An Efficient Algorithm for the 2-D Discrete Cosine Transform

PeiZong Lee [a,1,2] and Gau-Shin Liu [b]

[a] *Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C.*

[b] *Vanguard International Semiconductor Corporation, Hsin Chu, Taiwan, R.O.C.*

Abstract

In this paper, we present an efficient algorithm for computing the 2-D discrete cosine transform (DCT). First, we arrange the input data matrix in a certain order. Second, we formulate the output data matrix to a block structured matrix-vector multiplication, and show that the block structured matrix has the same properties as the 1-D DCT kernel matrix. Therefore, the 2-D DCT problem is reduced to the 1-D DCT problem. We thus can provide a procedure for computing the 2-D DCT which is similar to that of the 1-D DCT. The primary appeal of this algorithm lies in not only its low computational cost but also its regular structure as the transform size increases.

Zusammenfassung

In dieser Arbeit wird ein effizienter Algorithmus zur Berechnung der 2-D diskreten Cosinustransformation (DCT) vorgestellt. Die Input-Datenmatrix wird zuerst in einer speziellen Ordnung arrangiert. Dann formulieren wir die Outputmatrix als eine blockstrukturierte Matrix-Vektor Multiplikation und zeigen, daß letztere die gleichen Eigenschaften wie die 1-D DCT Kernmatrix aufweist. Das 2-D DCT Problem reduziert sich somit auf ein 1-D DCT Problem. Dies gibt eine Prozedur zur Berechnung der 2-D DCT, die ähnlich der für die 1-D DCT ist. Der Hauptvorteil dieses Algorithmus liegt in seiner Effizienz wie auch in seiner regulären Struktur bei wachsender Transformationsgröße.

Résumé

Dans cet article, nous présentons un algorithme efficace pour calculer le transforme de cosinus discretisé en 2 dimensions (DCT). Pour commencer, nous arrangeons la matrice de données dans un certain ordre. Ensuite nous formulons la matrice qu'on désire calculer dans une structure de blocs de multiplications de matrices et vecteurs, et nous montrons que la matrice en structure de blocs a les mêmes propriétés que le DCT en une dimension. Le problème DCT en 2 dimensions est ainsi réduit à un problême DCT en 1 dimension. L'avantage principal de cet algorithme est non seulement son coût de calcul bas, mais aussi sa structure régulière quand la taille du transforme augmente.

Preprint submitted to Elsevier Science *7 August 1996*

Keywords: 2-D discrete cosine transform, recursive algorithm, signal-flow graph.

1 Introduction

This paper is concerned with designing a fast algorithm for computing the 2-D discrete cosine transform (DCT). Because the DCT performs much like the theoretically optimal Karhunen-Loeve transform for first-order Markov stationary random data, the DCT has found wide applications in speech and image processing, as well as telecommunication signal processing for the purpose of data compression, feature extraction, image reconstruction, and filtering. As these applications are key techniques used in multi-media research, DCT algorithms thus play an important role in many multi-media applications. A complete survey of DCT algorithms is provided in [15].

For the fast computation of the 2-D DCT, the conventional approach is the row-column method, which requires evaluating $2N$ sets of N-point 1-D DCT's for an $(N \times N)$-point 2-D DCT. Kamangar and Rao [9], who arranged the 2-D input data and output data into 1-D arrays in lexicographical order, wrote the needed 2-D transform coefficients as the Kronecker product of the two 1-D DCT coefficient matrices, which yielded the sparse matrix factorization for that 2-D coefficient matrix. Haque [7], Chan and Ho [1], and Lee and Huang [12] derived 2-D DCT recursive algorithms based on Lee's [10], Hou's [8], and Lee and Huang's [12] 1-D DCT algorithms, respectively.

Vetterli mapped the 2-D DCT into a 2-D DFT plus a number of rotations, and the 2-D DFT was then computed by polynomial transform techniques [16]. His algorithm, while achieving substantial computational savings, had a rather involved structure. Duhamel and Guillemot provided a direct 2-D DCT algorithm based on polynomial transform techniques [5]. They showed that the resulting signal-flow graphs were simple and repetitive; however, they also mentioned in their paper that its derivation was complicated and it was fairly difficult to obtain a closed-form expression for the arithmetic complexity of such an algorithm.

Cho and Lee [2,3] proposed a fast 2-D DCT algorithm, in which an $(N \times N)$-point 2-D DCT could be obtained by computing N sets of N-point 1-D DCT's as well as some additions. That is, the number of multiplications required for the proposed algorithm is the same as that required for computing N sets of

[1] Corresponding author. Fax: +886 (2) 782-4814; e-mail: leepe@iis.sinica.edu.tw.

[2] Partially supported by the NSC under Grants NSC 84-2213-E-001-003, NSC 85-2213-E-001-006 and 86-2213-E-001-006.

N-point 1-D DCT's. Therefore, the complexity of their algorithm is greatly reduced. However, according to their later comments [4], the signal-flow graph for the post-addition stage seems very complicated, and the order of the output index seems to be irregular because the post-addition stage was not based on the mathematical expressions. Consequently, derivation of the signal-flow graph becomes complicated as the transform size increases.

Recently, Cho, Yun and Lee elaborated their previous work and presented systematic expressions for the post-addition stage [4]. The structure of the signal-flow graph shows a regularity similar to that of Hou's 1-D DCT algorithm [8]. However, from our point of view, their algorithm still has a somewhat involved structure. First, before performing 1-D DCT's, they need to calculate a pre-addition stage, in which the plus operations and the minus operations cross each other and they appear in an interleaved manner. This is because the pre-addition stage contains two types of butterfly operations. We found that the pre-addition stage would be less complex if it contains only one kind of butterfly operation, similar to the ones in the 1-D discrete Fourier transform or in the 1-D DCT. Second, they divide the post-addition stage into two parts and solve them using two methods independently. We found that it is possible to design a 2-D DCT algorithm which does not require any pre-addition stage, and in which the computation of its post-addition stage is similar to the computation of the 1-D DCT.

It is our goal in this paper to develop a fast and simple 2-D DCT algorithm. We will first arrange the input data matrix in a certain order. Then we formulate the output data matrix as a block structured matrix-vector multiplication, and show that the block structured matrix has the same properties as the 1-D DCT kernel matrix. Hence, the 2-D DCT problem is reduced to the 1-D DCT problem. We are thus able to provide a procedure for computing the 2-D DCT which is similar to that of the 1-D DCT.

The rest of this paper is organized as follows. In Section 2, we identify intrinsic properties of the 2-D DCT. In Section 3 we provide an efficient 2-D DCT algorithm. We also compare our algorithm with related research works. Finally, some concluding remarks are given in Section 4.

2 Properties of the 2-D DCT

For a given input data matrix $\mathbf{x}_{N\times N} = [x_{i,j}]$, $0 \leq i,\ j \leq N-1$, the 2-D DCT output matrix $\mathbf{y}_{N\times N} = [y_{m,n}]$, $0 \leq m,\ n \leq N-1$, is defined by

$$y_{m,n} = \frac{2}{N}\,\epsilon(m)\,\epsilon(n)\,\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} x_{i,j}\,\cos(\frac{(2i+1)m}{2N}\pi)\cos(\frac{(2j+1)n}{2N}\pi)\,, \quad (1)$$

where

$$\epsilon(k) = \begin{cases} \frac{1}{\sqrt{2}}\ , & \text{for } k = 0 \\ 1\ , & \text{for } 1 \leq k \leq N-1 \end{cases}$$

and $k = m$ or n.

In this paper, we want to derive a recursive formula for the 2-D DCT. For convenience, we assume throughout this paper that N has a value of 2^i, where i is an integer and $i \geq 1$. We will remove $\epsilon(m)$, $\epsilon(n)$, and also the normalization factor $\frac{2}{N}$ in Equation (1), since they can be introduced in a subsequent step. Therefore, from now on, we deal with a simplified version of Equation (1):

$$y_{m,n} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x_{i,j} \cos(\frac{(2i+1)m}{2N}\pi) \cos(\frac{(2j+1)n}{2N}\pi). \tag{2}$$

Using the trigonometric identity $\cos A \cos B = \frac{1}{2}[\cos(A+B) + \cos(A-B)]$ in (2) results in

$$y_{m,n} = \frac{1}{2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x_{i,j} [\cos(\frac{(2i+1)m + (2j+1)n}{2N}\pi) + \cos(\frac{(2i+1)m - (2j+1)n}{2N}\pi)]. \tag{3}$$

We now study the relationship between $(2i+1)$ and $(2j+1)$, for $0 \leq i,\ j \leq N-1$. For every pair of (i,j), there exists exactly one p, such that exactly one of the following conditions holds:

$$a: \quad (2j+1) = p(2i+1) - 2Nq_a \tag{4}$$

$$b: \quad (2j+1) = 2Nq_b - p(2i+1), \tag{5}$$

where p is an odd integer, $p \in W = \{1,\ 3,\ 5,\ \cdots,\ N-1\}$, q_a and q_b are integers depending on p and i. Hence, j is a function of p and i depending on Equations (4) and (5). Table 1 shows the 64 corresponding pairs of (i,j) when $N = 8$.

We now further elaborate upon Equations (4) and (5). We separate Equation (4) into two cases depending on whether q_a is even or odd. In (4), if q_a is even, we represent Equation (4) as

$$(2j_{(p,a,i,e)} + 1) = p(2i+1) - 2Nq_{(p,a,i,e)}, \tag{6}$$

Table 1
When $N = 8$ '(a)' indicates that (i, j) is based on Equation (4), and '(b)' specifies that (i, j) is based on Equation (5).

$p = 1$		$p = 3$		$p = 5$		$p = 7$	
(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
(0,0)	(0,7)	(0,1)	(0,6)	(0,2)	(0,5)	(0,3)	(0,4)
(1,1)	(1,6)	(1,4)	(1,3)	(1,7)	(1,0)	(1,2)	(1,5)
(2,2)	(2,5)	(2,7)	(2,0)	(2,4)	(2,3)	(2,1)	(2,6)
(3,3)	(3,4)	(3,2)	(3,5)	(3,1)	(3,6)	(3,0)	(3,7)
(4,4)	(4,3)	(4,5)	(4,2)	(4,6)	(4,1)	(4,7)	(4,0)
(5,5)	(5,2)	(5,0)	(5,7)	(5,3)	(5,4)	(5,6)	(5,1)
(6,6)	(6,1)	(6,3)	(6,4)	(6,0)	(6,7)	(6,5)	(6,2)
(7,7)	(7,0)	(7,6)	(7,1)	(7,5)	(7,2)	(7,4)	(7,3)

and if q_a is odd we represent Equation (4) as

$$(2j_{(p,a,i,o)} + 1) = p(2i + 1) - 2Nq_{(p,a,i,o)}, \tag{7}$$

where the indices $''a''$ and $''e''$ specify even q_a and the indices $''a''$ and $''o''$ stand for old q_a. Likewise we separate Equation (5) into two cases depending on whether q_b is even or odd. In (5), if q_b is even, we represent Equation (5) as

$$(2j_{(p,b,i,e)} + 1) = 2Nq_{(p,b,i,e)} - p(2i + 1), \tag{8}$$

and if q_b is odd we represent Equation (5) as

$$(2j_{(p,b,i,o)} + 1) = 2Nq_{(p,b,i,o)} - p(2i + 1), \tag{9}$$

where the indices $''b''$ and $''e''$ mean even q_b and the indices $''b''$ and $''o''$ stand for old q_b. Note that, for every pair of (p, i), $q_b = q_a + 1$. Therefore, if q_a is even then q_b is odd, and if q_a is odd then q_b is even. Hence, either both Equations (6) and (9) hold and both Equations (7) and (8) do not hold, or both Equations (6) and (9) do not hold and both Equations (7) and (8) hold.

We then replace $(2j + 1)$ in (3) with the equations in (6)–(9), yielding

$$y_{m,n} = \frac{1}{2}\sum_{p\in W}\sum_{i=0}^{N-1}[(x_{i,j_{(p,a,i,e)}} + x_{i,j_{(p,b,i,e)}}) + (-1)^n(x_{i,j_{(p,a,i,o)}} + x_{i,j_{(p,b,i,o)}})] \times [\cos(\tfrac{(2i+1)(m+np)}{2N}\pi) + \cos(\tfrac{(2i+1)(m-np)}{2N}\pi)],$$

where $x_{i,j_{(p,a,i,e)}} = x_{i,j}$ if j satisfies (6), and otherwise $x_{i,j_{(p,a,i,e)}} = 0$. Similarly, $x_{i,j_{(p,a,i,o)}} = x_{i,j}$ if j satisfies (7), and otherwise $x_{i,j_{(p,a,i,o)}} = 0$; $x_{i,j_{(p,b,i,e)}} = x_{i,j}$ if j satisfies (8), and otherwise $x_{i,j_{(p,b,i,e)}} = 0$; and $x_{i,j_{(p,b,i,o)}} = x_{i,j}$ if j satisfies (9), and otherwise $x_{i,j_{(p,b,i,o)}} = 0$. Let

$$\begin{aligned} A_{p,i} &= (x_{i,j_{(p,a,i,e)}} + x_{i,j_{(p,b,i,e)}}) \\ B_{p,i} &= (x_{i,j_{(p,a,i,o)}} + x_{i,j_{(p,b,i,o)}}). \end{aligned}$$

Then if n is even,

$$y_{m,n} = \frac{1}{2} \sum_{p \in W} \sum_{i=0}^{N-1} (A_{p,i} + B_{p,i}) \, [\cos(\frac{(2i+1)(m+np)}{2N}\pi) + \cos(\frac{(2i+1)(m-np)}{2N}\pi)], \tag{10}$$

and if n is odd,

$$y_{m,n} = \frac{1}{2} \sum_{p \in W} \sum_{i=0}^{N-1} (A_{p,i} - B_{p,i}) \, [\cos(\frac{(2i+1)(m+np)}{2N}\pi) + \cos(\frac{(2i+1)(m-np)}{2N}\pi)]. \tag{11}$$

Before formulating the output data matrix as a block structured matrix-vector multiplication, we introduce two sets of 1-D DCT output sequences one for each of $A_{p,i}$ and $B_{p,i}$, where $p \in W$ and $0 \leq i \leq N-1$. The motivation for these constructs becomes clear as we discuss the details of the block structured matrix-vector multiplication for the output data matrix.

Define $f_{(p-1)/2,r}$ to be the 1-D DCT output sequence of a given input data sequence $A_{p,i}$, where $0 \leq i,\ r \leq N-1$. That is,

$$f_{(p-1)/2,r} = \sum_{i=0}^{N-1} A_{p,i} \cos(\frac{(2i+1)r}{2N}\pi).$$

Similarly, define $f_{N-1-(p-1)/2,r}$ to be the 1-D DCT output sequence of a given input data sequence $B_{p,i}$, where $0 \leq i,\ r \leq N-1$. That is,

$$f_{N-1-(p-1)/2,r} = \sum_{i=0}^{N-1} B_{p,i} \cos(\frac{(2i+1)r}{2N}\pi).$$

For example, for $N = 8$,

$$
\begin{aligned}
\{A_{1,i}\}_{i=0}^{7} &= \{x_{0,0},\ x_{1,1},\ x_{2,2},\ x_{3,3},\ x_{4,4},\ x_{5,5},\ x_{6,6},\ x_{7,7}\},\\
\{A_{3,i}\}_{i=0}^{7} &= \{x_{0,1},\ x_{1,4},\ x_{2,7},\ x_{3,5},\ x_{4,2},\ x_{5,0},\ x_{6,3},\ x_{7,6}\},\\
\{A_{5,i}\}_{i=0}^{7} &= \{x_{0,2},\ x_{1,7},\ x_{2,3},\ x_{3,1},\ x_{4,6},\ x_{5,4},\ x_{6,0},\ x_{7,5}\},\\
\{A_{7,i}\}_{i=0}^{7} &= \{x_{0,3},\ x_{1,5},\ x_{2,1},\ x_{3,7},\ x_{4,0},\ x_{5,6},\ x_{6,2},\ x_{7,4}\},\\
\{B_{7,i}\}_{i=0}^{7} &= \{x_{0,4},\ x_{1,2},\ x_{2,6},\ x_{3,0},\ x_{4,7},\ x_{5,1},\ x_{6,5},\ x_{7,3}\},\\
\{B_{5,i}\}_{i=0}^{7} &= \{x_{0,5},\ x_{1,0},\ x_{2,4},\ x_{3,6},\ x_{4,1},\ x_{5,3},\ x_{6,7},\ x_{7,2}\},\\
\{B_{3,i}\}_{i=0}^{7} &= \{x_{0,6},\ x_{1,3},\ x_{2,0},\ x_{3,2},\ x_{4,5},\ x_{5,7},\ x_{6,4},\ x_{7,1}\},\\
\{B_{1,i}\}_{i=0}^{7} &= \{x_{0,7},\ x_{1,6},\ x_{2,5},\ x_{3,4},\ x_{4,3},\ x_{5,2},\ x_{6,1},\ x_{7,0}\}.
\end{aligned}
$$

$$
\begin{aligned}
\{f_{0,r}\}_{r=0}^{7} &= DCT(\{A_{1,i}\}_{i=0}^{7}), & \{f_{4,r}\}_{r=0}^{7} &= DCT(\{B_{7,i}\}_{i=0}^{7}),\\
\{f_{1,r}\}_{r=0}^{7} &= DCT(\{A_{3,i}\}_{i=0}^{7}), & \{f_{5,r}\}_{r=0}^{7} &= DCT(\{B_{5,i}\}_{i=0}^{7}),\\
\{f_{2,r}\}_{r=0}^{7} &= DCT(\{A_{5,i}\}_{i=0}^{7}), & \{f_{6,r}\}_{r=0}^{7} &= DCT(\{B_{3,i}\}_{i=0}^{7}),\\
\{f_{3,r}\}_{r=0}^{7} &= DCT(\{A_{7,i}\}_{i=0}^{7}), & \{f_{7,r}\}_{r=0}^{7} &= DCT(\{B_{1,i}\}_{i=0}^{7}).
\end{aligned}
$$

Let

$$
\begin{aligned}
m + np &= 2Nk_1 + r_1; && (12)\\
m - np &= 2Nk_2 + r_2, && (13)
\end{aligned}
$$

where $-(N-1) \le r_1,\ r_2 \le N$, and k_1 and k_2 are integers. Then

$$
\sum_{i=0}^{N-1} A_{p,i} \cos(\frac{(2i+1)(m+np)}{2N}\pi) = \begin{cases} (-1)^{k_1} f_{(p-1)/2,|r_1|} & \text{if } r_1 \ne N \\ 0 & \text{if } r_1 = N \end{cases}
$$

$$
\sum_{i=0}^{N-1} A_{p,i} \cos(\frac{(2i+1)(m-np)}{2N}\pi) = \begin{cases} (-1)^{k_2} f_{(p-1)/2,|r_2|} & \text{if } r_2 \ne N \\ 0 & \text{if } r_2 = N \end{cases}
$$

$$
\sum_{i=0}^{N-1} B_{p,i} \cos(\frac{(2i+1)(m+np)}{2N}\pi) = \begin{cases} (-1)^{k_1} f_{N-1-(p-1)/2,|r_1|} & \text{if } r_1 \ne N \\ 0 & \text{if } r_1 = N \end{cases}
$$

$$
\sum_{i=0}^{N-1} B_{p,i} \cos(\frac{(2i+1)(m-np)}{2N}\pi) = \begin{cases} (-1)^{k_2} f_{N-1-(p-1)/2,|r_2|} & \text{if } r_2 \ne N \\ 0 & \text{if } r_2 = N. \end{cases}
$$

Define $f_{i,N} = 0$ for $0 \le i \le N-1$. Then, if n is even, from (10),

$$y_{m,n} = \frac{1}{2}\sum_{p\in W}[(-1)^{k_1} f_{(p-1)/2,|r_1|} + (-1)^{k_2} f_{(p-1)/2,|r_2|} + \\ (-1)^{k_1} f_{N-1-(p-1)/2,|r_1|} + (-1)^{k_2} f_{N-1-(p-1)/2,|r_2|}]; \quad (14)$$

if n is odd, from (11),

$$y_{m,n} = \frac{1}{2}\sum_{p\in W}[(-1)^{k_1} f_{(p-1)/2,|r_1|} + (-1)^{k_2} f_{(p-1)/2,|r_2|} - \\ (-1)^{k_1} f_{N-1-(p-1)/2,|r_1|} - (-1)^{k_2} f_{N-1-(p-1)/2,|r_2|}]. \quad (15)$$

Let $\underline{\mathbf{y}}_n$ represent the n-th column entries of $\mathbf{y}_{N\times N}$. Then, $\underline{\mathbf{y}}_n = (y_{0,n}, y_{1,n}, \cdots, y_{N-1,n})^t$. Let $\underline{\mathbf{f}}_i$ represent the sequence of $f_{i,r}$, for $0 \le i,\ r \le N-1$. Then, $\underline{\mathbf{f}}_i = (f_{i,0}, f_{i,1}, \cdots, f_{i,N-1})^t$, which is the i-th 1-D DCT output vector. Since the length of the vectors $\underline{\mathbf{y}}_n$ and $\underline{\mathbf{f}}_i$ is N, we can represent Equations (14) and (15) as a block structured matrix-vector multiplication:

$$\mathbf{Y}_{N^2\times 1} = (\mathbf{E}_N)_{N^2}\mathbf{F}_{N^2\times 1}, \quad (16)$$

where

$$\mathbf{Y}_{N^2\times 1} = \begin{bmatrix} \underline{\mathbf{y}}_0 \\ \underline{\mathbf{y}}_1 \\ \vdots \\ \underline{\mathbf{y}}_{N-1} \end{bmatrix}, \qquad \mathbf{F}_{N^2\times 1} = \begin{bmatrix} \underline{\mathbf{f}}_0 \\ \underline{\mathbf{f}}_1 \\ \vdots \\ \underline{\mathbf{f}}_{N-1} \end{bmatrix},$$

and $(\mathbf{E}_N)_{N^2} =$

$$\begin{bmatrix} (G_{0,1})_N & (G_{0,3})_N & \cdots & (G_{0,N-1})_N & (G'_{0,N-1})_N & \cdots & (G'_{0,1})_N \\ (G_{1,1})_N & (G_{1,3})_N & \cdots & (G_{1,N-1})_N & -(G'_{1,N-1})_N & \cdots & -(G'_{1,1})_N \\ (G_{2,1})_N & (G_{2,3})_N & \cdots & (G_{2,N-1})_N & (G'_{2,N-1})_N & \cdots & (G'_{2,1})_N \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (G_{N-1,1})_N & (G_{N-1,3})_N & \cdots & (G_{N-1,N-1})_N & -(G'_{N-1,N-1})_N & \cdots & -(G'_{N-1,1})_N \end{bmatrix}. \quad (17)$$

In Equations (16) and (17), $(G_{n,p})_N$ and $(G'_{n,p})_N$ represent $N \times N$ matrices, $(\mathbf{E}_N)_{N^2}$ represents an $N^2 \times N^2$ matrix, and $\mathbf{Y}_{N^2\times 1}$ and $\mathbf{F}_{N^2\times 1}$ represent vectors of length N^2.

Note that, in Equations (12) and (13), r_1 and r_2 will not be equal to N at the same time. If $m = 0$ or $n = 0$, then $|k_1| = |k_2|$ and $|r_1| = |r_2|$; if $m \neq 0$ and $n \neq 0$, then $|r_1| \neq |r_2|$. Therefore, from Equations (14) and (15), we can observe that each of the rows of $(G_{n,p})_N$ or $(G'_{n,p})_N$ has one or two nonzero elements, and the location of the nonzero elements is dependent on np, where $0 \leq n \leq N-1$ and $p \in W$. Specifically, in both $(G_{n,p})_N$ and $(G'_{n,p})_N$, for $0 \leq m,\ r \leq N-1$,

the element in the position (m, r)

$$= \begin{cases} \frac{1}{2} \text{ if either } \cos\frac{m+np}{2N}\pi = \cos\frac{r}{2N}\pi \text{ or } \cos\frac{m-np}{2N}\pi = \cos\frac{r}{2N}\pi; \\ -\frac{1}{2} \text{ if either } \cos\frac{m+np}{2N}\pi = -\cos\frac{r}{2N}\pi \text{ or } \cos\frac{m-np}{2N}\pi = -\cos\frac{r}{2N}\pi; \\ 1 \text{ if } \cos\frac{m+np}{2N}\pi = \cos\frac{m-np}{2N}\pi = \cos\frac{r}{2N}\pi; \\ -1 \text{ if } \cos\frac{m+np}{2N}\pi = \cos\frac{m-np}{2N}\pi = -\cos\frac{r}{2N}\pi; \\ 0 \text{ otherwise.} \end{cases}$$

Therefore,

$$(G_{n,p})_N = (G'_{n,p})_N.$$

In the following we will show that the block structured matrix $(\mathbf{E}_N)_{N^2}$ has the same properties as the N-point 1-D DCT kernel matrix. First, the block structured matrix $(\mathbf{E}_N)_{N^2}$ in Equation (17) has the same structure as the N-point 1-D DCT kernel matrix shown below.

$(\text{1-D DCT})_N = \mathbf{C}_N$

$$= \begin{bmatrix} c(0,1) & c(0,3) & \cdots & c(0,N-1) & c(0,N+1) & \cdots & c(0,2N-1) \\ c(1,1) & c(1,3) & \cdots & c(1,N-1) & c(1,N+1) & \cdots & c(1,2N-1) \\ c(2,1) & c(2,3) & \cdots & c(2,N-1) & c(2,N+1) & \cdots & c(2,2N-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c(N-1,1) & c(N-1,3) & \cdots & c(N-1,N-1) & c(N-1,N+1) & \cdots & c(N-1,2N-1) \end{bmatrix}$$

$$= \begin{bmatrix} c(0,1) & c(0,3) & \cdots & c(0,N-1) & c(0,N-1) & \cdots & c(0,1) \\ c(1,1) & c(1,3) & \cdots & c(1,N-1) & -c(1,N-1) & \cdots & -c(1,1) \\ c(2,1) & c(2,3) & \cdots & c(2,N-1) & c(2,N-1) & \cdots & c(2,1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c(N-1,1) & c(N-1,3) & \cdots & c(N-1,N-1) & -c(N-1,N-1) & \cdots & -c(N-1,1) \end{bmatrix},$$

where $c(n,p)$ represents $\cos\frac{n\times p}{2N}\pi$.

Second, we will show that each entry of $(\mathbf{E}_N)_{N^2}$ has the same properties as the corresponding entry in the N-point 1-D DCT kernel matrix. Since the elements in $(G_{n,p})_N$ only depend on np, for $0 \leq n,\ n' \leq N-1$ and $p,\ p' \in W$, if $np = n'p'$ then $(G_{n,p})_N = (G_{n',p'})_N$. Define

$$(H_{np})_N = (G_{n,p})_N,$$

for $0 \leq n \leq N-1$ and $p \in W$. Then, for $i \geq 0$,

the element (m, r) of $(H_i)_N$

$$= \begin{cases} \frac{1}{2} \text{ if either } \cos\frac{m+i}{2N}\pi = \cos\frac{r}{2N}\pi \text{ or } \cos\frac{m-i}{2N}\pi = \cos\frac{r}{2N}\pi \\ -\frac{1}{2} \text{ if either } \cos\frac{m+i}{2N}\pi = -\cos\frac{r}{2N}\pi \text{ or } \cos\frac{m-i}{2N}\pi = -\cos\frac{r}{2N}\pi \\ 1 \text{ if } \cos\frac{m+i}{2N}\pi = \cos\frac{m-i}{2N}\pi = \cos\frac{r}{2N}\pi \\ -1 \text{ if } \cos\frac{m+i}{2N}\pi = \cos\frac{m-i}{2N}\pi = -\cos\frac{r}{2N}\pi \\ 0 \text{ otherwise.} \end{cases} \tag{18}$$

In addition as shown in Figure 1, $(H_i)_N$ has four properties, which are the same as those of $\cos\frac{i}{2N}\pi$. Property 1 holds because

$$\cos\frac{m+(i-2Nk)}{2N}\pi = (-1)^k \cos\frac{m+i}{2N}\pi$$
$$\cos\frac{m-(i-2Nk)}{2N}\pi = (-1)^k \cos\frac{m-i}{2N}\pi.$$

Property 2 holds because

$$\cos\frac{m+(2Nk-i)}{2N}\pi = (-1)^k \cos\frac{m-i}{2N}\pi$$
$$\cos\frac{m-(2Nk-i)}{2N}\pi = (-1)^k \cos\frac{m+i}{2N}\pi.$$

Property 3 was proved in [4]. Property 4 follows from Property 3.

The purpose of showing that the block structured matrix $(\mathbf{E}_N)_{N^2}$ has the same properties as the N-point 1-D DCT kernel matrix $\mathbf{C}_N$, is that we can compute the remaining post-addition stage of the 2-D DCT using a procedure similar to that of the 1-D DCT.

3 The 2-D DCT Algorithm

In [12] a recursive algorithm for computing the 1-D DCT $\mathbf{Z}_N = \mathbf{C}_N \mathbf{z}_N$ was derived, where $\mathbf{Z}_N = (Z_0, Z_1, \ldots, Z_{N-1})^t$ is the 1-D DCT output vector, $\mathbf{z}_N =$

Prop.	$(H_i)_N$	$\cos \frac{i}{2N}\pi$
1	$(H_{i-2Nk})_N = (-1)^k (H_i)_N$ for $i,\ i-2Nk \geq 0$	$\cos \frac{i-2Nk}{2N}\pi = (-1)^k \cos \frac{i}{2N}\pi$ for $i,\ i-2Nk \geq 0$
2	$(H_{2Nk-i})_N = (-1)^k (H_i)_N$ for $i,\ 2Nk-i \geq 0$	$\cos \frac{2Nk-i}{2N}\pi = (-1)^k \cos \frac{i}{2N}\pi$ for $i,\ 2Nk-i \geq 0$
3	$2(H_i)_N(H_j)_N = \begin{cases} (H_{\lvert i-j\rvert})_N, & \text{if } i+j=N \\ (H_{\lvert i-j\rvert})_N + (H_{i+j})_N, & \text{if } i+j \neq N \end{cases}$ for $0 \leq i,\ j \leq N-1$	$2\cos \frac{i}{2N}\pi \cos \frac{j}{2N}\pi = \begin{cases} \cos \frac{\lvert i-j\rvert}{2N}\pi, & \text{if } i+j=N \\ \cos \frac{\lvert i-j\rvert}{2N}\pi + \cos \frac{i+j}{2N}\pi, & \text{if } i+j \neq N \end{cases}$ for $0 \leq i,\ j \leq N-1$
4	$(H_i)_N(H_j)_N = (H_j)_N(H_i)_N$ for $0 \leq i,\ j \leq N-1$	$\cos \frac{i}{2N}\pi \cos \frac{j}{2N}\pi = \cos \frac{j}{2N}\pi \cos \frac{i}{2N}\pi$ for $0 \leq i,\ j \leq N-1$

Fig. 1. $(H_i)_N$ has four properties, which are the same as those of $\cos \frac{i}{2N}\pi$.

$(z_0, z_1, \ldots, z_{N-1})^t$ is the input vector, and $\mathbf{C}_N$ is the N-point 1-D DCT kernel matrix. The derivation was based on the four cosine properties listed in Figure 1. Because our 2-D DCT algorithm is based on this recursive algorithm, we describe the procedure in [12] due to its relevance to deriving our 2-D DCT algorithm.

3.1 A Recursive Algorithm for Computing the 1-D DCT

Definition:

(i) Let P_N be the column permutation which arranges columns of a $N \times N$ matrix in the order of

$$0,\ 2,\ 4,\ 6,\ \ldots,\ N-2,\ N-1,\ N-3,\ \ldots,\ 5,\ 3,\ 1.$$

(ii) Let S_N^t be the row permutation which arranges rows of a $N \times N$ matrix in the order of

$$0,\ 2,\ 4,\ 6,\ \ldots,\ N-2,\ 1,\ 3,\ 5,\ 7,\ \ldots,\ N-1.$$

(iii) Let $L_{N/2}$ be a lower triangular matrix defined by

$$L_{N/2} = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ -1 & 2 & 0 & 0 & \ldots & 0 \\ 1 & -2 & 2 & 0 & \ldots & 0 \\ -1 & 2 & -2 & 2 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & 2 & -2 & 2 & \ldots & 2 \end{bmatrix}.$$

(iv) Let $Q_{N/2}$ be a diagonal matrix defined by

$$Q_{N/2} = \text{diagonal}[\cos(\theta_n)],$$

where $\theta_n = (4n+1)\,\pi/(2N)$, for $0 \le n \le N/2 - 1$.

Define $\hat{\mathbf{C}}_N = S_N^t\, \mathbf{C}_N\, P_N$. Then $\hat{\mathbf{C}}_N$ has a recursive structure,

$$\hat{\mathbf{C}}_N = \begin{bmatrix} S_{N/2}\, \hat{\mathbf{C}}_{N/2} & S_{N/2}\, \hat{\mathbf{C}}_{N/2} \\ L_{N/2}\, S_{N/2}\, \hat{\mathbf{C}}_{N/2}\, Q_{N/2} & -L_{N/2}\, S_{N/2}\, \hat{\mathbf{C}}_{N/2}\, Q_{N/2} \end{bmatrix}.$$

A complete proof of the main theorem in [12], which shows that $\hat{\mathbf{C}}_N$ has a recursive structure, is based on the four cosine properties listed in Figure 1, and can be found in [11].

Let $[Z_e,\ Z_o]^t$ be $(Z_0, Z_2, \ldots, Z_{N-2}, Z_1, Z_3, \ldots, Z_{N-1})^t$ and $[z_e,\ z_{\bar{o}}]^t$ be $(z_0, z_2, \ldots, z_{N-2}, z_{N-1}, \ldots, z_3, z_1)^t$. We will use the subscript $''e''$ to represent the even entries of a vector, $''o''$ the odd entries, and $''\bar{o}''$ the odd entries, but in reversed order.

1-D DCT recursive algorithm [12] :
Step 1: Evaluate $[z_e,\ z_{\bar{o}}]^t = P_N^t\, \mathbf{z}$;
Step 2: recursively compute $[Z_e,\ Z_o]^t = \hat{\mathbf{C}}_N [z_e,\ z_{\bar{o}}]^t$:
  2.1 if N is 2 **then** compute $Z_e = (z_e + z_{\bar{o}})$ and $Z_o = \cos \frac{N/2}{2N}\pi\,(z_e - z_{\bar{o}})$;
  2.2 else recursively perform $Z_e = S_{\frac{N}{2}}\, \hat{\mathbf{C}}_{\frac{N}{2}}\,(z_e + z_{\bar{o}})$ and
$Z_o = L_{\frac{N}{2}}\, S_{\frac{N}{2}}\, \hat{\mathbf{C}}_{\frac{N}{2}}\, Q_{\frac{N}{2}}\,(z_e - z_{\bar{o}})$;
Step 3: calculate $\mathbf{Z} = S_N [Z_e,\ Z_o]^t$. $\square$

3.2 The New 2-D DCT Algorithm

The steps of our 2-D DCT algorithm is similar to the 1-D DCT algorithm described in the last subsection. We start from Equation (16). Suppose that N' also has a value of 2^k, where k is an integer, $k \geq 1$, and $2 \leq N' \leq N$. First, let $[Y_e,\ Y_o]^t$ be $(\underline{\mathbf{y}}_0^t, \underline{\mathbf{y}}_2^t, \ldots, \underline{\mathbf{y}}_{N-2}^t, \underline{\mathbf{y}}_1^t, \underline{\mathbf{y}}_3^t, \ldots, \underline{\mathbf{y}}_{N-1}^t)^t$ and $[F_e,\ F_{\bar{o}}]^t$ be $(\underline{\mathbf{f}}_0^t, \underline{\mathbf{f}}_2^t, \ldots, \underline{\mathbf{f}}_{N-2}^t, \underline{\mathbf{f}}_{N-1}^t, \ldots, \underline{\mathbf{f}}_3^t, \underline{\mathbf{f}}_1^t)^t$. Second, let $(P_N)_{NN'}$, $(S_N)_{NN'}$ and $(L_N)_{NN'/2}$ be the block matrix versions of $P_{N'}$, $S_{N'}$ and $L_{N'/2}$, respectively. That is, $(P_N)_{NN'} = P_{N'} \otimes I_N$, $(S_N)_{NN'} = S_{N'} \otimes I_N$ and $(L_N)_{NN'/2} = L_{N'/2} \otimes I_N$, where $\otimes$ denotes the Kronecker product and I_N is the $N \times N$ identity matrix.

Third, let $(Q_N)_{NN'/2}$ be a block matrix with block size N, where $2 \leq N' \leq N$, and $(Q_N)_{NN'/2} = diagonal[(G_{N/N',4n+1})_N] = diagonal[(H_{(4n+1)N/N'})_N]$, for $0 \leq n \leq N'/2 - 1$. Fourth, define

$$(\hat{\mathbf{E}}_N)_{N^2} = (S_N^t)_{N^2}\,(\mathbf{E}_N)_{N^2}\,(P_N)_{N^2}.$$

Then, $(\hat{\mathbf{E}}_N)_{N^2}$ also has a recursive structure, where $\Psi = (S_N)_{N^2/2}\,(\hat{\mathbf{E}}_N)_{N^2/2}$ and $\Omega = (L_N)_{N^2/2}\,(S_N)_{N^2/2}\,(\hat{\mathbf{E}}_N)_{N^2/2}\,(Q_N)_{N^2/2}$:

$$(\hat{\mathbf{E}}_N)_{N^2} = \begin{bmatrix} \Psi & \Psi \\ \Omega & -\Omega \end{bmatrix}.$$

In the following, we describe the procedure for computing the 2-D DCT. It is a recursive algorithm, and its derivation is based on the four properties of $(H_i)_N$ listed in Figure 1. Note that the derivation of $(\hat{\mathbf{E}}_N)_{N^2}$ having a recursive structure is not provided due to its similarity to $\hat{C}_N$; however, it has been verified.

2-D DCT algorithm:

Step 1: Compute N sets of 1-D DCT's for each of the input data sequences $\{A_{p,i}\}_{i=0}^{N-1}$ and $\{B_{p,i}\}_{i=0}^{N-1}$, where $p \in W$; and let the output vectors be $\underline{\mathbf{f}}_{(p-1)/2}$ and $\underline{\mathbf{f}}_{N-1-(p-1)/2}$, respectively;

Step 2: evaluate $[F_e,\ F_{\bar{o}}]^t = (P_N^t)_{N^2}\,\mathbf{F}$;

let $N' = N$;

Step 3: with respect to N', recursively compute $[Y_e, Y_o]^t = (\hat{\mathbf{E}}_N)_{NN'}[F_e, F_{\bar{o}}]^t$:

3.1 if N' is 2 **then** compute $Y_e = (F_e + F_{\bar{o}})$ and $Y_o = (H_{N/2})_N(F_e - F_{\bar{o}})$;

3.2 else recursively perform

$$Y_e = (S_N)_{NN'/2}\,(\hat{\mathbf{E}}_N)_{NN'/2}\,(F_e + F_{\bar{o}});$$

$$Y_o = (L_N)_{NN'/2}\,(S_N)_{NN'/2}\,(\hat{\mathbf{E}}_N)_{NN'/2}\,(Q_N)_{NN'/2}\,(F_e - F_{\bar{o}});$$

Step 4: calculate $\mathbf{Y} = (S_N)_{N^2}[Y_e,\ Y_o]^t$. $\square$

3.3 Examples

We now illustrate our 2-D DCT algorithm with three lower order examples. Let $\alpha = 1/2$.

For $N = 2$,

Step 1:

$$\{A_{1,i}\}_{i=0}^{1} = \{x_{0,0},\ x_{1,1}\}, \qquad \underline{\mathbf{f}}_0 = DCT(\{A_{1,i}\}_{i=0}^{1});$$

$$\{B_{1,i}\}_{i=0}^{1} = \{x_{0,1},\ x_{1,0}\}, \qquad \underline{\mathbf{f}}_1 = DCT(\{B_{1,i}\}_{i=0}^{1}).$$

Step 2:

$$P_2^t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad (P_2^t)_4 = P_2^t \otimes I_2 = \begin{bmatrix} I_2 & 0_2 \\ 0_2 & I_2 \end{bmatrix}, \quad \begin{bmatrix} F_e \\ F_{\bar{o}} \end{bmatrix} = (P_2^t)_4 \mathbf{F} = \begin{bmatrix} \underline{\mathbf{f}}_0 \\ \underline{\mathbf{f}}_1 \end{bmatrix}.$$

$$(\mathbf{E}_2)_4 = \begin{bmatrix} (H_0)_2 & (H_0)_2 \\ (H_1)_2 & -(H_1)_2 \end{bmatrix}, \quad (H_0)_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad (H_1)_2 = \begin{bmatrix} 0 & 1 \\ \alpha & 0 \end{bmatrix}.$$

$$S_2^t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad (S_2^t)_4 = S_2^t \otimes I_2 = \begin{bmatrix} I_2 & 0_2 \\ 0_2 & I_2 \end{bmatrix}.$$

$$(\hat{\mathbf{E}}_2)_4 = (S_2^t)_4\,(\mathbf{E}_2)_4\,(P_2)_4 = (\mathbf{E}_2)_4.$$

Step 3: $N' = 2$, therefore,

$$Y_e = \underline{\mathbf{y}}_0 = (F_e + F_{\bar{o}}) = (\underline{\mathbf{f}}_0 + \underline{\mathbf{f}}_1);$$
$$Y_o = \underline{\mathbf{y}}_1 = (H_1)_2(F_e - F_{\bar{o}}) = (H_1)_2(\underline{\mathbf{f}}_0 - \underline{\mathbf{f}}_1).$$

Step 4:

$$\mathbf{Y} = \begin{bmatrix} \underline{\mathbf{y}}_0 \\ \underline{\mathbf{y}}_1 \end{bmatrix} = (S_2)_4 \begin{bmatrix} Y_e \\ Y_o \end{bmatrix} = \begin{bmatrix} (\underline{\mathbf{f}}_0 + \underline{\mathbf{f}}_1) \\ (H_1)_2(\underline{\mathbf{f}}_0 - \underline{\mathbf{f}}_1) \end{bmatrix}.$$

Figures 2-(a) and 2-(b) show the signal-flow graph and the abstract signal-flow graph for $N = 2$.

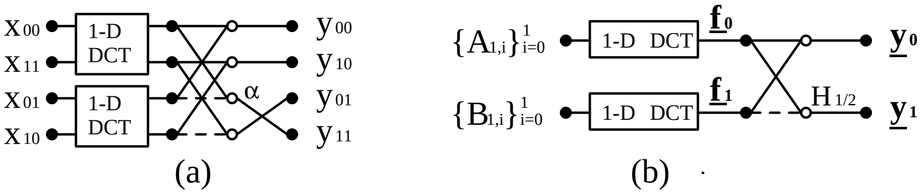


Fig. 2. (a) A signal-flow graph and (b) an abstract signal-flow graph for the 2-D DCT for $N = 2$. Solid lines represent transfer factor 1, while dashed lines represent transfer factor -1. Circles $\circ$ represent adders. $H_{i/N}$ means $(H_i)_N$. $\alpha = 1/2$. A line in the abstract signal-flow graph represents N (parallel) lines in the signal-flow graph.

For $N = 4$,

Step 1:

$$\{A_{1,i}\}_{i=0}^{3} = \{x_{0,0},\ x_{1,1},\ x_{2,2},\ x_{3,3}\}, \qquad \underline{\mathbf{f}}_0 = DCT(\{A_{1,i}\}_{i=0}^{3});$$

$$\{A_{3,i}\}_{i=0}^{3} = \{x_{0,1},\ x_{1,3},\ x_{2,0},\ x_{3,2}\}, \qquad \underline{\mathbf{f}}_1 = DCT(\{A_{3,i}\}_{i=0}^{3});$$

$$\{B_{3,i}\}_{i=0}^{3} = \{x_{0,2},\ x_{1,0},\ x_{2,3},\ x_{3,1}\}, \qquad \underline{\mathbf{f}}_2 = DCT(\{B_{3,i}\}_{i=0}^{3});$$

$$\{B_{1,i}\}_{i=0}^{3} = \{x_{0,3},\ x_{1,2},\ x_{2,1},\ x_{3,0}\}, \qquad \underline{\mathbf{f}}_3 = DCT(\{B_{1,i}\}_{i=0}^{3}).$$

Step 2:

$$P_4^t = \begin{bmatrix} 1&0&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 0&1&0&0 \end{bmatrix}, \quad (P_4^t)_{16} = P_4^t \otimes I_4 = \begin{bmatrix} I_4 & 0_4 & 0_4 & 0_4 \\ 0_4 & 0_4 & I_4 & 0_4 \\ 0_4 & 0_4 & 0_4 & I_4 \\ 0_4 & I_4 & 0_4 & 0_4 \end{bmatrix}.$$

$$\begin{bmatrix} F_e \\ F_{\bar{o}} \end{bmatrix} = (P_4^t)_{16}\mathbf{F} = \begin{bmatrix} \underline{\mathbf{f}}_0 \\ \underline{\mathbf{f}}_2 \\ \underline{\mathbf{f}}_3 \\ \underline{\mathbf{f}}_1 \end{bmatrix}, \quad (\mathbf{E}_4)_{16} = \begin{bmatrix} (H_0)_4 & (H_0)_4 & (H_0)_4 & (H_0)_4 \\ (H_1)_4 & (H_3)_4 & -(H_3)_4 & -(H_1)_4 \\ (H_2)_4 & -(H_2)_4 & -(H_2)_4 & (H_2)_4 \\ (H_3)_4 & -(H_1)_4 & (H_1)_4 & -(H_3)_4 \end{bmatrix}.$$

$$(H_0)_4 = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}, \quad (H_1)_4 = \begin{bmatrix} 0&1&0&0 \\ \alpha&0&\alpha&0 \\ 0&\alpha&0&\alpha \\ 0&0&\alpha&0 \end{bmatrix}, \quad (H_2)_4 = \begin{bmatrix} 0&0&1&0 \\ 0&\alpha&0&\alpha \\ \alpha&0&0&0 \\ 0&\alpha&0&-\alpha \end{bmatrix},$$

$$
(H_3)_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & \alpha & 0 \\ 0 & \alpha & 0 & -\alpha \\ \alpha & 0 & -\alpha & 0 \end{bmatrix} . \; S_4^t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} , \; (S_4^t)_{16} = S_4^t \otimes I_4 = \begin{bmatrix} I_4 & 0_4 & 0_4 & 0_4 \\ 0_4 & 0_4 & I_4 & 0_4 \\ 0_4 & I_4 & 0_4 & 0_4 \\ 0_4 & 0_4 & 0_4 & I_4 \end{bmatrix}
$$

$$
(\hat{\mathbf{E}}_4)_{16} = (S_4^t)_{16} \, (\mathbf{E}_4)_{16} \, (P_4)_{16} = \begin{bmatrix} (H_0)_4 & (H_0)_4 & (H_0)_4 & (H_0)_4 \\ (H_2)_4 & -(H_2)_4 & (H_2)_4 & -(H_2)_4 \\ (H_1)_4 & -(H_3)_4 & -(H_1)_4 & (H_3)_4 \\ (H_3)_4 & (H_1)_4 & -(H_3)_4 & -(H_1)_4 \end{bmatrix} .
$$

Step 3: $N' = 4$, we compute Y_e and Y_o recursively.

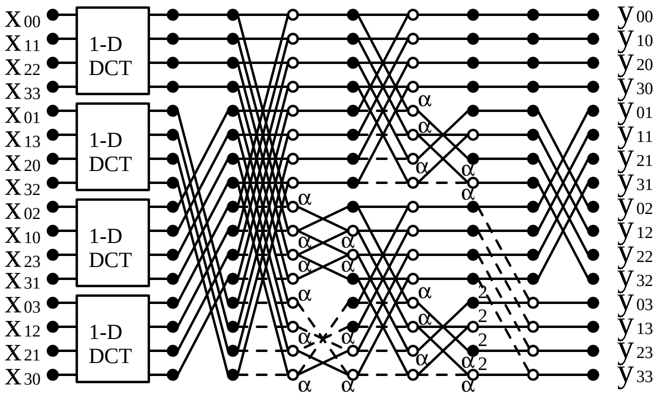
$$
\begin{aligned}
Y_e &= \begin{bmatrix} \underline{\mathbf{y}}_0 \\ \underline{\mathbf{y}}_2 \end{bmatrix} = (S_4)_8 \, (\hat{\mathbf{E}}_4)_8 \, (F_e + F_{\bar{o}}) \\
&= \begin{bmatrix} I_4 & 0_4 \\ 0_4 & I_4 \end{bmatrix} \begin{bmatrix} (H_0)_4 & (H_0)_4 \\ (H_2)_4 & -(H_2)_4 \end{bmatrix} \left(\begin{bmatrix} \underline{\mathbf{f}}_0 \\ \underline{\mathbf{f}}_2 \end{bmatrix} + \begin{bmatrix} \underline{\mathbf{f}}_3 \\ \underline{\mathbf{f}}_1 \end{bmatrix} \right);
\end{aligned}
$$

$$
\begin{aligned}
Y_o &= \begin{bmatrix} \underline{\mathbf{y}}_1 \\ \underline{\mathbf{y}}_3 \end{bmatrix} = (L_4)_8 \, (S_4)_8 \, (\hat{\mathbf{E}}_4)_8 \, (Q_4)_8 \, (F_e - F_{\bar{o}}) \\
&= \begin{bmatrix} I_4 & 0_4 \\ -I_4 & 2I_4 \end{bmatrix} \begin{bmatrix} I_4 & 0_4 \\ 0_4 & I_4 \end{bmatrix} \begin{bmatrix} (H_0)_4 & (H_0)_4 \\ (H_2)_4 & -(H_2)_4 \end{bmatrix} \begin{bmatrix} (H_1)_4 & 0_4 \\ 0_4 & (H_5)_4 \end{bmatrix} \left(\begin{bmatrix} \underline{\mathbf{f}}_0 \\ \underline{\mathbf{f}}_2 \end{bmatrix} - \begin{bmatrix} \underline{\mathbf{f}}_3 \\ \underline{\mathbf{f}}_1 \end{bmatrix} \right) \\
&= \begin{bmatrix} (H_1)_4 & -(H_3)_4 \\ (H_3)_4 & (H_1)_4 \end{bmatrix} \left(\begin{bmatrix} \underline{\mathbf{f}}_0 \\ \underline{\mathbf{f}}_2 \end{bmatrix} - \begin{bmatrix} \underline{\mathbf{f}}_3 \\ \underline{\mathbf{f}}_1 \end{bmatrix} \right).
\end{aligned}
$$

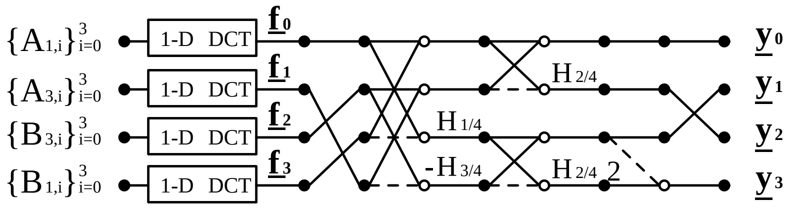
Step 4:

$$
\mathbf{Y} = \begin{bmatrix} \underline{\mathbf{y}}_0 \\ \underline{\mathbf{y}}_1 \\ \underline{\mathbf{y}}_2 \\ \underline{\mathbf{y}}_3 \end{bmatrix} = (S_4)_{16} \begin{bmatrix} Y_e \\ Y_o \end{bmatrix} = \begin{bmatrix} I_4 & 0_4 & 0_4 & 0_4 \\ 0_4 & 0_4 & I_4 & 0_4 \\ 0_4 & I_4 & 0_4 & 0_4 \\ 0_4 & 0_4 & 0_4 & I_4 \end{bmatrix} \begin{bmatrix} \underline{\mathbf{y}}_0 \\ \underline{\mathbf{y}}_2 \\ \underline{\mathbf{y}}_1 \\ \underline{\mathbf{y}}_3 \end{bmatrix} .
$$

Figures 3-(a) and 3-(b) show the signal-flow graph and the abstract signal-flow graph for $N = 4$.



(a)



(b)

Fig. 3. (a) A signal-flow graph and (b) an abstract signal-flow graph for the 2-D DCT for $N = 4$.

For $N = 8$,

Step 1:

$$\{A_{1,i}\}_{i=0}^{7} = \{x_{0,0},\ x_{1,1},\ x_{2,2},\ x_{3,3},\ x_{4,4},\ x_{5,5},\ x_{6,6},\ x_{7,7}\},$$
$$\{A_{3,i}\}_{i=0}^{7} = \{x_{0,1},\ x_{1,4},\ x_{2,7},\ x_{3,5},\ x_{4,2},\ x_{5,0},\ x_{6,3},\ x_{7,6}\},$$
$$\{A_{5,i}\}_{i=0}^{7} = \{x_{0,2},\ x_{1,7},\ x_{2,3},\ x_{3,1},\ x_{4,6},\ x_{5,4},\ x_{6,0},\ x_{7,5}\},$$
$$\{A_{7,i}\}_{i=0}^{7} = \{x_{0,3},\ x_{1,5},\ x_{2,1},\ x_{3,7},\ x_{4,0},\ x_{5,6},\ x_{6,2},\ x_{7,4}\},$$
$$\{B_{7,i}\}_{i=0}^{7} = \{x_{0,4},\ x_{1,2},\ x_{2,6},\ x_{3,0},\ x_{4,7},\ x_{5,1},\ x_{6,5},\ x_{7,3}\},$$
$$\{B_{5,i}\}_{i=0}^{7} = \{x_{0,5},\ x_{1,0},\ x_{2,4},\ x_{3,6},\ x_{4,1},\ x_{5,3},\ x_{6,7},\ x_{7,2}\},$$
$$\{B_{3,i}\}_{i=0}^{7} = \{x_{0,6},\ x_{1,3},\ x_{2,0},\ x_{3,2},\ x_{4,5},\ x_{5,7},\ x_{6,4},\ x_{7,1}\},$$
$$\{B_{1,i}\}_{i=0}^{7} = \{x_{0,7},\ x_{1,6},\ x_{2,5},\ x_{3,4},\ x_{4,3},\ x_{5,2},\ x_{6,1},\ x_{7,0}\}.$$

$$\{f_{0,r}\}_{r=0}^{7} = DCT(\{A_{1,i}\}_{i=0}^{7}), \qquad \{f_{4,r}\}_{r=0}^{7} = DCT(\{B_{7,i}\}_{i=0}^{7}),$$
$$\{f_{1,r}\}_{r=0}^{7} = DCT(\{A_{3,i}\}_{i=0}^{7}), \qquad \{f_{5,r}\}_{r=0}^{7} = DCT(\{B_{5,i}\}_{i=0}^{7}),$$
$$\{f_{2,r}\}_{r=0}^{7} = DCT(\{A_{5,i}\}_{i=0}^{7}), \qquad \{f_{6,r}\}_{r=0}^{7} = DCT(\{B_{3,i}\}_{i=0}^{7}),$$
$$\{f_{3,r}\}_{r=0}^{7} = DCT(\{A_{7,i}\}_{i=0}^{7}), \qquad \{f_{7,r}\}_{r=0}^{7} = DCT(\{B_{1,i}\}_{i=0}^{7}).$$

Step 2:

$$(P_8^t)_{64} = P_8^t \otimes I_8 = \begin{bmatrix} I_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & I_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & 0_8 & I_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & I_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & I_8 \\ 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & I_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & I_8 & 0_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & I_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 \end{bmatrix}, \quad \begin{bmatrix} F_e \\ F_{\bar{o}} \end{bmatrix} = (P_8^t)_{64}\mathbf{F} = \begin{bmatrix} \underline{\mathbf{f}}_0 \\ \underline{\mathbf{f}}_2 \\ \underline{\mathbf{f}}_4 \\ \underline{\mathbf{f}}_6 \\ \underline{\mathbf{f}}_7 \\ \underline{\mathbf{f}}_5 \\ \underline{\mathbf{f}}_3 \\ \underline{\mathbf{f}}_1 \end{bmatrix}.$$

$$(\mathbf{E}_8)_{64} = \begin{bmatrix} (H_0)_8 & (H_0)_8 & (H_0)_8 & (H_0)_8 & (H_0)_8 & (H_0)_8 & (H_0)_8 & (H_0)_8 \\ (H_1)_8 & (H_3)_8 & (H_5)_8 & (H_7)_8 & -(H_7)_8 & -(H_5)_8 & -(H_3)_8 & -(H_1)_8 \\ (H_2)_8 & (H_6)_8 & -(H_6)_8 & -(H_2)_8 & -(H_2)_8 & -(H_6)_8 & (H_6)_8 & (H_2)_8 \\ (H_3)_8 & -(H_7)_8 & -(H_1)_8 & -(H_5)_8 & (H_5)_8 & (H_1)_8 & (H_7)_8 & -(H_3)_8 \\ (H_4)_8 & -(H_4)_8 & -(H_4)_8 & (H_4)_8 & (H_4)_8 & -(H_4)_8 & -(H_4)_8 & (H_4)_8 \\ (H_5)_8 & -(H_1)_8 & (H_7)_8 & (H_3)_8 & -(H_3)_8 & -(H_7)_8 & (H_1)_8 & -(H_5)_8 \\ (H_6)_8 & -(H_2)_8 & (H_2)_8 & -(H_6)_8 & -(H_6)_8 & (H_2)_8 & -(H_2)_8 & (H_6)_8 \\ (H_7)_8 & -(H_5)_8 & (H_3)_8 & -(H_1)_8 & (H_1)_8 & -(H_3)_8 & (H_5)_8 & -(H_7)_8 \end{bmatrix}.$$

$$(H_0)_8 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}, \quad (H_1)_8 = \begin{bmatrix} 0&1&0&0&0&0&0&0 \\ \alpha&0&\alpha&0&0&0&0&0 \\ 0&\alpha&0&\alpha&0&0&0&0 \\ 0&0&\alpha&0&\alpha&0&0&0 \\ 0&0&0&\alpha&0&\alpha&0&0 \\ 0&0&0&0&\alpha&0&\alpha&0 \\ 0&0&0&0&0&\alpha&0&\alpha \\ 0&0&0&0&0&0&\alpha&0 \end{bmatrix},$$

$$(H_2)_8 = \begin{bmatrix} 0&0&1&0&0&0&0&0 \\ 0&\alpha&0&\alpha&0&0&0&0 \\ \alpha&0&0&0&\alpha&0&0&0 \\ 0&\alpha&0&0&0&\alpha&0&0 \\ 0&0&\alpha&0&0&0&\alpha&0 \\ 0&0&0&\alpha&0&0&0&\alpha \\ 0&0&0&0&\alpha&0&0&0 \\ 0&0&0&0&0&\alpha&0&-\alpha \end{bmatrix}, \quad (H_3)_8 = \begin{bmatrix} 0&0&0&1&0&0&0&0 \\ 0&0&\alpha&0&\alpha&0&0&0 \\ 0&\alpha&0&0&0&\alpha&0&0 \\ \alpha&0&0&0&0&0&\alpha&0 \\ 0&\alpha&0&0&0&0&0&\alpha \\ 0&0&\alpha&0&0&0&0&0 \\ 0&0&0&\alpha&0&0&0&-\alpha \\ 0&0&0&0&\alpha&0&-\alpha&0 \end{bmatrix},$$

$$(H_4)_8 = \begin{bmatrix} 0&0&0&0&1&0&0&0 \\ 0&0&0&\alpha&0&\alpha&0&0 \\ 0&0&\alpha&0&0&0&\alpha&0 \\ 0&\alpha&0&0&0&0&0&\alpha \\ \alpha&0&0&0&0&0&0&0 \\ 0&\alpha&0&0&0&0&0&-\alpha \\ 0&0&\alpha&0&0&0&-\alpha&0 \\ 0&0&0&\alpha&0&-\alpha&0&0 \end{bmatrix}, \quad (H_5)_8 = \begin{bmatrix} 0&0&0&0&0&1&0&0 \\ 0&0&0&0&\alpha&0&\alpha&0 \\ 0&0&0&\alpha&0&0&0&\alpha \\ 0&0&\alpha&0&0&0&0&0 \\ 0&\alpha&0&0&0&0&0&-\alpha \\ \alpha&0&0&0&0&0&-\alpha&0 \\ 0&\alpha&0&0&0&-\alpha&0&0 \\ 0&0&\alpha&0&-\alpha&0&0&0 \end{bmatrix},$$

$$
(H_6)_8 = \begin{bmatrix}
0&0&0&0&0&0&1&0\\
0&0&0&0&0&\alpha&0&\alpha\\
0&0&0&0&\alpha&0&0&0\\
0&0&0&\alpha&0&0&0&-\alpha\\
0&0&\alpha&0&0&0&-\alpha&0\\
0&\alpha&0&0&0&-\alpha&0&0\\
\alpha&0&0&0&-\alpha&0&0&0\\
0&\alpha&0&-\alpha&0&0&0&0
\end{bmatrix}, \quad
(H_7)_8 = \begin{bmatrix}
0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&\alpha&0\\
0&0&0&0&0&\alpha&0&-\alpha\\
0&0&0&0&\alpha&0&-\alpha&0\\
0&0&0&\alpha&0&-\alpha&0&0\\
0&0&\alpha&0&-\alpha&0&0&0\\
0&\alpha&0&-\alpha&0&0&0&0\\
\alpha&0&-\alpha&0&0&0&0&0
\end{bmatrix}.
$$

$$
(S_8^t)_{64} = S_8^t \otimes I_8 = \begin{bmatrix}
I_8&0_8&0_8&0_8&0_8&0_8&0_8&0_8\\
0_8&0_8&I_8&0_8&0_8&0_8&0_8&0_8\\
0_8&0_8&0_8&0_8&I_8&0_8&0_8&0_8\\
0_8&0_8&0_8&0_8&0_8&0_8&I_8&0_8\\
0_8&I_8&0_8&0_8&0_8&0_8&0_8&0_8\\
0_8&0_8&0_8&I_8&0_8&0_8&0_8&0_8\\
0_8&0_8&0_8&0_8&0_8&I_8&0_8&0_8\\
0_8&0_8&0_8&0_8&0_8&0_8&0_8&I_8
\end{bmatrix}.
$$

$$
(\hat{\mathbf{E}}_8)_{64} = (S_8^t)_{64}\,(\mathbf{E}_8)_{64}\,(P_8)_{64} =
$$

$$
\begin{bmatrix}
(H_0)_8&(H_0)_8&(H_0)_8&(H_0)_8&(H_0)_8&(H_0)_8&(H_0)_8&(H_0)_8\\
(H_2)_8&-(H_6)_8&-(H_2)_8&(H_6)_8&(H_2)_8&-(H_6)_8&-(H_2)_8&(H_6)_8\\
(H_4)_8&-(H_4)_8&(H_4)_8&-(H_4)_8&(H_4)_8&-(H_4)_8&(H_4)_8&-(H_4)_8\\
(H_6)_8&(H_2)_8&-(H_6)_8&-(H_2)_8&(H_6)_8&(H_2)_8&-(H_6)_8&-(H_2)_8\\
(H_1)_8&(H_5)_8&-(H_7)_8&-(H_3)_8&-(H_1)_8&-(H_5)_8&(H_7)_8&(H_3)_8\\
(H_3)_8&-(H_1)_8&(H_5)_8&(H_7)_8&-(H_3)_8&(H_1)_8&-(H_5)_8&-(H_7)_8\\
(H_5)_8&(H_7)_8&-(H_3)_8&(H_1)_8&-(H_5)_8&-(H_7)_8&(H_3)_8&-(H_1)_8\\
(H_7)_8&(H_3)_8&(H_1)_8&(H_5)_8&-(H_7)_8&-(H_3)_8&-(H_1)_8&-(H_5)_8
\end{bmatrix}.
$$

Step 3: $N' = 8$, we compute Y_e and Y_o recursively.

$$
Y_e = \begin{bmatrix} \underline{\mathbf{y}}_0\\ \underline{\mathbf{y}}_2\\ \underline{\mathbf{y}}_4\\ \underline{\mathbf{y}}_6 \end{bmatrix} = (S_8)_{32}\,(\hat{\mathbf{E}}_8)_{32}\,(F_e + F_{\bar{o}})
$$

$$
= \begin{bmatrix}
I_8&0_8&0_8&0_8\\
0_8&0_8&I_8&0_8\\
0_8&I_8&0_8&0_8\\
0_8&0_8&0_8&I_8
\end{bmatrix}
\begin{bmatrix}
(H_0)_8&(H_0)_8&(H_0)_8&(H_0)_8\\
(H_4)_8&-(H_4)_8&(H_4)_8&-(H_4)_8\\
(H_2)_8&-(H_6)_8&-(H_2)_8&(H_6)_8\\
(H_6)_8&(H_2)_8&-(H_6)_8&-(H_2)_8
\end{bmatrix}
\left(
\begin{bmatrix} \underline{\mathbf{f}}_0\\ \underline{\mathbf{f}}_2\\ \underline{\mathbf{f}}_4\\ \underline{\mathbf{f}}_6 \end{bmatrix}
+
\begin{bmatrix} \underline{\mathbf{f}}_7\\ \underline{\mathbf{f}}_5\\ \underline{\mathbf{f}}_3\\ \underline{\mathbf{f}}_1 \end{bmatrix}
\right)
$$

$$
= \begin{bmatrix}
(H_0)_8&(H_0)_8&(H_0)_8&(H_0)_8\\
(H_2)_8&-(H_6)_8&-(H_2)_8&(H_6)_8\\
(H_4)_8&-(H_4)_8&(H_4)_8&-(H_4)_8\\
(H_6)_8&(H_2)_8&-(H_6)_8&-(H_2)_8
\end{bmatrix}
\left(
\begin{bmatrix} \underline{\mathbf{f}}_0\\ \underline{\mathbf{f}}_2\\ \underline{\mathbf{f}}_4\\ \underline{\mathbf{f}}_6 \end{bmatrix}
+
\begin{bmatrix} \underline{\mathbf{f}}_7\\ \underline{\mathbf{f}}_5\\ \underline{\mathbf{f}}_3\\ \underline{\mathbf{f}}_1 \end{bmatrix}
\right);
$$

$$
Y_o = \begin{bmatrix} \underline{\mathbf{y}}_1 \\ \underline{\mathbf{y}}_3 \\ \underline{\mathbf{y}}_5 \\ \underline{\mathbf{y}}_7 \end{bmatrix} = (L_8)_{32}\,(S_8)_{32}\,(\hat{\mathbf{E}}_8)_{32}\,(Q_8)_{32}\,(F_e - F_{\bar{o}})
$$

$$
= \begin{bmatrix} I_8 & 0_8 & 0_8 & 0_8 \\ -I_8 & 2I_8 & 0_8 & 0_8 \\ I_8 & -2I_8 & 2I_8 & 0_8 \\ -I_8 & 2I_8 & -2I_8 & 2I_8 \end{bmatrix} \begin{bmatrix} I_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & I_8 & 0_8 \\ 0_8 & I_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & I_8 \end{bmatrix} \begin{bmatrix} (H_0)_8 & (H_0)_8 & (H_0)_8 & (H_0)_8 \\ (H_4)_8 & -(H_4)_8 & (H_4)_8 & -(H_4)_8 \\ (H_2)_8 & -(H_6)_8 & -(H_2)_8 & (H_6)_8 \\ (H_6)_8 & (H_2)_8 & -(H_6)_8 & -(H_2)_8 \end{bmatrix}
$$

$$
\times \begin{bmatrix} (H_1)_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & (H_5)_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & (H_9)_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & (H_{13})_8 \end{bmatrix} \left(\begin{bmatrix} \underline{\mathbf{f}}_0 \\ \underline{\mathbf{f}}_2 \\ \underline{\mathbf{f}}_4 \\ \underline{\mathbf{f}}_6 \end{bmatrix} - \begin{bmatrix} \underline{\mathbf{f}}_7 \\ \underline{\mathbf{f}}_5 \\ \underline{\mathbf{f}}_3 \\ \underline{\mathbf{f}}_1 \end{bmatrix} \right)
$$

$$
= \begin{bmatrix} (H_1)_8 & (H_5)_8 & -(H_7)_8 & -(H_3)_8 \\ (H_3)_8 & -(H_1)_8 & (H_5)_8 & (H_7)_8 \\ (H_5)_8 & (H_7)_8 & -(H_3)_8 & (H_1)_8 \\ (H_7)_8 & (H_3)_8 & (H_1)_8 & (H_5)_8 \end{bmatrix} \left(\begin{bmatrix} \underline{\mathbf{f}}_0 \\ \underline{\mathbf{f}}_2 \\ \underline{\mathbf{f}}_4 \\ \underline{\mathbf{f}}_6 \end{bmatrix} - \begin{bmatrix} \underline{\mathbf{f}}_7 \\ \underline{\mathbf{f}}_5 \\ \underline{\mathbf{f}}_3 \\ \underline{\mathbf{f}}_1 \end{bmatrix} \right).
$$

Step 4:

$$
\mathbf{Y} = \begin{bmatrix} \underline{\mathbf{y}}_0 \\ \underline{\mathbf{y}}_1 \\ \underline{\mathbf{y}}_2 \\ \underline{\mathbf{y}}_3 \\ \underline{\mathbf{y}}_4 \\ \underline{\mathbf{y}}_5 \\ \underline{\mathbf{y}}_6 \\ \underline{\mathbf{y}}_7 \end{bmatrix} = (S_8)_{64} \begin{bmatrix} Y_e \\ Y_o \end{bmatrix} = \begin{bmatrix} I_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & 0_8 & I_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & I_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & I_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & I_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & I_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & I_8 & 0_8 & 0_8 & 0_8 & 0_8 \\ 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & 0_8 & I_8 \end{bmatrix} \begin{bmatrix} \underline{\mathbf{y}}_0 \\ \underline{\mathbf{y}}_2 \\ \underline{\mathbf{y}}_4 \\ \underline{\mathbf{y}}_6 \\ \underline{\mathbf{y}}_1 \\ \underline{\mathbf{y}}_3 \\ \underline{\mathbf{y}}_5 \\ \underline{\mathbf{y}}_7 \end{bmatrix}.
$$

Figure 4 shows the abstract signal-flow graph for $N = 8$.

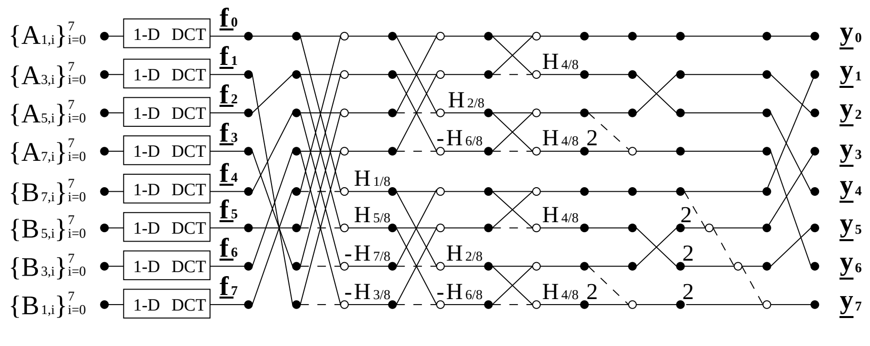


Fig. 4. A signal-flow graph for the 2-D DCT for $N = 8$.

3.4 Comparisons with Related Works

In the following, we analyze the algorithm's complexity. As shown in [8,12], it requires $\frac{1}{2}N\log N$ multiplications and $\frac{3}{2}N\log N - N + 1$ additions for computing an N-point 1-D DCT, where the base of the log is two. In our algorithm for computing an $(N\times N)$-point 2-D DCT, N sets of N-point 1-D DCT's and a post-addition stage must be computed, where the block version of this post-addition stage has the same structure as the 1-D DCT. For evaluating N sets of N-point 1-D DCT's, it requires $N(\frac{1}{2}N\log N)$ multiplications and $N(\frac{3}{2}N\log N - N+1)$ additions. For calculating the post-addition stage, it requires $(N-2)(\frac{1}{2}N\log N) + N(\frac{3}{2}N\log N - N + 1)$ additions. Therefore, it requires $\frac{1}{2}N^2\log N$ multiplications and $\frac{7}{2}N^2\log N - 2N^2 + 2N - N\log N$ additions to compute an $(N\times N)$-point 2-D DCT. Note that it only requires $N-2$ additions for calculating $(H_i)_N\,\mathbf{v}_N$, where $\mathbf{v}_N$ is a vector of length N and $0 < i \le N-1$; we also ignore the computation time of multiplying a factor 2 or $\frac{1}{2}$ by a number, because it only involves a shift operation which is much faster than performing an addition operation.

Table 2 shows the comparisons with the computational complexities among those 2-D DCT algorithms found in related research works. This reveals that our algorithm performs competitively with existing ones. However, the primary appeal of our algorithm lies in not only its low computational cost but also its regular structure as the transform size increases. Although the computational complexity of our algorithm is not better than those in [2,4], the computational structure is more regular than those in [2,4]. Thus, as mentioned in [2,3], this proposed 2-D DCT algorithm may be more suitable for VLSI implementations.

Table 2
Computational complexities for the 2-D DCT.

	# of multiplications	# of additions
this paper	$\frac{1}{2}N^2\log N$	$\frac{7}{2}N^2\log N - 2N^2 + 2N - N\log N$
[4]	$\frac{1}{2}N^2\log N$	$\frac{7}{2}N^2\log N - 2N^2 + 2N - N\log N$
[2]	$\frac{1}{2}N^2\log N$	$\frac{5}{2}N^2\log N - 2N + 2$
[16]	$\frac{1}{2}N^2\log N + \frac{1}{3}N^2 - 2N + \frac{8}{3}$	$\frac{5}{2}N^2\log N + \frac{1}{3}N^2 - 6N + \frac{62}{3}$
[12]	$\frac{3}{4}N^2\log N - \frac{1}{4}N^2$	$3N^2\log N - 2N^2 + 2N$
[1,7]	$\frac{3}{4}N^2\log N$	$3N^2\log N - 2N^2 + 2N$
row-column method	$N^2\log N$	$3N^2\log N - 2N^2 + 2N$

Table 3 lists experimental results of implementing our algorithm to deal with five well-known image examples: *lena, baboon, peppers, ship* and *sunny*, on a Sparc 20/81 workstation. All of the image sizes of these five images are 512×512 pixels and each pixel is represented by an 8-bit grey scale. We implemented an 8×8 2-D DCT and a 16×16 2-D DCT and compared results with the conventional row-column method. Experimental studies show that the execution time of our algorithm is slightly better than the conventional row-column method.

Table 3
The execution time in units of seconds of running the proposed 2-D DCT algorithm (our algo.) and an algorithm based on the row-column method (row-colu.) with five 512×512 images.

Block size	algorithm	five images				
		lena	baboon	peppers	ship	sunny
8×8	our algo.	1.167	1.167	1.150	1.183	1.167
	row-colu.	1.267	1.283	1.250	1.250	1.267
16×16	our algo.	1.533	1.517	1.600	1.600	1.583
	row-colu.	1.650	1.633	1.633	1.650	1.650

We now compare our algorithm with the algorithm derived by Cho, Yun and Lee [4]. First, the order in which we arrange the input data matrix is different from theirs. Second, as mentioned in Section 1, in their algorithm, before performing 1-D DCT's, they need to calculate a pre-addition stage; also, the computation of their post-addition stage is similar to that of Hou's 1-D DCT algorithm [8], except that they divide the post-addition stage into two parts and solve them independently. In contrast, in our algorithm, no pre-addition stages are needed before performing 1-D DCT's; in addition, the computation of our post-addition stage is similar to the computation of the 1-D DCT algorithm (proposed by Lee and Huang [12]).

There is also one similarity between these two algorithms in that after the first reordering and addition stage following the 1-D DCT's, our algorithm arrives at the same vectors f_p and g_p defined in [2,4]. Here we provide the relations between our $\underline{\mathbf{f}}_i$ and their f_p and g_p for an easy reference, where $0 \leq i \leq N-1$ and $p \in W$:

$$
\begin{aligned}
f_p &= \underline{\mathbf{f}}_{(p-1)/2} + \underline{\mathbf{f}}_{N-1-(p-1)/2}; \\
g_p &= \underline{\mathbf{f}}_{(p-1)/2} - \underline{\mathbf{f}}_{N-1-(p-1)/2}.
\end{aligned}
$$

Note that for a specific N, if we only consider the number of operations used in an algorithm and don't care about the computational structure of the post-addition stage, then we may apply algorithms by Feig and Winograd [6] or McGovern, Woods and Yan [14] for computing 1-D DCT's and the post-addition stage. Feig and Winograd applied matrix factorization techniques to the DCT kernel matrix, the structure of tensor products of their matrices can be viewed as elements in certain fields defined by polynomial multiplication modulo irreducible polynomials. Therefore, for different length N, the computational structures are different. McGovern, Woods and Yan adopted Feig and Winograd's 1-D DCT algorithm to implement their 2-D DCT algorithm based on the row-column method. These design approaches where the computational structures for different problem sizes are different, however, are beyond the scope of this presentation.

4 Conclusions

A systematic method for designing a 2-D DCT algorithm has been presented in this paper. The method, which is based on certain intrinsic properties of the 2-D DCT, allows us to reduce the 2-D DCT problem to the 1-D DCT problem. We thus can design a 2-D DCT algorithm in a manner similar to a 1-D DCT algorithm.

A contribution of this paper is to provide a clear procedure for the computation. This allows us to testify that the algorithm is simple and the corresponding signal-flow graphs are regular. It also allows us to give a closed-form expression for the arithmetic complexity of this algorithm. In practice, for some special purpose hardware devices for implementing DSP applications, such as VLSI implementations, the cost of performing a multiplication may be higher than that of performing an addition. If we are only interested in the number of scalar multiplications used in an algorithm, then our algorithm is one of the best algorithms found in published results.

Acknowledgement

This work was partially supported by the NSC under Grants NSC 84-2213-E-001-003, NSC 85-2213-E-001-006 and 86-2213-E-001-006. The authors would like to thank Professor Murat Kunt and the anonymous referees for their valuable comments on this paper.

References

[1] S. C. Chan and K. L. Ho, "A New Two-dimensional Fast Cosine Transform Algorithm", *IEEE Trans. Signal Processing*, Vol. 39, No. 2, Feb. 1991, pp. 481–485.

[2] N. I. Cho and S. U. Lee, "Fast Algorithm and Implementation of 2-D Discrete Cosine Transform", *IEEE Trans. Circuits and Syst.*, Vol. 38, No. 3, Mar. 1991, pp. 297–305.

[3] N. I. Cho and S. U. Lee, "A Fast 4×4 DCT Algorithm for the Recursive 2-D DCT", *IEEE Trans. Signal Processing*, Vol. 40, No. 9, Sep. 1992, pp. 2166–2173.

[4] N. I. Cho, I. D. Yun, and S. U. Lee, "On the Regular Structure for the Fast 2-D DCT Algorithm", *IEEE Trans. Circuits and Syst.*, Vol. 40, No. 4, Apr. 1993, pp. 259–266.

[5] P. Duhamel and C. Guillemot, "Polynomial Transform Computation of the 2-D DCT", *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, Apr. 1990, pp. 1515–1518.

[6] E. Feig and S. Winograd, "Fast Algorithms for the Discrete Cosine Transform", *IEEE Trans. Signal Processing*, Vol. 40, No. 9, Sep. 1992, pp. 2174–2193.

[7] M. A. Haque, "A Two-dimensional Fast Cosine Transform", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. 33, No. 6, Dec. 1985, pp. 1532–1539.

[8] H. S. Hou, "A Fast Recursive Algorithm for Computing the Discrete Cosine Transform", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. 35, No. 10, Oct. 1987, pp. 1455–1461.

[9] F. A. Kamangar and K. R. Rao, "Fast Algorithms for the 2-D Discrete Cosine Transform", *IEEE Trans. Comput.*, Vol. 31, No. 9, Sep. 1982, pp. 899–906.

[10] B. G. Lee, "A New Algorithm to Compute the Discrete Cosine Transform", *IEEE Trans. Acoust., Speech, and Signal Processing*, Vol. 32, No. 6, Dec. 1984, pp. 1243–1245.

[11] P.-Z. Lee and F. Y. Huang, "New Recursive Algorithms for Computing the 1-D and 2-D Discrete Cosine Transforms", Technical Report TR-93-001, Institute of Information Science, Academia Sinica, Taipei, Taiwan, 1993. An extended-abstract version is in *Proc. of Fifth Digital Signal Processing Workshop*, Starved Rock State Park, IL., Sep. 1992, pp. 3.9.1–3.9.2.

[12] P.-Z. Lee and F. Y. Huang, "Restructured Recursive DCT and DST Algorithms", *IEEE Trans. Signal Processing*, Vol. 42, No. 7, July 1994, pp. 1600–1609.

[13] P.-Z. Lee and G. S. Liu, "An Efficient Algorithm for the 2-D Discrete Cosine Transform", Technical Report TR-94-001, Institute of Information Science, Academia Sinica, Taipei, Taiwan, 1994.

[14] F. A. McGovern, R. F. Woods, and M. Yan, "Novel VLSI Implementation of (8×8) Point 2-D DCT", *Electronics Letters*, Vol. 30, No, 8, Apr. 1994, pp. 624–626.

[15] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, and Applications.* Academic Press, Inc., San Diego, CA, 1990.

[16] M. Vetterli, "Fast 2-D Discrete Cosine Transform", *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, Mar. 1985, pp. 1538–1541.