1994年7月 第17卷第4期 重庆大学学报 JOURNAL OF CHONGQING UNIVERSITY Vol. 17, №. 4 Jul. 1994

一种新的快速离散余弦变换算法*

A New Fast Algorithm for the Discrete Cosine Transform

吴晓芸 Wu Xiaoyun　张太怡 Zhang Taiyi　张双腾 Zhang Shuangteng

(重庆大学电子信息工程学院，重庆，630044)

摘　要　提出一种快速的离散余弦变换(DCT)算法。由于计算机中整数运算远快于浮点运算，所以DCT算法采用整数运算，并且通过矩阵变换来减少乘加次数，提高了运算速度。本算法用于我们开发的JPEG图像编码算法上，取得了满意的效果。

关键词　离散余弦变换(DCT)；反离散余弦变换(IDCT)；矩阵运算；线性变换；直积

中国图书资料分类法分类号　O235

ABSTRACT　We prepose a new fast algorithm for computing discrete cosine transform and its inverse on two dimensional input sizes which are powers of 2. Because the integer operation is more faster than the float operation in computer, the integer operation is used in this algorithm. Through the matrix linear transform the number of operation for multiplications and additions is reduced, so that the speed is greatly raised. This algorithm is applied to our developmented JPEG image coding, and yields very good results.

KEYWORDS　discrete cosine transform; inverse discrete cosine transform; matrix operation; linear transform; tensor product

0 引　言

离散余弦变换在图像处理，特别是图像压缩中得到了广泛的应用。有关离散余弦变换及快速算法在文献[1～5]中进行了深入的讨论，文献[1]对8X8的DCT作了全面的算法推导。离散余弦变换(DCT)是一种正交变换，变换前后信号的能量不变，但能量的分布发生了改变，变换后的能量会集中在少数几项上；并且DCT易于实现，所以图像压缩编码的国际标准JPEG算法就是以DCT为基础的。在图像压缩中，用得最多的是8×8点的DCT。因为在整个压缩编码的实现过程中占用时间最多的是DCT运算，所以要缩短编码时间实现快速的DCT就是关键。在计算机中，整数运算远快于浮点运算。DCT运算原为浮点运算，本文针对图像压缩提出了一种快速DCT算法(下称FDCT)，将浮点运算转化为整数运算，并通过矩阵变换以减少乘加次数，提高了图像处理速度。

* 收文日期　1993-12-13

1 8点的 FDCT

设 $f(m)$ $(m=0,1,\cdots 8-1)$ 是一个离散序列，那么 $f(m)$ 的 DCT 为当 $s=0$ 时，$F(0)=\sqrt{1/8}\sum_{m=0}^{8-1}f(m)$

当 $s=1,2,\cdots,8-1$ 时，$F(s)=\sqrt{2/8}\sum_{m=0}^{8-1}f(m)\cos[(\pi/16)(2m+1)s]$

变换核为：$\begin{cases}[g(m,0)]=\sqrt{1/8}\\ [g(m,s)]=1/2\cdot\cos[(\pi/16)(2m+1)s]\end{cases}$

将变换核写成矩阵形式，并设 $r(k)=\cos(2\pi k/32)$。由于在计算机中，浮点数运算要比整数运算占的时间长，所以将 $[g(m,s)]$ 整数化。这里，我们对每个元素乘以 $2^{13}\cdot\sqrt{2}$，得：

$$M=\begin{bmatrix} ri(4) & ri(4) & ri(4) & ri(4) & ri(4) & ri(4) & ri(4) & ri(4)\\ ri(1) & ri(3) & ri(5) & ri(7) & -ri(7) & -ri(5) & -ri(3) & -ri(1)\\ ri(2) & ri(6) & -ri(6) & -ri(2) & -ri(2) & -ri(6) & ri(6) & ri(2)\\ ri(3) & -ri(7) & -ri(1) & -ri(5) & ri(5) & ri(1) & ri(7) & -ri(3)\\ ri(4) & -ri(4) & -ri(4) & ri(4) & ri(4) & -ri(4) & -ri(4) & ri(4)\\ ri(5) & -ri(1) & ri(7) & ri(3) & -ri(3) & -ri(7) & ri(1) & -ri(5)\\ ri(6) & -ri(2) & ri(2) & -ri(6) & -ri(6) & ri(2) & -ri(2) & ri(6)\\ ri(7) & -ri(5) & ri(3) & -ri(1) & ri(1) & -ri(3) & ri(5) & -ri(7)\end{bmatrix} \tag{1}$$

对 M 阵进行线性变换得：

$$M_1=\begin{bmatrix} ri(4) & ri(4) & ri(4) & ri(4) & 0 & 0 & 0 & 0\\ ri(2) & ri(6) & -ri(6) & -ri(2) & 0 & 0 & 0 & 0\\ ri(4) & -ri(4) & -ri(4) & ri(4) & 0 & 0 & 0 & 0\\ ri(6) & -ri(2) & ri(2) & -ri(6) & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & ri(5) & -ri(7) & ri(3) & ri(1)\\ 0 & 0 & 0 & 0 & -ri(1) & ri(5) & -ri(7) & ri(3)\\ 0 & 0 & 0 & 0 & -ri(3) & -ri(1) & ri(5) & -ri(7)\\ 0 & 0 & 0 & 0 & ri(7) & -ri(3) & -ri(1) & ri(5)\end{bmatrix} \tag{2}$$

对 M_1 进行线性变换得 M_2：

$$M_2=\begin{bmatrix} ri(4) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & ri(4) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & ri(6) & ri(2) & 0 & 0 & 0 & 0 \\ 0 & 0 & -ri(2) & ri(6) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ri(5) & -ri(7) & ri(3) & ri(1) \\ 0 & 0 & 0 & 0 & -ri(1) & ri(5) & -ri(7) & ri(3) \\ 0 & 0 & 0 & 0 & -ri(3) & -ri(1) & ri(5) & -ri(7) \\ 0 & 0 & 0 & 0 & ri(7) & -ri(3) & -ri(1) & ri(5) \end{bmatrix} \tag{3}$$

经过上述变换得到了(3)式的矩阵 $M2$，由此，可得到下式：

$$M = P \cdot M_2 \cdot B \tag{4}$$

其中，$B = B_1 \cdot B_2 \cdot B_3$，经过运算得：

$$P=\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{bmatrix} \quad B_1=\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B_2=\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad B_3=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \end{bmatrix}$$

在进行 DCT 变换时，图像数据先和 B 相乘，这是加法运算；然后和 M_2 相乘(乘法运算)；最后和 P 相乘(位置交换)。整个运算过程就是一种线性变换。

2　8×8 DCT 的实现

将上面的8点 DCT 运算推广到8×8的 DCT 运算。

8×8的 DCT 变换核是8点的 DCT 变换核自身的直积，而对8×8 点的图像数据进行行向量的堆叠，就可实现8×8的 DCT.

由（4)式可得到8×8的 DCT 变换核，即：

$$M \otimes M = (P \otimes P) \cdot (M_2 \otimes M_2) \cdot (B \otimes B) \tag{5}$$

下面通过框图说明 8×8 DCT 的实现。

2.1 前加过程

这个过程是图像数据（8 × 8个像素点）和$(B_1 \cdot B_2 \cdot B_3) \otimes (B_1 \cdot B_2 \cdot B_3)$的乘积，事实上可用加法实现，如图 1 所示。

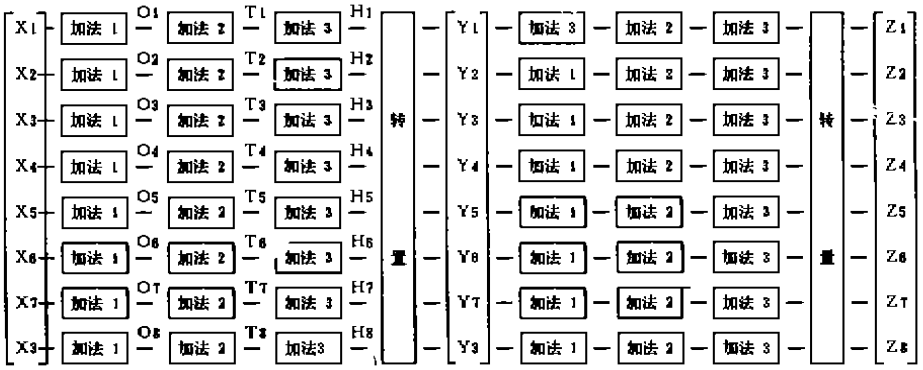

图 1 前加过程

其中，X_i是输入的图像数据（8 × 8 像素）的行向量，即：

$$X_i = \{X_{i0}, X_{i1}, X_{i2}, X_{i3}, X_{i4}, X_{i5}, X_{i6}, X_{i7}\}$$

对每个 X_i 进行加法 1，加法 2，加法 3(见图 2)，然后将结果转置，转置后矩阵的行向量是 Y_i，即：

$$Y_i = \{Y_{i0}, Y_{i1}, Y_{i2}, Y_{i3}, Y_{i4}, Y_{i5}, Y_{i6}, Y_{i7}\}$$

对每个 Y_i 又进行加法 1，加法 2，加法 3，然后再将结果转置，转置后矩阵的行向量是 Z_i，即：

$$Z_i = \{Z_{i0}, Z_{i1}, Z_{i2}, Z_{i3}, Z_{i4}, Z_{i5}, Z_{i6}, Z_{i7}\}$$

Z_i 是前加过程的输出。图 2 为三个加法的框图。

2.2 乘法过程的实现

这个过程从上面第 1 个过程的输出结果 Z 和矩阵 M2 的乘法来实现。如图 3 所示。右移 12 位及右移 15 位是为了补偿划为整数时左移的位数。

下面是各乘法的详细框图。其中，→ 代表乘以 ×.

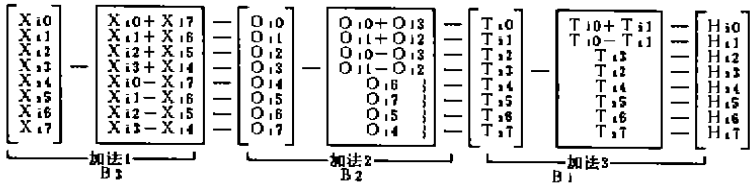

图 2　加法框图

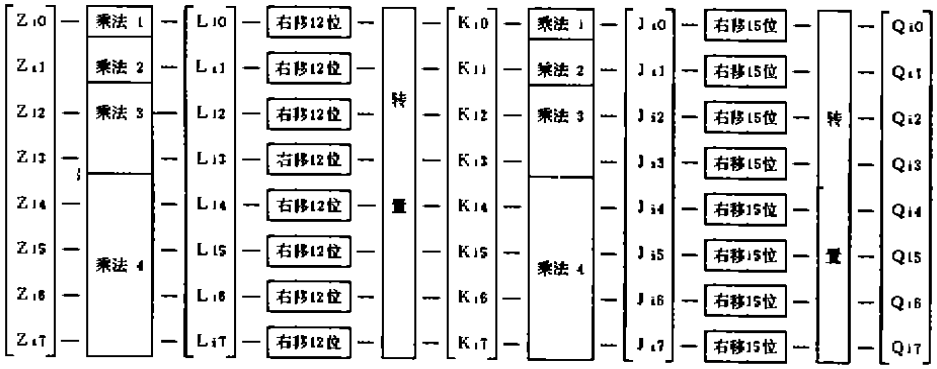

图 3　乘法过程

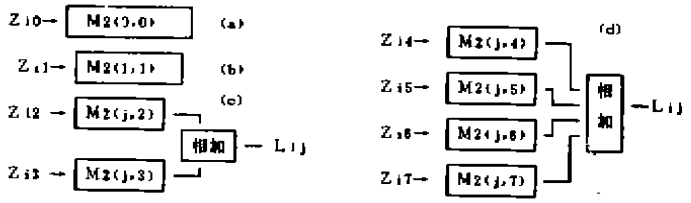

图 4　(a) 乘法 1；(b) 乘法 2；(c) 乘法 3 ($j = 2,3$)；(d) 乘法 4 ($j = 4,5,6,7$)

2.3　后置变换过程

这个过程是乘法过程的输出 Q 和矩阵 P 的乘，实际上是矩阵 Q 简单的线性变换。如图 5 所示。图 6 是图 5 中的变换模块的详细框图。

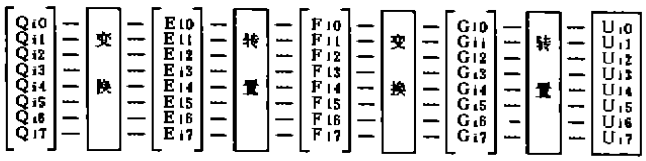

图 5　后置变换过程

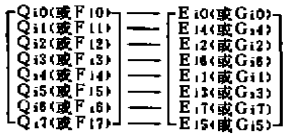

图 6　换模块框图

3 逆 DCT 过程(IDCT)

由(4)式 $M \otimes M = (P \cdot M_2 \cdot B) \otimes (P \cdot M_2 \cdot B)$ 可得逆 DCT 为：

$$(M \otimes M)^{-1} = (P \cdot M_2 \cdot B)^{-1} \otimes (P \cdot M_2 \cdot B)^{-1}$$

因为 M 是正交矩阵，所以 M 的逆阵 M^{-1} 等于 M 的转置矩阵 M^T，所以上式可变为

$$(M \otimes M)^T = (B^T \otimes B^T) \cdot (M_2^T \otimes M_2^T) \cdot (P^T \otimes P^T)$$

其中

$$B^T \otimes B^T = (B_3^T \cdot B_2^T \cdot B_1^T) \otimes (B_3^T \cdot B_2^T \cdot B_1^T)$$

模仿正 DCT 过程可很容易地实现逆 DCT(IDCT) 运算，图像数据先与 $P^T \otimes P^T$ 相乘(位置变换)，然后与 $M_2^T \otimes M_2^T$ 相乘 (乘法运算)，最后和 $B^T \otimes B^T$ 相乘 (加法运算) 即可。

4 结 论

对于8×8的 DCT 运算，采用直接计算要1024次乘法和896次加法。本文提出的算法通过线性变换，共用了352次乘法和320次加法；同时采用整数运算，DCT 变换采用长整型数4个字节，变换系数采用短整型数2个字节，图象数据及变换结果截取1个字节，这样，使整个运算速度大大提高，满足图像处理的速度和精度要求，并且易于实现。本算法用于我们开发的 JPEG 图像编码算法上，取得了满意的效果，提高了图像处理的实时性。

参 考 文 献

1 Feig E, Winograd S, Fast Algorithm for the Discrete Cosine Transform, IEEE Trans. on Signal Process. ,1992. 40(9):2174—2193

2 Hsteh S Hou. A Fast Resursive Algorithm for Computing the Discrete Cosine Transform, IEEE Trans. on Acoust. ,Speech, and Signal Process. , 1987,ASSP—35,(10);

3 Cho N, Lee S, A Fast 4×4 DCT Algorithm for the Recursive 2—D DCT, IEEE Trans. on Signal Process. , 1992,40(9);

4 Ahmed N, Natarajan T, Rao K R, Discrete Cosine Transform, IEEE Trans. on Computers,1974,C—23,90—93

5 余松煜，周源华，李时光. 数字图像处理，电子工业出版社，1989