TR-93-001

New Recursive Algorithms for Computing the 1-D and 2-D Discrete Cosine Transforms

PeiZong Lee and Fang-Yu Huang

New Recursive Algorithms for Computing the 1-D and 2-D Discrete Cosine Transforms*

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.

January 16, 1993

Abstract

It has been observed by many researchers that the discrete cosine transform (DCT) has wide applications in speech and image processing, as well as telecommunication signal processing for the purpose of data compression, feature extraction, and filtering. In this paper, we present a new method to design fast recursive algorithms for the DCT. 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.

It is worthwhile to show that the recursive algorithms we design are able to decompose the DCT into two balanced lower-order subproblems in comparison to previous research works. Therefore, our algorithms are especially suitable for the parallel computation. We have proposed two parallel algorithms, which can reduce the parallel computation steps from linear order down to logarithmic order. In addition, when converting our algorithms into hardware implementations, we require fewer hardware components than other DCT algorithms.

*** A preliminary version *A New Method to Design Recursive Algorithms for Computing the 1-D and 2-D Discrete Cosine Transforms* is presented in the **Fifth Digital Signal Processing Workshop**, Starved Rock State Park, IL., September, 1992.

*This work was partially supported by the NSC under Grants NSC 81-0408-E-001-505.
†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). Ahmed and Rao [2], Elliott and Rao [10], Jain [18], Rao and Yip [32], and other 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 [2] [11] [18] [32]. For the applications of the DCT in image processing, speech processing, and filtering, consult [3] [8] [14] [21] [22] [31] [33] [34].

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] [26] [30] [37] [39] or the fast Hartley transforms [16] [27] [28], or convert the DCT's into circular convolutions which can be computed very efficiently using distributed arithmetics [25]. On the other hand, direct computation algorithms use techniques such as matrix factorizations [7] [35] [40] [41] [42] [43], divide and conquer method [23], recursive decomposition [17], prime-factor decomposition [24], and small odd-length DCT modules which are derived from Winograd's small modules of real-valued discrete Fourier transforms (DFT's) [15] [44].

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 give a clear procedure to describe their computation.

Hardware implementations also have been proposed. Jalali and Rao [19] designed a processor, which was based on a matrix factorization algorithm developed by Chen, Smith, and Fralick [7]. Vetterli and Ligtenberg [38] designed a Fourier-Cosine transform chip based on Vetterli and Nussbaumer's algorithm [39], which could compute an N-point DCT by an N-point DFT and several rotations. Cho and Lee [9] proposed two different implementations for computing the DCT on

the existing VLSI DFT architectures. Chakrabarti and JaJa [4] designed a DCT systolic architecture, which was based on their discrete Hartley transform systolic architecture. Chang and Wu [6] proposed a systolic array, which could recursively generate coefficients of the DCT matrix; the systolic array performed operations similar to implement a matrix-vector multiplication. Sun, Wu, and Liou [36] used bit-serial and bit-parallel data structures to implement vector inner products; their architecture did not require multipliers, instead, a large table was required to store cosine coefficients in binary form. In general, the requirements for the hardware implementations they emphasized included regular functional units, regular data flows, minimum number of multipliers, and fast throughput. However, these requirements could not be satisfied at the same time.

2-D DCT algorithms also have been studied. 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 [26], by applying the similar idea in Haralic [13], derived a 1-D DCT algorithm which was essentially identical to that of Narasimha and Peterson [30]; 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.

Besides, there also existed algorithms working directly on the 2-D input data set and not separately on rows and columns. Kamangar and Rao [20], 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 $(\frac{N_1}{2} \times \frac{N_2}{2})$-point subproblems. Each of the subproblems was a linear combination of four $(\frac{N_1}{2} \times \frac{N_2}{2})$-point DCT 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 [23]; however, both of them suffered from the same problem of numerical instabilities because of the round-off error. Based on Hou's 1-D algorithm [17], Chan and Ho [5] proposed an 2-D algorithm, in which the numbers of multiplications and

additions were the same as that of Haque's algorithm; however, unlike Haque's algorithm, their algorithm was numerically stable.

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 operations in the algorithms, for instance, the memory reference operations and the shift operations. In effect, none of the above algorithm designers have considered the complexities of the effort for arranging data in appropriate memory locations; in addition, none of them have considered the complexities of doing shift operations. Although the memory reference operations or the shift operations may be unimportant in conventional computers, they must be considered if we want to design hardware implementations.

There are other important issues for designing a good DCT 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, 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 recursive algorithm. Moreover, these algorithms should be easily converted into hardware implementations.

Although Hou [17] also designed a 1-D DCT recursive algorithm previously, his method was quite different from ours. Based on the work of Narasimha and Peterson [30], 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. Unlike his method, 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 properties of the DCT coefficient matrix. In our 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 [30] and used by Hou. However, our intermediate steps for arranging data into appropriate memory locations require shuffle-exchanges, rather than Hou's bit-reversed shuffling. Finally, our algorithm is more balanced than Hou's algorithm when decomposing the DCT into several functional submodules. Therefore, our algorithm is more attractive than his algorithm when performing a parallel computation. In effect, when comparing the hardware implementations, our algorithm requires fewer hardware components than that of Hou's implementation. In addition, the parallel computation time of our implementation is also less than that of his implementation.

Detailed analysis of our algorithms' complexities is also provided in this paper. The complexity criteria discussed in this paper include the numbers of multiplications, additions, shifts, and memory references. For the 1-D DCT implementation, the number of multiplications and the number of additions of the proposed algorithm are the same as that of Hou's algorithm. However, for the 2-D DCT implementation, the number of multiplications required in our algorithm is less than that of Chan and Ho's implementation [5], which is based on Hou's 1-D DCT algorithm; while the number of additions required in our algorithm is the same as that of their algorithm. Detailed comparisons of the numbers of operations required are given in Section 6 and Section 8.

We also propose parallel algorithms and hardware implementations for the DCT. In order to reduce the parallel computation steps, we introduce two parallel algorithms: one is based on the recursive doubling technique, the other is based on the cyclic reduction technique. These two algorithms, which show the trade-offs between cost and performance, can reduce the parallel computation steps from linear order down to logarithmic order. In addition, they can be easily converted into hardware implementations.

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 algorithm, and we give a formal procedure to describe the algorithm. In Section 4, analysis of the DCT algorithm is given. In Section 5, we propose parallel algorithms and hardware implementations of the DCT. In Section 6, we generalize our method to derive a 2-D DCT recursive algorithm. Finally, some

concluding remarks are given in Section 7.

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

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

where

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

Equation (1) can be written in a matrix-vector multiplication form $\mathbf{X} = \check{\mathbf{C}}_N\,\mathbf{x}$, 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 $\check{\mathbf{C}}_N = [\check{C}_N(k,n)]$, $0 \le k,\ n \le N-1$, which is defined by

$$\check{C}_N(k,n) = \begin{cases} \sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}} \cos(\frac{\pi(2n+1)k}{2N}), & \text{for } k = 0,\ 0 \le n \le N-1; \\ \sqrt{\frac{2}{N}} \cos(\frac{\pi(2n+1)k}{2N}), & \text{for } 1 \le k \le N-1,\ 0 \le n \le N-1. \end{cases} \tag{3}$$

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 will remove $\epsilon(k)$ and the normalization factor $\sqrt{\frac{2}{N}}$ for every entry $\check{C}_N(k,n)$ defined in Equation (3), since they can be done in a separate step. Therefore, from now on, we deal with a simplified DCT version $\mathbf{X} = \mathbf{C}_N\,\mathbf{x}$, where

$$C_N(k,n) = \cos(\frac{\pi(2n+1)k}{2N}), \quad \text{for } 0 \le k,\ n \le N-1. \tag{4}$$

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.

Lemma 1 : *For* $0 \le n \le N-1$,

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

Proof: From Equation (4),

$$\begin{aligned} C_N(k, N-1-n) &= \cos(\frac{\pi(2(N-1-n)+1)k}{2N}) \\ &= \cos(k\pi - \frac{\pi(2n+1)k}{2N}) \\ &= \begin{cases} C_N(k,n)\,, & \text{if } k \text{ is even;} \\ -C_N(k,n)\,, & \text{if } k \text{ is odd.} \end{cases} \end{aligned}$$

□

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 $\hat{\mathbf{C}}_N$. Lemma 2 can be obtained immediately from the definitions of $\bar{\mathbf{C}}_N$ and $\hat{\mathbf{C}}_N$.

Lemma 2 :

1. *For $0 \le n \le \frac{N}{2} - 1$,* $\bar{C}_N(k,n) = C_N(k,2n)$, *and* $\bar{C}_N(k, n+\frac{N}{2}) = C_N(k, N-1-2n)$.

2. *For $0 \le k \le \frac{N}{2} - 1$,* $\hat{C}_N(k,n) = \bar{C}_N(2k,n)$, *and* $\hat{C}_N(k+\frac{N}{2}, n) = \bar{C}_N(2k+1,n)$.

□

Lemma 3 : *For* $0 \le k \le \frac{N}{2} - 1$ *and* $0 \le n \le \frac{N}{2} - 1$,

$$\hat{C}_N(k,n) = \hat{C}_N(k, n+\tfrac{N}{2}), \quad \textit{and}$$
$$\hat{C}_N(k+\tfrac{N}{2}, n) = -\hat{C}_N(k+\tfrac{N}{2}, n+\tfrac{N}{2}).$$

Proof:

For $0 \le k \le \frac{N}{2} - 1$ and $0 \le n \le \frac{N}{2} - 1$,

$$\begin{aligned}
\hat{C}_N(k, n+\tfrac{N}{2}) &= \bar{C}_N(2k, n+\tfrac{N}{2}) && \text{(from Lemma 2)}\\
&= C_N(2k, N-1-2n) && \text{(from Lemma 2)}\\
&= C_N(2k, 2n) && \text{(from Lemma 1)}\\
&= \bar{C}_N(2k, n) && \text{(from Lemma 2)}\\
&= \hat{C}_N(k, n) && \text{(from Lemma 2)}
\end{aligned}$$

$$\begin{aligned}
\hat{C}_N(k+\tfrac{N}{2}, n+\tfrac{N}{2}) &= \bar{C}_N(2k+1, n+\tfrac{N}{2}) && \text{(from Lemma 2)}\\
&= C_N(2k+1, N-1-2n) && \text{(from Lemma 2)}\\
&= -C_N(2k+1, 2n) && \text{(from Lemma 1)}\\
&= -\bar{C}_N(2k+1, n) && \text{(from Lemma 2)}\\
&= -\hat{C}_N(k+\tfrac{N}{2}, n) && \text{(from Lemma 2)}
\end{aligned}$$

□

We now consider the $\frac{N}{2} \times \frac{N}{2}$ upper-left corner submatrix of $\hat{\mathbf{C}}_N$, and we find that it is equal to $\bar{\mathbf{C}}_{\frac{N}{2}}$.

Lemma 4 :

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

Proof: We first prove that $\hat{C}_N(k,n) = C_N(2k,2n)$. Then we prove that $\bar{C}_{\frac{N}{2}}(k,n) = C_N(2k,2n)$. For $0 \le k,\ n \le \frac{N}{2} - 1$, we have

$$\begin{aligned}
\hat{C}_N(k,n) &= \bar{C}_N(2k,n) && \text{(from Lemma 2)}\\
&= C_N(2k,2n) && \text{(from Lemma 2).}
\end{aligned}$$

On the other hand, for $0 \le k \le \frac{N}{2} - 1$ and $0 \le n \le \frac{N}{4} - 1$,

$$\begin{aligned}
\bar{C}_{\frac{N}{2}}(k,n) &= C_{\frac{N}{2}}(k,2n) && \text{(from Lemma 2)}\\
&= \cos(\tfrac{\pi(4n+1)k}{2(\frac{N}{2})}) && \text{(from Equation (4))}\\
&= C_N(2k,2n).
\end{aligned}$$

And for $0 \le k \le \frac{N}{2} - 1$ and $\frac{N}{4} \le n \le \frac{N}{2} - 1$,

$$\begin{aligned}
\bar{C}_{\frac{N}{2}}(k,n) &= \bar{C}_{\frac{N}{2}}(k,(n-\tfrac{N}{4})+\tfrac{N}{4}) \\
&= C_{\frac{N}{2}}(k,\tfrac{N}{2}-1-2(n-\tfrac{N}{4})) \qquad \text{(from Lemma 2)} \\
&= C_{\frac{N}{2}}(k,N-1-2n) \\
&= \cos(\tfrac{\pi(2N-4n-2+1)k}{2(\frac{N}{2})}) \qquad \text{(from Equation (4))} \\
&= \cos(\tfrac{\pi(4n+1)k}{2(\frac{N}{2})}) \\
&= C_N(2k,2n). \qquad \square
\end{aligned}$$

Furthermore, if we denote the submatrix

$$[\hat{C}_N(k+\frac{N}{2},n)]\ \begin{matrix} 0 \le k \le \frac{N}{2}-1 \\ 0 \le n \le \frac{N}{2}-1 \end{matrix}$$

as $D_{\frac{N}{2}}$, then from Lemma 3 and Lemma 4 we obtain the following theorem.

Theorem 5 :

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

In Theorem 5, $\bar{\mathbf{C}}_{\frac{N}{2}}$ can be arranged to get $\hat{\mathbf{C}}_{\frac{N}{2}}$. However, in order to get a recursive algorithm, $D_{\frac{N}{2}}$ must be converted into a certain form related to either $\bar{\mathbf{C}}_{\frac{N}{2}}$ or $\hat{\mathbf{C}}_{\frac{N}{2}}$. In the following lemma, we show that $D_{\frac{N}{2}}$ can be decomposed into the product of three matrices, including one regular lower triangular matrix $L_{\frac{N}{2}}$, one $\bar{\mathbf{C}}_{\frac{N}{2}}$, and one diagonal matrix $Q_{\frac{N}{2}}$.

Lemma 6 :

$$D_{\frac{N}{2}} = L_{\frac{N}{2}}\, \bar{\mathbf{C}}_{\frac{N}{2}}\, Q_{\frac{N}{2}}$$

where L is a lower triangular matrix defined by

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

and

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

$\theta_n = \frac{(4n+1)\pi}{2N}$, *for* $0 \le n \le \frac{N}{2} - 1$.

Proof: It is sufficient to prove the following general form:

For some fixed n, we have

$$D_{\frac{N}{2}}(k,n) = (L_{\frac{N}{2}}(k,0), L_{\frac{N}{2}}(k,1), \ldots, L_{\frac{N}{2}}(k,k))(\bar{C}_{\frac{N}{2}}(0,n), \bar{C}_{\frac{N}{2}}(1,n), \ldots, \bar{C}_{\frac{N}{2}}(k,n))^T \cos(\theta_n),$$

for $k = 0, 1, \ldots, \frac{N}{2} - 1$,

where

$$L_{\frac{N}{2}}(k,i) = \begin{cases} (-1)^{(k-i)}, & \text{if } i = 0; \\ (-1)^{(k-i)}2, & \text{if } 1 \le i \le k. \end{cases}$$

First, we can write the values of both $\bar{C}_{\frac{N}{2}}(k,n)$ and $D_{\frac{N}{2}}(k,n)$ in terms of θ_n. For $0 \le k,\ n \le \frac{N}{2} - 1$,

$$\begin{aligned} \bar{C}_{\frac{N}{2}}(k,n) &= \hat{C}_N(k,n) && \text{(from Theorem 5)} \\ &= \bar{C}_N(2k,n) && \text{(from Lemma 2)} \\ &= C_N(2k,2n) && \text{(from Lemma 2)} \\ &= \cos(\tfrac{\pi(4n+1)2k}{2N}) && \text{(from Equation (4))} \\ &= \cos(2k\theta_n). \end{aligned}$$

$$\begin{aligned} D_{\frac{N}{2}}(k,n) &= \hat{\mathbf{C}}_N(k+\tfrac{N}{2},n) && \text{(from Theorem 5)} \\ &= \bar{C}_N(2k+1,n) && \text{(from Lemma 2)} \\ &= C_N(2k+1,2n) && \text{(from Lemma 2)} \\ &= \cos(\tfrac{\pi(4n+1)(2k+1)}{2N}) && \text{(from Equation (4))} \\ &= \cos((2k+1)\theta_n). \end{aligned}$$

Having the above two relations, we can begin to prove the validity of the general form by induction on k.

1. For $k \stackrel{?}{=} 0$, since $L_{\frac{N}{2}}(0,0) = \bar{C}_{\frac{N}{2}}(0,n) = 1$, we have

$$D_{\frac{N}{2}}(0,n) = \cos(\theta_n) = L_{\frac{N}{2}}(0,0)\,\bar{C}_{\frac{N}{2}}(0,n)\,\cos(\theta_n).$$

2. Suppose that it is true for $k = m$. That is,

$$\begin{aligned} D_{\frac{N}{2}}(m,n) &= (L_{\frac{N}{2}}(m,0), L_{\frac{N}{2}}(m,1), \ldots, L_{\frac{N}{2}}(m,m))(\bar{C}_{\frac{N}{2}}(0,n), \bar{C}_{\frac{N}{2}}(1,n), \ldots, \bar{C}_{\frac{N}{2}}(m,n))^T \cos(\theta_n) \\ &= [(-1)^m \bar{C}_{\frac{N}{2}}(0,n) + (-1)^{m-1} 2\,\bar{C}_{\frac{N}{2}}(1,n) + \cdots + (-1)^{m-m} 2\,\bar{C}_{\frac{N}{2}}(m,n)]\,\cos(\theta_n). \end{aligned}$$

3. When $k = m + 1$,

$$D_{\frac{N}{2}}(m+1,n) = \cos((2m+3)\theta_n) = 2\,\cos(2(m+1)\theta_n)\,\cos(\theta_n) - \cos((2m+1)\theta_n).$$

In the above equation, we have utilized the trigonometric identity:

$$\cos A = 2\cos(\frac{A+B}{2})\cos(\frac{A-B}{2}) - \cos B,$$

where $A = 2m+3$ and $B = 2m+1$.

Therefore,

$$\begin{aligned}
D_{\frac{N}{2}}(m+1,n) &= 2\,\bar{C}_{\frac{N}{2}}(m+1,n)\cos(\theta_n) + (-1)\,D_{\frac{N}{2}}(m,n) \\
&= (-1)\,[(-1)^m\,\bar{C}_{\frac{N}{2}}(0,n) + (-1)^{m-1}\,2\,\bar{C}_{\frac{N}{2}}(1,n) + \cdots + (-1)^{m-m}\,2\,\bar{C}_{\frac{N}{2}}(m,n)]\cos(\theta_n) \\
&\qquad + 2\,\bar{C}_{\frac{N}{2}}(m+1,n)\cos(\theta_n) \\
&= [(-1)^{m+1}\,\bar{C}_{\frac{N}{2}}(0,n) + (-1)^{(m+1)-1}\,2\,\bar{C}_{\frac{N}{2}}(1,n) + \cdots + (-1)^{(m+1)-m}\,2\,\bar{C}_{\frac{N}{2}}(m,n) \\
&\qquad + (-1)^{(m+1)-(m+1)}\,2\,\bar{C}_{\frac{N}{2}}(m+1,n)]\cos(\theta_n) \\
&= (L_{\frac{N}{2}}(m+1,0), L_{\frac{N}{2}}(m+1,1), \ldots, L_{\frac{N}{2}}(m+1,m), L_{\frac{N}{2}}(m+1,m+1)) \\
&\qquad (\bar{C}_{\frac{N}{2}}(0,n), \bar{C}_{\frac{N}{2}}(1,n), \ldots, \bar{C}_{\frac{N}{2}}(m,n), \bar{C}_{\frac{N}{2}}(m+1,n))^T\cos(\theta_n).
\end{aligned}$$

The proof is completed. □

The following theorem shows the recursive structure of the coefficient matrix, and we will use it to design our DCT recursive algorithms.

Theorem 7 :

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

Proof: Immediately follows from Theorem 5 and Lemma 6. □

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 (4). First, apply the column permutation P_N and the row permutation S_N^T to the matrix $\mathbf{C}_N$. Because P_N is orthonomal,

$P_N^T = P_N^{-1}$. We have

$$\begin{bmatrix} X_e \\ X_o \end{bmatrix} = S_N^T \mathbf{X} = S_N^T \mathbf{C}_N \mathbf{x} = S_N^T \mathbf{C}_N \underline{P_N P_N^T} \mathbf{x} = \hat{\mathbf{C}}_N P_N^T \mathbf{x} = \hat{\mathbf{C}}_N \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix} \tag{5}$$

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 7 and Equation (5) we have

$$\begin{bmatrix} X_e \\ X_o \end{bmatrix} = \begin{bmatrix} S_{\frac{N}{2}} \hat{\mathbf{C}}_{\frac{N}{2}} & S_{\frac{N}{2}} \hat{\mathbf{C}}_{\frac{N}{2}} \\ L_{\frac{N}{2}} S_{\frac{N}{2}} \hat{\mathbf{C}}_{\frac{N}{2}} Q_{\frac{N}{2}} & -L_{\frac{N}{2}} S_{\frac{N}{2}} \hat{\mathbf{C}}_{\frac{N}{2}} Q_{\frac{N}{2}} \end{bmatrix} \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix}. \tag{6}$$

From Equation (6), X_e and X_o can be easily derived.

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

In order to compute the simplified DCT version $\mathbf{X} = \mathbf{C}_N \mathbf{x}$, two more multiplications are needed. First, before computing Equation (6), we compute $P_N^T \mathbf{x}$, which is $\begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix}$. Second, because $\begin{bmatrix} X_e \\ X_o \end{bmatrix}$ $= S_N^T \mathbf{X}$, therefore, after computing Equation (6), we compute $S_N \begin{bmatrix} X_e \\ X_o \end{bmatrix}$, which is $\mathbf{X}$. In the following, we describe the procedure for computing the DCT.

DCT recursive algorithm:

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

Step 2: /* Recursively compute $\begin{bmatrix} X_e \\ X_o \end{bmatrix} = \hat{\mathbf{C}}_N \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix}$, where
$X_e = S_{\frac{N}{2}} \hat{\mathbf{C}}_{\frac{N}{2}} (x_e + x_{\bar{o}})$ and $X_o = L_{\frac{N}{2}} S_{\frac{N}{2}} \hat{\mathbf{C}}_{\frac{N}{2}} Q_{\frac{N}{2}} (x_e - x_{\bar{o}})$. */

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

2.2 else perform Steps from 2.2.1 to 2.2.8:

2.2.1 compute $(x_e + x_{\bar{o}})$, and let the result be u_1;

2.2.2 solve $\hat{\mathbf{C}}_{\frac{N}{2}} u_1$ recursively, and let the result be u_2;

2.2.3 compute $S_{\frac{N}{2}} u_2$, then the the result is X_e;

2.2.4 compute $(x_e - x_{\bar{o}})$, and let the result be v_1;

2.2.5 compute $Q_{\frac{N}{2}}\, v_1$, and let the result be v_2;

2.2.6 solve $\hat{\mathbf{C}}_{\frac{N}{2}}\, v_2$ recursively, and let the result be v_3;

2.2.7 compute $S_{\frac{N}{2}}\, v_3$, and let the result be v_4;

2.2.8 compute $L_{\frac{N}{2}}\, v_4$, then the result is X_o;

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

In the DCT recursive algorithm, Step 2, which implements Equation (6), is a recursive procedure. In general, an N-point DCT problem will be decomposed recursively until $N = 2$. Note that, the 2-point DCT is regarded as a basic computation block and will not be decomposed further. Step 2.1 tests the condition whether $N = 2$; if $N = 2$, then computes the 2-point DCT problem directly. Step 2.2 first decomposes the problem into two lower-order subproblems, and then solves them independently. Fig. 1 shows the general signal-flow graph for the implementation of Step 2.2.

In the following, we illustrate our DCT recursive algorithm with three lower order examples.

For $N = 2$,

$$\mathbf{X} = \mathbf{C}_2\mathbf{x}$$

which implies

$$\begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} (x_0 + x_1) \\ \cos(\frac{\pi}{4})(x_0 - x_1) \end{bmatrix}.$$

In this case, two additions (or subtractions) and one multiplication are required; see Fig. 2-(a) for the signal-flow graph for $N = 2$.

For $N = 4$,

$$\mathbf{X} = \mathbf{C}_4\mathbf{x}.$$

From Equation (6),

$$\begin{bmatrix} X_e \\ X_o \end{bmatrix} = \begin{bmatrix} S_2\,\hat{\mathbf{C}}_2 & S_2\,\hat{\mathbf{C}}_2 \\ L_2\,S_2\,\hat{\mathbf{C}}_2\,Q_2 & -L_2\,S_2\,\hat{\mathbf{C}}_2\,Q_2 \end{bmatrix} \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix} = \begin{bmatrix} S_2\,\hat{\mathbf{C}}_2\,(x_e + x_{\bar{o}}) \\ L_2\,S_2\,\hat{\mathbf{C}}_2\,Q_2\,(x_e - x_{\bar{o}}) \end{bmatrix}.$$

Since $\hat{\mathbf{C}}_2 = \mathbf{C}_2$ and $S_2 = I_2$, we have

$$\begin{bmatrix} X_e \\ X_o \end{bmatrix} = \begin{bmatrix} \mathbf{C}_2\,(x_e + x_{\bar{o}}) \\ L_2\,\mathbf{C}_2\,Q_2\,(x_e - x_{\bar{o}}) \end{bmatrix}. \tag{7}$$

In this case, first, four additions are required for computing $u_1 = (x_e + x_{\bar{o}})$ and $v_1 = (x_e - x_{\bar{o}})$. Second, two multiplications are required for computing $v_2 = Q_2\,v_1$. Third, for computing either $X_e = \mathbf{C}_2\,u_1$ or $v_3 = \mathbf{C}_2\,v_2$, each one requires one multiplication and two additions, as was in the case when $N = 2$. And fourth, one shift and one addition are required for computing $X_o = L_2\,v_3$. Therefore, we require four multiplications, nine additions, and one shift for computing Equation (7); see Fig. 2-(b) for the signal-flow graph for $N = 4$.

For $N = 8$,

$$\mathbf{X} = \mathbf{C}_8\,\mathbf{x}.$$

From Equation (6),

$$\begin{bmatrix} X_e \\ X_o \end{bmatrix} = \begin{bmatrix} S_4\,\hat{\mathbf{C}}_4 & S_4\,\hat{\mathbf{C}}_4 \\ L_4\,S_4\,\hat{\mathbf{C}}_4\,Q_4 & -L_4\,S_4\,\hat{\mathbf{C}}_4\,Q_4 \end{bmatrix} \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix} = \begin{bmatrix} S_4\,\hat{\mathbf{C}}_4\,(x_e + x_{\bar{o}}) \\ L_4\,S_4\,\hat{\mathbf{C}}_4\,Q_4\,(x_e - x_{\bar{o}}) \end{bmatrix}. \tag{8}$$

In this case, first, eight additions are required for computing $u_1 = (x_e + x_{\bar{o}})$ and $v_1 = (x_e - x_{\bar{o}})$. Second, four multiplications are required for computing $v_2 = Q_4\,v_1$. Third, for computing either $u_2 = \hat{\mathbf{C}}_4\,u_1$ or $v_3 = \hat{\mathbf{C}}_4\,v_2$, each involves four multiplications, nine additions, and one shift, as was in the case when $N = 4$. Fourth, eight memory references are required for computing $X_e = S_4\,u_2$ and $v_4 = S_4\,v_3$. And Fifth, three shifts and three additions are required for computing $X_o = L_4\,v_4$. Therefore, we require twelve multiplications, twenty-nine additions, five shifts, and eight memory references for computing Equation (8); see Fig. 2-(c) for the signal-flow graph for $N = 8$.

4 Analysis of the Algorithm

In this section, we will analyze the complexities of the DCT algorithm. The DCT algorithm contains four steps. Step 1 and Step 3 are concerned with data arrangement, and each can be done with N memory references. Step 4 deals with multiplying constant factors to each of the data entries, which can be done with N multiplications.

We now consider the complexity of Step 2. Step 2.1 deals with the basic 2-point DCT case. As shown in Section 4, the basic 2-point DCT problem can be solved by using one multiplication and two additions. Step 2.2 includes eight substeps:

- Steps 2.2.1 and 2.2.4 are concerned with two vector additions (or subtractions), which can be done with N scalar additions (or subtractions).
- Steps 2.2.2 and 2.2.6 are concerned with two lower-order subproblems, which can be solved recursively.
- Steps 2.2.3 and 2.2.7 deal with data arrangement, which can be done with N memory references.
- Step 2.2.5 deals with a matrix vector multiplication $Q_{\frac{N}{2}} v_1$. Because $Q_{\frac{N}{2}}$ is a diagonal matrix, this step can be done by using $\frac{N}{2}$ multiplications.
- Step 2.2.8 also deals with a matrix vector multiplication $L_{\frac{N}{2}} v_4$. Because of the regularity of the matrix $L_{\frac{N}{2}}$, this step can be done with $\frac{N}{2} - 1$ shifts and $\frac{N}{2} - 1$ additions. This is because the operation of multiplying a number by a constant factor 2 can be done by using a shift operation in the computer.

Theorem 8 : *Let $v = (v_0, v_1, \ldots, v_{\frac{N}{2}-1})^T$, which is a vector of length $\frac{N}{2}$. Then, $y = L_{\frac{N}{2}} v$ can be computed with $\frac{N}{2} - 1$ shift operations and $\frac{N}{2} - 1$ additions.*

Proof: Because of the regularity of the matrix $L_{\frac{N}{2}}$, the resulting vector y can be computed in the following way:

$$\begin{aligned} y_0 &= v_0; \\ y_i &= 2v_i - y_{i-1}, \text{ for } 1 \leq i \leq \tfrac{N}{2} - 1. \end{aligned}$$

Therefore, $\frac{N}{2} - 1$ shift operations and $\frac{N}{2} - 1$ additions (subtractions) are enough for computing $y = L_{\frac{N}{2}} v$. □

We now analyze the exact numbers of operations, including multiplications, additions, shifts, and memory references, required for computing Step 2.

1. Multiplications.

   As shown in Section 4, when $N = 2$, one multiplication is required. From Steps 2.1, 2.2.2, 2.2.6, and 2.2.5, the recursive formula for the number of multiplications required is

$$\begin{aligned} M(2) &= 1, \text{ and} \\ M(N) &= 2M(\tfrac{N}{2}) + \tfrac{N}{2}, \text{ for } N > 2; \end{aligned}$$

   which implies

$$M(N) = \frac{N}{2} \log N, \text{ for } N \geq 2.$$

2. Additions.

   As shown in Section 4 that, when $N = 2$, two additions are required. From Steps 2.1, 2.2.2, 2.2.6, 2.2.1, 2.2.4, and 2.2.8, the recursive formula for the number of additions required is

$$\begin{aligned} A(2) &= 2, \text{ and} \\ A(N) &= 2A(\tfrac{N}{2}) + \tfrac{3}{2}N - 1, \text{ for } N > 2; \end{aligned}$$

   which implies

$$A(N) = \frac{3}{2} N \log N - N + 1, \text{ for } N \geq 2.$$

3. Shifts.

   As shown in Section 4 that, when $N = 4$, one shift is required. From Steps 2.2.2, 2.2.6, and 2.2.8, the recursive formula for the number of shifts required is

$$\begin{aligned} S(4) &= 1, \text{ and} \\ S(N) &= 2S(\tfrac{N}{2}) + \tfrac{N}{2} - 1, \text{ for } N > 4; \end{aligned}$$

   which implies

$$S(N) = \frac{N}{2} \log N - N + 1, \text{ for } N \geq 4.$$

4. Memory references.

   As shown in Section 4 that, when $N = 8$, eight memory references are required. From Steps 2.2.2, 2.2.6, 2.2.3, and 2.2.7, the recursive formula for the number of memory references required is

$$\begin{aligned} R(8) &= 8, \text{ and} \\ R(N) &= 2R(\tfrac{N}{2}) + N, \text{ for } N > 8; \end{aligned}$$

   which implies

$$R(N) = N \log N - 2N, \text{ for } N \geq 8.$$

	Our method				[17, 23, 35] [39, 41]		[7]		[27]		[29]		[40]	
N	×	+	S	R	×	+	×	+	×	+	×	+	×	+
2	1	2	0	0							5	9	5	9
4	4	9	1	0	4	9	6	8			13	35	13	29
8	12	29	5	8	12	29	16	26	12	31	33	95	35	83
16	32	81	17	32	32	81	44	74	34	85	81	251	91	219
32	80	209	49	96	80	209	116	194	88	211	193	615	227	547
64	192	513	129	256	192	513	292	482	218	509	449	1467	547	1315
128	448	1217	321	640	448	1217	708	1154	520	1187	1025	3399	1283	3075
256	1024	2817	769	1536	1024	2817	1668	2690	1210	2717	2305	7739	2947	7043
512	2304	6401	1793	3584	2304	6401	3844	6146	2760	6115	5121	38459	6659	15875
1024	5120	14337	4097	8192	5120	14337	8708	13826	6202	13597				

Table 1: The comparison of the numbers of operations required for computing the DCT. '×' means 'number of multiplications', '+' means 'number of additions or subtractions', 'S' means 'number of shifts', and 'R' means 'number of memory references'.

Table 1 shows the numbers of multiplications, additions, shifts, and memory references required for the DCT algorithm, when $N = 2, 4, 8, 16, 32, 64, 128, 256, 512$, and 1024. The numbers of multiplications and additions are the same as those required in the best and well-known algorithms of [17] [23] [35] [39] [41]. In addition, the number of multiplications is less than that required in [7] [27] [40]. The comparison of the numbers of operations is shown in Table 1, although the numbers of shifts and memory references are not provided in other research papers.

5 Parallel Implementations and Hardware Considerations

In previous sections, we have studied our sequential DCT algorithm. In this section, we first analyze the bottleneck in the parallel computation, then we propose parallel algorithms and hardware implementations for the DCT.

5.1 Parallel DCT Implementations

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

- Steps 2.2.1 and 2.2.4 are concerned with two vector additions, which can be done in one unit of time with two parallel additions simultaneously.
- Steps 2.2.2 and 2.2.6 are concerned with two lower-order subproblems, which can be solved independently.
- Steps 2.2.3 and 2.2.7 deal with data arrangement, which can be done in one unit of time using two shuffle-exchanges simultaneously.
- Step 2.2.5 deals with a matrix-vector multiplication $Q_{\frac{N}{2}} v_1$. Because $Q_{\frac{N}{2}}$ is a diagonal matrix, this step can be done in one parallel multiplication.
- Step 2.2.8 also deals with a matrix-vector multiplication $L_{\frac{N}{2}} v_4$. However, as shown in Theorem 8, the computation involves a first-order linear recurrence. Therefore, it also requires $\frac{N}{2} - 1$ units of time in parallel computation for shifts and additions.

It is clear that to compute $L_{\frac{N}{2}} v_4$ in Step 2.2.8 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.

From Theorem 8, $y = L_N v$ can be computed in the following way:

$$\begin{aligned} y_0 &= v_0; \\ y_i &= 2v_i - y_{i-1}, \text{ for } 1 \le i \le N-1. \end{aligned}$$

Let w be a vector of length N, and

$$\begin{aligned} w_0 &= v_0; \\ w_i &= 2v_i, \text{ for } 1 \le i \le N-1. \end{aligned}$$

Then, we have

$$\begin{aligned} y_0 &= w_0; \\ y_i &= w_i - y_{i-1}, \text{ for } 1 \le i \le N-1. \end{aligned} \tag{9}$$

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 (9) can be computed by only $O(\log N)$ steps of parallel additions. For convenience, let us call the problem in Equation (9) to be *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. 3 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.

Theorem 9 : *When given sufficient processing elements, the recursive doubling algorithm can compute the prefix difference problem of size N in only* $\log N$ *parallel addition steps.*

Proof: We show by induction on Step k that if $2^{(k-1)} \leq i \leq 2^k - 1$, then after step k, $y_i = w_i - w_{i-1} + w_{i-2} \cdots (-1)^i w_0$; if $i > 2^k - 1$, then after step k, $y_i = w_i - w_{i-1} + w_{i-2} \cdots - w_{i-(2^k-1)}$. Therefore, after $\log N$ steps, the prefix difference problem is solved.

Denote $y_i^{(k)}$ to be y_i after step k.

When $k = 1$, $2^k - 1 = 1$. As shown in Step 1 that $y_0^{(1)} = w_0$ and $y_i^{(1)} = w_i - w_{i-1}$ for $1 \leq i \leq N-1$.

When $k > 1$, $y_i^{(k)} = y_i^{(k-1)} + y_{i-2^{(k-1)}}^{(k-1)}$, for $2^{(k-1)} \leq i \leq N-1$. There are three cases:

1. If $i < 2^{(k-1)}$, y_i is not changed. Therefore, $y_i^{(k)} = y_i^{(k-1)}$.

2. If $2^{(k-1)} \leq i \leq 2^k - 1$, then $0 \leq i - 2^{(k-1)} \leq 2^{(k-1)} - 1$. By induction, $y_i^{(k-1)} = w_i - w_{i-1} + w_{i-2} \cdots - w_{i-(2^{(k-1)}-1)}$; and $y_{i-2^{(k-1)}}^{(k-1)} = w_{i-2^{(k-1)}} - w_{i-2^{(k-1)}-1} + \cdots (-1)^{(i-2^{(k-1)})} w_0$. Therefore, $y_i^{(k)} = w_i - w_{i-1} + w_{i-2} \cdots (-1)^i w_0$.

3. If $i > 2^k - 1$, then $i - 2^{(k-1)} > 2^{(k-1)} - 1$. By induction, $y_i^{(k-1)} = w_i - w_{i-1} + w_{i-2} \cdots - w_{i-(2^{(k-1)}-1)}$; and $y_{i-2^{(k-1)}}^{(k-1)} = w_{i-2^{(k-1)}} - w_{i-2^{(k-1)}-1} + \cdots - w_{i-2^{(k-1)}-(2^{(k-1)}-1)}$. Therefore, $y_i^{(k)} = w_i - w_{i-1} + \cdots - w_{i-(2^k-1)}$. □

The second algorithm, which uses the cyclic reduction technique, 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 $0 < i < \frac{N}{2}$, **do** $y_{2i} = w_{2i} - w_{2i-1}$;

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

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

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

for all $0 \leq i < 2^k$, **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** $0 \leq i < \frac{N}{2}$, **do** $y_{2i+1} = w_{2i+1} - y_{2i}$;

Fig. 4 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.

Theorem 10 : *When given sufficient processing elements, the cyclic reduction algorithm can compute the prefix difference problem of size N in only $2(\log N) - 1$ parallel addition steps.*

Proof: We briefly sketch the idea of proving this theorem. If we ignore the operations of computing $y_{\frac{N}{2}}^{(k)}$ in step k, for all $1 \leq k \leq \log N$, the algorithm is to compute two subproblems of size $\frac{N}{2}$. They

are

$$(1)\ y_0 = w_0 \text{ and } y_i = w_i - y_{i-1}, \text{ for } 1 \le i < \tfrac{N}{2}; \text{ and} \tag{10}$$

$$(2)\ y_{\frac{N}{2}} = w_{\frac{N}{2}} \text{ and } y_i = w_i - y_{i-1}, \text{ for } \tfrac{N}{2}+1 \le i < N. \tag{11}$$

However, $y_{\frac{N}{2}} = w_{\frac{N}{2}} - w_{\frac{N}{2}-1} + \cdots - w_0$ is computed only in Step 1 through Step $\log N$. In addition, $y_{\frac{N}{2}}$ is used (on the right-hand side of the assignment) only after Step $\log N$. Therefore, if we include the operations of computing $y_{\frac{N}{2}}$, we can replace $y_{\frac{N}{2}} = w_{\frac{N}{2}}$ by $y_{\frac{N}{2}} = w_{\frac{N}{2}} - w_{\frac{N}{2}-1} + \cdots - w_0$ in Equation (11). Thus, the algorithm computes the prefix difference problem correctly.

For formally proving the theorem, we could further divide each of the problems in Equations (10) and (11) into two subproblems, and repeatedly use the same techniques as shown in the last paragraph. Because of the space limitation, we omit it in this presentation. □

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 $O(\log N)$.

In the following, we show the complexities for the parallel computation.

1. Parallel multiplication steps.

   The recursive formula for the number of parallel multiplication steps required is

$$\begin{aligned} M(2) &= 1, \text{ and} \\ M(N) &= M(\tfrac{N}{2}) + 1, \text{ for } N > 2; \end{aligned}$$

   which implies

$$M(N) = \log N, \text{ for } N \ge 2.$$

2. Parallel addition steps.

   The recursive formula for the number of parallel addition steps required is

$$\begin{aligned} A(2) &= 1, \text{ and} \\ A(N) &= A(\tfrac{N}{2}) + O(\log N), \text{ for } N > 2; \end{aligned}$$

which implies

$$A(N) = O(\log^2 N), \quad \text{for } N \geq 2.$$

3. Parallel shift steps.

   The recursive formula for the number of parallel shift steps required is

$$\begin{aligned} S(4) &= 1, \quad \text{and} \\ S(N) &= S(\tfrac{N}{2}) + 1, \quad \text{for } N > 4; \end{aligned}$$

   which implies

$$S(N) = \log N - 1, \quad \text{for } N \geq 4.$$

4. Parallel memory reference steps.

   The recursive formula for the number of parallel memory reference steps required is

$$\begin{aligned} R(8) &= 1, \quad \text{and} \\ R(N) &= R(\tfrac{N}{2}) + 1, \quad \text{for } N > 8; \end{aligned}$$

   which implies

$$R(N) = \log N - 2, \quad \text{for } N \geq 8.$$

5.2 Hardware Implementation Considerations

We now consider the hardware implementations of our DCT algorithm. Fig. 1 shows the general signal-flow graph for the implementation of Equation (6). The block $L_{\frac{N}{2}}$, which performs shifts and additions, can be implemented by either the recursive doubling algorithm as shown in Fig. 3 or the cyclic reduction algorithm as shown in Fig. 4 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 $\hat{\mathrm{C}}_{\frac{N}{2}}$ and $S_{\frac{N}{2}}$ as shown in Fig. 5 instead of two $\hat{\mathrm{C}}_{\frac{N}{2}}$'s and two $S_{\frac{N}{2}}$'s as shown in Fig. 1. Note that $S_{\frac{N}{2}}$ is a shuffle-exchange network of size $\frac{N}{2}$.

In Fig. 5, the hardware is divided into four parts: one coefficient multiplier $Q_{\frac{N}{2}}$, one $\hat{\mathrm{C}}_{\frac{N}{2}}$, one shuffle-exchange network $S_{\frac{N}{2}}$, and one shift and adder $L_{\frac{N}{2}}$. 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 $\hat{\mathrm{C}}_{\frac{N}{2}}$ and the

	1	2	3	4	5	6	7	8	9	10	11	12
$Q_{\frac{N}{2}}$	B^1		B^2		B^3		B^4		B^5			
$\hat{\mathbf{C}}_{\frac{N}{2}}$	T^1	B^1	T^2	B^2	T^3	B^3	T^4	B^4	T^5	B^5		
$S_{\frac{N}{2}}$		T^1	B^1	T^2	B^2	T^3	B^3	T^4	B^4	T^5	B^5	
$L_{\frac{N}{2}}$				B^1		B^2		B^3		B^4		B^5

Table 2: Five sets of input data are pipelined entering into the hardware: $Q_{\frac{N}{2}}$, $\hat{\mathbf{C}}_{\frac{N}{2}}$, $S_{\frac{N}{2}}$, and $L_{\frac{N}{2}}$, and are executed in a pipelined fashion in twelve units of time. T^k means the k-th top-half input signals, B^k means the k-th bottom-half input signals.

$S_{\frac{N}{2}}$ functional units first; while the bottom half will first perform coefficient multiplications by a coefficient multiplier $Q_{\frac{N}{2}}$, then pass through the $\hat{\mathbf{C}}_{\frac{N}{2}}$ and the $S_{\frac{N}{2}}$ functional units, and finally pass through the shift and adder processor $L_{\frac{N}{2}}$. Table 2 shows that succeeding sets of input data can enter the hardware and can be executed in a pipelined fashion. Note that, the functional units $\hat{\mathbf{C}}_{\frac{N}{2}}$ and $S_{\frac{N}{2}}$ are at full utilization.

It is worthwhile illustrating that our hardware implementation is better than the implementation proposed by Hou [17]. In addition, we propose fast hardware implementation for computing $L_{\frac{N}{2}} v_4$, 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. This is because our algorithm is more balanced than Hou's algorithm when decomposing the DCT into several functional submodules. Therefore, our algorithm is more attractive than his algorithm when performing the parallel computation. In effect, the parallel computation time of our implementation is also less than that of his implementation.

6 2-D DCT

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

$$X_{k_1,k_2} = \frac{2}{\sqrt{N_1}\sqrt{N_2}}\,\epsilon(k_1)\,\epsilon(k_2) \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2}\, \cos(\frac{\pi(2n_1+1)k_1}{2N_1}) \cos(\frac{\pi(2n_2+1)k_2}{2N_2}), \qquad (12)$$

where

$$\epsilon(k_i) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for } k_i = 0 \\ 1, & \text{for } 1 \le k_i \le N_i - 1 \end{cases} \qquad (13)$$

and $i = 1$ or 2.

Similar to the 1-D case, we will also ignore $\epsilon(k_i)$ and the normalization factor $\frac{2}{\sqrt{N_1}\sqrt{N_2}}$ for convenience. Therefore, from now on, we deal with the simplified version of Equation (12):

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2}\, \cos(\frac{\pi(2n_1+1)k_1}{2N_1}) \cos(\frac{\pi(2n_2+1)k_2}{2N_2}). \qquad (14)$$

According to the conventional row-column method, we can express the simplified 2-D DCT version in Equation (14) as the following matrix form:

$$\mathbf{X}_{N_1 \times N_2} = \mathbf{C}_{N_1} (\mathbf{C}_{N_2} \mathbf{x}^T_{N_1 \times N_2})^T = \mathbf{C}_{N_1} \mathbf{x}_{N_1 \times N_2} \mathbf{C}^T_{N_2}. \qquad (15)$$

Again, we assume throughout this paper that N_1 and N_2 have values of 2 to a power.

6.1 The 2-D DCT Recursive Algorithm

We now construct our 2-D DCT recursive formula. We start with the simplified 2-D DCT version in Equation (15). First, by arranging appropriate row permutations and column permutations, we have

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

where S_{N_i}, $S_{N_i}^T$, P_{N_i}, and $P_{N_i}^T$ are similarly defined as in Section 3, for $i = 1$ or 2; in addition, $S_{N_1} S_{N_1}^T = P_{N_1} P_{N_1}^T = I_{N_1}$ and $S_{N_2} S_{N_2}^T = P_{N_2} P_{N_2}^T = I_{N_2}$. Next, multiply $S_{N_1}^T$ and S_{N_2} to both sides of the above equation, and we have

$$\begin{aligned} S_{N_1}^T \mathbf{X}_{N_1 \times N_2} S_{N_2} &= S_{N_1}^T S_{N_1} \hat{\mathbf{C}}_{N_1} P_{N_1}^T \mathbf{x}_{N_1 \times N_2} P_{N_2} \hat{\mathbf{C}}_{N_2}^T S_{N_2}^T S_{N_2} \\ &= \hat{\mathbf{C}}_{N_1} (P_{N_1}^T \mathbf{x}_{N_1 \times N_2} P_{N_2}) \hat{\mathbf{C}}_{N_2}^T. \end{aligned} \tag{16}$$

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_e represent the submatrix formed by the even rows of $\mathbf{x}_{N_1 \times N_2}$, and $x_{\bar{o}}$ represent the submatrix formed by the odd rows of $\mathbf{x}_{N_1 \times N_2}$ but in reversed order. Then,

$$P_{N_1}^T \mathbf{x}_{N_1 \times N_2} = \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix}_{N_1 \times N_2}.$$

Let x_{ee} represent the submatrix formed by the even columns of x_e; $x_{e\bar{o}}$ represent the submatrix formed by the odd columns of x_e but in reversed order; $x_{\bar{o}e}$ represent the submatrix formed by the even columns of $x_{\bar{o}}$; and $x_{\bar{o}\bar{o}}$ represent the submatrix formed by the odd columns of $x_{\bar{o}}$ but in reversed order. Then,

$$P_{N_1}^T \mathbf{x}_{N_1 \times N_2} P_{N_2} = \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix} P_{N_2} = \begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix}_{N_1 \times N_2}. \tag{17}$$

Similarly, let X_e represent the submatrix formed by the even rows of $\mathbf{X}_{N_1 \times N_2}$, and X_o represent the submatrix formed by the odd rows of $\mathbf{X}_{N_1 \times N_2}$. Then,

$$S_{N_1}^T \mathbf{X}_{N_1 \times N_2} = \begin{bmatrix} X_e \\ X_o \end{bmatrix}_{N_1 \times N_2}.$$

Let X_{ee} represent the submatrix formed by the even columns of X_e; X_{eo} represent the submatrix formed by the odd columns of X_e; X_{oe} represent the submatrix formed by the even columns of X_o; and X_{oo} represent the submatrix formed by the odd columns of X_o. Then,

$$S_{N_1}^T \mathbf{X}_{N_1 \times N_2} S_{N_2} = \begin{bmatrix} X_e \\ X_o \end{bmatrix} S_{N_2} = \begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix}_{N_1 \times N_2}. \tag{18}$$

Therefore, from Equations (16), (17), and (18), we have

$$\begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix} = \hat{\mathbf{C}}_{N_1} \begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix} \hat{\mathbf{C}}_{N_2}^T. \tag{19}$$

Equation (19) is our basic 2-D DCT recursive formula. Now, by applying Theorem 7, we have

$$\begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix} =$$

$$\begin{bmatrix} S_{\frac{N_1}{2}}\hat{\mathbf{C}}_{\frac{N_1}{2}} & S_{\frac{N_1}{2}}\hat{\mathbf{C}}_{\frac{N_1}{2}} \\ L_{\frac{N_1}{2}}S_{\frac{N_1}{2}}\hat{\mathbf{C}}_{\frac{N_1}{2}}Q_{\frac{N_1}{2}} & -L_{\frac{N_1}{2}}S_{\frac{N_1}{2}}\hat{\mathbf{C}}_{\frac{N_1}{2}}Q_{\frac{N_1}{2}} \end{bmatrix} \begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{C}}^T_{\frac{N_2}{2}}S^T_{\frac{N_2}{2}} & Q_{\frac{N_2}{2}}\hat{\mathbf{C}}^T_{\frac{N_2}{2}}S^T_{\frac{N_2}{2}}L^T_{\frac{N_2}{2}} \\ \hat{\mathbf{C}}^T_{\frac{N_2}{2}}S^T_{\frac{N_2}{2}} & -Q_{\frac{N_2}{2}}\hat{\mathbf{C}}^T_{\frac{N_2}{2}}S^T_{\frac{N_2}{2}}L^T_{\frac{N_2}{2}} \end{bmatrix}.$$

By expanding the matrix multiplications on the right-hand side of the above equation, we obtain the following four equations:

$$X_{ee} = S_{\frac{N_1}{2}}[\hat{\mathbf{C}}_{\frac{N_1}{2}}(x_{ee}+x_{e\bar{o}}+x_{\bar{o}e}+x_{\bar{o}\bar{o}})\hat{\mathbf{C}}^T_{\frac{N_2}{2}}]S^T_{\frac{N_2}{2}}; \quad (20)$$

$$X_{eo} = S_{\frac{N_1}{2}}[\hat{\mathbf{C}}_{\frac{N_1}{2}}(x_{ee}-x_{e\bar{o}}+x_{\bar{o}e}-x_{\bar{o}\bar{o}})Q_{\frac{N_2}{2}}\hat{\mathbf{C}}^T_{\frac{N_2}{2}}]S^T_{\frac{N_2}{2}}L^T_{\frac{N_2}{2}}; \quad (21)$$

$$X_{oe} = L_{\frac{N_1}{2}}S_{\frac{N_1}{2}}[\hat{\mathbf{C}}_{\frac{N_1}{2}}Q_{\frac{N_1}{2}}(x_{ee}+x_{e\bar{o}}-x_{\bar{o}e}-x_{\bar{o}\bar{o}})\hat{\mathbf{C}}^T_{\frac{N_2}{2}}]S^T_{\frac{N_2}{2}}; \quad (22)$$

$$X_{oo} = L_{\frac{N_1}{2}}S_{\frac{N_1}{2}}[\hat{\mathbf{C}}_{\frac{N_1}{2}}Q_{\frac{N_1}{2}}(x_{ee}-x_{e\bar{o}}-x_{\bar{o}e}+x_{\bar{o}\bar{o}})Q_{\frac{N_2}{2}}\hat{\mathbf{C}}^T_{\frac{N_2}{2}}]S^T_{\frac{N_2}{2}}L^T_{\frac{N_2}{2}}. \quad (23)$$

Note that, the terms inside the bracket of the above four equations are four subproblems with a reduced size $\frac{N_1}{2} \times \frac{N_2}{2}$ of Equation (19). 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 order to design a fast algorithm, some optimizations are considered. First, reduce the number of matrix additions.

Theorem 11 : *The four matrices* $x_a = x_{ee} + x_{e\bar{o}} + x_{\bar{o}e} + x_{\bar{o}\bar{o}}$, $x_b = x_{ee} - x_{e\bar{o}} + x_{\bar{o}e} - x_{\bar{o}\bar{o}}$, $x_c = x_{ee} + x_{e\bar{o}} - x_{\bar{o}e} - x_{\bar{o}\bar{o}}$, *and* $x_d = x_{ee} - x_{e\bar{o}} - x_{\bar{o}e} + x_{\bar{o}\bar{o}}$ *can be computed by using only eight matrix additions.*

Proof: We show eight matrix additions to compute the four matrices: x_a, x_b, x_c, and x_d.

1. $t_1 = x_{ee} + x_{e\bar{o}}$, 2. $t_2 = x_{ee} - x_{e\bar{o}}$, 3. $t_3 = x_{\bar{o}e} + x_{\bar{o}\bar{o}}$, 4. $t_4 = x_{\bar{o}e} - x_{\bar{o}\bar{o}}$,
5. $x_a = t_1 + t_3$, 6. $x_b = t_2 + t_4$, 7. $x_c = t_1 - t_3$, 8. $x_d = t_2 - t_4$. □

Second, reduce the number of multiplications. We now consider the term $Q_{\frac{N_1}{2}}x_dQ_{\frac{N_2}{2}}$, which is required to compute X_{oo} in Equation (23). Note that, $Q_{\frac{N_1}{2}}$ and $Q_{\frac{N_2}{2}}$ are diagonal matrices, and

x_d is a $\frac{N_1}{2} \times \frac{N_2}{2}$ matrix. Let $q_{\frac{N_1}{2}}$ be the vector in the diagonal of $Q_{\frac{N_1}{2}}$, and $q_{\frac{N_2}{2}}$ be the vector in the diagonal of $Q_{\frac{N_2}{2}}$. Then, $Q_{\frac{N_1}{2}}\, x_d\, Q_{\frac{N_2}{2}} = (q_{\frac{N_1}{2}}\, q_{\frac{N_2}{2}}^T) \odot x_d$, where '$\odot$' means pointwise multiplications of two matrices. Since $q_{\frac{N_1}{2}}$ and $q_{\frac{N_2}{2}}$ are already known by its definition, we can compute $q_{\frac{N_1}{2}}\, q_{\frac{N_2}{2}}^T$ and store the results in the memory in advance. Therefore, we can compute $Q_{\frac{N_1}{2}}\, x_d\, Q_{\frac{N_2}{2}}$ in only $\frac{N_1}{2} \times \frac{N_2}{2}$ scalar multiplications.

Theorem 12 : *If the value of $q_{\frac{N_1}{2}}\, q_{\frac{N_2}{2}}^T$ is stored in the memory in advance, then $Q_{\frac{N_1}{2}}\, x_d\, Q_{\frac{N_2}{2}}$ can be computed in only $\frac{N_1}{2} \times \frac{N_2}{2}$ scalar multiplications.* □

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

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 x_a, x_b, x_c, and x_d in Theorem 11. */
$t_1 = x_{ee} + x_{e\bar{o}}$, $t_2 = -x_{ee} - x_{e\bar{o}}$, $t_3 = x_{\bar{o}e} + x_{\bar{o}\bar{o}}$, $t_4 = x_{\bar{o}e} - x_{\bar{o}\bar{o}}$,
$x_a = t_1 + t_3$, $x_b = t_2 + t_4$, $x_c = t_1 - t_3$, $x_d = t_2 - t_4$;

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

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

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

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

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

In the 2-D DCT recursive algorithm, Step 2, which implements Equation (19), is a recursive procedure. In general, an $(N_1 \times N_2)$-point DCT problem will be decomposed recursively until $N_1 = N_2 = 2$. Note that, in our opinion, the (2×2)-point DCT problem should be regarded as a basic

computation block and should be computed directly with the optimized operation count. Step 2.1 tests the condition whether $N_1 = 2$ or $N_2 = 2$; if this is true, then handles this special case until $N_1 = N_2 = 2$. Note that, this special case does not need to be illustrated due to space constraints; however, this has been verified. Step 2.2 deals with the four subproblems, which can be solved independently. Fig. 6 shows the general signal-flow graph for the implementation of Step 2.2.

In the following, we illustrate our 2-D DCT recursive algorithm with three simple examples.

For $N_1 = N_2 = 2$,

$$\mathbf{X}_{2\times 2} = \mathbf{C}_2\, \mathbf{x}_{2\times 2}\, \mathbf{C}_2^T$$

which implies

$$\begin{aligned}
\begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ \cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{bmatrix} \begin{bmatrix} 1 & \cos(\frac{\pi}{4}) \\ 1 & -\cos(\frac{\pi}{4}) \end{bmatrix} \\
&= \begin{bmatrix} (x_{00}+x_{10}) & (x_{01}+x_{11}) \\ \cos(\frac{\pi}{4})(x_{00}-x_{10}) & \cos(\frac{\pi}{4})(x_{01}-x_{11}) \end{bmatrix} \begin{bmatrix} 1 & \cos(\frac{\pi}{4}) \\ 1 & -\cos(\frac{\pi}{4}) \end{bmatrix} \\
&= \begin{bmatrix} (x_{00}+x_{01}+x_{10}+x_{11}) & \cos(\frac{\pi}{4})(x_{00}-x_{01}+x_{10}-x_{11}) \\ \cos(\frac{\pi}{4})(x_{00}+x_{01}-x_{10}-x_{11}) & \frac{1}{2}(x_{00}-x_{01}-x_{10}+x_{11}) \end{bmatrix}.
\end{aligned}$$

In this case, eight additions as shown in Theorem 11, two multiplications, and one shift are required; see Fig. 7-(a) for the signal-flow graph for $N_1 = N_2 = 2$.

For $N_1 = N_2 = 4$,

$$\mathbf{X}_{4\times 4} = \mathbf{C}_4\, \mathbf{x}_{4\times 4}\, \mathbf{C}_4^T.$$

From Equation (19),

$$\begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix} = \begin{bmatrix} S_2\hat{\mathbf{C}}_2 & S_2\hat{\mathbf{C}}_2 \\ L_2S_2\hat{\mathbf{C}}_2Q_2 & -L_2S_2\hat{\mathbf{C}}_2Q_2 \end{bmatrix} \begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{C}}_2^T S_2^T & Q_2\hat{\mathbf{C}}_2^T S_2^T L_2^T \\ \hat{\mathbf{C}}_2^T S_2^T & -Q_2\hat{\mathbf{C}}_2^T S_2^T L_2^T \end{bmatrix}. \quad (24)$$

Since $\hat{\mathbf{C}}_2 = \mathbf{C}_2$ and $S_2 = S_2^T = I_2$, we have

$$\begin{aligned}
X_{ee} &= \mathbf{C}_2\,(x_{ee} + x_{e\bar{o}} + x_{\bar{o}e} + x_{\bar{o}\bar{o}})\,\mathbf{C}_2^T; \\
X_{eo} &= \mathbf{C}_2\,(x_{ee} - x_{e\bar{o}} + x_{\bar{o}e} - x_{\bar{o}\bar{o}})\,Q_2\,\mathbf{C}_2^T\,L_2^T;
\end{aligned}$$

$$X_{oe} = L_2\,\mathrm{C}_2\,Q_2\,(x_{ee} + x_{e\bar{o}} - x_{\bar{o}e} - x_{\bar{o}\bar{o}})\,\mathrm{C}_2^T;$$
$$X_{oo} = L_2\,\mathrm{C}_2\,Q_2\,(x_{ee} - x_{e\bar{o}} - x_{\bar{o}e} + x_{\bar{o}\bar{o}})\,Q_2\,\mathrm{C}_2^T\,L_2^T.$$

In this case, first, eight 2×2 matrix additions are required for computing x_a, x_b, x_c, and x_d as shown in Theorem 11. Second, for deriving the terms $u_1 = x_b\,Q_2$, $u_2 = Q_2\,x_c$, and $u_3 = Q_2\,x_d\,Q_2$, each requires 2×2 multiplications. Note that we can compute $Q_2 x_d Q_2$ in 2×2 multiplications as shown in Theorem 12. Third, for computing either $X_{ee} = \mathrm{C}_2\,x_a\,\mathrm{C}_2^T$, or $u_4 = \mathrm{C}_2\,u_1\,\mathrm{C}_2^T$, or $u_5 = \mathrm{C}_2\,u_2\,\mathrm{C}_2^T$, or $u_6 = \mathrm{C}_2\,u_3\,\mathrm{C}_2^T$, each one requires two multiplications, eight additions, and one shift, as was in the case when $N_1 = N_2 = 2$. Fourth, computation of $X_{eo} = u_4\,L_2^T$ needs one shift and one addition twice; determination of $X_{oe} = L_2\,u_5$ embodies one shift and one addition twice; derivation of $X_{oo} = L_2\,u_6\,L_2^T$ requires one shift and one addition four times. Therefore, we require twenty multiplications, seventy-two additions, and twelve shifts for computing Equation (24); see Fig. 7-(b) for the signal-flow graph for $N_1 = N_2 = 4$.

For $N_1 = N_2 = 8$,

$$\mathrm{X}_{8\times 8} = \mathrm{C}_8\,\mathrm{x}_{8\times 8}\,\mathrm{C}_8^T.$$

From Equation (19),

$$\begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix} = \begin{bmatrix} S_4\hat{\mathrm{C}}_4 & S_4\hat{\mathrm{C}}_4 \\ L_4S_4\hat{\mathrm{C}}_4Q_4 & -L_4S_4\hat{\mathrm{C}}_4Q_4 \end{bmatrix} \begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix} \begin{bmatrix} \hat{\mathrm{C}}_4^T S_4^T & Q_4\hat{\mathrm{C}}_4^T S_4^T L_4^T \\ \hat{\mathrm{C}}_4^T S_4^T & -Q_4\hat{\mathrm{C}}_4^T S_4^T L_4^T \end{bmatrix}. \quad (25)$$

We have

$$X_{ee} = S_4[\hat{\mathrm{C}}_4\,(x_{ee} + x_{e\bar{o}} + x_{\bar{o}e} + x_{\bar{o}\bar{o}})\,\hat{\mathrm{C}}_4^T]\,S_4^T;$$
$$X_{eo} = S_4[\hat{\mathrm{C}}_4\,(x_{ee} - x_{e\bar{o}} + x_{\bar{o}e} - x_{\bar{o}\bar{o}})\,Q_4\,\hat{\mathrm{C}}_4^T]\,S_4^T\,L_4^T;$$
$$X_{oe} = L_4\,S_4[\hat{\mathrm{C}}_4\,Q_4\,(x_{ee} + x_{e\bar{o}} - x_{\bar{o}e} - x_{\bar{o}\bar{o}})\,\hat{\mathrm{C}}_4^T]\,S_4^T;$$
$$X_{oo} = L_4\,S_4[\hat{\mathrm{C}}_4\,Q_4\,(x_{ee} - x_{e\bar{o}} - x_{\bar{o}e} + x_{\bar{o}\bar{o}})\,Q_4\,\hat{\mathrm{C}}_4^T]\,S_4^T\,L_4^T.$$

In this case, first, eight 4×4 matrix additions are required for computing x_a, x_b, x_c, and x_d as shown in Theorem 11. Second, for computing either $u_1 = x_b\,Q_4$, or $u_2 = Q_4\,x_c$, or $u_3 = Q_4\,x_d\,Q_4$, each one requires 4×4 multiplications. Third, for deriving the terms $u_4 = \hat{\mathrm{C}}_4\,x_a\,\hat{\mathrm{C}}_4^T$, $u_5 = \hat{\mathrm{C}}_4\,u_1\,\hat{\mathrm{C}}_4^T$, $u_6 = \hat{\mathrm{C}}_4\,u_2\,\hat{\mathrm{C}}_4^T$, and $u_7 = \hat{\mathrm{C}}_4\,u_3\,\hat{\mathrm{C}}_4^T$, each requires twenty multiplications, seventy two additions, and twelve shifts, as was in the case when $N_1 = N_2 = 4$. Fourth, eight 4×4 memory reference

operations are required for computing $X_{ee} = S_4\, u_4\, S_4^T$, $u_8 = S_4\, u_5\, S_4^T$, $u_9 = S_4\, u_6\, S_4^T$, and $u_{10} = S_4\, u_7\, S_4^T$. Fifth, computation of $X_{eo} = u_8\, L_4^T$ needs three shifts and three additions four times; derivation of $X_{oe} = L_4\, u_9$ embodies three shifts and three additions four times; and determination of $X_{oo} = L_4\, u_{10}\, L_4^T$ requires three shifts and three additions eight times. Therefore, we require 128 multiplications, 464 additions, 96 shifts, and 128 memory references for computing Equation (25).

6.2 Analysis of the 2-D DCT Algorithm

In this subsection, we will analyze the complexities of the 2-D DCT algorithm. The 2-D DCT algorithm contains four steps. Step 1 and Step 3 are concerned with data arrangement, and each can be done in $2(N_1 \times N_2)$ memory references. Step 4 deals with multiplying constant factors to each of the data entries, which can be done in $N_1 \times N_2$ multiplications.

We now consider the complexity of Step 2. For simplicity of this presentation, we only discuss the case when $N_1 = N_2 = N$. Therefore, Step 2.1 deals with the basic (2×2)-point DCT. As shown in Section 6.1, the basic (2×2)-point DCT can be solved by using two multiplications, eight additions, and one shift. Step 2.2 includes the following operations:

- Eight $\frac{N}{2} \times \frac{N}{2}$ additions are required for computing x_a, x_b, x_c, and x_d as shown in Theorem 11.
- Three $\frac{N}{2} \times \frac{N}{2}$ multiplications are required for computing $u_1 = x_b\, Q_{\frac{N}{2}}$, $u_2 = Q_{\frac{N}{2}}\, x_c$, and $u_3 = Q_{\frac{N}{2}}\, x_d\, Q_{\frac{N}{2}}$.
- Four subproblems, which include $u_4 = \hat{\mathbf{C}}_{\frac{N}{2}}\, x_a\, \hat{\mathbf{C}}_{\frac{N}{2}}^T$, $u_5 = \hat{\mathbf{C}}_{\frac{N}{2}}\, u_1\, \hat{\mathbf{C}}_{\frac{N}{2}}^T$, $u_6 = \hat{\mathbf{C}}_{\frac{N}{2}}\, u_2\, \hat{\mathbf{C}}_{\frac{N}{2}}^T$, and $u_7 = \hat{\mathbf{C}}_{\frac{N}{2}}\, u_3\, \hat{\mathbf{C}}_{\frac{N}{2}}^T$, can be solved recursively.
- Eight $\frac{N}{2} \times \frac{N}{2}$ memory reference operations are required for computing $X_{ee} = S_{\frac{N}{2}}\, u_4\, S_{\frac{N}{2}}^T$, $u_8 = S_{\frac{N}{2}}\, u_5\, S_{\frac{N}{2}}^T$, $u_9 = S_{\frac{N}{2}}\, u_6\, S_{\frac{N}{2}}^T$, and $u_{10} = S_{\frac{N}{2}}\, u_7\, S_{\frac{N}{2}}^T$.
- Four times of $\frac{N}{2}(\frac{N}{2} - 1)$ shifts and additions are required for computing $X_{eo} = u_8\, L_{\frac{N}{2}}^T$, $X_{oe} = L_{\frac{N}{2}}\, u_9$, and $X_{oo} = L_{\frac{N}{2}}\, u_{10}\, L_{\frac{N}{2}}^T$.

We now analyze the exact numbers of operations, including multiplications, additions, shifts, and memory references, required for computing Step 2.

1. Multiplications.

   As shown in Section 7, that when $N = 2$, two multiplications are required. The recursive formula for the number of multiplications required is

$$\begin{aligned} M(2) &= 2, \text{ and} \\ M(N) &= 4M(\tfrac{N}{2}) + \tfrac{3}{4}N^2, \quad \text{for } N > 2; \end{aligned}$$

   which implies

$$M(N) = \frac{3}{4}N^2 \log N - \frac{1}{4}N^2, \quad \text{for } N \geq 2.$$

2. Additions.

   As shown in Section 7, that when $N = 2$, eight additions are required. The recursive formula for the number of additions required is

$$\begin{aligned} A(2) &= 8, \text{ and} \\ A(N) &= 4A(\tfrac{N}{2}) + 3N^2 - 2N, \quad \text{for } N > 2; \end{aligned}$$

   which implies

$$A(N) = 3N^2 \log N - 2N^2 + 2N, \quad \text{for } N \geq 2.$$

3. Shifts.

   As shown in Section 7, that when $N = 2$, one shift is required. The recursive formula for the number of shifts required is

$$\begin{aligned} S(2) &= 1, \text{ and} \\ S(N) &= 4S(\tfrac{N}{2}) + N^2 - 2N, \quad \text{for } N > 2; \end{aligned}$$

   which implies

$$S(N) = N^2 \log N - \frac{7}{4}N^2 + 2N, \quad \text{for } N \geq 2.$$

4. Memory references.

   As shown in Section 7, that when $N = 8$, 128 memory references are required. The recursive formula for the number of memory references required is

$$\begin{aligned} R(8) &= 128, \text{ and} \\ R(N) &= 4R(\tfrac{N}{2}) + 2N^2, \quad \text{for } N > 8; \end{aligned}$$

	Our method				Row-Column with 1-D DCT [23, 17]		[20]		[5, 12]	
$N \times N$	$\times$	$+$	S	R	$\times$	$+$	$\times$	$+$	$\times$	$+$
2×2	2	8	1	0			4	8		
4×4	20	72	12	0	32	72	28	66	24	72
8×8	128	464	96	128	192	464	128	430	144	464
16×16	704	2592	608	1024	1024	2592			768	2592
32×32	3584	13376	3392	6144	5120	13376			3840	13376
64×64	17408	65664	17536	32768	24576	65664			18432	65664
128×128	81920	311552	86272	163840	114688	311552			86016	311552
256×256	376832	1442304	410112	786432	524288	1442304			393216	1442304

Table 3: The comparison of the numbers of operations required for computing the 2-D DCT. '$\times$' means 'number of multiplications', '$+$' means 'number of additions or subtractions', 'S' means 'number of shifts', and 'R' means 'number of memory references'.

which implies

$$R(N) = 2N^2 \log N - 4N^2, \quad \text{for } N \geq 8.$$

Table 3 shows the numbers of multiplications, additions, shifts, and memory references required for the 2-D DCT algorithm, when $N = 2$, 4, 8, 16, 32, 64, 128, and 256. The numbers of multiplications are less than those of [5] [12] [20] and the conventional row-column method. The numbers of additions are the same as those of [5] [12] and the conventional row-column method. The comparison of the numbers of operations is shown in Table 3, although the numbers of shifts and memory references are not provided in other research papers.

6.3 Implementation Considerations for the 2-D DCT

Suppose that we have sufficient processing elements to implement the 2-D DCT. It is clear that the parallel operation complexities for the 2-D DCT are the same as those for the 1-D DCT. That is, the complexity for the parallel multiplication steps is $O(\log N)$, the complexity for the parallel addition steps is $O(\log^2 N)$, the complexity for the parallel shift steps is $O(\log N)$, and the complexity for the parallel memory reference steps is $O(\log N)$. Note that, in the 2-D DCT parallel computation, we use more processing elements than those for the 1-D DCT. However, their complexity order for the parallel operations are the same.

In the following, we consider a hardware implementation which computes Equation (19). We show how to use only one copy of functional units to implement the 2-D DCT in a pipelined fashion. In Theorem 12 (in Section 6.1), we show that $Q_{\frac{N_1}{2}} x_d Q_{\frac{N_2}{2}}$ can be done by a 'pointwise' matrix multiplication. In effect, $x_b Q_{\frac{N_2}{2}}$ and $Q_{\frac{N_1}{2}} x_c$ also can be done by using pointwise matrix multiplications, provided we have prepared necessary coefficient matrices in the memory in advance. Therefore, we use $Q_{\frac{N_1}{2} \times \frac{N_2}{2}}$ to represent the functional unit for implementing the three pointwise matrix multiplications.

Let (x) represent the current input $\frac{N_1}{2} \times \frac{N_2}{2}$ submatrix. Then, Fig. 8 shows the proposed hardware implementation. In Fig. 8, the hardware is divided into five parts: one pointwise matrix multiplier $Q_{\frac{N_1}{2} \times \frac{N_2}{2}}$, one $\hat{\mathbf{C}}_{\frac{N_1}{2}}(\mathbf{x})\,\hat{\mathbf{C}}^T_{\frac{N_2}{2}}$, one $S_{\frac{N_1}{2}}(\mathbf{x})\,S^T_{\frac{N_2}{2}}$, one $L_{\frac{N_1}{2}}(\mathbf{x})$, and one $(\mathbf{x})\,L^T_{\frac{N_2}{2}}$. Input data are read from the host computer and are executed in a pipelined fashion. Table 4 shows that succeeding sets of input data can enter the hardware and can be executed in a pipelined fashion. Note that the functional units $\hat{\mathbf{C}}_{\frac{N_1}{2}}(\mathbf{x})\,\hat{\mathbf{C}}^T_{\frac{N_2}{2}}$ and $S_{\frac{N_1}{2}}(\mathbf{x})\,S^T_{\frac{N_2}{2}}$ are at full utilization.

7 Conclusions

A systematic method for designing the DCT recursive algorithm 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 the 2-D DCT recursive algorithm. Since the DCT is orthonormal [1] [2], the inverse DCT transform can be realized by taking the transpose of the forward transform.

An important contribution of this paper is to derive a more balanced recursive formula when decomposing the DCT into several functional submodules. This allows us to design fast parallel algorithms. It also allows us to design fast hardware implementations but with fewer hardware components than other designers.

Detailed analysis of the algorithms is provided in this paper. We consider operation complexities including those for multiplications, additions, shifts, and memory references, because all of these must be considered if we want to design parallel algorithms and hardware implementations. For the

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Q_{\frac{N_1}{2}\times\frac{N_2}{2}}$	D^1	C^1	B^1		D^2	C^2	B^2		D^3	C^3	B^3			
$\hat{\mathbf{C}}_{\frac{N_1}{2}}(\mathbf{x})\,\hat{\mathbf{C}}^T_{\frac{N_2}{2}}$	A^1	D^1	C^1	B^1	A^2	D^2	C^2	B^2	A^3	D^3	C^3	B^3		
$S_{\frac{N_1}{2}}(\mathbf{x})\,S^T_{\frac{N_2}{2}}$		A^1	D^1	C^1	B^1	A^2	D^2	C^2	B^2	A^3	D^3	C^3	B^3	
$L_{\frac{N_1}{2}}(\mathbf{x})$				D^1	C^1			D^2	C^2			D^3	C^3	
$(\mathbf{x})\,L^T_{\frac{N_2}{2}}$					D^1	B^1			D^2	B^2			D^3	B^3

Table 4: Three sets of input data are pipelined entering into the hardware: $Q_{\frac{N_1}{2}\times\frac{N_2}{2}}$, $\hat{\mathbf{C}}_{\frac{N_1}{2}}(\mathbf{x})\,\hat{\mathbf{C}}^T_{\frac{N_2}{2}}$, $S_{\frac{N_1}{2}}(\mathbf{x})\,S^T_{\frac{N_2}{2}}$, $L_{\frac{N_1}{2}}(\mathbf{x})$, and $(\mathbf{x})\,L^T_{\frac{N_2}{2}}$, and are executed in a pipelined fashion in fourteen units of time. Note that, initially, let $A = x_a$, $B = x_b$, $C = x_c$, and $D = x_d$. After passing through the hardware, $X_{ee} = A$, $X_{eo} = B$, $X_{oe} = C$, and $X_{oo} = D$.

sequential algorithms, we require $\frac{N}{2}\log N$ multiplications, $\frac{3}{2}N\log N - N + 1$ additions, $\frac{N}{2}\log N - N + 1$ shifts, and $N\log N - 2N$ memory references, for computing an N-point DCT. The numbers of multiplications and additions are the same as those required in the best and well-known algorithms of [17] [23] [35] [39] [41].

For the sequential 2-D algorithms, we require $\frac{3}{4}N^2\log N - \frac{1}{4}N^2$ multiplications, $3N^2\log N - 2N^2 + 2N$ additions, $N^2\log N - \frac{7}{4}N^2 + 2N$ shifts, and $2N^2\log N - 4N^2$ memory references, for computing an $(N \times N)$-point DCT. The number of multiplications is less than those of [5] [12] [20]; while the number of additions are the same as those of [5] [12].

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 both the 1-D and the 2-D DCT. Note that, in the 2-D DCT parallel computation, we use more processing elements than those for the 1-D DCT. However, the complexity order for the parallel operations of these two are the same.

References

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

[2] N. Ahmed and K. R. Rao. *Orthogonal Transforms for Digital Signal Processing*. Springer-Verlag, 1975.

[3] G. Bertocci, B. W. Schoenherr, and D. G. Messerschmitt. An approach to the implementation of a DCT. *IEEE Transactions on Communications*, COM-30(4):635–641, 1982.

[4] C. Chakrabarti and J. JaJa. Systolic architectures for the computation of the discrete Hartley and the discrete Cosine transforms based on prime factor decomposition. *IEEE Transactions on Computers*, C-39(11):1359–1368, November 1990.

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

[6] L. W. Chang and M. C. Wu. A unified systolic array for discrete Cosine and Sine transforms. *IEEE Transactions on Signal Processing*, 39(1):192–194, January 1991.

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

[8] B. Chitprasert and K. R. Rao. Discrete Cosine transform filtering. *Signal Processing*, 19:233–245, 1990.

[9] N. I. Cho and S. U. Lee. DCT algorithms for VLSI parallel implementations. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(1):121–127, January 1990.

[10] D. F. Elliott and K. R. Rao. *Fast Transforms: Algorithms, Analysis, Application.* Academic Press, Inc., 1982.

[11] M. Hamidi and J. Pearl. Comparison of the Cosine and Fourier transforms of markov-1 signals. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-25:428–429, October 1976.

[12] M. A. Haque. A two-dimensional fast Cosine transform. *IEEE Transactions on Acoustics, 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 Transactions on Computers*, C-25(7):764–765, July 1976.

[14] H. Hassanein and M. Rudko. On the use of discrete Cosine transform in cepstral analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-32(4):922–925, August 1984.

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

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

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

[18] A. K. Jain. *Fundamentals of digital Image Processing.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[19] A. Jalali and K. R. Rao. A high-speed FDCT processor for real-time processing of NTSC color TV signal. *IEEE Transactions on Electromagn. Compat.*, EMC-24(2):270–286, 1982.

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

[21] D. S. Kim and S. U. Lee. Image vector quantizer based on a classification in the DCT domain. *IEEE Transactions on Communications*, 39(4):549–556, April 1991.

[22] T. Kinoshita and T. Nakahashi. A 130 Mb/s compact HDTV CODEC based on a motion-adaptive DCT algorithm. *IEEE Journal on Selected Areas in Communications*, 10(1):122–129, January 1992.

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

[24] B. G. Lee. Input and output mappings for a prime-factor-decomposed computation of discrete Cosine transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(2):237–244, February 1989.

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

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

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

[28] H. S. Malvar. Fast computation of the discrete Cosine transform and the discrete Hartley transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, TASSP-35(10):1484–1485, October 1987.

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

[30] M. J. Narashimha and A. M. Peterson. On the computation of the discrete Cosine transform. *IEEE Transactions on Communications*, COM-26:934–936, June 1978.

[31] K. N. Ngan. Image display techniques using the Cosine transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-32(1):173–177, February 1984.

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

[33] R. C. Reininger and J. D. Gibson. Distributions of the two-dimensional DCT coefficients for images. *IEEE Transactions on Communications*, COM-31(6):835–839, June 1983.

[34] J. A. Roese, W. K. Pratt, and G. S. Robinson. Interframe Cosine transform image coding. *IEEE Transactions on Communications*, COM-25:1329–1339, November 1977.

[35] N. Suehiro and M. Hatori. Fast algorithms for the DFT and other sinusoidal transforms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-34(3):642–644, June 1986.

[36] M. T. Sun, L. Wu, and M. L. Liou. A concurrent architecture for VLSI implementation of DCT. *IEEE Transactions on Circuits and Systems*, CAS-34(8):992–994, August 1987.

[37] B. D. Tseng and W. C. Miller. On computing the discrete Cosine transform. *IEEE Transactions on Computers*, C-27(10):966–968, October 1978.

[38] M. Vetterli and A. Ligtenberg. A discrete Fourier-Cosine transform chip. *IEEE Journal on Selected Areas in Communications*, SAC-4(1):49–61, January 1986.

[39] M. Vetterli and H. J. Nussbaumer. Simple FFT and DCT algorithm with reduced number of operations. *Signal Processing*, 6(4):267–278, 1984.

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

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

[42] Z. Wang. Pruning the fast discrete Cosine transform. *IEEE Transactions on Communications*, 39(5):640–643, May 1991.

[43] Z. Wang and B. R. Hunt. The discrete W transform. *Applied Mathematics and Computation*, 16:19–48, 1985.

[44] S. Winograd. On computing the discrete Fourier transform. *Mathematics of Computation*, 32(141):175–199, January 1978.

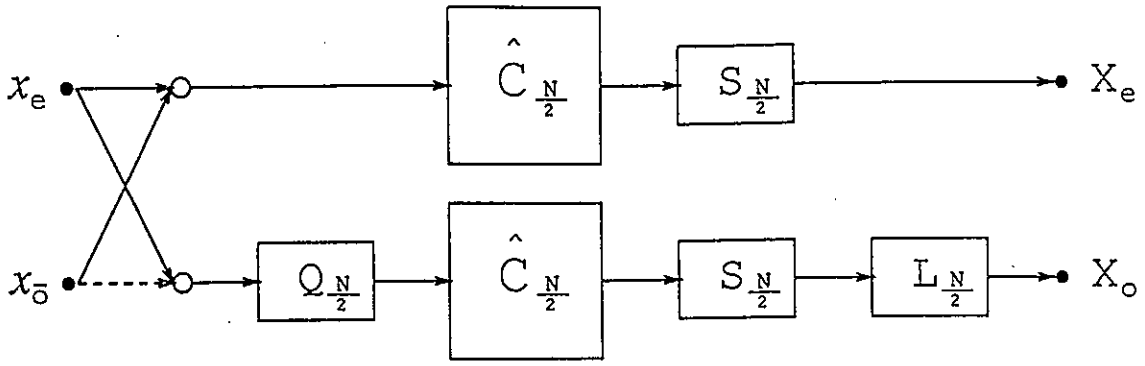
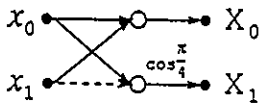
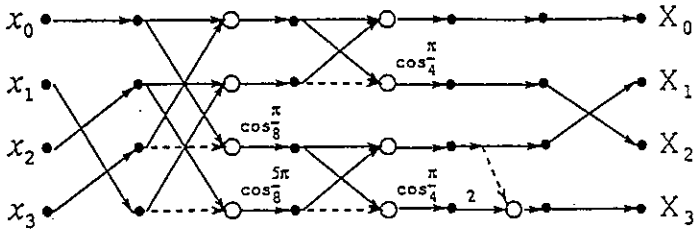


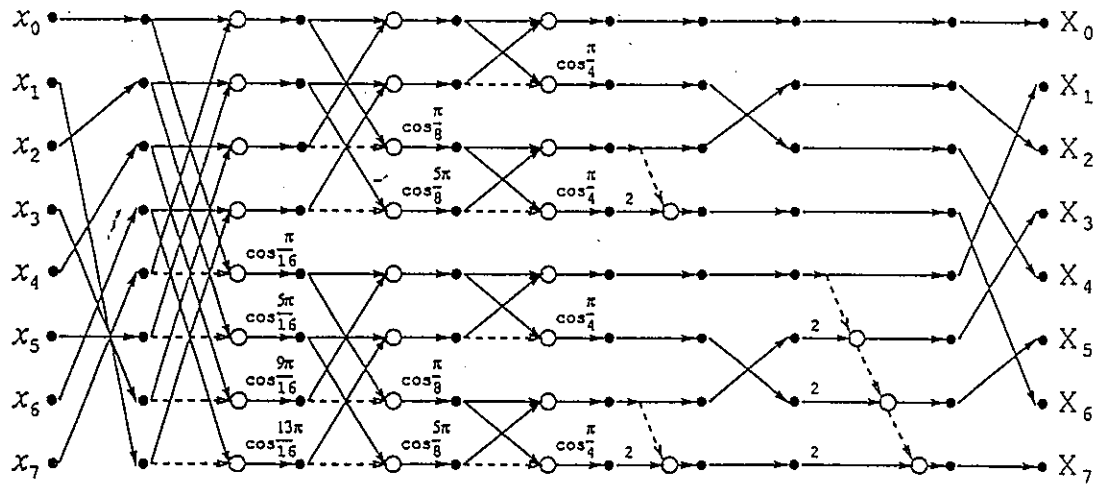
Fig. 1. Implementation of Step 2, which computes $\begin{bmatrix} X_e \\ X_o \end{bmatrix} = \hat{C}_N \begin{bmatrix} x_e \\ x_{\bar{o}} \end{bmatrix}$, in the DCT recursive algorithm. Solid lines represent transfer factor 1, while dashed lines represent transfer factor -1. Circles o represent adders.



(a)



(b)



(c)

Fig. 2. The DCT signal-flow graphs for (a) $N = 2$, (b) $N = 4$, and (c) $N = 8$.

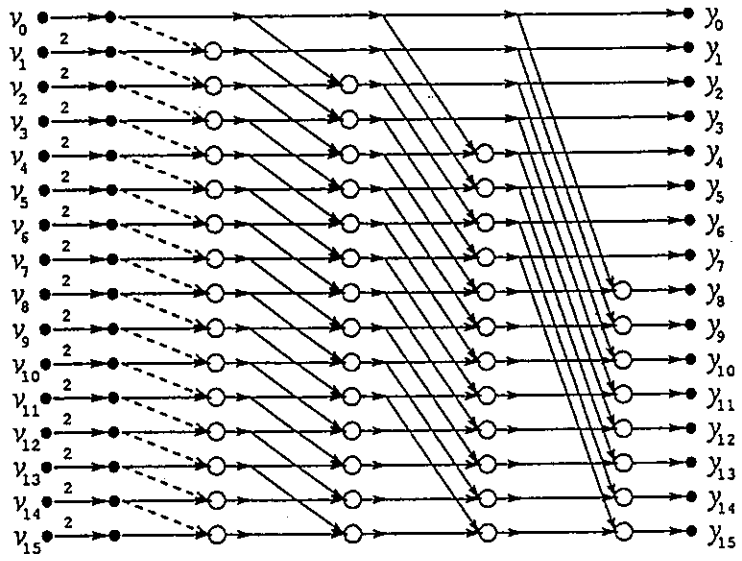


Fig. 3. The signal-flow graph for $N = 16$, in which the problem $L_N v$ can be computed in only one parallel shift step and $\log N$ $(= 4)$ parallel addition steps based on the recursive doubling algorithm.

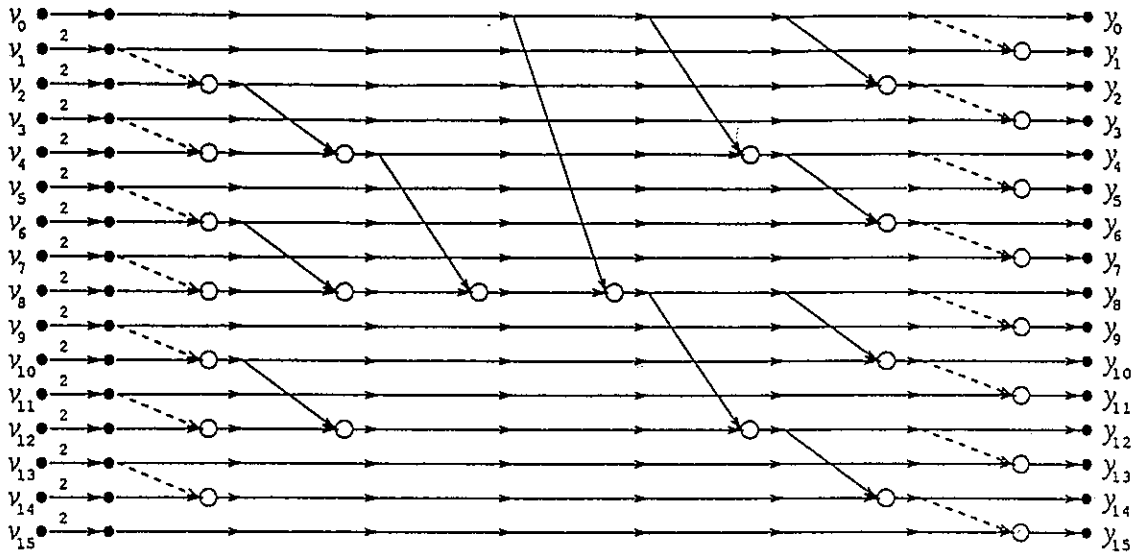


Fig. 4. The signal-flow graph for $N = 16$, in which the problem $L_N v$ can be computed in only one parallel shift step and $2(\log N) - 1$ $(= 7)$ parallel addition steps based on the cyclic reduction algorithm.

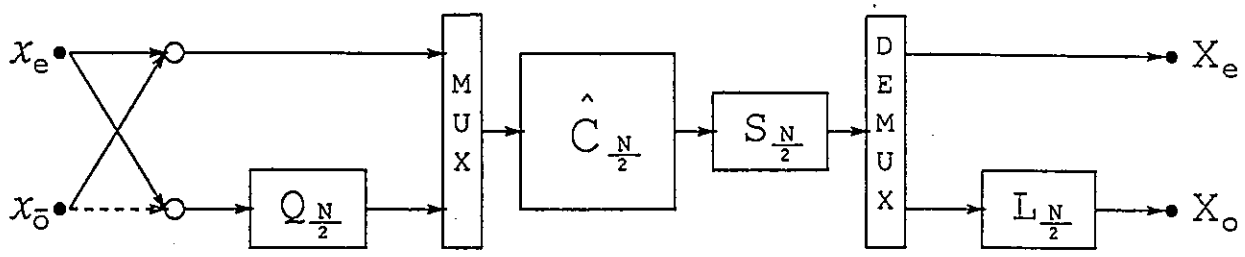


Fig. 5. Proposed hardware implementation for the DCT.

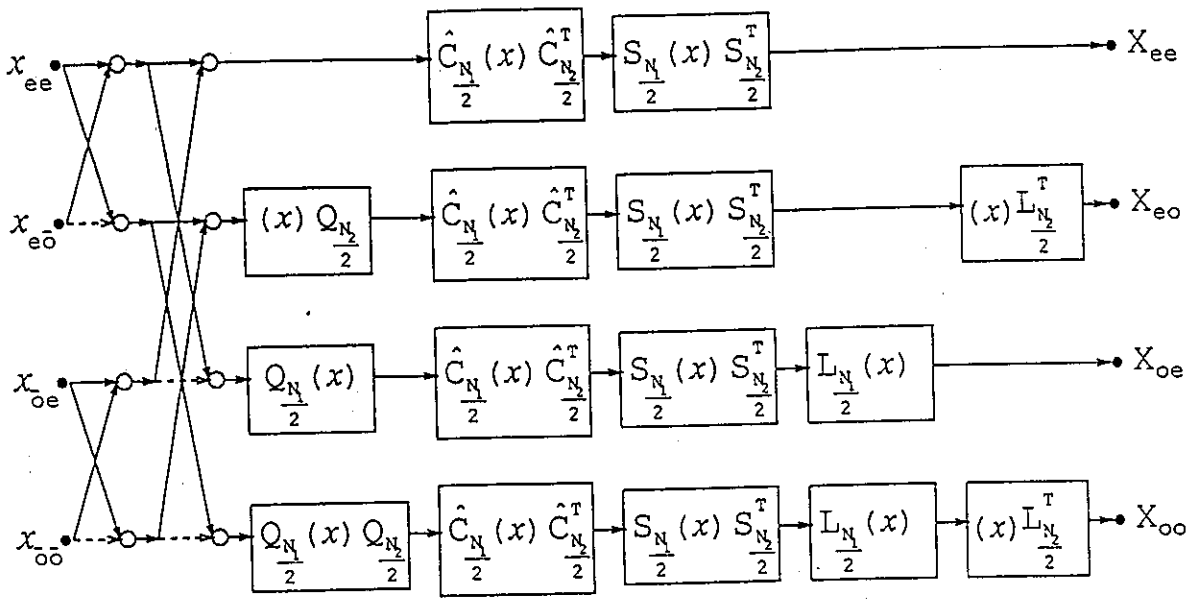
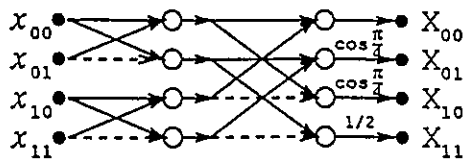
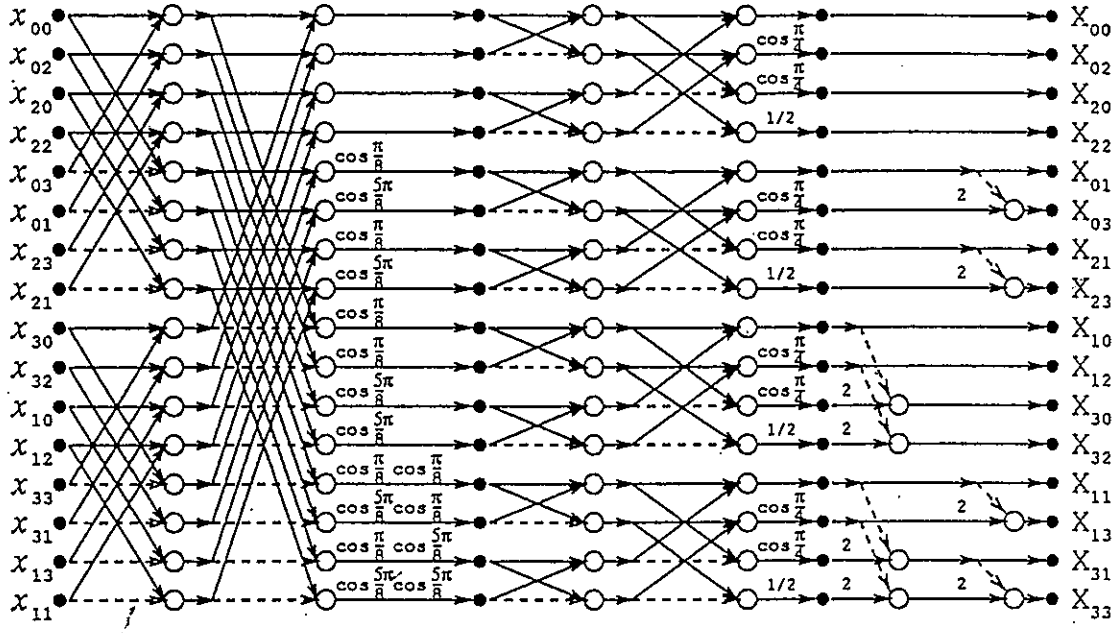


Fig. 6. Implementation of Step 2, which computes $\begin{bmatrix} X_{ee} & X_{eo} \\ X_{oe} & X_{oo} \end{bmatrix} = \hat{C}_{N_1} \begin{bmatrix} x_{ee} & x_{e\bar{o}} \\ x_{\bar{o}e} & x_{\bar{o}\bar{o}} \end{bmatrix} \hat{C}^T_{N_2}$, in the 2-D DCT recursive algorithm. (x)'s represent current input $\frac{N_1}{2} \times \frac{N_2}{2}$ submatrices.



(a)



(b)

Fig. 7. The $N \times N$ 2-D DCT signal-flow graphs for (a) $N = 2$ and (b) $N = 4$, where each of the $\frac{N}{2} \times \frac{N}{2}$ data subsets are arranged into a 1-D array in lexicographical order for illustration.

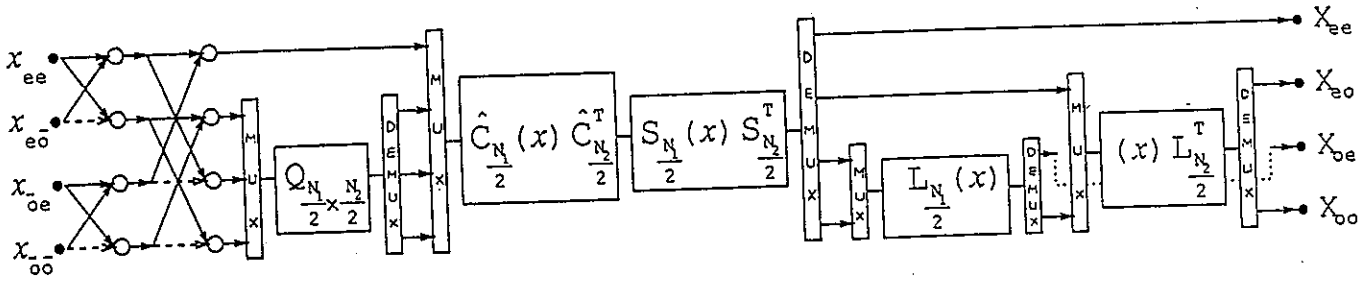


Fig. 8. Proposed hardware implementation for the 2-D DCT.