Two-stage circular-convolution-like algorithm/architecture for the discrete cosine transform

W.-J. Duh
J.-L. Wu

Indexing terms: Signal processing, Algorithms, Transforms

Abstract: Because of the great improvements in computer engineering, digital signal processing has been getting more and more important in recent years. Since discrete transformations play a significant role in digital signal processing, they have found many applications in various fields. The discrete cosine transform (DCT) is well known for its usefulness in the fields of image processing and data compression. With recent advances in the ISDN, limited communication bandwidth has become a new bottleneck, a possible solution to which may be an efficient encoding algorithm/architecture. The paper presents a two-stage algorithm and its corresponding architectures for efficient computation of a power-of-two length DCT. In this approach, the transform matrix of the DCT is decomposed into the product of two matrices, the preprocessing and the postprocessing ones. The elements in the preprocessing stage consist of 1, −1, and 0 only; the postprocessing stage is of block diagonal form in which each block performs a circular-convolution-like (CCL) operation. Thus, both stages can be implemented efficiently either by software or hardware. Details of the matrix decomposition are described and several corresponding architectures are also presented.

1 Introduction

Since its introduction the discrete cosine transform (DCT) [1] has found a number of applications in image processing [2] and in speech processing [3]. It has also been shown that the DCT has a performance very close to the statistically optimal Karhunen–Loève transform (KLT) for a large number of signal classes [4, 5]. Therefore, many fast algorithms for computing the DCT have been derived. A comprehensive review of various fast DCT algorithms was given in Reference 6 and a few novel recursive DCT algorithms were presented in References 7 and 8. Since they are mostly used in various signal-processing applications, DCT processors with real-time computation capabilities are urgently required. DCT architectures have been proposed to meet this requirement, such as the distributed arithmetics DCT of Sun *et al.* [9], the concurrent DCT [10] and the constant-rotation DCT of the authors [11].

Paper 7514F (E5), first received 19th December 1989 and in revised form 24th April 1990

The authors are with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 10764, Taiwan, Republic of China

In this paper, a two-stage algorithm/architecture for computing a power-of-two length DCT is proposed. The preprocessing stage consists of values 1, −1, and 0. The postprocessing stage is of block diagonal form with maximum block size $N/2$. Each block is of circular-convolution-like (CCL) form. The difference between the circular-convolution and the CCL lies in the sign of the elements. The postprocessing stage is totally parallel among blocks and can be realised by using the modified contemporary circular-convolution architecture [12, 13] within each block. Thus, both the pre- and postprocessing stages can be implemented efficiently. The system block diagram of the proposed algorithm/architecture is shown in Fig. 1.

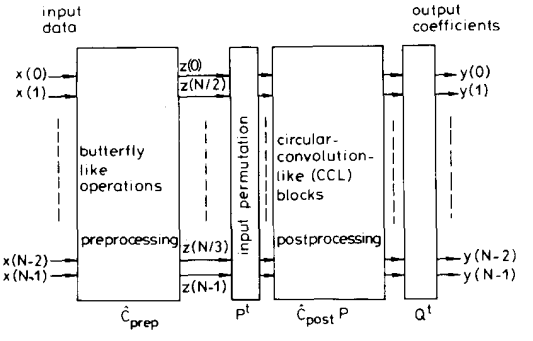

Fig. 1 *System block diagram of two-stage CCL DCT*

2 Two-stage matrix decomposition

The N-point DCT is defined as follows:

$$y(n) = \frac{2c(n)}{N} \sum_{k=0}^{N-1} x(k) \cos\left(\frac{\pi(2k+1)n}{2N}\right) \quad (1)$$

where N is the transform length, $n, k \in \{0, 1, \ldots, N-1\}$, and

$$c(n) = \begin{cases} \dfrac{1}{\sqrt{2}}, & \text{if } n = 0 \\ 1, & \text{otherwise} \end{cases} \quad (2)$$

The corresponding inverse transformation is

$$x(k) = \sum_{n=0}^{N-1} c(n)y(n) \cos\left(\frac{\pi(2k+1)n}{2N}\right) \quad (3)$$

For the purpose of simplicity, the transformation is represented in matrix-vector form. The DCT transform matrix, with length N, is denoted as C_N. An input column vector $X = [x_0, x_1, \ldots, x_{N-1}]^t$ is transformed to

the output vector, denoted by Y, as

$$Y = D_N C_N X \tag{4}$$

where $D_N = (2/N)$ diag $(c(0), c(1), \ldots, c(N-1))$.

For the purpose of simplicity, the algorithm discussed in this paper is focussed on the transform matrix C_N.

2.1 Index partitions

Consider the DCT with power-of-two length only, that is $N = 2^m$, where m is a positive integer. Some definitions and lemmas for the index partition are given below.

Definition 1: Z_m stands for the set of integers $\{0, 1, \ldots, 2^m - 1\}$, where m is a positive integer.

Lemma 1: The integer set Z_m is the disjoint union of the subsets A_i:

$$A_i = \{2^i(2j+1)\}$$

where $j \in Z_{m-i-1}$ and $i \in \{0, 1, \ldots, m-1\}$, and $A_m = \{0\}$, i.e.

$$Z_m = \bigcup_{i=0}^{m} A_i$$

and $A_i \cap A_j = \phi$, for $i \neq j$.

Proof: See Appendix 8.1.

For example, when $m = 4$, the set Z_4 can be decomposed into $A_0 = \{1, 3, 5, 7, 9, 11, 13, 15\}$, $A_1 = \{2, 6, 10, 14\}$, $A_2 = \{4, 12\}$, $A_3 = \{8\}$ and $A_4 = \{0\}$.

Definition 2: Let $g(x) = 2x + 1$, where $x \in Z_{m-1}$ and g is a mapping from Z_{m-1} to A_0.

Since the transform kernels of DCT are cosine functions, they possess the same symmetric properties, such as $\cos(\pi/2 - \theta) = -\cos(\pi/2 + \theta)$ for $\theta \in [0, \pi/2]$. Using the symmetric property, a function $f(g(k), n)$ can be defined to explore the symmetry of the transform matrix.

Definition 3:

$$f(k, n) = \text{abs}\,(kn \bmod_{2N}) \tag{5}$$

where k, n, and $f(k, n) \in \{0, 1, \ldots, N-1, N\}$. The function abs (.) denotes absolute value and the operation $\bmod_{2N}$ is similar to that of modulo $2N$ except that the valid range is in $\{-N+1, \ldots, -1, 0, 1, \ldots, N\}$.

The definition of $f(g(k), n)$ is to formalise the index mapping of the DCT. We ignore the sign of the cosine functions in the transform kernels: only the absolute values are considered. Hence, the argument k, n of function $f(g(k), n)$ denote the input and output indices, respectively, and the corresponding value is the angle to be computed in the transform. As indicated in definition 3, only the absolute values are involved. Thus, the sign signals are treated as additional control signals for the convolvers and are described in detail in the next Section. With the aid of the index mapping function defined in definition 3, we proceed to partition the index space to achieve higher order parallelism.

Lemma 2: For all $n \in A_i$ and $k \in A_0$, the values of $f(k, n) \in A_i$.

Proof: See Appendix 8.2.

Lemma 2 explores the parallelism implied in the transform kernels of the DCT. That is, to compute any output value $y(n)$, only those angles in A_i are required where $n \in A_i$. Since lemma 1 forms an index partition of the index set Z_m into $m+1$ subsets with maximum size $N/2$, the transform matrix C_N can be decomposed into $m+1$ independent submatrices which can be computed totally in parallel.

2.2 Transform matrix decomposition

For simplicity, let $\hat{C}_N = QC_N$ where Q is a proper permutation matrix for output indices. In the following, subscripts are used to denote the corresponding ranks of the matrices. The matrix $\hat{C}_N$ can be factorised as follows [14]:

$$\begin{aligned}\hat{C}_N &= \begin{bmatrix} \hat{C}_{N/2} & \hat{C}_{N/2} I'_{N/2} \\ \hat{R}_{N/2} & -\hat{R}_{N/2} I'_{N/2} \end{bmatrix} \\ &= \begin{bmatrix} \hat{C}_{N/2} & 0 \\ 0 & \hat{R}_{N/2} \end{bmatrix} \begin{bmatrix} I_{N/2} & I_{N/2} \\ I_{N/2} & -I_{N/2} \end{bmatrix} \begin{bmatrix} I_{N/2} & 0 \\ 0 & I_{N/2} \end{bmatrix}\end{aligned} \tag{6}$$

where $I'_{N/2}$ denotes the opposite diagonal identity matrix. It is clear that the submatrix $\hat{C}_{N/2}$ can be factorised recursively. The submatrix $\hat{R}_{N/2}$ is left unfactorised as the maximum block of the postprocessing stage. After $(\log_2 N) - 1$ iterations, the matrix is decomposed into two stages. The preprocessing stage consists of data reversing and butterfly operations only, however the postprocessing stage is of block diagonal form. In eqn. 6, the second matrix on the right-hand side performs the butterfly operations often seen in FFT algorithms, and the third matrix will reverse the lower half data.

For simplicity, we denote $\cos(\theta\pi/32)$ as C_θ. Therefore, for example, $\hat{C}_{16}$ becomes

$$\begin{bmatrix}
C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 \\
C_8 & -C_8 & -C_8 & C_8 & C_8 & -C_8 & -C_8 & C_8 & C_8 & -C_8 & -C_8 & C_8 & C_8 & -C_8 & -C_8 & C_8 \\
C_4 & C_{12} & -C_{12} & -C_4 & -C_{12} & -C_4 & C_4 & C_{12} & C_4 & C_{12} & -C_{12} & -C_4 & -C_{12} & -C_4 & C_4 & C_{12} \\
C_{12} & -C_4 & C_4 & -C_{12} & -C_4 & C_{12} & -C_{12} & C_4 & C_{12} & -C_4 & C_4 & -C_{12} & -C_4 & C_{12} & -C_{12} & C_4 \\
C_2 & C_6 & C_{10} & C_{14} & -C_{14} & -C_{10} & -C_6 & -C_2 & C_2 & C_6 & C_{10} & C_{14} & -C_{14} & -C_{10} & -C_6 & -C_2 \\
C_6 & -C_{14} & -C_2 & -C_{10} & C_{10} & C_2 & C_{14} & -C_6 & C_6 & -C_{14} & -C_2 & -C_{10} & C_{10} & C_2 & C_{14} & -C_6 \\
C_{14} & -C_{10} & C_6 & -C_2 & C_2 & -C_6 & C_{10} & -C_{14} & C_{14} & -C_{10} & C_6 & -C_2 & C_2 & -C_6 & C_{10} & -C_{14} \\
C_{10} & -C_2 & C_{14} & C_6 & -C_6 & -C_{14} & C_2 & -C_{10} & C_{10} & -C_2 & C_{14} & C_6 & -C_6 & -C_{14} & C_2 & -C_{10} \\
C_1 & C_3 & C_5 & C_7 & C_9 & C_{11} & C_{13} & C_{15} & -C_{15} & -C_{13} & -C_{11} & -C_9 & -C_7 & -C_5 & -C_3 & -C_1 \\
C_3 & C_9 & C_{15} & -C_{11} & -C_5 & C_1 & C_7 & C_{13} & -C_{13} & -C_7 & -C_1 & C_5 & C_{11} & -C_{15} & -C_9 & -C_3 \\
C_9 & -C_5 & C_{13} & -C_1 & -C_{15} & C_3 & -C_{11} & C_7 & -C_7 & C_{11} & -C_3 & C_{15} & C_1 & -C_{13} & C_5 & -C_9 \\
C_5 & C_{15} & -C_7 & C_3 & C_{13} & -C_9 & C_1 & C_{11} & -C_{11} & -C_1 & C_9 & -C_{13} & -C_3 & C_7 & -C_{15} & -C_5 \\
C_{15} & C_{13} & C_{11} & C_9 & C_7 & C_5 & C_3 & C_1 & -C_1 & -C_3 & -C_5 & -C_7 & -C_9 & -C_{11} & -C_{13} & -C_{15} \\
C_{13} & C_7 & C_1 & -C_5 & -C_{11} & C_{15} & C_9 & C_3 & -C_3 & -C_9 & -C_{15} & C_{11} & C_5 & -C_1 & -C_7 & -C_{13} \\
C_7 & -C_{11} & C_3 & -C_{15} & -C_1 & C_{13} & -C_5 & C_9 & -C_9 & C_5 & -C_{13} & C_1 & C_{15} & -C_3 & C_{11} & -C_7 \\
C_{11} & C_1 & -C_9 & C_{13} & C_3 & -C_7 & C_{15} & C_5 & -C_5 & -C_{15} & C_7 & -C_3 & -C_{13} & C_9 & -C_1 & -C_{11}
\end{bmatrix} \tag{7}$$

Now, if the permutation matrix $\boldsymbol{Q}$ is chosen to be

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0
\end{bmatrix} \tag{8}$$

then the preprocessing stage of a 16-point DCT can be factorised as

$$\left[\begin{array}{cc|c} 1 & 1 & \\ 1 & -1 & \mathbf{0} \\ \hline \multicolumn{2}{c|}{\mathbf{0}} & I_{14} \end{array}\right]\left[\begin{array}{cc|c} I_2 & I_2 & \\ I_2 & -I_2 & \mathbf{0} \\ \hline \multicolumn{2}{c|}{\mathbf{0}} & I_{12} \end{array}\right]\left[\begin{array}{cc|c} I_2 & \mathbf{0} & \\ \mathbf{0} & I_2' & \mathbf{0} \\ \hline \multicolumn{2}{c|}{\mathbf{0}} & I_{12} \end{array}\right]$$

$$\left[\begin{array}{cc|c} I_4 & I_4 & \\ I_4 & -I_4 & \mathbf{0} \\ \hline \multicolumn{2}{c|}{\mathbf{0}} & I_8 \end{array}\right]\left[\begin{array}{cc|c} I_4 & \mathbf{0} & \\ \mathbf{0} & I_4' & \mathbf{0} \\ \hline \multicolumn{2}{c|}{\mathbf{0}} & I_8 \end{array}\right]\begin{bmatrix} I_8 & I_8 \\ I_8 & -I_8 \end{bmatrix}\begin{bmatrix} I_8 & \mathbf{0} \\ \mathbf{0} & I_8' \end{bmatrix} \tag{9}$$

The corresponding postprocessing matrix is of block diagonal form as shown below:

$$\begin{bmatrix}
C_0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&C_8&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&C_4&C_{12}&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&C_{12}&-C_4&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&C_2&C_6&C_{10}&C_{14}&0&0&0&0&0&0&0&0\\
0&0&0&0&C_6&-C_{14}&-C_2&-C_{10}&0&0&0&0&0&0&0&0\\
0&0&0&0&C_{10}&-C_2&C_{14}&C_6&0&0&0&0&0&0&0&0\\
0&0&0&0&C_{14}&-C_{10}&C_6&-C_2&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&C_1&C_3&C_5&C_7&C_9&C_{11}&C_{13}&C_{15}\\
0&0&0&0&0&0&0&0&C_3&C_9&C_{15}&-C_{11}&-C_5&C_1&C_7&C_{13}\\
0&0&0&0&0&0&0&0&C_9&-C_5&C_{13}&-C_1&-C_{15}&C_3&-C_{11}&C_7\\
0&0&0&0&0&0&0&0&C_5&C_{15}&-C_7&C_3&C_{13}&-C_9&C_1&C_{11}\\
0&0&0&0&0&0&0&0&C_{15}&C_{13}&C_{11}&C_9&C_7&C_5&C_3&C_1\\
0&0&0&0&0&0&0&0&C_{13}&C_7&C_1&-C_5&-C_{11}&C_{15}&C_9&C_3\\
0&0&0&0&0&0&0&0&C_7&-C_{11}&C_3&-C_{15}&-C_1&C_{13}&-C_5&C_9\\
0&0&0&0&0&0&0&0&C_{11}&C_1&-C_9&C_{13}&C_3&-C_7&C_{15}&C_5
\end{bmatrix} \tag{10}$$

From eqn. 6, a recursive algorithm for computing the preprocessing stage can be derived easily. Fig. 2 shows the signal flow graph of the resultant fast algorithm of the preprocessing stage of a 16-point example, Fig. 3 shows the postprocessing stage.

2.3 Circular-convolution-like postprocessing stage

One way to achieve fast computations is to decompose further the submatrices in expr. 10 [8, 14]. In this paper, we present another approach from the view point of architecture rather than the number of operations. That is, with appropriate input/output permutations, each block in the postprocessing stage can be computed utilising the circular-convolution hardwares [12, 13].

Since A_i corresponds to a block in the postprocessing stage of size $|A_i|$, only A_0, the largest block, will be discussed here. Following the same strategy, other blocks can be constructed without difficulty.

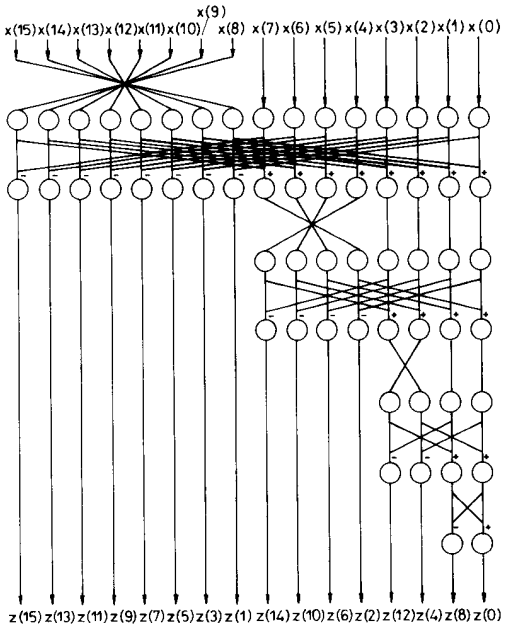

Fig. 2 *Preprocessing stage of 16-point two-stage CCL DCT*

From lemma 2, it is clear that the input indices $g(k) \in A_0$, the output indices $n \in A_0$, and the corresponding rotation angle $f(g(k), n) \in A_0$. Thus, according to the definition of $f(g(k), n)$, a binary operator can be defined as shown in definition 4.

Definition 4: $\otimes_N$ is a binary operator defined over the set A_0. The definition of $\otimes_N$ is

$$k \otimes_N n = \text{abs}\,(kn \bmod_{2N}) \tag{11}$$

where $\bmod_{2N}$ is defined in definition 3.

After defining the operator $\otimes_N$, the relationship between input and output indices has been explored. The following lemma illustrates the so-called circular-convolution-like (CCL) properties.

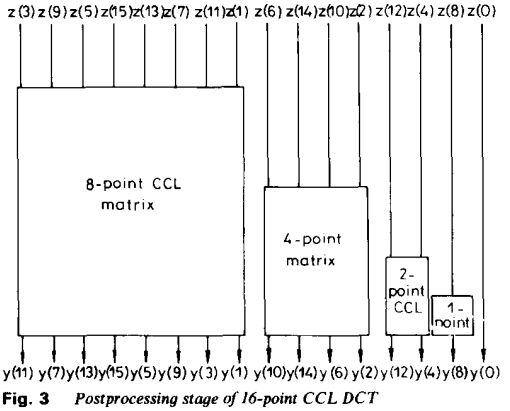

Fig. 3 *Postprocessing stage of 16-point CCL DCT*

Lemma 3: $G_N = \{A_0; \otimes_N\}$ is a cyclic group, and 3 is a generator of the group.

Proof: See Appendix 8.3.

From lemma 3, the generator 3 can generate the sequence of the cyclic group as shown in Table 1 [15, 16]. It is clear that the circular-correlation matrix is of the same form as given in Table 1. It is well known that time-reversion of circular-correlation becomes circular-convolution. Therefore, if the output indices are permuted according to the sequence generated by 3 and input indices are permuted as time reversal of the new output orders, each block in the postprocessing stage is a CCL block whose length is the size of the block. Note that in definition 4 only the absolute values of the cosine functions are formalised, hence sign factors are excluded from the operator $\otimes_N$. Therefore, the cyclic group forms a circular-convolution-like matrix and can be computed efficiently by using modified circular-convolution hardwares [12, 13].

Table 1: Operation table of a cylic group

$\otimes$	*e*	*a*	*b*	*c*
e	*e*	*a*	*b*	*c*
a	*a*	*b*	*c*	*e*
b	*b*	*c*	*e*	*a*
c	*c*	*e*	*a*	*b*

e denotes identity and *a* denotes generator

3 Additional controls of the convolver

From the preceding section, it is clear that in the proposed two-stage DCT, the postprocessing stage is a block diagonal matrix with maximum block size $N/2$. Each block is of CCL form. The reason why we call it CCL is that the matrix is of circular-convolution form except for some sign changes. The CCL form can be formulated as follows:

$$y(n) = \sum_{k=0}^{N-1} \text{sg}(k, n)x((n-k) \bmod N)h(k) \tag{12}$$

where $y(n)$ denotes the output, and $x(.)$ and $h(.)$ denote the two input sequences for convolving. The function sg(.) controls the type of convolution. When sg(.) = 1, eqn. 12 represents a circular convolution, otherwise, if sg (k, n) = sign $(n - k)$, eqn. 12 becomes a skew circular convolution, where

$$\text{sign}(x) = \begin{cases} 1, & \text{if } x \geqslant 0 \\ -1, & \text{if } x < 0 \end{cases} \tag{13}$$

The function sg(.) for CCL is defined in eqn. 14. It can be treated as the control signals of the CCL architecture [12, 13]. Note that the magnitudes of the postprocessing functions are defined by the operator $\otimes_N$ and the corresponding signs are defined by sg(.).

$$\text{sg}(k, n) = \text{sign}(g(k)n + N \bmod_{4N}) \tag{14}$$

As the transform kernels of the DCT are defined on the $4N$ equal divisions of the unit circle, the function sg(.) is defined according to the fundamental properties of the cosine function. The first and the fourth quadrants are positive, the others are negative.

4 Realisation considerations for the CCL matrix

Since the postprocessing stage is composed of $m + 1$ CCL submatrices, some simple architectures, such as semisystolic [12] and systolic arrays, and transversal filters [13] can be used to calculate the results. The relationship among various hardware structures and the corresponding matrix representations will be discussed in detail in this Section.

4.1 Semisystolic arrays for the CCL matrix

The operation of the maximal CCL block of the 16-point DCT can be expressed as a matrix vector product as follows:

$$\begin{bmatrix} y(1) \\ y(3) \\ y(9) \\ y(5) \\ y(15) \\ y(13) \\ y(7) \\ y(11) \end{bmatrix} = \begin{bmatrix} C_1 & C_{11} & C_7 & C_{13} & C_{15} & C_5 & C_9 & C_3 \\ C_3 & C_1 & -C_{11} & C_7 & C_{13} & C_{15} & -C_5 & C_9 \\ C_9 & C_3 & -C_1 & -C_{11} & C_7 & C_{13} & -C_{15} & -C_5 \\ C_5 & -C_9 & C_3 & C_1 & C_{11} & -C_7 & C_{13} & C_{15} \\ C_{15} & C_5 & C_9 & C_3 & C_1 & C_{11} & C_7 & C_{13} \\ C_{13} & C_{15} & -C_5 & C_9 & C_3 & C_1 & -C_{11} & C_7 \\ C_7 & C_{13} & -C_{15} & -C_5 & C_9 & C_3 & -C_1 & -C_{11} \\ C_{11} & -C_7 & C_{13} & C_{15} & C_5 & -C_9 & C_3 & C_1 \end{bmatrix} \begin{bmatrix} z(1) \\ z(11) \\ z(7) \\ z(13) \\ z(15) \\ z(5) \\ z(9) \\ z(3) \end{bmatrix} \tag{15}$$

Since the DCT kernels are used in the matrix and the input signals in the column vector, then summing column-wise will lead to the semisystolic array implementation shown in Fig. 4. Note that in Fig. 4 input data broadcasting is necessary. The sequence of input signals are rearranged according to the sequence generated by 3. The partial sums are pumped through the array at each clock pulse. Since the circular convolution operation is commutative, interchanging the roles of input signals and DCT kernels will still result in the desired DCT coefficients.

In practical usage, the transform kernels are fixed and input signals are changed with time. Therefore, specific multipliers can be used to reduce the complexities of the implementation. Table look-up and power-of-two approximation [13] are good alternatives.

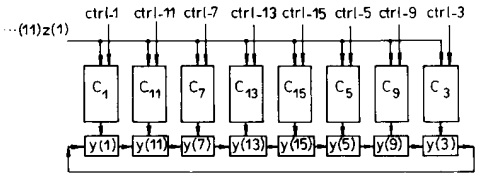

Fig. 4 *Semisystolic structure for realising the CCL matrix*
Input signal broadcasting is used

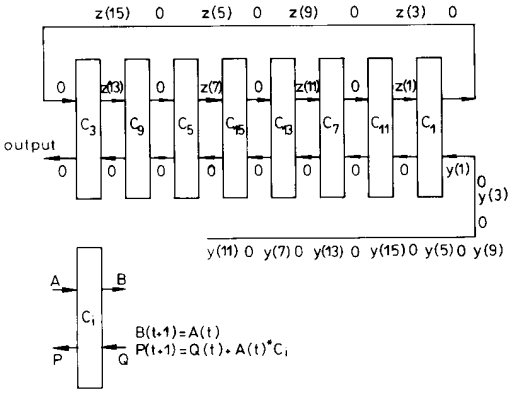

Fig. 5 *Systolic array for computing the CCL matrix using input signal circulation*

4.2 Systolic arrays for CCL matrix

Eqn. 15 can be reformulated as

$$\begin{bmatrix} y(1) \\ y(3) \\ y(9) \\ y(5) \\ y(15) \\ y(13) \\ y(7) \\ y(11) \end{bmatrix} = \begin{bmatrix} C_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_{11} & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_7 & -C_{11} & -C_1 & 0 & 0 & 0 & 0 & 0 \\ C_{13} & C_7 & -C_{11} & C_1 & 0 & 0 & 0 & 0 \\ C_{15} & C_{13} & C_7 & C_{11} & C_1 & 0 & 0 & 0 \\ C_5 & C_{15} & C_{13} & -C_7 & C_{11} & C_1 & 0 & 0 \\ C_9 & -C_5 & -C_{15} & C_{13} & C_7 & -C_{11} & -C_1 & 0 \\ C_3 & C_9 & -C_5 & C_{15} & C_{13} & C_7 & -C_{11} & C_1 \\ 0 & C_3 & C_9 & C_5 & C_{15} & C_{13} & C_7 & C_{11} \\ 0 & 0 & C_3 & -C_9 & C_5 & C_{15} & C_{13} & -C_7 \\ 0 & 0 & 0 & C_3 & C_9 & -C_5 & -C_{15} & C_{13} \\ 0 & 0 & 0 & 0 & C_3 & C_9 & -C_5 & C_{15} \\ 0 & 0 & 0 & 0 & 0 & C_3 & C_9 & C_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & C_3 & -C_9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & C_3 \end{bmatrix}^t \begin{bmatrix} z(1) \\ z(11) \\ z(7) \\ z(13) \\ z(15) \\ z(5) \\ z(9) \\ z(3) \\ z(1) \\ z(11) \\ z(7) \\ z(13) \\ z(15) \\ z(5) \\ z(9) \end{bmatrix} \tag{16}$$

Therefore the 1-D systolic array for computing CCL can be easily derived as shown in Fig. 5. From Fig. 5, it is obvious that the hardware utilisation is only 50%. To promote the utilisation, one can interleave two input sequences and results in two interleaved convolved sequences. Another solution is to use a two-phase clock instead of data interleaving.

Interchanging input signals and DCT kernels results in another structure. Another matrix vector product form for CCL is shown in eqn. 17 and leads to the array in Fig. 6, which circulates partial sums instead of input signals.

4.3 Transversal filter structures for the CCL matrix

From the CCL matrix shown in eqn. 15, summing row-wise instead of column-wise leads to the transversal filter structures shown in Fig. 7. Interchanging DCT kernels and input signals leads to another filter implementation.

4.4 Comparisons

The above-mentioned three structures used to realise the CCL matrix are compared in Table 2.

The systolic array implementation is free from pin-out limitation but requires additional latches and takes about

$$\begin{bmatrix} y(1) \\ y(3) \\ y(9) \\ y(5) \\ y(15) \\ y(13) \\ y(7) \\ y(11) \\ y(1) \\ y(3) \\ y(9) \\ y(5) \\ y(15) \\ y(13) \\ y(7) \end{bmatrix} = \begin{bmatrix} C_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_3 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_9 & C_3 & -C_1 & 0 & 0 & 0 & 0 & 0 \\ C_5 & -C_9 & C_3 & C_1 & 0 & 0 & 0 & 0 \\ C_{15} & C_5 & C_9 & C_3 & C_1 & 0 & 0 & 0 \\ C_{13} & C_{15} & -C_5 & C_9 & C_3 & C_1 & 0 & 0 \\ C_7 & C_{13} & -C_{15} & -C_5 & C_9 & C_3 & -C_1 & 0 \\ C_{11} & -C_7 & C_{13} & C_{15} & C_5 & -C_9 & C_3 & C_1 \\ 0 & C_{11} & C_7 & C_{13} & C_{15} & C_5 & C_9 & C_3 \\ 0 & 0 & -C_{11} & C_7 & C_{13} & C_{15} & -C_5 & C_9 \\ 0 & 0 & 0 & -C_{11} & C_7 & C_{13} & -C_{15} & -C_5 \\ 0 & 0 & 0 & 0 & C_{11} & -C_7 & C_{13} & C_{15} \\ 0 & 0 & 0 & 0 & 0 & C_{11} & C_7 & C_{13} \\ 0 & 0 & 0 & 0 & 0 & 0 & -C_{11} & C_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -C_{11} \end{bmatrix} \begin{bmatrix} z(1) \\ z(11) \\ z(7) \\ z(13) \\ z(15) \\ z(5) \\ z(9) \\ z(3) \end{bmatrix} \tag{17}$$

twice the number of clock periods. The semisystolic array suffers from signal broadcasting and the shortcoming of the transversal filter is the requirement for global summation. It is clear that for low-speed applications the systolic array is a good choice, yet for high-speed usages the semisystolic array and transversal filter structures are more attractive.

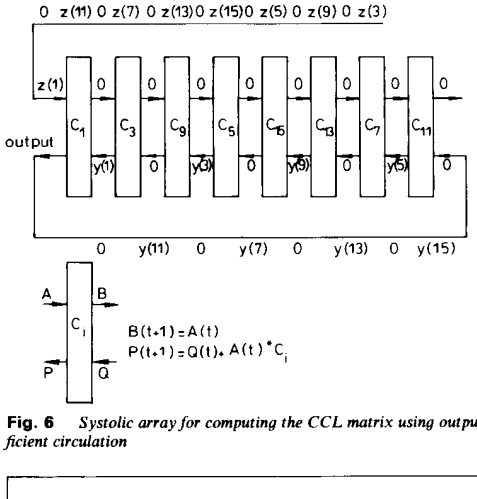

Fig. 6 *Systolic array for computing the CCL matrix using output coefficient circulation*

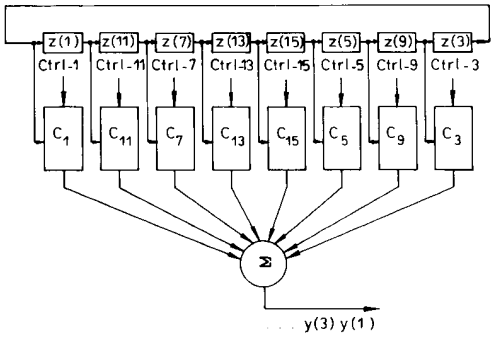

Fig. 7 *Transversal filter for computing CCL matrix using input signal circulation*

Table 2: Comparison between semisystolic array, systolic array and transversal filter

	Semisystolic	Systolic	Transversal filter
operation mode	serial-in parallel-out	serial-in serial-out	parallel-in serial-out
no. of clocks	N	$2N-1$	N
no. of adders	N	N	$N-1$
disadvantage	broadcasting	low-speed additional latches	global summation

5 16-point DCT example

From the discussions above, a two-stage decomposition of the DCT transform matrix has been derived completely. The preprocessing stage consists of data permutations and butterfly operations; the postprocessing stage comprises several CCL blocks which can be realised by modified convolvers in parallel. To clarify the operation of the algorithm, a specific 16-point DCT example will now be given.

5.1 Two-stage decomposition

The 16-point DCT transform matrix C_{16} can be factorised into preprocessing, input permutation, postprocessing and output permutation stages as

$$\begin{aligned} C_{16} &= Q^t \hat{C}_{16} \\ &= Q^t(\hat{C}_{post} P) P^t \hat{C}_{pre} \end{aligned} \tag{18}$$

The permutation matrices P^t and Q^t denote the transposition matrices of P and Q, respectively. P^t reorders the input and Q^t the output, with respect to CCL matrices. The matrix Q can be generated by using generator 3 under the operator $\otimes_{16}$ described in lemma 3.

The permutation matrix P is partitioned into $m+1$ submatrices using lemma 1 and its transposition is shown in eqn. 21. By time-reversing the input indices, the cyclic group form shown in Table 1 becomes a circular-convolution form as shown in Table 3. Hence, P reverses the orders for each nonzero block and can be computed using eqns. 19 and 20:

$$P^t = TQ \tag{19}$$

where T, for the 16-point example, is

$$\begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \end{bmatrix} \tag{20}$$

Therefore, the input permutation matrix P^t becomes

$$\begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0 \end{bmatrix} \tag{21}$$

Table 3: Operation table of the 4-point CCL block showing circular-convolution form

⊗	1	5	7	3
1	1	5	7	3
3	3	1	5	7
7	7	3	1	5
5	5	7	3	1

From eqn. 18, it is obvious that the computation of the 16-point DCT can be divided into four stages as shown in Fig. 1. Hence, the algorithm/architecture for computing the DCT becomes:

(*a*) compute the simple preprocessing stage $\hat{C}_{pre}$. The fast computational algorithm is illustrated in Fig. 2 and described in eqn. 9.

(*b*) permute the results of $\hat{C}_{prep}$ based on P^t as given in eqn. 21

(*c*) compute the postprocessing stage, say $\hat{C}_{post}P$, using CCL hardwares described in the previous Section. The block diagram of the stage is shown in Fig. 3

(*d*) reorder the output coefficients based on Q^t, which is given in expr. 8.

In the 16-point DCT example, there are 8-point, 4-point, 2-point and two 1-point blocks. Each block corresponds to one of the subsets described in lemma 1. It can be easily verified that the 4-point block of the 16-point DCT is the same as that of the 8-point DCT. Therefore, each CCL block can be constructed using the same strategy. Since 3 is used to generate the output order of the 8-point CCL block, that is $\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5, 3^6, 3^7\}_{under\ \otimes_{16}}$, the result is {1, 3, 9, 5, 15, 13, 7, 11}. In other words, the relative output order within the block can be renumbered using $g^{-1}(3^i\ under\ \otimes_{16})$ and becomes {0, 1, 4, 2, 7, 6, 3, 5} in this example; the corresponding input order is {1, 11, 7, 13, 15, 5, 9, 3}, the time-reversal of {1, 3, 9, 5, 15, 13, 7, 11}. Thus, the relative input order within the 8-point block is {0, 5, 3, 6, 7, 2, 4, 1}, the time-reversal of {0, 1, 4, 2, 7, 6, 3, 5}. The block diagram of the example is illustrated in Fig. 8.

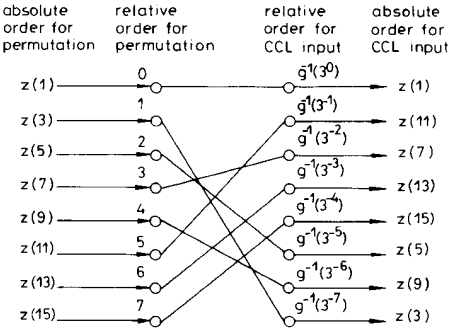

Fig. 8 *Input ordering for {0, 5, 3, 6, 7, 2, 4, 1}*

6 Conclusions

A novel, fast, parallel algorithm/architecture for computing the DCT has been proposed. The derivation of the fast algorithm is based on the decomposition of the DCT transform matrix into pre- and postprocessing stages. The preprocessing stage consists of +1, −1, and 0 and it can be factorised into $(2\log_2 N) - 1$ successive stages (including $(\log_2 N) - 1$ half data-reversing stages and $\log_2 N$ stages for butterflies). The postprocessing stage is a block-diagonal matrix with maximum block size $N/2$. Each block in the postprocessing stage is a circular-convolution-like (CCL) operation.

The algorithm/architecture derived in this paper is confined to the power-of-two length DCT. The upper limit of the transform length is dependent upon current VLSI technology. Since the implementation of the post-processing stage is similar to that of a transversal filter, the upper limit on the transform length is about twice the upper bound on existing FIR filter sizes.

The preprocessing stage can be implemented by simple additions/subtractions. The postprocessing stage can be implemented by using existing convolvers/correlators [12, 13]. Since the coefficients of the CCL matrix are fixed, table lookup and mixed power-of-two approximation are effective alternatives. Flexibility is one of the advantages of this proposed algorithm/architecture, that is, if an N-point DCT is available, one needs only to add the maximum block of the $2N$-point DCT and one additional reversing/butterfly stage to form a $2N$-point DCT.

7 References

1 AHMED, N., NATARAJAN, T., and RAO, K.R.: 'The discrete cosine transform', *IEEE Trans.*, January 1974, **C-25,** pp. 90–93
2 JAIN, A.K.: 'Image data compression: a review', *Proc. IEEE*, 1981, pp. 349–389
3 ZELINSKI, R., and NOLL, P.: 'Adaptive transform coding of speech signals', *IEEE Trans.*, August 1977, **ASSP-25,** pp. 299–309
4 HAMIDI, M., and PEARL, J.: 'Comparison of the cosine and Fourier transforms of Markov-1 signals', *IEEE Trans.*, October 1976, **ASSP-24,** pp. 428–429
5 CLARK, R.J.: 'Relation between the Karhunen–Loéve and cosine transform', *IEE Proc. F, Commun., Radar & Signal Process.*, November 1981, **128,** pp. 359–360
6 MAKHOUL, J.: 'A fast cosine transform in one and two dimensions', *IEEE Trans.*, February 1980, **ASSP-28,** pp. 27–34
7 LEE, B.G.: 'A new algorithm to compute the discrete cosine transform', *IEEE Trans.*, December 1984, **ASSP-32,** (6), pp. 1243–1247
8 HOU, H.S.: 'A fast recursive algorithm for computing the discrete cosine transform', *IEEE Trans.*, October 1987, **ASSP-35,** pp. 1455–1461
9 SUN, M.T., WU, L., and LIOU, M.L.: 'A concurrent architecture for VLSI implementation of discrete cosine transform', *IEEE Trans.*, August 1987, **CAS-34,** (8), pp. 992–994
10 WU, J.-L., and DUH, W.-J.: 'A novel concurrent architecture to implement discrete cosine transform based on index partitions', to appear in *Int. J.* Electron.
11 DUH, W.-J., and WU, J.-L.: 'A constant-rotation DCT architecture based on CORDIC techniques', to appear in *Int. J. Electron.*
12 ERSOY, O.: 'Semisystolic array implementation of circular, skew circular, and linear convolutions', *IEEE Trans.*, February 1985, **C-34,** (2), pp. 190–194
13 BOUSSAKTA, S., and HOLT, A.G.J.: 'Prime-factor Hartley and Hartley-like transform calculation using transversal filter-type structures', *IEE Proc. G., Circuits, Devices & Syst.*, October 1989, **136,** (5), pp. 269–277
14 CHEN, W.-H., HARRISON SMITH, C., and FRALICK, S.C.: 'A fast computational algorithm for the discrete cosine transform', *IEEE Trans.*, September 1977, **COM-25,** (9), pp. 1004-1009
15 GILBERT, J., and GILBERT, L.: 'Elements of modern algebra' (PWS Publishers, Boston, USA, 1984)
16 SHANKS, D.: 'Solved and unsolved problems in number theory' (Chelsea Publishing Co., NY, 1978)

8 Appendix

8.1 Proof of lemma 1

We complete the proof of Lemma 1 by using Euler's ϕ function.

Theorem 1:

$$\sum_{d\backslash N} \phi(d) = N \tag{22}$$

where N is a positive integer and $d\backslash N$ denotes 'd divides N'.

The index set of an N-point transformation is $\{0, 1, 2, \ldots, N-1\}$. If the transform length is power-of-two, denoted as 2^m, the index set can be partitioned into $m+1$ subsets, corresponding to $\phi(d)$, where $d \in \{2^0, 2^1, 2^2, \ldots, 2^m\}$, respectively.

8.2 Proof of lemma 2

Since $n \in A_i$ and $k \in A_0$, it is obvious that gcd $(n, N) = 2^i$ for $i \in \{0, 1, \ldots, m-1\}$. Therefore, gcd $(f(k, n), N) = 2^i$, that is, the value $f(k, n)$ is still in the set A_i.

8.3 Proof of lemma 3

First, we prove that $G_N = \{A_0; \otimes_N\}$ is a group. The binary operator $\otimes_N$ is similar to modulo multiplication. If the number is first normalised to the set, say $\{-N+1, -N+2, \ldots, -1, 0, 1, \ldots, N-1, N\}$ mod $2N$, then the operation of $\otimes_N$ is the absolute value of corresponding modulo multiplications. Since 'modulo multiplication group' is a group, it is obvious that the four properties of a group structure are satisfied in G_N.

Next, we prove that 3 is a generator of group G_N and

$$\begin{aligned} 3^{N/2} &\equiv 1 \ (\textit{under} \otimes_N) \\ 3^{N/4} &\equiv N-1 \ (\textit{under} \otimes_N) \end{aligned} \tag{23}$$

Eqn. 23 means that 3 is a generator of order $N/2$. The reason for using $N-1$ in the second equation is that $(N-1)\otimes_N(N-1) = 1$. The proof of eqn. 23 is as follows:

(*a*) From Euler's function, we have

$$3^{\phi(2N)} \equiv 1 (\text{mod } 2N)$$

$$3^N \equiv 1 (\text{mod } 2N)$$

(*b*) Since $N = 2^m$, we have

$$3^{N/2} \equiv -1 (\text{mod } 2N)$$

$$\equiv 1 (\textit{under} \otimes_N)$$

Therefore, the first of eqns. 23 is verified.

(*c*) If $N = 2^m$, for $m < 4$ the proof is trivial and for $m \geqslant 4$ the proof is as follows:

$$\begin{aligned} 3^{N/4} &= (2+1)^{N/4} \\ &= \sum_{k=0}^{N/4} \binom{N/4}{k} 2^k \\ &= 1 + \frac{N}{2} + \frac{N}{2}\left(\frac{N}{4} - 1\right) + \cdots + 2^{N/4} \\ &\equiv 1 + 2^{m-1} + (2^{2m-3} - 2^{m-1}) + 0 \\ &\quad + \tfrac{1}{3}(2^{4m-7} - 12 \times 2^{3m-6} + 22 \\ &\quad \times 2^{2m-4} - 12 \times 2^{m-2}) + 0 \\ &\quad + \cdots + 0 (\text{mod } 2^{m+1}) \\ &\equiv 1 - 4 \times 2^{m-2} (\text{mod } 2^{m+1}) \\ &\equiv 1 - 2^m (\text{mod } 2^{m+1}) \\ &\equiv N - 1 (\textit{under} \otimes_N) \end{aligned} \tag{24}$$

Thus, G_N is a cyclic group and 3 is a generator.