Restructured Recursive DCT and DST Algorithms*

PeiZong Lee†and Fang-Yu Huang
Institute of Information Science
Academia Sinica
#128, Sec. 2, Yen-Chiu-Yun Road
Nankang, Taipei, Taiwan, R. O. C.

June 15, 1994

Abstract

The discrete cosine transform (DCT) and the discrete sine transform (DST) have 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. In this paper, we present new recursive algorithms for the DCT and the DST. The proposed method is based on certain recursive properties of the DCT coefficient matrix, and can be generalized to design recursive algorithms for the 2-D DCT and the 2-D DST. These new structured recursive algorithms are able to decompose the DCT and the DST into two balanced lower-order subproblems in comparison to previous research works. Therefore, when converting our algorithms into hardware implementations, we require fewer hardware components than other recursive algorithms. Finally, we propose two parallel algorithms for accelerating the computation.

EDICS number: 4.1, 4.1.1, 6.1, 6.1.3.

Index Terms: discrete cosine transform, discrete sine transform, hardware implementation, recursive algorithm, signal-flow graph.

*This work was partially supported by the NSC under Grants NSC 81-0408-E-001-505 and NSC 82-0408-E-001-016.

†PeiZong Lee: leepe@iis.sinica.edu.tw, TEL: +886 (2) 788-3799, FAX: +886 (2) 782-4814.

1 Introduction

This paper is concerned with designing fast recursive algorithms for the discrete cosine transform (DCT) and the discrete sine transform (DST). Many researchers pointed out that the DCT is especially suitable for implementing data compression, feature extraction, and filtering. This is because the DCT performs much like the theoretically optimal Karhunen-Loeve transform for the first-order Markov stationary random data [27]. In order to compute the DCT efficiently, fast algorithms have been intensively studied. In general, they can be classified into two approaches: indirect computation; and direct computation.

Indirect computation algorithms adopt the fast Fourier transforms [1] [13] [23] [26] or the fast Hartley transforms [16] [25], or convert the DCT's into circular convolutions which can be computed very efficiently using distributed arithmetics [22]. On the other hand, direct computation algorithms use techniques such as matrix factorizations [3] [9] [29] [30], divide and conquer method [20], recursive decomposition [7] [17], and small odd-length DCT modules which are derived from Winograd's small modules of real-value discrete Fourier transforms (DFT's) [14]. In general, indirect computation algorithms took advantage of using existing fast algorithms. However, additional operations were often involved in the computation steps. On the other hand, direct computation algorithms generally required fewer computation steps than indirect computation algorithms; however, it was tedious to prove their correctness. Moreover, it was often not easy to provide a clear procedure to describe their computation.

2-D DCT algorithms also have been studied[1]. The conventional approach for their fast computation is the row-column method. This method requires N_1 sets of N_2-point and N_2 sets of N_1-point 1-D DCT's for the computation of an $(N_1 \times N_2)$-point DCT. Makhoul [23], by applying a similar idea in Haralick [13], derived a 1-D DCT algorithm which was essentially identical to that in [26]; however, the input data points could be even or odd. Then, by generalizing his 1-D derivation method, he showed how an $(N_1 \times N_2)$-point DCT could be computed using an $(N_1 \times N_2)$-point real DFT. However, the 2-D DFT was still computed by the row-column method.

[1] Perhaps because the 2-D DST has not found enough applications, the only literature we found for discussing the 2-D DST adopts the conventional row-column method [4].

Further, there also existed algorithms working directly on the 2-D input data set and not separately on rows and columns. Kamangar and Rao [19], 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 and yielded the sparse matrix factorization for that 2-D coefficient matrix. Their algorithm performed fewer multiplications and additions than that of Makhoul's algorithm. In the Haque's algorithm [12], an $(N_1 \times N_2)$-point DCT was decomposed into four $(N_1/2 \times N_2/2)$-point subproblems. Each of the subproblems was a linear combination of four $(N_1/2 \times N_2/2)$-point DCTs and scaled by two diagonal matrices, whose diagonal entries were all fractions in the form of one over a certain cosine coefficient. His algorithm might be regarded as an extended 2-D version of Lee [20]; however, both of them suffered from the same problem of numerical instability because of the round-off error.

On the other hand, although the DST has not found as many applications as the DCT, many researchers have discussed its properties and its fast computation. For instance, based on the sparse matrix factorization, Yip and Rao [31] proposed a fast algorithm for a varied DST; Wang [28] showed that the DST can be computed based on the DCT; Gupta and Rao [11] showed that the DFT can be computed based on the DST. Gupta and Rao [10] as well as Huang and Lee [18] developed fast recursive algorithms for the DST. Wang [29], Ersoy and Hu [8], and Yip and Rao [32] [33] proposed systematic methods for computing a family of discrete cosine and sine transforms, which was introduced by Wang [29]. In this family, there are four versions each of the DCTs and the DSTs. The DCT and the DST discussed in this paper, as well as their inverse transforms also belong to it.

Generally, the above algorithms are efficient, if we only count the number of multiplications and the number of additions required. However, there are other important issues for designing a good DCT and a good DST algorithm. Besides proving the correctness of the algorithm, the procedure of the algorithm must be clear. Furthermore, the algorithm should be easily generalized to solve higher-order problems, provided we have solved lower-order problems. This property is especially attractive when we consider hardware implementations, because it allows us to use regular components to synthesize hardware to solve high-order problems.

It is our goal in this paper to present a recursive DCT algorithm and a recursive DST algorithm, which can be used to solve a higher-order problem from lower-order problems. In addition, these algorithms can be easily generalized as a 2-D DCT and a 2-D DST recursive algorithm. Although Hou [17] has designed a 1-D DCT recursive algorithm previously, his method was quite different from ours. Based on the work of Narasimha and Peterson [26], Hou wrote the angle of the DCT kernel as the angle of the DFT kernel plus a variable phase angle. Then, by using this particular form of the kernel, he permuted rows and columns in a certain order and proved the recursive property for the DCT coefficient matrix. Using a similar technique, Gupta and Rao [10] also designed a 1-D DST recursive algorithm. Based on Hou's 1-D algorithm, Chan and Ho [2] also proposed a 2-D DCT algorithm. They used the sequence splitting method to derive their algorithm and then used the Kronecker matrix product to formulate it. However, the former algebraic derivation, especially in the 2-D case, was somewhat tedious and didn't appropriately reflect the intrinsic structure of the latter matrix formulation. In addition, how to implement it is not clear.

Unlike their methods, we investigate the recursive property of the DCT coefficient matrix directly from the definition of the DCT kernel. Moreover, we can derive our recursive algorithm directly from the recursive property of the DCT coefficient matrix. The merit is that the whole procedure is very clear and easy to understand and to implement. In our DCT algorithm, the input data are arranged in the following order: the even indexes are first placed in increasing order; then the odd indexes are arranged but in decreasing order. This order was also suggested in other papers [2] [17] [26]. However, our intermediate steps for arranging data into appropriate memory locations require shuffle-exchanges, rather than Hou's bit-reversed shuffling. Finally, our algorithm gives rise to another structure which is different from Hou's algorithm when decomposing the DCT into several functional submodules. Based on this proposed DCT algorithm, we derive a new DST recursive algorithm whose structure is much simpler than that of the research works [10] [18].

The rest of this paper is organized as follows. In Section 2, we will explore the recursive property of the DCT coefficient matrix. In Section 3, we derive the 1-D DCT recursive algorithm. In Section 4, we generalize our method to derive a 2-D DCT recursive algorithm. In Section 5, we derive the 1-D DST recursive algorithm. In Section 6, we compare our results with other research works. In Section 7, we propose parallel algorithms and hardware implementations. Finally, some concluding remarks are given in Section 8.

2 Properties of the DCT Coefficient Matrix

For a given input data sequence x_n, $0 \le n \le N-1$, the DCT output sequence X_k, $0 \le k \le N-1$, is defined by

$$X_k = \sqrt{2/N}\ \epsilon(k) \sum_{n=0}^{N-1} x_n \cos(\pi(2n+1)k/2N)\,, \tag{1}$$

where

$$\epsilon(k) = \begin{cases} 1/\sqrt{2}\,, & \text{for } k=0; \\ 1\,, & \text{for } 1 \le k \le N-1. \end{cases}$$

We will remove $\epsilon(k)$ and the normalization factor $\sqrt{2/N}$ in (1), since they can be done in a separate step. Therefore, from now on, we deal with a simplified DCT version

$$\mathbf{X} = \mathbf{C}_N\,\mathbf{x}, \tag{2}$$

where $\mathbf{X} = (X_0, X_1, \ldots, X_{N-1})^T$, $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})^T$, and the $N \times N$ DCT coefficient matrix $\mathbf{C}_N$ is defined by

$$C_N(k,n) = \cos(\pi(2n+1)k/2N)\,,$$

for $0 \le k,\ n \le N-1$.

In this section, we want to derive a recursive formula for the DCT coefficient matrix. For convenience, we assume throughout this paper that N has a value of 2 to a power. We now analyze the DCT coefficient matrix $\mathbf{C}_N$. We will arrange both columns and rows in a certain order so that the resulting coefficient matrix has a recursive structure. First, we notice that in each of the even rows an entry is equal to the reflected-symmetric one, i.e., each row has mirror image terms which have corresponding values. While in each of the odd rows an entry is equal to the reflected-symmetric one but with an opposite sign. That is, for $0 \le n \le N-1$,

$$C_N(k,n) = \begin{cases} C_N(k, N-1-n)\,, & \text{if } k \text{ is even;} \\ -C_N(k, N-1-n)\,, & \text{if } k \text{ is odd.} \end{cases} \tag{3}$$

We now introduce two auxiliary matrices which are used to construct the recursive formula of the DCT coefficient matrix.

Definition:

1. *Let P_N be the column permutation which arranges columns of a $N \times N$ matrix in the order of*

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

Define $\bar{\mathbf{C}}_N = \mathbf{C}_N\, P_N$, and whose (k,n)-th entry is $\bar{C}_N(k,n)$.

2. *Let S_N^T be the row permutation which arranges rows of a $N \times N$ matrix in the order of*

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

Define $\hat{\mathbf{C}}_N = S_N^T\, \bar{\mathbf{C}}_N$, and whose (k,n)-th entry is $\hat{C}_N(k,n)$.

We prefer to use S_N^T to denote the row permutation matrix, because when performing S_N^T on a vector, the operation is an unshuffled exchange. The symbol S_N is reserved for performing a shuffle exchange operation. In the following, we show the relationship among $\mathbf{C}_N$, $\bar{\mathbf{C}}_N$, and $\hat{\mathbf{C}}_N$. From the definitions of $\bar{\mathbf{C}}_N$ and $\hat{\mathbf{C}}_N$, we have

$$\text{for } 0 \le n \le N/2-1, \qquad \begin{array}{l} \bar{C}_N(k,n) = C_N(k,2n), \ \text{ and} \\ \bar{C}_N(k,n+N/2) = C_N(k,N-1-2n); \end{array} \tag{4}$$

$$\text{for } 0 \le k \le N/2-1, \qquad \begin{array}{l} \hat{C}_N(k,n) = \bar{C}_N(2k,n), \ \text{ and} \\ \hat{C}_N(k+N/2,n) = \bar{C}_N(2k+1,n). \end{array} \tag{5}$$

From (3)—(5), for $0 \le k \le N/2-1$ and $0 \le n \le N/2-1$,

$$\begin{array}{l} \hat{C}_N(k,n) = \hat{C}_N(k,n+N/2), \ \text{ and} \\ \hat{C}_N(k+N/2,n) = -\hat{C}_N(k+N/2,n+N/2). \end{array} \tag{6}$$

In addition, the $N/2 \times N/2$ upper-left corner submatrix of $\hat{\mathbf{C}}_N$ is equal to $\bar{\mathbf{C}}_{N/2}$. That is,

$$[\hat{C}_N(k,n)]_{\substack{0 \le k \le N/2-1 \\ 0 \le n \le N/2-1}} = \bar{\mathbf{C}}_{N/2}. \tag{7}$$

Furthermore, if we denote the submatrix

$$[\hat{C}_N(k+N/2,n)]_{\substack{0 \le k \le N/2-1 \\ 0 \le n \le N/2-1}}$$

as $D_{N/2}$, then from Equations (6) and (7) we obtain the following theorem.

Theorem 1 :

$$\hat{\mathbf{C}}_N = \begin{bmatrix} \bar{\mathbf{C}}_{N/2} & \bar{\mathbf{C}}_{N/2} \\ D_{N/2} & -D_{N/2} \end{bmatrix}. \qquad \square$$

In Theorem 1, $\bar{\mathbf{C}}_{N/2}$ can be arranged to get $\hat{\mathbf{C}}_{N/2}$. However, in order to get a recursive algorithm, $D_{N/2}$ must be converted into a certain form related to either $\bar{\mathbf{C}}_{N/2}$ or $\hat{\mathbf{C}}_{N/2}$. In [21], we show that $D_{N/2}$ can be decomposed into the product of three matrices, including one regular lower triangular matrix $L_{N/2}$, one $\bar{\mathbf{C}}_{N/2}$, and one diagonal matrix $Q_{N/2}$. That is,

$$D_{N/2} = L_{N/2}\,\bar{\mathbf{C}}_{N/2}\,Q_{N/2}, \tag{8}$$

where L is a lower triangular matrix defined by

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

and

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

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

The following theorem, which follows immediately from Theorem 1 and Equation (8), shows the recursive structure of the coefficient matrix, and we will use it to design our DCT recursive algorithms.

Theorem 2 :

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

3 The DCT Recursive Algorithm

In this section, we will provide our DCT recursive algorithm. Before that, we first explain the relationship of $\mathbf{C}$, $\bar{\mathbf{C}}$, and $\hat{\mathbf{C}}$ when performing the DCT. We know that $\bar{\mathbf{C}}_N = \mathbf{C}_N\,P_N$ and $\hat{\mathbf{C}}_N =$

$S_N^T \bar{\mathbf{C}}_N$. We start with the simplified DCT version $\mathbf{X} = \mathbf{C}_N \mathbf{x}$ in Equation (2). First, we apply the column permutation P_N and the row permutation S_N^T to the matrix $\mathbf{C}_N$. Because P_N is orthonomal, $P_N^T = P_N^{-1}$, and we have

$$\begin{bmatrix} X_e \\ X_o \end{bmatrix} = S_N^T \mathbf{X} = S_N^T \mathbf{C}_N \mathbf{x} = S_N^T \mathbf{C}_N \underline{P_N P_N^T} \mathbf{x} = \hat{\mathbf{C}}_N P_N^T \mathbf{x} = \hat{\mathbf{C}}_N \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix} \tag{9}$$

where $\begin{bmatrix} X_e \\ X_o \end{bmatrix}$ means $(X_0, X_2, \ldots, X_{N-2}, X_1, X_3, \ldots, X_{N-1})^T$ and $\begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix}$ means $(x_0, x_2, \ldots, x_{N-2}, x_{N-1}, \ldots, x_3, x_1)^T$. We will use the subindex $'e'$ to represent the even entries of a vector, $'o'$ the odd entries, and $'\bar{o}'$ the odd entries but in reversed order. Therefore, from Theorem 2 and Equation (9) we have

$$\begin{bmatrix} X_e \\ X_o \end{bmatrix} = \begin{bmatrix} S_{N/2} \hat{\mathbf{C}}_{N/2} & S_{N/2} \hat{\mathbf{C}}_{N/2} \\ L_{N/2} S_{N/2} \hat{\mathbf{C}}_{N/2} Q_{N/2} & -L_{N/2} S_{N/2} \hat{\mathbf{C}}_{N/2} Q_{N/2} \end{bmatrix} \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix}. \tag{10}$$

From Equation (10), X_e and X_o can be easily derived.

$$X_e = S_{N/2} \hat{\mathbf{C}}_{N/2} (x_e + x_{\bar{o}}) \text{ and } X_o = L_{N/2} S_{N/2} \hat{\mathbf{C}}_{N/2} Q_{N/2} (x_e - x_{\bar{o}}).$$

In the following, we describe the procedure for computing the DCT.

1-D DCT recursive algorithm:

Step 1: Compute $\begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix} = P_N^T \mathbf{x}$;

Step 2: /* Recursively compute $\begin{bmatrix} X_e \\ X_o \end{bmatrix} = \hat{\mathbf{C}}_N \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix}$. */

2.1 if N is 2, then compute $\mathbf{X} = \begin{bmatrix} X_e \\ X_o \end{bmatrix} = \hat{\mathbf{C}}_2 \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix} = \mathbf{C}_2 \mathbf{x}$ directly;

2.2 else recursively perform $X_e = S_{N/2} \hat{\mathbf{C}}_{N/2} (x_e + x_{\bar{o}})$, $X_o = L_{N/2} S_{N/2} \hat{\mathbf{C}}_{N/2} Q_{N/2} (x_e - x_{\bar{o}})$;

Step 3: compute $\mathbf{X} = S_N \begin{bmatrix} X_e \\ X_o \end{bmatrix}$. $\square$

Fig. 1 shows the general signal-flow graph for the implementation of Step 2.2. Fig. 2 shows signal flow graphs for three lower order 1-D DCT's. Since the DCT is an orthogonal transform, the inverse DCT also can be realized simply by taking transposition of the above DCT algorithm.

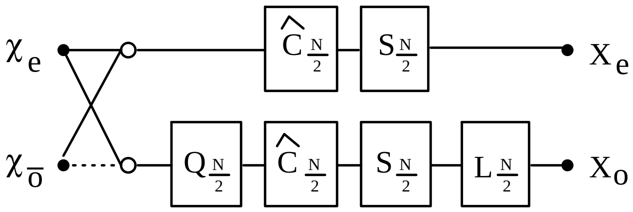

Figure 1: Implementation of Step 2 in the DCT recursive algorithm. Solid lines represent transfer factor 1, while dashed lines represent transfer factor -1. Circles $\circ$ represent adders.

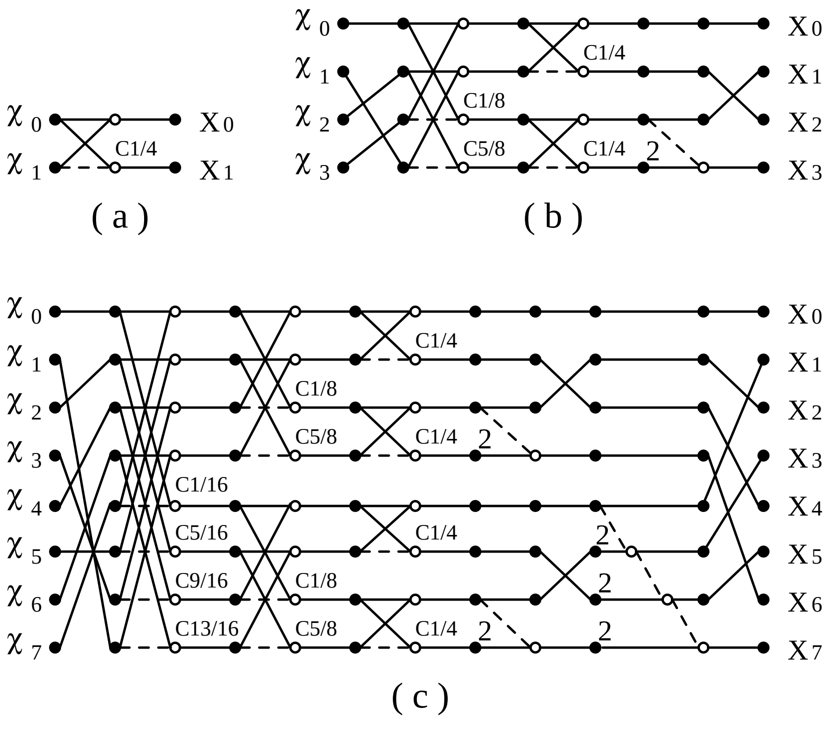

Figure 2: The DCT signal-flow graphs for (a) $N = 2$, (b) $N = 4$, and (c) $N = 8$. Ca/b means $\cos(a\pi/b)$.

4 The 2-D DCT Recursive Algorithm

For a given input data matrix $\mathbf{x}_{N_1 \times N_2} = [x_{n_1,n_2}]$, $0 \leq n_1 \leq N_1 - 1$, $0 \leq n_2 \leq N_2 - 1$, the 2-D DCT output matrix $\mathbf{X}_{N_1 \times N_2} = [X_{k_1,k_2}]$, $0 \leq k_1 \leq N_1 - 1$, $0 \leq k_2 \leq N_2 - 1$, which is defined by

$$X_{k_1,k_2} = (2/\sqrt{N_1 N_2})\, \epsilon(k_1)\, \epsilon(k_2) \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos(\pi(2n_1+1)k_1/2N_1) \cos(\pi(2n_2+1)k_2/2N_2) \,, \tag{11}$$

where

$$\epsilon(k_i) = \begin{cases} 1/\sqrt{2}\,, & \text{for } k_i = 0 \\ 1\,, & \text{for } 1 \leq k_i \leq N_i - 1 \end{cases}$$

and $i = 1$ or 2.

Similar to the 1-D case, we will also ignore $\epsilon(k_i)$ and the normalization factor $2/\sqrt{N_1 N_2}$. Therefore, from now on, we deal with the simplified version of Equation (11):

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos(\pi(2n_1+1)k_1/2N_1) \cos(\pi(2n_2+1)k_2/2N_2) \,. \tag{12}$$

According to the conventional row-column method, we can express the simplified 2-D DCT version in Equation (12) as the following matrix form:

$$\mathbf{X}_{N_1 \times N_2} = \mathbf{C}_{N_1} \,(\mathbf{C}_{N_2}\, \mathbf{x}^T_{N_1 \times N_2})^T = \mathbf{C}_{N_1}\, \mathbf{x}_{N_1 \times N_2}\, \mathbf{C}^T_{N_2}. \tag{13}$$

Again, we assume throughout this paper that N_1 and N_2 have values of 2 to a power. We now construct our 2-D DCT recursive formula. We start with the simplified 2-D DCT version in Equation (13). First, by arranging appropriate row and column permutations, we have

$$\begin{aligned} S^T_{N_1}\, \mathbf{X}_{N_1 \times N_2}\, S_{N_2} &= S^T_{N_1}\, \mathbf{C}_{N_1}\, \mathbf{x}_{N_1 \times N_2}\, \mathbf{C}^T_{N_2}\, S_{N_2} \\ &= S^T_{N_1}\, \mathbf{C}_{N_1}\, \underline{P_{N_1}\, P^T_{N_1}}\, \mathbf{x}_{N_1 \times N_2}\, (\underline{S_{N_2}\, S^T_{N_2}}\, \mathbf{C}_{N_2}\, \underline{P_{N_2}\, P^T_{N_2}})^T\, S_{N_2} \\ &= \hat{\mathbf{C}}_{N_1}\, P^T_{N_1}\, \mathbf{x}_{N_1 \times N_2}\, (S_{N_2}\, \hat{\mathbf{C}}_{N_2}\, P^T_{N_2})^T\, S_{N_2} \\ &= \hat{\mathbf{C}}_{N_1}\, P^T_{N_1}\, \mathbf{x}_{N_1 \times N_2}\, P_{N_2}\, \hat{\mathbf{C}}^T_{N_2} \end{aligned} \tag{14}$$

where S_{N_i}, $S^T_{N_i}$, P_{N_i}, and $P^T_{N_i}$ are similarly defined as in Section 2, for $i = 1$ or 2; in addition, $S_{N_1}\, S^T_{N_1} = P_{N_1}\, P^T_{N_1} = I_{N_1}$ and $S_{N_2}\, S^T_{N_2} = P_{N_2}\, P^T_{N_2} = I_{N_2}$.

We now consider the structures of the two matrices: $P_{N_1}^T\,\mathbf{x}_{N_1\times N_2}\,P_{N_2}$ and $S_{N_1}^T\,\mathbf{X}_{N_1\times N_2}\,S_{N_2}$. Let x_{ee} be the submatrix formed by the even rows and the even columns of $\mathbf{x}_{N_1\times N_2}$; $x_{e\bar{o}}$ represent the submatrix formed by the even rows and the odd columns of $\mathbf{x}_{N_1\times N_2}$ but the odd columns are in reversed order; $x_{\bar{o}e}$ represent the submatrix formed by the odd rows and the even columns of $\mathbf{x}_{N_1\times N_2}$ but the odd rows are in reversed order; and $x_{\bar{o}\bar{o}}$ represent the submatrix formed by the odd rows and the odd columns of $\mathbf{x}_{N_1\times N_2}$, but both are in reversed order. Then,

$$P_{N_1}^T\,\mathbf{x}_{N_1\times N_2}\,P_{N_2} = \begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix}_{N_1\times N_2}. \tag{15}$$

Similarly, let X_{ee} represent the submatrix formed by the even rows and the even columns of $\mathbf{X}_{N_1\times N_2}$; X_{eo} represent the submatrix formed by the even rows and the odd columns of $\mathbf{X}_{N_1\times N_2}$; X_{oe} represent the submatrix formed by the odd rows and the even columns of $\mathbf{X}_{N_1\times N_2}$; and X_{oo} represent the submatrix formed by the odd rows and the odd columns of $\mathbf{X}_{N_1\times N_2}$. Then,

$$S_{N_1}^T\,\mathbf{X}_{N_1\times N_2}\,S_{N_2} = \begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix}_{N_1\times N_2}. \tag{16}$$

Therefore, from Equations (14), (15), and (16), we have

$$\begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix} = \hat{\mathbf{C}}_{N_1} \begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix} \hat{\mathbf{C}}_{N_2}^T. \tag{17}$$

Equation (17) is our basic 2-D DCT recursive formula. Now, by applying Theorem 2, we obtain the following four equations:

$$\begin{aligned}
X_{ee} &= S_{N_1/2}\,[\hat{\mathbf{C}}_{N_1/2}\,(x_{ee} + x_{e\bar{o}} + x_{\bar{o}e} + x_{\bar{o}\bar{o}})\,\hat{\mathbf{C}}_{N_2/2}^T]\,S_{N_2/2}^T; \\
X_{eo} &= S_{N_1/2}\,[\hat{\mathbf{C}}_{N_1/2}\,(x_{ee} - x_{e\bar{o}} + x_{\bar{o}e} - x_{\bar{o}\bar{o}})\,Q_{N_2/2}\,\hat{\mathbf{C}}_{N_2/2}^T]\,S_{N_2/2}^T\,L_{N_2/2}^T; \\
X_{oe} &= L_{N_1/2}\,S_{N_1/2}\,[\hat{\mathbf{C}}_{N_1/2}\,Q_{N_1/2}\,(x_{ee} + x_{e\bar{o}} - x_{\bar{o}e} - x_{\bar{o}\bar{o}})\,\hat{\mathbf{C}}_{N_2/2}^T]\,S_{N_2/2}^T; \\
X_{oo} &= L_{N_1/2}\,S_{N_1/2}\,[\hat{\mathbf{C}}_{N_1/2}\,Q_{N_1/2}\,(x_{ee} - x_{e\bar{o}} - x_{\bar{o}e} + x_{\bar{o}\bar{o}})\,Q_{N_2/2}\,\hat{\mathbf{C}}_{N_2/2}^T]\,S_{N_2/2}^T\,L_{N_2/2}^T.
\end{aligned}$$

Note that the terms inside the bracket of the above four equations are four subproblems with a reduced size $N_1/2 \times N_2/2$ of Equation (17). By applying the same procedure repeatedly, we can decompose each of these reduced-size subproblems until $N_1 = N_2 = 2$, and we obtain the desired recursive algorithm. In the following, we give the procedure for computing the 2-D DCT.

2-D DCT recursive algorithm:

Step 1: Compute $\begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix} = P_{N_1}^T \, \mathbf{x}_{N_1 \times N_2} \, P_{N_2};$

Step 2: /* recursively compute $\begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix} = \hat{\mathbf{C}}_{N_1} \begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix} \hat{\mathbf{C}}_{N_2}^T.$ */

2.1 if N_1 is 2 or N_2 is 2, then this is a special case and we omit details here;

2.2 else perform Steps from 2.2.1 to 2.2.5:

2.2.1 compute $\begin{array}{llll} t_1 = x_{ee} + x_{e\bar{o}}, & t_2 = x_{ee} - x_{e\bar{o}}, & t_3 = x_{\bar{o}e} + x_{\bar{o}\bar{o}}, & t_4 = x_{\bar{o}e} - x_{\bar{o}\bar{o}}, \\ x_a = t_1 + t_3, & x_b = t_2 + t_4, & x_c = t_1 - t_3, & x_d = t_2 - t_4; \end{array}$

2.2.2 recursively compute $X_{ee} = S_{N_1/2} \, (\hat{\mathbf{C}}_{N_1/2} \, x_a \, \hat{\mathbf{C}}_{N_2/2}^T) \, S_{N_2/2}^T;$

2.2.3 recursively compute $X_{eo} = S_{N_1/2} \, (\hat{\mathbf{C}}_{N_1/2} \, x_b \, Q_{N_2/2} \, \hat{\mathbf{C}}_{N_2/2}^T) \, S_{N_2/2}^T \, L_{N_2/2}^T;$

2.2.4 recursively compute $X_{oe} = L_{N_1/2} \, S_{N_1/2} \, (\hat{\mathbf{C}}_{N_1/2} \, Q_{N_1/2} \, x_c \, \hat{\mathbf{C}}_{N_2/2}^T) \, S_{N_2/2}^T;$

2.2.5 recursively compute $X_{oo} = L_{N_1/2} \, S_{N_1/2} \, (\hat{\mathbf{C}}_{N_1/2} \, Q_{N_1/2} \, x_d \, Q_{N_2/2} \, \hat{\mathbf{C}}_{N_2/2}^T) \, S_{N_2/2}^T \, L_{N_2/2}^T;$

Step 3: compute $\mathbf{X}_{N_1 \times N_2} = S_{N_1} \begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix} S_{N_2}^T.$ $\square$

Fig. 3 shows the general signal-flow graph for the implementation of Step 2.2. Fig. 4 shows signal flow graphs for two lower order 2-D DCT's.

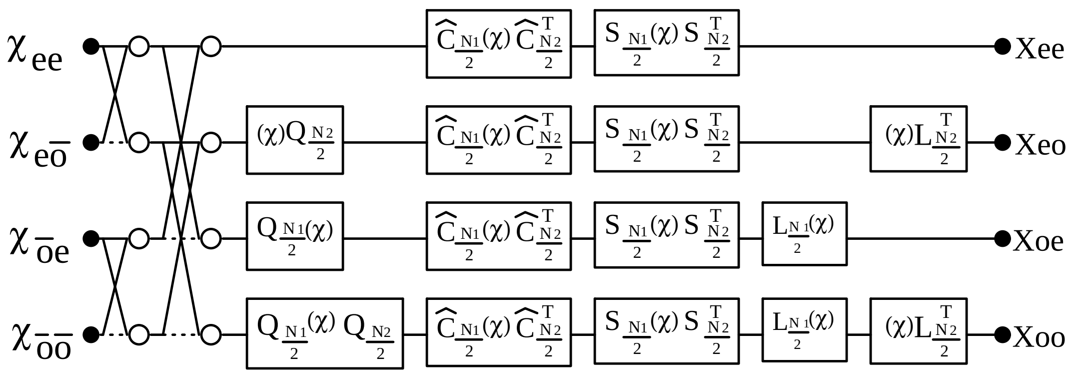

Figure 3: Implementation of Step 2 in the 2-D DCT recursive algorithm. ($\mathbf{x}$)'s represent current input $N_1/2 \times N_2/2$ submatrices.

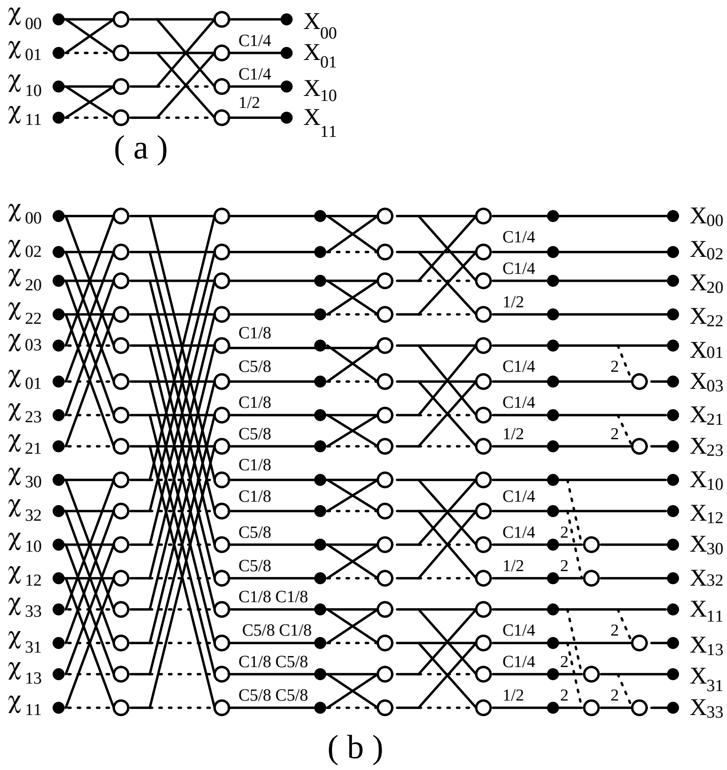

Figure 4: The $N \times N$ 2-D DCT signal-flow graphs for (a) $N = 2$ and (b) $N = 4$, where each of the $N/2 \times N/2$ data subsets are arranged into a 1-D array in lexicographical order for illustration.

5 The DST Recursive Algorithm

In this section, we show how to derive the DST recursive algorithm. We first state Wang's method for computing an N-point DST by an N-point DCT [28]. Then, we derive our DST recursive algorithm which is based on our proposed DCT recursive algorithm. The generalization of designing a 2-D DST recursive algorithm is similar to that of designing a 2-D DCT recursive algorithm described in Section 4, and we omit it in this presentation.

For a given input data sequence x_n, $0 \leq n \leq N-1$, the DST output sequence X_m^s, $0 \leq m \leq N-1$, is defined by

$$X_m^s = \sqrt{2/N}\; \epsilon^s(m) \sum_{n=0}^{N-1} x_n \, \sin(\pi(2n+1)(m+1)/2N)\,, \tag{18}$$

where

$$\epsilon^s(m) = \begin{cases} 1\,, & \text{for } 0 \le m \le N-2; \\ 1/\sqrt{2}\,, & \text{for } m = N-1. \end{cases}$$

Similar to the DCT case, we will also remove $\epsilon(k)$ and the normalization factor $\sqrt{2/N}$, and rewrite Equation (18) in a simplified version

$$X_m^s = \sum_{n=0}^{N-1} x_n \, \sin(\pi(2n+1)(m+1)/2N)\,.$$

We now distil Wang's method into a procedure for the computation of the DST by using the DCT. Note that, for the sake of distinction, we use the superscript $'s'$ for denoting the sine transform, and $'c'$ for denoting the cosine transform.

Computation of the DST by the DCT:

Step 1: Compute the data sequence $\{y_n\}_{n=0}^{N-1}$, where $y_n = (-1)^n x_n$;

Step 2: compute the DCT output sequence $\{Y_m^c\}_{m=0}^{N-1}$ for the data sequence $\{y_n\}_{n=0}^{N-1}$ by using a 1-D DCT algorithm;

Step 3: compute $X_m^s = Y_{N-1-m}^c$, for $0 \le m \le N-1$. $\square$

With the above procedure, we begin to derive our DST recursive algorithm. According to Step 1, we first compute $\mathbf{y} = A_N\,\mathbf{x}$, where A_N is an $N \times N$ matrix defined by

$$A_N(i,j) = \begin{cases} (-1)^i & \text{for } i = j\,; \\ 0 & \text{for } i \ne j\,. \end{cases}$$

Then, by utilizing our proposed DCT recursive algorithm, we perform $\mathbf{Y}^c = S_N\,\hat{\mathbf{C}}_N\,P_N^T\,\mathbf{y}$. After that, according to Step 3, we calculate $\mathbf{X}^s = B_N\,\mathbf{Y}^c$, where B_N is an $N \times N$ matrix defined by

$$B_N(i,j) = \begin{cases} 1 & \text{for } i+j = N-1\,; \\ 0 & \text{for } i+j \ne N-1\,. \end{cases}$$

Consequently, we have the following relation:

$$\mathbf{X}^s = B_N\,S_N\,\hat{\mathbf{C}}_N\,P_N^T\,A_N\,\mathbf{x}\,. \tag{19}$$

Since $B_N\, B_N^T = I_N$ and $S_N\, S_N^T = I_N$, Equation (19) is equivalent to

$$\underline{S_N^T\, B_N^T}\, \mathbf{X}^s = \hat{\mathbf{C}}_N\, \underline{P_N^T\, A_N}\, \mathbf{x}\,. \tag{20}$$

This equation means that, by appropriately arranging and adapting the input data vector $\mathbf{x}$, we can compute the DST output, which will be in a certain permuted order, via the DCT coefficient kernel. In effect, we can write Equation (20) as

$$\begin{bmatrix} X^s_{\bar{o}} \\ X^s_{\bar{e}} \end{bmatrix} = \hat{\mathbf{C}}_N \begin{bmatrix} x_e \\ \tilde{x}_{\bar{o}} \end{bmatrix},$$

where $\begin{bmatrix} x_e \\ \tilde{x}_{\bar{o}} \end{bmatrix} = P_N^T\, A_N\, \mathbf{x}$ and $\begin{bmatrix} X^s_{\bar{o}} \\ X^s_{\bar{e}} \end{bmatrix} = S_N^T\, B_N^T\, \mathbf{X}^s$. Here we use the symbol $'\tilde{\cdot}'$ to represent the negation of a vector, the subindex $'e'$ to represent the even entries of a vector, $'\bar{e}'$ the even entries but in reversed order, and $'\bar{o}'$ the odd entries but in reversed order. For illustration, we see that $P_8^T\, A_8\, \mathbf{x} = (x_0, x_2, x_4, x_6, -x_7, -x_5, -x_3, -x_1)$; $S_8^T\, B_8^T\, \mathbf{X}^s = (X_7, X_5, X_3, X_1, X_6, X_4, X_2, X_0)$. The following algorithm gives the procedure for computing the DST. Fig. 5 shows the general signal-flow graph for the implementation of Step 2.

DST recursive algorithm:

Step 1: Compute $\begin{bmatrix} x_e \\ \tilde{x}_{\bar{o}} \end{bmatrix} = P_N^T\, A_N\, \mathbf{x}$;

Step 2: Recursively compute $\begin{bmatrix} X^s_{\bar{o}} \\ X^s_{\bar{e}} \end{bmatrix} = \hat{\mathbf{C}}_N \begin{bmatrix} x_e \\ \tilde{x}_{\bar{o}} \end{bmatrix}$;

Step 3: compute $\mathbf{X}^s = B_N\, S_N \begin{bmatrix} X^s_{\bar{o}} \\ X^s_{\bar{e}} \end{bmatrix}$. □

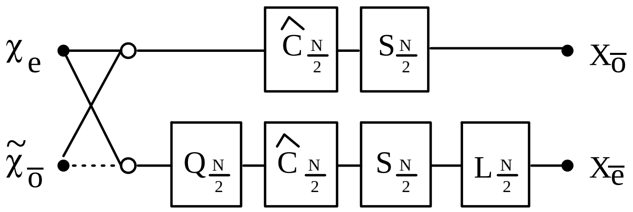

Figure 5: Implementation of Step 2 in the DST recursive algorithm.

6 Comparisons with Related Works

6.1 Comparison of the DCT Algorithms

A detailed analysis of the proposed 1-D and 2-D DCT algorithms is given in [21]. For computing an N-point 1-D DCT, it requires $(1/2)N \log N$ multiplications and $(3/2)N \log N - N + 1$ additions. For computing an $(N \times N)$-point 2-D DCT, it requires $(3/4)N^2 \log N - (1/4)N^2$ multiplications and $3N^2 \log N - 2N^2 + 2N$ additions. Table 1 and Table 2 show the comparisons of the operation counts among those 1-D and 2-D DCT algorithms found in the related research works, respectively.

	Our method		[7, 17] [20, 30]		[3]		[25]		[29]	
N	$\times$	$+$	$\times$	$+$	$\times$	$+$	$\times$	$+$	$\times$	$+$
2	1	2								
4	4	9	4	9	6	8	5	9	5	9
8	12	29	12	29	16	26	13	35	13	29
16	32	81	32	81	44	74	33	95	35	83
32	80	209	80	209	116	194	81	251	91	219
64	192	513	192	513	292	482	193	615	227	547

Table 1: The comparison of the numbers of operations required for computing the 1-D DCT. '$\times$' means 'number of multiplications', '$+$' means 'number of additions or subtractions'.

	Our method		Row-Column method		[2, 12]		[5]	
$N \times N$	$\times$	$+$	$\times$	$+$	$\times$	$+$	$\times$	$+$
2×2	2	8						
4×4	20	72	32	72	24	72	16	74
8×8	128	464	192	464	144	464	96	466
16×16	704	2592	1024	2592	768	2592	512	2530
32×32	3584	13376	5120	13376	3840	13376	2560	12738
64×64	17408	65664	24576	65664	18432	65664		

Table 2: The comparison of the numbers of operations required for computing the 2-D DCT.

We see that, in the 1-D DCT part, the numbers of multiplications and additions are the same as those required in the best and well-known algorithms of [17] [20] [30]. In addition, the number of multiplications is less than that required in [3] [24] [29]. In the 2-D DCT part, the numbers of multiplications are less than those of [2] [12] and the conventional row-column method. The numbers of additions are the same as those of [2] [12] and the conventional row-column method.

Cho and Lee [5] claimed a fast 2-D algorithm, in which an $(N \times N)$-point DCT could be obtained by computing N sets of N-point 1-D DCT's and a post-addition stage. So the complexity of their algorithm is greatly reduced. however, the required arrangement of the input data regrouping was an obvious bottleneck for the fast computation in a conventional computer and it was also difficult to write a simple procedure to describe the convoluted additions. Recently, they provided regular expressions for the input-output relations of the post-addition stage [6]. However, the number of additions increases at the expense of improving the regularity in the structure.

After the submission of this paper, the authors learned that Cvetković and Popović [7] have independently given another recursive DCT algorithm. The structure of the proposed algorithm is the same as Hou's, but the new way of derivation they proposed allows the shift (multiply by 2) operations required in Hou's algorithm to be completely eliminated. However, this needs to pay some price. Since the required multiplication coefficients are all fractions in the form of one over a certain cosine coefficient, these will involve a lot of round-off errors as we have pointed out previously in the introduction.

6.2 Comparison of the DST Algorithms

In Section 5, we have derived a 1-D DST recursive algorithm which is based on our 1-D DCT recursive algorithm. Table 3 gives the computation complexities for the DST algorithms found in the related research works. We see that the numbers of multiplications and additions are less than those of [10] [18] [29].

The DST algorithms proposed by Ersoy and Hu [8] are also based on the DCT. As for the DCT, Ersoy and Hu designed a fast algorithm for a real DFT and used it to compute the DCT. The operation counts they provided are the same as that of the proposed one, however, they employed Given's plane rotations as the basic computation unit. Yip and Rao [32] [33] provided the decimation-in-time and the decimation-in-frequency algorithms for the DST and the DCT. The

	# of multiplications	# of additions
Our algorithm	$(1/2)N \log N$	$(3/2)N \log N - N + 1$
[10] [18]	$N \log N - (1/2)N$	$(5/2)N \log N - 3N + 3$
[29]	$(3/4)N \log N - N + 3$	$(7/4)N \log N - 2N + 3$
[7] [8] [32]	$(1/2)N \log N$	$(3/2)N \log N - N + 1$
[33]	$(1/2)N \log N + (1/4)N - 1$	$(3/2)N \log N + (1/2)N - 2$

Table 3: Computation complexities for the 1-D DST.

operation counts of their algorithms are as good as that of our DST algorithm. However, their algorithms involve multiplications with fractions in the form of one over a certain sine or cosine coefficient. Therefore, their algorithms are numerically unstable because of the round-off error.

As mentioned in Section 6.1, Cvetković and Popović also proposed two DST recursive algorithms in [7]. Both of them improved the operation counts required in [10] and [18], but still had the same algorithm structure as Hou's. Our proposed DST recursive algorithm is as good as the first DST recursive algorithm. Although the second DST recursive algorithm could get rid of shift operations as the authors did in their DCT recursive algorithm, the required multiplication coefficients are all fractions in the form of one over a certain cosine coefficient. It also confronted the problem of a large round-off error.

7 Parallel Implementations and Hardware Considerations

In this section, we first analyze the bottleneck in the parallel computation, and then we propose parallel algorithms and hardware implementations for the DCT. The DST also can be dealt with in a likewise fashion.

7.1 Parallel DCT Implementations

Suppose that we have sufficient processing elements to implement the DCT. We now analyze the complexities for the parallel computation. Consider the sequential DCT algorithm:

$$X_e = S_{N/2}\,\hat{\mathbf{C}}_{N/2}\,(x_e + x_{\bar{o}}) \;\text{ and }\; X_o = L_{N/2}\,S_{N/2}\,\hat{\mathbf{C}}_{N/2}\,Q_{N/2}\,(x_e - x_{\bar{o}}).$$

- $u_1 = (x_e + x_{\bar{o}})$ and $v_1 = (x_e - x_{\bar{o}})$ are concerned with two vector additions, which can be done in one unit of time with two parallel additions simultaneously.

- $v_2 = Q_{N/2}\, v_1$ deals with a matrix-vector multiplication. Because $Q_{N/2}$ is a diagonal matrix, this step can be done in one parallel multiplication.

- $u_2 = \hat{\mathbf{C}}_{N/2}\, u_1$ and $v_3 = \hat{\mathbf{C}}_{N/2}\, v_2$ are concerned with two lower-order subproblems, which can be solved independently.

- $X_e = S_{N/2}\, u_2$ and $v_4 = S_{N/2}\, v_3$ deal with data arrangement, which can be done in one unit of time using two shuffle-exchanges simultaneously.

- $X_o = L_{N/2}\, v_4$ also deals with a matrix-vector multiplication. Because the computation involves a first-order linear recurrence, a naive implementation requires $N/2-1$ units of time in parallel computation for shifts and additions.

It is clear that to compute $X_o = L_{N/2}\, v_4$ is the bottleneck for the parallel computation. This is because it incurs linear time steps for shifts and additions in the parallel computation. We now introduce two fast parallel algorithms, which can compute the general problem $y = L_N\, v$ in only one parallel shift step and $O(\log N)$ parallel addition steps.

Because of the regularity of the matrix L_N, $y = L_N\, v$ can be computed in the following way:

$$\begin{aligned} y_0 &= w_0 = v_0; \\ y_i &= w_i - y_{i-1} = 2v_i - y_{i-1}, \quad \text{for } 1 \leq i \leq N-1, \end{aligned} \tag{21}$$

where w is a vector of length N. Vector w can be obtained in only one step by doing a parallel shift $w_i = 2v_i$ for all $1 \leq i \leq N-1$. We now show that Equation (21) can be computed by only $O(\log N)$ steps of parallel additions. For convenience, let us call the problem in Equation (21) the *prefix difference problem.*

The first algorithm, which uses the recursive doubling technique, can compute the prefix difference problem in only $\log N$ parallel addition steps.

Recursive doubling algorithm for the prefix difference problem:

Step 1: $y_0 = w_0$, and

for all $1 \leq i \leq N-1$, **do** $y_i = w_i - w_{i-1}$;

Step k**:** (for $2 \leq k \leq \log N$)

for all $2^{(k-1)} \leq i \leq N-1$, **do** $y_i = y_i + y_{i-2^{(k-1)}}$. □

Fig. 6 shows a signal-flow graph that when $N = 16$, the problem $L_N v$ can be computed in only one parallel shift step and $\log N$ (= 4) parallel addition steps based on the recursive doubling algorithm.

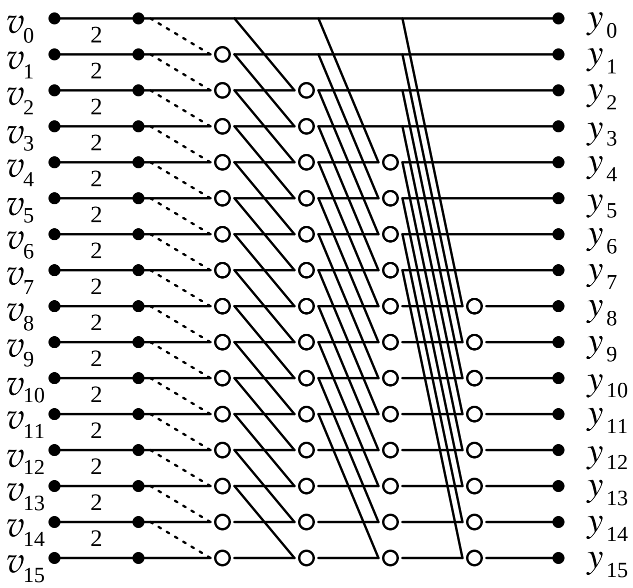

Figure 6: The signal-flow graph for computing $L_{16}\, v$ by the recursive doubling algorithm.

The second algorithm uses the cyclic reduction technique, which was first introduced by Hockney [15] to solve regular tridiagonal linear systems in conventional computers. The parallel version can compute the prefix difference problem in only $2(\log N) - 1$ parallel addition steps.

Cyclic reduction algorithm for the prefix difference problem:

Step 1: $y_0 = w_0$, and

for all $0 < i < N/2$, **do** $y_{2i} = w_{2i} - w_{2i-1}$;

Step k**:** (for $2 \le k \le (\log N) - 1$)

for all $0 < i < N/2^k$, **do** $y_{i2^k} = y_{i2^k} + y_{i2^k - 2^{(k-1)}}$;

Step $k + \log N$**:** (for $0 \le k \le (\log N) - 2$)

for all $0 \le i < 2^k$, **do** $y_{i2^{((\log N)-k)}+2^{((\log N)-k-1)}} = y_{i2^{((\log N)-k)}+2^{((\log N)-k-1)}} + y_{i2^{((\log N)-k)}}$;

Step $2(\log N) - 1$**: for all** $0 \le i < N/2$, **do** $y_{2i+1} = w_{2i+1} - y_{2i}$. $\square$

Fig. 7 shows a signal-flow graph that when $N = 16$, the problem $L_N\, v$ can be computed in only one parallel shift step and $2(\log N) - 1$ $(= 7)$ parallel addition steps based on the cyclic reduction algorithm.

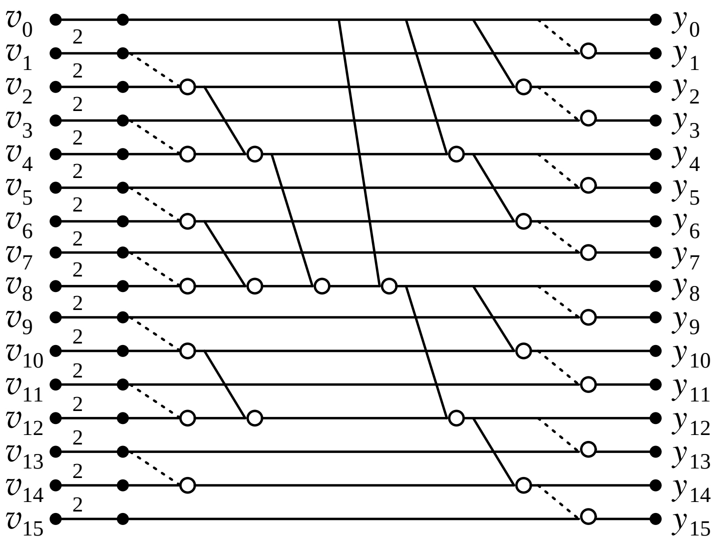

Figure 7: The signal-flow graph for computing $L_{16}\, v$ by the cyclic reduction algorithm.

The recursive doubling algorithm and the cyclic reduction algorithm in actuality show the trade-offs between cost and performance. Although the recursive doubling algorithm only requires $\log N$ steps, it requires in total $N(\log N) - N + 1$ additions. On the other hand, the cyclic reduction algorithm requires $2(\log N) - 1$ steps; however, it only requires $2N - (\log N) - 2$ additions. In any case, both algorithms can reduce the parallel addition steps for solving $L_N v$ from $O(N)$ down to $O(\log N)$. The proofs of the correctness of these two parallel algorithms are given in [21], in which a detailed analysis of these two parallel algorithms is also given. Table 4 shows the complexities of parallel computation steps, including multiplications, additions, shifts, and memory references, required for the DCT.

Operations	Complexities	
parallel multiplication	$\log N$,	for $N \geq 2$
parallel addition	$O(\log^2 N)$,	for $N \geq 2$
parallel shift	$\log N - 1$,	for $N \geq 4$
parallel memory reference	$\log N$,	for $N \geq 4$

Table 4: Parallel operation complexities for computing the proposed sequential DCT algorithm.

7.2 Hardware Implementation Considerations

We now consider the hardware implementations of our proposed DCT algorithm. Fig. 1 shows the general signal-flow graph for the implementation of Equation (10). The block $L_{N/2}$, which performs shifts and additions, can be implemented by either the recursive doubling algorithm as shown in Fig. 6 or the cyclic reduction algorithm as shown in Fig. 7 depending on the cost-performance trade-offs. In practice, in order to save hardware without slowing down the processing speed, we can use only one $\hat{\mathbf{C}}_{N/2}$ and $S_{N/2}$ as shown in Fig. 8 instead of two $\hat{\mathbf{C}}_{N/2}$'s and two $S_{N/2}$'s as shown in Fig. 1. Note that $S_{N/2}$ is a shuffle-exchange network of size $N/2$.

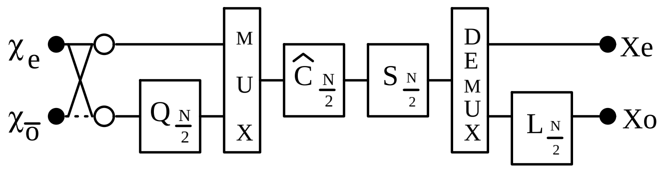

Figure 8: Proposed hardware implementation for the DCT.

In Fig. 8, the hardware is divided into four parts: one coefficient multiplier $Q_{N/2}$, one $\hat{\mathbf{C}}_{N/2}$, one shuffle-exchange network $S_{N/2}$, and one shift and adder $L_{N/2}$. A multiplexer is used to multiplex the top and the bottom halves of the signal flow. Input data are read from the host computer and are executed in a pipelined fashion. The top half of the signals will pass through the $\hat{\mathbf{C}}_{N/2}$ and the $S_{N/2}$ functional units first; while the bottom half will first perform coefficient multiplications by a coefficient multiplier $Q_{N/2}$, then pass through the $\hat{\mathbf{C}}_{N/2}$ and the $S_{N/2}$ functional units, and finally pass through the shift and adder processor $L_{N/2}$. It can be verified that succeeding sets of input data can enter the hardware and can be executed in a pipelined fashion [21]. In addition, the functional units $\hat{\mathbf{C}}_{N/2}$ and $S_{N/2}$ will be at full utilization.

It is worthwhile illustrating that our hardware implementation may be more attractive than the implementation proposed by Hou [17]. In addition, we propose fast hardware implementation for computing $L_{N/2}\, v_4$, and we require fewer hardware components than Hou. In our implementation, we only require one shuffle-exchange network, rather than two bit-reversed shuffling networks proposed by Hou.

8 Conclusions

A systematic method for designing the DCT and the DST recursive algorithms has been presented in this paper. The method is based on certain recursive properties of the DCT coefficient matrix that are proved in the paper. This method can be generalized to design a 2-D DCT and a 2-D DST recursive algorithms. Since the DCT and the DST are orthonormal, the inverse DCT transform and the inverse DST transform also can be realized by taking the transpose of the forward transforms.

Detailed analysis of our recursive algorithms can be seen in [21]. For the sequential algorithms, $(1/2)N\log N$ multiplications and $(3/2)N\log N - N + 1$ additions are required for computing an N-point DCT or an N-point DST; $(3/4)N^2\log N - (1/4)N^2$ multiplications and $3N^2\log N - 2N^2 + 2N$ additions are required for computing an $(N\times N)$-point DCT or an $(N\times N)$-point DST.

We also consider the parallel computation. We propose two parallel algorithms: one is based on the recursive doubling technique; the other is based on the cyclic reduction technique. These two algorithms, showing the trade-offs between cost and performance, can reduce the parallel computation steps from linear order down to logarithmic order. Suppose that we have sufficient processing elements for the parallel computation. We require $O(\log N)$ parallel multiplication steps, $O(\log^2 N)$ parallel addition steps, $O(\log N)$ parallel shift steps, and $O(\log N)$ parallel memory reference steps, for computing the 1-D DCT or DST. The parallel operation complexities for the 2-D DCT are the same as that of the 1-D DCT. Because of the space limitation, the parallel implementations and hardware considerations for the 2-D DCT are omitted in this presentation, and interested readers can see this elsewhere [21].

Acknowledgements

The authors would like to thank Professor H. V. Sorensen and the anonymous referees for their valuable comments on this paper.

References

[1] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete Cosine transform. *IEEE Trans. Comput.*, C-23(1):90–93, January 1974.

[2] S. C. Chan and K. L. Ho. A new two-dimensional fast Cosine transform algorithm. *IEEE Trans. Signal Processing*, 39(2):481–485, February 1991.

[3] W. H. Chen, C. H. Smith, and S. C. Fralick. A fast computational algorithm for the discrete Cosine transform. *IEEE Trans. Commun.*, COM-25(9):1004–1009, September 1977.

[4] S. Cheng. Application of the Sine-transform method in time-of-flight positron-emission image reconstruction algorithms. *IEEE Transactions on Biome. Eng.*, BME-32:185–192, March 1985.

[5] N. I. Cho and S. U. Lee. Fast algorithm and implementation of 2-D discrete Cosine transform. *IEEE Trans. Circuits and Syst.*, 38(3):297–305, March 1991.

[6] 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.*, 40(4):259–266, April 1993.

[7] Z. Cvetković and M. V. Popović. New fast recursive algorithms for the computation of discrete Cosine and Sine transforms. *IEEE Trans. Signal Processing*, 40(8):2083–2086, August 1992.

[8] O. Ersoy and N. C. Hu. A unified approach to the fast computation of all discrete trigonometric transforms. In *Proc. of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pages 1843–1846. IEEE, 1987.

[9] E. Feig and S. Winograd. Fast algorithms for the discrete Cosine transform. *IEEE Trans. Signal Processing*, 40(9):2174–2193, September 1992.

[10] A. Gupta and K. R. Rao. A fast recursive algorithm for the discrete Sine transform. *IEEE Trans. Acoust., Speech, and Signal Processing*, 38(3):553–557, March 1990.

[11] A. Gupta and K. R. Rao. An efficient FFT algorithm based on the discrete Sine transform. *IEEE Trans. Signal Processing*, 39(2):486–490, February 1991.

[12] M. A. Haque. A two-dimensional fast Cosine transform. *IEEE Trans. Acoust., Speech, and Signal Processing*, ASSP-33(6):1532–1539, December 1985.

[13] R. M. Haralick. A storage efficient way to implement the discrete Cosine transform. *IEEE Trans. Comput.*, C-25(7):764–765, July 1976.

[14] M. T. Heideman. Computation of an odd-length DCT from a real-valued DFT of the same length. *IEEE Trans. Signal Processing*, 40(1):54–61, January 1992.

[15] R. W. Hockney. A fast direct solution of Poisson's equation using Fourier analysis. *JACM*, 12(1):95–113, January 1965.

[16] H. S. Hou. The fast Hartley transform algorithm. *IEEE Trans. Comput.*, C-36(2):147–156, February 1987.

[17] H. S. Hou. A fast recursive algorithm for computing the discrete Cosine transform. *IEEE Trans. Acoust., Speech, and Signal Processing*, ASSP-35(10):1455–1461, October 1987.

[18] F. Y. Huang and P.-Z. Lee. Design of algorithms for the discrete Sine transform. In *Proc. of International Computer Symposium*, pages 733–742, Taichung, Taiwan, R.O.C., December 1992.

[19] F. A. Kamangar and K. R. Rao. Fast algorithms for the 2-D discrete Cosine transform. *IEEE Trans. Comput.*, C-31(9):899–906, September 1982.

[20] B. G. Lee. A new algorithm to compute the discrete Cosine transform. *IEEE Trans. Acoust., Speech, and Signal Processing*, ASSP-32(6):1243–1245, December 1984.

[21] 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, 1993. An extended-abstract version is in *Proc. of Fifth Digital Signal Processing Workshop*, pages 3.9.1–3.9.2, Starved Rock State Park, IL., September, 1992.

[22] W. Li. A new algorithm to compute the DCT and its inverse. *IEEE Trans. Signal Processing*, 39(6):1305–1313, June 1991.

[23] J. Makhoul. A fast Cosine transform in one and two dimensions. *IEEE Trans. Acoust., Speech, and Signal Processing*, ASSP-28(1):27–34, February 1980.

[24] H. S. Malvar. Fast computation of discrete Cosine transform through fast Hartley transform. *Electronics Letters*, 22(7):352–353, March 1986.

[25] H. S. Malvar. Corrections to 'fast computation of the discrete Cosine transform and the discrete Hartley transform'. *IEEE Trans. Acoust., Speech, and Signal Processing*, 36(4):610–612, April 1988.

[26] M. J. Narasimha and A. M. Peterson. On the computation of the discrete Cosine transform. *IEEE Trans. Commun.*, COM-26:934–936, June 1978.

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

[28] Z. Wang. A fast algorithm for the discrete Sine transform implemented by the fast Cosine transform. *IEEE Trans. Acoust., Speech, and Signal Processing*, ASSP-30:814–815, 1982.

[29] Z. Wang. Fast algorithms for the discrete W transform and for the discrete Fourier transform. *IEEE Trans. Acoust., Speech, and Signal Processing*, ASSP-32(4):803–816, August 1984.

[30] Z. Wang. On computing the discrete Fourier and Cosine transforms. *IEEE Trans. Acoust., Speech, and Signal Processing*, 33(4):1341–1344, October 1985.

[31] P. Yip and K. R. Rao. A fast computational algorithm for the discrete Sine transform. *IEEE Trans. Commun.*, COM-28:304–307, February 1980.

[32] P. Yip and K. R. Rao. Fast decimation-in-time algorithms for a family of discrete Sine and Cosine transforms. *Circuits Systems Signal Process*, 3:387–408, 1984.

[33] P. Yip and K. R. Rao. The decimation-in-frequency algorithms for a family of discrete Sine and Cosine transforms. *Circuits Systems Signal Process*, 7(1):3–19, 1988.