2^m$ DHTs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 10, pp. 2031-2043, Oct. 2004.

[4] P. K. Meher, T. Srikanthan, and J. C. Patra, "Scalable and modular memory-based systolic architectures for discrete Hartley transform," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 5, pp. 1065-1077, May 2006.

[5] P. K. Meher, J. C. Patra, and M. N. S. Swamy, "High-throughput memory-based architecture for DHT using a new convolutional formulation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 7, pp. 606-610, Jul. 2007.

[6] A. Amira and S. Chandrasekaran, "Power modeling and efficient FPGA implementation of FHT for signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 3, pp. 286-295, Mar. 2007.

[7] H. Shu, Y. Wang, L. Senhadji, and L. Luo, "Direct computation of type-II discrete Hartley transform, " *IEEE Signal Process. Lett.*, vol. 14, no. 5, pp. 329-332, May 2007.

[8] P. Jain, B. Kumar, and S. B. Jain, "Fast computation of the discrete Hartley transform," *Int. J. Circ. Theor. Appl,*. doi: 10.1002/cta.574.

[9] M. A. Richards, "On hardware implementation of the split-radix FFT," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 36, no. 10, pp. 1575-1581, Oct. 1988.

[10] W. -C. Yeh and C. -W. Jen, "High-speed and low-power split-radix FFT," *IEEE Trans. Signal Process.*, vol. 51, no. 3, pp. 864–874, Mar. 2003.

[11] P. K. Meher, "Efficient systolic implementation of DFT using a low-complexity convolution-like formulation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 8, pp. 702-706, Aug. 2006.

[12] C. Cheng and K. K. Parhi, "Low-cost fast VLSI algorithm for discrete Fourier transform," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 4, pp. 791-806, Apr. 2007.

[13] C. Cheng and K. K. Parhi, "High-throughput VLSI architecture for FFT computation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 10, pp. 863-867, Oct. 2007.

[14] 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.

[15] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "A general class of split-radix FFT algorithms for the computation of the DFT of length-2^m," *IEEE Trans. Signal Process.*, vol. 55, no. 8, pp. 4127-4138, Aug. 2007.

[16] Y. -W. Lin and C. -Y. Lee, "Design of an FFT/IFFT processor for MIMO OFDM systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 4, pp. 807–815, Apr. 2007.

[17] H. -Y. Lee and I. -C. Park, "Balanced binary-tree decomposition for area-efficient pipelined FFT processing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 4, pp. 889 – 900, Apr. 2007.

[18] Y. -N. Chang, "An efficient VLSI architecture for normal I/O order pipeline FFT design," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 12, pp. 1234-1238, Dec. 2008.

[19] A. Makur, "Computational Schemes for Warped DFT and Its Inverse," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 9, pp. 2686 – 2695, Oct. 2008.

[20] Y. Voronenko and M. Püschel, "Algebraic signal processing theory: Cooley–Tukey type algorithms for real DFTs," *IEEE Trans. Signal Process.*, vol. 57, no. 1, pp. 205–222, Jan. 2009.

[21] M. Li, D. Novo, B. Bougard, T. Carlson, L. Van Der Perre, and F. Catthoor, "Generic multiphase software pipelined partial FFT on instruction level parallel architectures," *IEEE Trans. Signal Process.*, vol. 57, no. 4, pp. 1604-1615, Apr. 2009.

[22] R. Kumaresan and P. K. Gupta, "Vector-radix algorithm for 2-D discrete Hartley transform," *Proc. IEEE*, vol. 74, no. 5, pp. 755–757, May 1986.

[23] S. Bouguezel, M. N. S. Swamy, and M. O. Ahmad, "Multidimensional vector radix FHT algorithms," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 4, pp. 905-917, Apr. 2006.

[24] N. -C. Hu and F. -F. Lu, "Fast computation of the two dimensional generalized Hartley transforms," *IEE Proc. Vis. Image. Signal Process.*, vol. 142, no. l, pp. 35-39, Feb.1995.

[25] S. -J. Huang, J. -G. Wang, and H. -Z. Qiu, "Split vector radix algorithm for two dimensional Hartley transform," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 27, no. 6, pp. 865-868, Nov. 1991.

[26] W. Ma, "Number of multiplications necessary to compute length-2^n two-dimensional discrete Hartley transform DHT (2^n; 2)," *Electron. Lett.*, vol. 28, no. 5, pp. 480-482, Feb. 1992.

[27] E. A. Jonkheere and C. Ma, "Split-radix fast Hartley transform in one and two dimensions," *IEEE Trans. Signal Process.*, vol. 39, no. 2, pp. 499-503, Feb. 1991.

[28] J. -L .Wu and S.-C. Pei, "The vector split-radix algorithm for 2-D DHT," *IEEE Trans. Signal Process.*, vol. 41, no. 2, pp. 960-965, Feb. 1993.

[29] G. Bi, "Split-radix algorithm for 2-D discrete Hartley transform," *Signal Process.*, vol. 63, no. 1, pp. 45-53, Nov. 1997.

[30] G. Bi, A.C. Kot and Z. Meng, "Computation of 2D discrete Hartley transform," *Electron. Lett.*, vol. 34, no. 11, pp. 1058-1059, May 1998.
[31] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "An efficient multidimensional decimation-in-frequency FHT algorithm based on the radix-2/4 approach," *Proc. IEEE ISCAS*, vol. 3, May 2005, pp. 2405-2408.
[32] Y. -H. Zeng, G. Bi, and A. R. Leyman, "New algorithms for multidimensional discrete Hartley transform," *Signal Process.*, vol. 82, no. 8, pp.1086–1095, Aug. 2002.
[33] S. C. Chan and K. L. Ho, "Polynomial Transform Fast Hartley transform," *Proc. IEEE ISCAS*, vol. 1, Jun. 1991, pp. 642-645.
[34] Y. -H. Zeng, G. Bi, and A. C. Kot, "Fast algorithm for multi-dimensional discrete Hartley transform with size $q^{l_1} \times q^{l_2} \times \cdots \times q^{l_r}$," *Signal Process.*, vol. 82, no. 3, pp. 497–502, Mar. 2002.
[35] S. Boussakta, O. H. Alshibami, and M. Y. Aziz, "Radix-2×2×2 algorithm for the 3-D discrete Hartley transform," *IEEE Trans. Signal Process.*, vol. 49, no. 12, pp. 3145-3156, Dec. 2001.
[36] O. Alshibami and S. Boussakta, "Fast 3-D decimation-in-frequency algorithm for 3-D Hartley transform," *Signal Process.,* vol. 82, no. 1, pp. 121-126, Jan. 2002.
[37] H. Z. Shu, J. S. Wu, L. Senhadji, and L. M. Luo, "Radix-2 algorithm for the fast computation of type-III 3-D discrete W transform," *Signal Process.*, vol. 88, no. 1, pp. 210-215, Jan. 2008.
[38] S. Bouguezel, M.O. Ahmad, and M.N.S. Swamy, "A split vector-radix algorithm for the 3-D discrete Hartley transform," *IEEE Trans. Circuits Syst.-I: Regular papers*, vol. 53, no. 9, pp. 1966-1976, Sept. 2006.
[39] J. S. Wu, H. Z. Shu, L. Senhadji, and L. M. Luo, "Radix-3×3 algorithm for the 2-D discrete Hartley transform," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 6, pp. 566-570, Jun. 2008.
[40] S. C. Pei and W. Y. Chen, "Split vector-radix-2/8 fast Fourier transform," *IEEE Signal Process. Lett.*, vol. 11, no. 5, pp. 459-462, 2004.
[41] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "A split-radix algorit hm for 2-D DFT," *Proc. IEEE ISCAS*, vol. 3, May 2003, pp. 698-701.
[42] J. Granata, M. Conner, and R. Tolimieri, "The tensor product: A mathematical programming language for FFT's and other fast DSP operations," *IEEE Signal Process. Mag.*, vol. 9, no. 1, pp. 40–48, Jan. 1992.
[43] S. Boussakta and H. O. Alshibami, "Fast algorithm for the 3-D DCT-II," *IEEE Trans. Signal Process.*, vol. 52, no. 4, pp. 992-1001, Apr. 2004.
[44] J. D. Villasenor, "Alternatives to the discrete cosine transform for irreversible tomographic image compression," *IEEE Trans. Med. Imag.*, vol. 12, no. 4, pp. 803–811, Dec. 1993.
[45] R. Shyam Sunder, C. Eswaran, and N. Sriraam, "Medical image compression using 3-D Hartley transform," *Comput. Biol. Med.*, vol. 36, no. 9, pp. 958–973, Sept. 2006.
[46] A. Said and W. A. Pearlman, "A new fast and efficient implementation of an image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 3, pp. 243-250, June 1996. SPIHT Matlab Program, [Online]. Available: http://www.cipr.rpi.edu/research/SPIHT/spiht3.html.

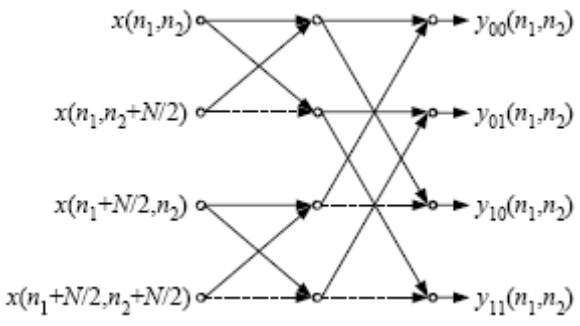
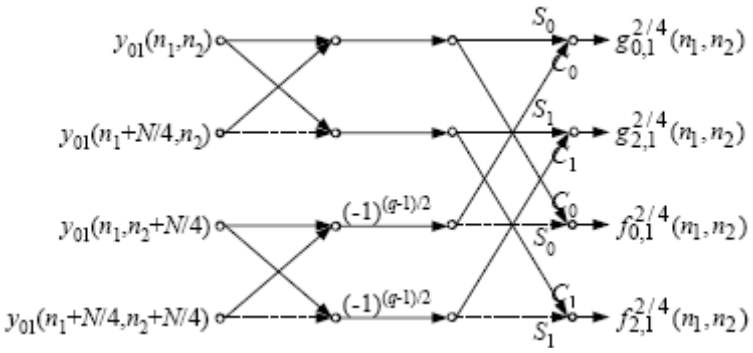
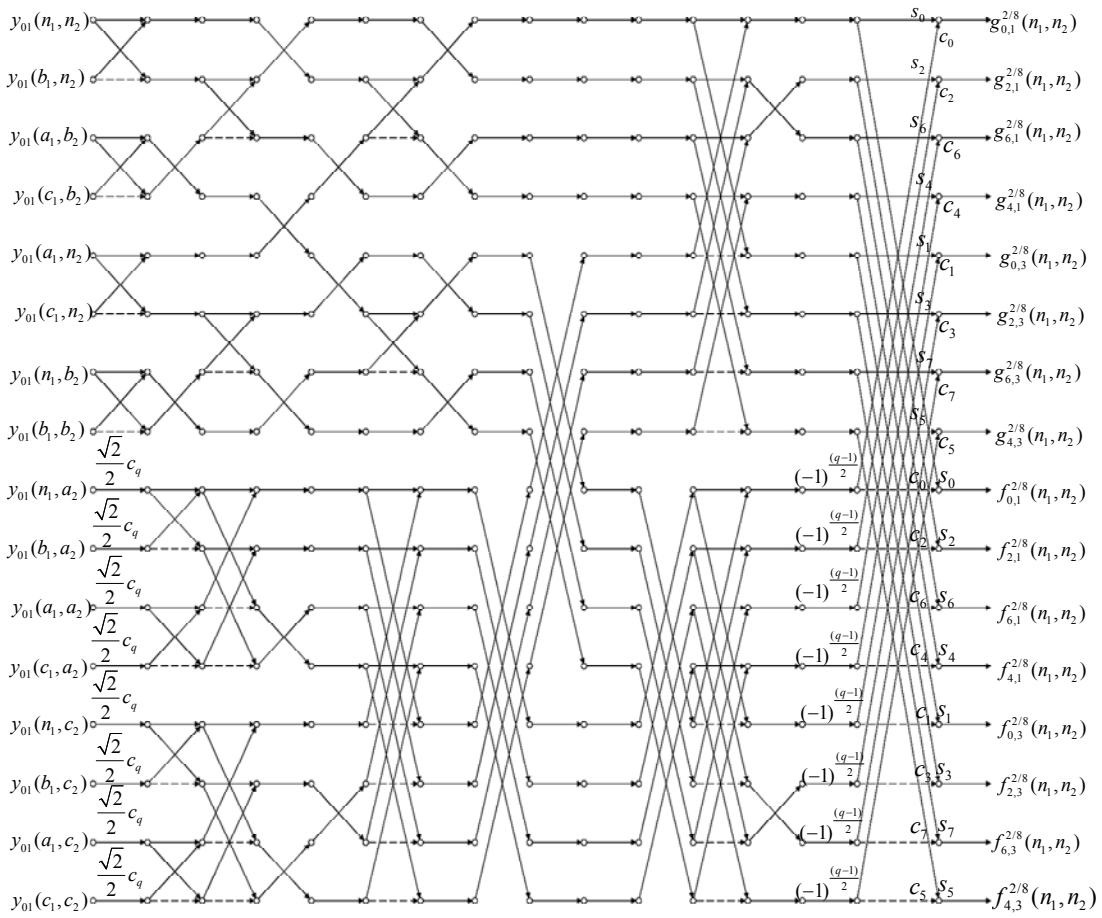


Fig. 1. Flowgraph for implementing equation (4)



$$C_0=\cos\frac{n_2\pi}{2}, S_0=\sin\frac{n_2\pi}{2}, C_1=(-1)^{n_1}\cos\frac{n_2\pi}{2}, S_1=(-1)^{n_1}\sin\frac{n_2\pi}{2}$$

Fig. 2. Flowgraph for implementing equation (17)



$a_1=n_1+N/8,\ a_2=n_2+N/8,\ b_1=n_1+N/4, b_2=n_2+N/4,\ c_1=n_1+3N/8,\ c_2=n_2+3N/8,$

$C_{eo}(p,p)$ and $S_{eo}(p,p)$ defined in (49) are as follows: $C_{eo}(p,p)=C_p, S_{eo}(p,p)=S_p,\ p=p_1+\frac{1}{2}(p_2-1)$

Fig. 3. Flowgraph for implementing equation (48)

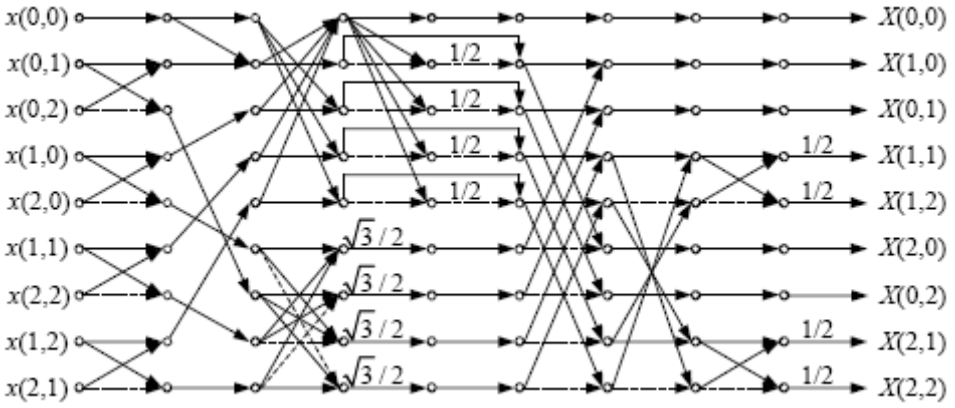
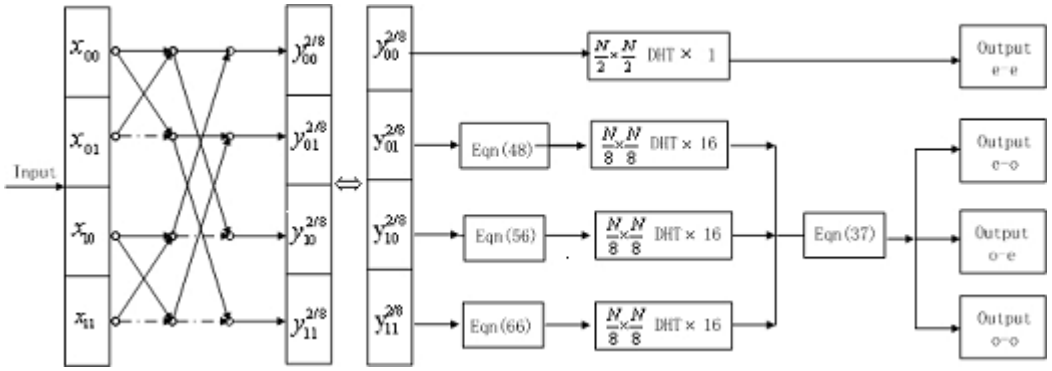


Fig. 4. Flowgraph of a length-3×3 DHT



$y_{p_1,p_2}(n_1,n_2) = y_{p_1,p_2}$, $p_1, p_2 = 0$ or 1, $x(n_1,n_2) = x_{00}, x(n_1,n_2+N/2) = x_{01}, x(n_1+N/2,n_2) = x_{10}, x(n_1+N/2,n_2+N/2) = x_{11}$

Fig. 5. The implementation scheme of the proposed algorithm

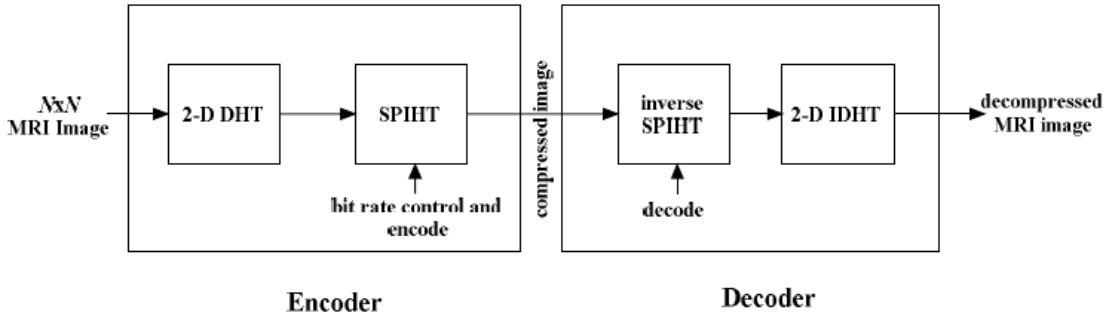


Fig. 6. Scheme of encoding and decoding process for MRI image compression

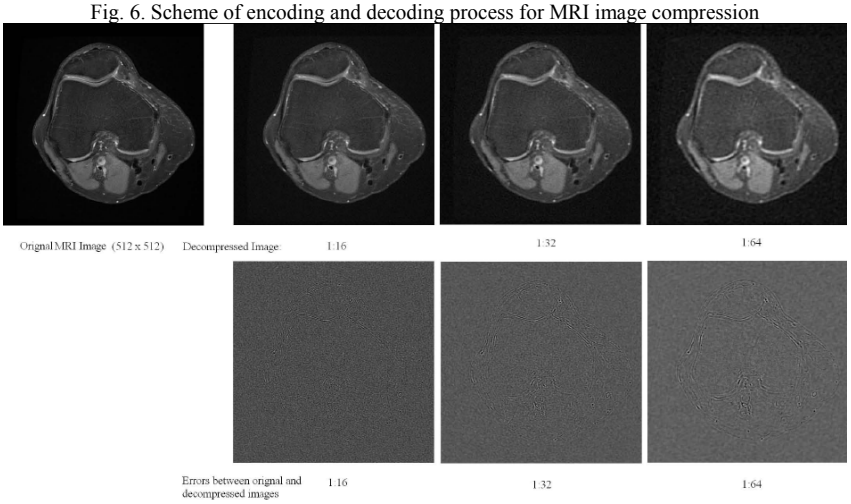


Fig. 7. Compression a 512×512 MRI image using FHT and SPIHT scheme

Table I Comparison of Arithmetic Complexities for $q = 1$ (Saving denotes the saving of arithmetic complexity compared to the algorithm in [29] and [30]).

	Algorithms in [29] and [30]			Row-column method based on [3]			Proposed Algorithm					
N×N (N)	*Muls*	*Adds*	*Total*	*Muls*	*Adds*	*Total*	*Muls*	Saving(%)	*Adds*	Saving(%)	*Total*	Saving(%)
8	24	408	432	32	608	640	24	0	408	0	432	0
16	264	2216	2480	384	3072	3456	264	0	2216	0	2480	0
32	1800	11368	13168	2688	14976	17664	1704	5.33	11272	0.84	12976	1.46
64	10536	55176	65712	15360	71168	86528	9576	9.11	55368	-0.35	64944	1.17
128	55560	260840	316400	81408	329216	410624	51048	8.12	260936	-0.04	311984	1.40
256	277992	1200712	1478704	407552	1495040	1902592	251880	9.39	1201096	-0.03	1452976	1.74
512	1333320	5443368	6776688	1951744	6703104	8654848	1195368	10.35	5459784	-0.30	6655152	1.79
1024	6232872	24305288	30538160	9109504	29696000	38805504	5596392	10.21	24398024	-0.38	29994416	1.78

Table II Comparison of Arithmetic Complexities for $q = 3$ (Saving denotes the saving of arithmetic complexity compared to the algorithm in [29] and [30]).

	Algorithms in [29] and [30]			Row-column method based on [3]			Proposed Algorithm					
N×N (N)	*Muls*	*Adds*	*Total*	*Muls*	*Adds*	*Total*	*Muls*	Saving(%)	*Adds*	Saving(%)	*Total*	Saving(%)
24	472	6680	7152	576	7776	8352	472	0	6680	0	7152	0
48	3400	31976	35376	4800	36864	41664	3400	0	31976	0	35376	0
96	20296	150440	170736	29952	171648	201600	19432	4.26	149576	0.57	169008	1.01
192	111208	689096	800304	158976	787968	946944	102568	7.77	690824	-0.25	793392	0.86
384	565576	3117608	3683184	817152	3552768	4369920	524968	7.18	3118472	-0.03	3643440	1.08
768	2764072	13886600	16650672	4021248	15814656	19835904	2529064	8.50	13890056	-0.02	16419120	1.39
1536	13048456	61311080	74359536	18935808	69765120	88700928	11806888	9.52	61458824	-0.24	73265712	1.47
3072	60290152	268030664	328320816	87429120	305012736	392441856	54561832	9.50	268865288	-0.31	323427120	1.49

Table III Comparison of Data Transfers for $q = 1$ (Saving1 and Saving2 denote respectively the saving of data transfers compared to the algorithm in [29] and [30], and the row-column algorithm based on [3]).

N×N (N)	Algorithms in [29] and [30]	Row-column method based on [3]	Proposed algorithm	Saving1 (%)	Saving2 (%)
8	180	448	84	53.33	81.25
16	868	2432	724	16.59	70.23
32	4820	9984	3476	27.88	65.18
64	22404	44544	14676	34.49	67.05
128	108916	207872	78100	28.29	62.43
256	492452	931840	359636	26.97	61.41
512	2258196	4075520	1522836	32.56	62.63
1024	10002628	17948672	7106644	28.95	60.41

Table IV Comparison of Data Transfers for $q = 3$(Saving1 and Saving2 denote respectively the saving of data transfers compared to the algorithm in [29] and [30], and the row-column algorithm based on [3]).

N×N (N)	Algorithms in [29] and [30]	Row-column method based on [3]	Proposed algorithm	Saving1 (%)	Saving2 (%)
24	2368	6144	1936	18.24	68.49
48	11776	28032	10048	14.67	64.16
96	56320	128256	47680	15.34	62.82
192	262144	563712	205120	21.75	63.61
384	1196032	2466816	945472	20.95	61.67
768	5373952	10807296	4266304	20.61	60.52
1536	23855100	46731264	18240830	23.53	60.97
3072	104857600	200712192	80138560	23.57	60.07

Table V Comparison of Twiddle Factors for $q = 1$ (Saving1 and Saving2 denote respectively the saving of twiddle factors compared to radix algorithm in [29] and [30], and the row-column algorithm based on [3])

N×N (N)	Algorithms in [29] and [30]	Row-column method based on [3]	Proposed algorithm	Saving1 (%)	Saving2 (%)
8	0	0	0	0	0
16	96	0	0	100	0
32	672	512	384	42.86	25.00
64	4512	4096	2688	40.43	34.38
128	24096	22528	13440	44.22	40.34
256	125856	122880	77952	38.06	36.56
512	608544	630784	397440	34.69	36.99
1024	2899104	3014656	1816704	37.34	39.74

Table VI Comparison of Twiddle Factors for $q = 3$ (Saving1 and Saving2 denote respectively the saving of twiddle factors compared to radix algorithm in [29] and [30], and the row-column algorithm based on [3])

$N\times N$ (N)	Algorithms in [29] and [30]	Row-column method based on [3]	Proposed algorithm	Saving1 (%)	Saving2 (%)
24	0	0	0	0	0
48	864	0	0	100	0
96	6048	4608	3456	42.86	25.00
192	40608	36864	24192	40.42	34.38
384	216864	202752	120960	44.22	40.34
768	1132704	1105920	701568	38.06	36.56
1536	5476896	5677056	3576960	34.69	36.99
3072	26091936	27131904	16350336	37.34	39.74

Table VII Comparison of Computational Time (based on hardware implementation analysis) for $q = 1$ and $q = 3$. T_M and T_A are the computational time of one multiplication and one addition, respectively.

q=1				q=3			
$N\times N$ (N)	Algorithms in [29] and [30]	Row-column method based on [3]	Proposed algorithm	$N\times N$ (N)	Algorithms in [29] and [30]	Row-column method based on [3]	Proposed algorithm
8	T_M+7T_A	4T_M+12T_A	2T_M+8T_A	24	2T_M+16T_A	6T_M+16T_A	3T_M+17T_A
16	T_M+9T_A	4T_M+16T_A	2T_M+10T_A	48	3T_M+19T_A	6T_M+20T_A	3T_M+19T_A
32	2T_M+12T_A	4T_M+20T_A	2T_M+12T_A	96	3T_M+21T_A	8T_M+22T_A	4T_M+22T_A
64	2T_M+14T_A	8T_M+22T_A	4T_M+16T_A	192	4T_M+24T_A	10T_M+26T_A	5T_M+25T_A
128	3T_M+17T_A	8T_M+26T_A	4T_M+18T_A	384	4T_M+26T_A	10T_M+30T_A	5T_M+27T_A
256	3T_M+19T_A	8T_M+30T_A	4T_M+20T_A	768	5T_M+29T_A	12T_M+32T_A	6T_M+30T_A
512	4T_M+22T_A	12T_M+32T_A	6T_M+24T_A	1536	5T_M+31T_A	14T_M+36T_A	7T_M+33T_A
1024	4T_M+24T_A	12T_M+36T_A	6T_M+26T_A	3072	6T_M+34T_A	14T_M+40T_A	7T_M+35T_A

Table VIII Comparison of Computer Run Time for $q = 1$ on an Intel Core2 Duo CPU using the VC++ compiler (Saving1, Saving2 and Saving3 denote respectively the saving of Computer run time compared to algorithm in [29], [30], and row-column algorithm based on [3])

	Computer run time (s)				Saving (%)		
$N\times N$ (N)	Algorithm in [29]	Algorithm in [30]	Row-column method based on [3]	Proposed Algorithm	Saving1	Saving2	Saving3
8	0.000197	0.000198	0.000453	0.0001489	24.42	24.80	67.13
16	0.000721	0.000715	0.002090	0.0006562	8.99	8.22	68.60
32	0.003163	0.003196	0.008625	0.0028138	11.04	11.96	67.38
64	0.012201	0.012277	0.032723	0.0102803	15.74	16.26	68.58
128	0.051230	0.052806	0.135540	0.0481164	6.08	8.88	65.50
256	0.203512	0.206572	0.556896	0.1897476	6.76	8.14	65.93
512	0.843680	0.863356	2.202470	0.7184260	14.85	16.79	67.38
1024	3.355970	3.4331520	8.798221	3.1265798	6.84	8.93	64.46

Table IX Comparison of Computer Run Time for $q = 3$ on an Intel Core2 Duo CPU using the VC++ compiler(Saving1, Saving2 and Saving3 denote respectively the saving of Computer run time compared to algorithm in [29], [30], and row-column algorithm based on [3])

	Computer run time (s)				Saving (%)		
$N\times N$ (N)	Algorithm in [29]	Algorithm in [30]	Row-column method based on [3]	Proposed Algorithm	Saving1	Saving2	Saving3
24	0.000855	0.000862	0.001668	0.000683	20.12	20.77	59.05
48	0.003731	0.003739	0.007536	0.003112	16.59	16.77	58.70
96	0.012270	0.012965	0.032201	0.011669	4.90	7.13	63.76
192	0.056782	0.055875	0.121261	0.050535	11.00	9.56	58.33
384	0.234900	0.234696	0.493468	0.213239	9.22	9.14	56.79
768	0.918632	0.939634	2.082430	0.840478	8.51	10.55	59.64
1536	3.843278	3.788372	8.141732	3.543753	7.79	6.46	56.47
3072	18.152742	18.214835	46.228839	16.788220	7.52	7.83	63.68

Table X Comparison of running time and PSNR for MRI image compression

		Algorithm in [29]	Algorithm in [30]	Row-column method based on [3]	Proposed Algorithm
1:16	Compress(DHT/Total)	0.8388/15.3988	0.8527/15.4127	2.1720/16.7320	0.7042/15.264
	Decompress(DHT/Total)	0.8407/12.6707	0.8542/12.6842	2.1900/14.0200	0.7067/12.5367
	PSNR(dB)	26.81	26.81	26.81	26.81
1:32	Compress (DHT/Total)	0.8388/5.4888	0.8527/5.5027	2.1720/6.8220	0.7042/5.3542
	Decompress (DHT/Total)	0.8481/4.5981	0.8482/4.5982	2.1951/5.9451	0.7075/4.4575
	PSNR(dB)	25.88	25.88	25.88	25.88
1:64	Compress (DHT/Total)	0.8388/2.9588	0.8527/2.9727	2.1720/4.2920	0.7042/2.8242
	Decompress (DHT/Total)	0.8479/1.9479	0.8563/1.9563	2.2005/3.3005	0.7100/1.8100
	PSNR(dB)	24.76	24.76	24.76	24.76