第12卷 第4期 电子科学学刊 Vol. 12, No. 4
1990年7月 JOURNAL OF ELECTRONICS July, 1990

通用顺序即位素因子FFT算法*,**

王 中 德
(北京邮电学院，北京)

摘要 本文从一维到多维的下标变换出发，得到了一种通用顺序，即位素因子FFT算法．与现在的素因子FFT算法相比较，这种算法不仅节省了约一半内存，而且有更高的计算效率．此外，这种算法能很方便地将逆变换也包括在同一程序内．

关键词 快速算法；离散付里叶变换；FFT；素因子算法

一、引 言

离散付里叶变换（DFT）的快速计算是各种各样数字信号处理应用中的一个基本问题．现有的快速付里叶变换（FFT）有两大类：一类适用于计算 $N=2^M$ 点的DFT，包括各种各样 Cooley-Tukey 型 FFT[1-3] 以及其他基2类 FFT[4,5]．另一类适合于计算 $N=N_1N_2\cdots N_m$ 点的DFT，其中 $N_1, N_2, \cdots, N_m$ 是 m 个两两互素的因子．这类算法又分为 Winograd 算法（WFTA）[6] 和素因子算法（PFA）[7] 两种．Morris[8]在各种型号的计算机上对基2类FFT与WFTA进行比较的结果说明，基2类FFT的效率要比WFTA的高．另一方面，Burrus 和 Eschenbacher[9] 将PFA编制成程序，表明PFA的效率比基2类FFT的效率更高，引人注目．

FFT的第二类算法中的关键，在于当 N 可以分解为 m 个互素因子时，就可以将一维DFT分解成 m 维DFT．这一思想是由 Good[10] 在1958年提出来的．然而，在 Cooley 和 Tukey[1] 的著名论文发表以前，Good 的算法并未引起人们的注意．直到1971年后，这一算法才重新受到重视[11]．Kolba 和 Panks[7] 用高速卷积将PFA具体化．Burrus 和 Eschenbacher[9] 提出一种简单的整序方法，将PFA程序化，实现了PFA的即位（in-place）计算．遗憾的是，尽管所提的整序方法非常简单，但在程序PFA1中，却不得不额外用两组存储器存储最终结果；文献[9]也编制出一个既是即位、且又顺序的程序PFA2，但这种即位、顺序算法的程序，需要对每一个特定的长度进行单独设计．Rothweiler[12] 认为已找到了通用的顺序、即位PFA算法，而且还公布了程序PFAT．然而，用PFAT不能给出正确的结果．Johnson 和 Burrus[13] 提出了用改变各小 N 点DFT的乘法系数来实现PFA的顺序计算．这种方法与文献[9]中的PFA2一样，没有通用性．Perez 和 Takaoka[14] 把所有数组都用简单变量直接写出，所有子程序的调用均逐条写出调用语句，而不用循环嵌套的方式．这样虽然可以提高程序的效率，却大大降低了编程的效率，且使

* 1989年3月10日收到，1990年2月9日修改定稿．
** 国家自然科学基金资助课题．

程序的长度随N而增长，而程序仍然不具备通用性。所以只有求助于计算机来编写程序。本文从一维到多维的下标变换出发，提出一种通用的顺序、即位 PFA 算法。它适用于任何适合于 PFA 计算的长度。由于免除了最后的整序操作，这种算法的效率比文献[9]的 PFA 算法的效率高，而且避免了使用额外的存储器。同时，这种算法可以很简单地把 DFT 的逆变换（IDFT）融合到同一个程序中。

二、下标变换

下标变换是 PFA 算法中最重要的，也是最令人费解的部分。N点输入序列 $x(n)$ 的 DFT 为

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk},\qquad W=e^{-j(2\pi/N)} \tag{1}$$

当N可分解为m个两两互素的因子

$$N=N_1N_2\cdots N_m \tag{2}$$

时，如何将一维 DFT 映射成多维 DFT，就是下标变换所要解决的问题。当 $m=2$ 时，Burrus[15] 提出了下列变换式：

$$n=\langle K_1n_1+K_2n_2\rangle_N \tag{3a}$$

$$k=\langle K_3k_1+K_4k_2\rangle_N \tag{3b}$$

其中，$\langle\cdot\rangle_N$ 表示 mod N；$n_i,k_i=0,1,\cdots,N_i-1$；$i=1,2$；K_1，K_2，K_3 和 K_4 满足以下条件：

$$\langle K_1K_3\rangle_N=N_2;\quad\langle K_2K_4\rangle_N=N_1 \tag{4a}$$

$$\langle K_1K_4\rangle_N=\langle K_2K_3\rangle_N=0 \tag{4b}$$

钱惠生和朱起秀[16]讨论了一维到多维这种更一般的情况。他们用与 Burrus 不同的方式，定义下标变换：

$$\begin{aligned}&n_i=\langle q_in\rangle_{N_i};\quad k_i=\langle r_ik\rangle_{N_i}\\&n_i,k_i=0,1,\cdots,N_i-1;\quad i=1,2,\cdots,m\end{aligned} \tag{5}$$

式中 q_i 和 r_i 满足

$$\langle q_ir_iM_i\rangle_{N_i}=1,\quad M_i=N/N_i \tag{6}$$

他们还用中国余数定理，得到

$$n=\left\langle\sum_{i=1}^{m}r_iM_in_i\right\rangle_N \tag{7a}$$

$$k=\left\langle\sum_{i=1}^{m}q_iM_ik_i\right\rangle_N \tag{7b}$$

本文作者把 Burrus 的下标变换式推广到多维，提出了下面的下标变换[17]：

$$n=\left\langle\sum_{i=1}^{m}K_in_i\right\rangle_N;\quad k=\left\langle\sum_{i=1}^{m}L_ik_i\right\rangle_N \tag{8}$$

在满足条件

$$\langle K_i L_i \rangle_N = M_i \tag{9a}$$

$$\langle K_i L_p \rangle_N = 0, \quad i \neq p \tag{9b}$$

时，就可以把由(1)式定义的一维 DFT 转换成一个 m 维 DFT

$$X(k_1, k_2, \cdots, k_m) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_m=0}^{N_m-1} x(n_1, n_2, \cdots, n_m) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} \cdots W_{N_m}^{n_m k_m} \tag{10}$$

式中

$$\begin{aligned} x(n_1, n_2, \cdots, n_m) &= x\left(\left\langle \sum_{i=1}^{m} K_i n_i \right\rangle_N\right) \\ X(k_1, k_2, \cdots, k_m) &= X\left(\left\langle \sum_{i=1}^{m} L_i k_i \right\rangle_N\right) \end{aligned} \tag{11}$$

N 和 M_i 分别由(2)和(6)式给出；$N_1, N_2, \cdots, N_m$ 为 m 个两两互素的因子。文献[17]还证明了下标变换(8)式和条件(9)式分别与下标变换(5)式和条件(6)式等价。

三、顺序素因子算法的实现

选用(7)式作为下标变换的标准表达式。为了使输入序列有一个统一的下标变换式，令 $r_i = 1$。这样，采用的下标变换为

$$n = \left\langle \sum_{i=1}^{m} M_i n_i \right\rangle_N \tag{17a}$$

$$k = \left\langle \sum_{i=1}^{m} q_i M_i k_i \right\rangle_N \tag{17b}$$

由(6)式可得到

$$q_i = \langle M_i \rangle_{N_i}^{-1} \tag{18}$$

(17b) 还可以写成以下等价的形式：

$$k = \left\langle \sum_{i=1}^{m} M_i \langle q_i k_i \rangle_{N_i} \right\rangle_N \tag{17c}$$

表 1

(a) $N = 15$ 时 n 的变换表

n_1 \ n_2	0	1	2	3	4
0	0	3	6	9	12
1	5	8	11	14	2
2	10	13	1	4	7

(b) $N = 15$ 时

k_1 \ k_2	0	1
0	0	6
1	10	1
2	5	11

令

$$k_i' = \langle q_i k_i \rangle_{N_i} \tag{19a}$$

就可以把 k 写成与（17a）式的 n 完全相同的形式

$$k = \left\langle \sum_{i=1}^{m} M_i k_i' \right\rangle_N \tag{20}$$

由(18)式知 q_i 与 N_i 互素，因此，k_i' 是 k_i 的一个重新排序．由(19a)式可得

$$k_i = \langle \langle q_i \rangle_{\bar{N}_i}' k_i' \rangle_{N_i} = \langle M_i k' \rangle_{N_i} \tag{19b}$$

(20)式表明，只要按照 k_i' 的自然顺序，即对 k_i 进行重新排序后，输出顺序 k 就能与输入顺序 n 保持一致．以 $N = N_1 N_2 = 3 \times 5$ 为例，来说明上述过程．由(18)式可得 $q_1 = q_2 = 2$，如果把 n 和 k 分别按 n_1、n_2 和 k_1、k_2 排列成一个二维变换表，则由(17)式可得 n 和 k 的变换表如表 1 (a) 和表 1 (b) 所示．另一方面，由(19)式得到 k_1、k_2 与 k_1'、k_2' 的对应关系为

$$k_1 = 0 \quad 1 \quad 2 \qquad k_2 = 0 \quad 1 \quad 2 \quad 3 \quad 4$$
$$k_1' = 0 \quad 2 \quad 1 \qquad k_2' = 0 \quad 2 \quad 4 \quad 1 \quad 3$$

按照 k_1' 和 k_2' 递增的顺序，将 k_1、k_2 进行重排以后的输出变换表如表 1 (c) 所示．比较表 1 (c) 和表 1 (a) 可以看出，k 和 n 的二维变换表完全相同．因此，按自然顺序的原始数据经过 DFT 变换以后，得到的将是按自然顺序排列的频谱数据．从而实现了 DFT 的顺序计算．

初看起来，上述方法要求对每一个小 N_i 点 DFT 的输出都要按(19)式进行重排，而文献[9]只要求在最后进行一次重排．前者需要重排的次数比后者多，因而效率可能不如后者．然而，对小 N_i 点 DFT 的输出进行重排，可以用选择适当的出口来实现；或者事先根据(19)式计算出一个输出地址表，然后用查表的方式确定输出的位置．在附录中，列出了一个简短的示范程序．其中 3 点与 4 点 DFT 采用不同的出口进行重排，而 5 点 DFT 则用输出地址表的方式进行重排．用第一种方式实现小 N_i 点输出的重排，几乎不需要任何额外的时间．用第二种方式，所需要的额外时间是查表的时间，与文献[9]中

k 的变换表			(c) 根据(19)式将 k_1 和 k_2 重排后 k 的变换表						
2	3	4	k_2' / k_2 / k_1' / k_1		0	1	2	3	4
12	3	9			0	3	1	4	2
7	11	4	0	0	0	3	6	9	12
2	8	14	1	2	5	8	11	14	2
			2	1	10	13	1	4	7

要求物理上的重排所需要的时间相比，也是微不足道的．因此，本文介绍的算法，有更高的效率．

由上述可知，如果按（17a）式对输入进行下标变换，就必须根据(19)式对各小 N_i DFT 的输出进行重排，才能得到顺序输出的素因子 DFT．文献[12]中的子程序 PFAT 是根据（17a）式对输入下标进行变换，但却按照

$$k' = \langle M_i k_i \rangle_{N_i}$$

对各小 N_i 点 DFT 的输出进行重排．这可以由 PFAT 中标号为 15 的循环语句所建立的输出下标表看出．由于正确的重排只能根据(19)式，所以文献[12]中的 PFAT 将得不到正确的结果．

四、逆变换的简单实现

DFT 的逆变换（IDFT）为

$$x(n) = \frac{1}{N} \sum_{k=1}^{N-1} X(k) W_N^{-nk} \tag{21}$$

式中 W_N 仍由(1)式给出．略去因子 $1/N$ 不计，(21)式和(1)式的主要区别，只在于 W_N 指数的符号．这时，采用下标变换

$$k = \left\langle \sum_{i=1}^{m} M_i k_i \right\rangle_N \tag{22a}$$

$$n = \left\langle - \sum_{i=1}^{m} q_i M_i n_i \right\rangle_N \tag{22b}$$

就可将(21)式的 IDFT 转换成一个 m 维的 DFT

$$x(n_1, n_2, \cdots, n_m) = \frac{1}{N} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \cdots \sum_{k_m=0}^{N_m-1} X(k_1, k_2, \cdots, k_m) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} \cdots W_{N_m}^{n_m k_m} \tag{23}$$

式中 q_i 仍由(18)式确定．只要按照文献[17]中的推理方法，这一论断是不难证明的．由于这时 k 为输入下标，n 为输出下标，因此，不计 $1/N$，只要用 $-q_i$ 代替 q_i，原来 DFT 的程序就变成了 IDFT 的程序．因为 q_i 是按模 N_i 计算，所以

$$\langle -q_i \rangle_{N_i} = N_i - q_i$$

按照本文方法，编制出一个名为 PFAF 的 FORTRAN 子程序．由于多出口的小 N_i 点整序方法比按地址表输出的方法效率更高，所以，当 $N_i \leqslant 8$ 时，均采用多出口的方式．当 $N_i > 9$ 时，若仍采用上述方式，则程序会显得冗长，采用第二种方式输出，程序就要简洁得多．附录中列出一个示范程序，只为了说明本文的方法，在 $N_i \geqslant 5$ 时即采用第二种方式输出．

五、运行时间比较

将 PFAF 与文献[9]中的 PFA1 以及文献[18]中按分裂基 FFT 编写的程序在一台

IBM-PC/XT 微机上进行运行时间的比较，结果列于表 2. 从表 2 可以看出，用本文介绍的方法，不仅可免除对额外存储器的要求，而且使计算效率得到提高.

表 2　运行时间比较(单位：ms)

	PFAF			PFA1[9]		(DITSR) FFT[18]	
	T_p	T_r	T_r/N	T_r	T_r/N	T_r	T_r/N
32						140.1	4.38
35	4.4	80.7	2.31	98.3	2.81		
63	6.5	154.4	2.45	184.7	3.03		
64						320.3	5.00
126	7.8	374.7	2.97	415.6	3.30		
128						710.3	5.55
252	9.2	797.9	3.17	881.3	3.50		
256						1548	6.05
504	9.2	1727	3.43	1899.7	3.77		
512						3336	6.52
1008	9.5	3788	3.76	4117	4.08		
1024						7138	6.97

注：T_p 为生成必要参数的时间；T_r 为运行时间；T_r/N 为平均每一点的运算时间.

附　　录

以下是一个通用的顺序、即位素因子 FFT 算法的示范程序. 输入序列的实部和虚部分别储存在数组 X 和 Y 中. 经过计算后，它们将被按顺序排列的频谱成份所代替. 序列的长度为 $N = N1(1) * N1(2) * N1(3) * N1(4)$. $N1(1) = 1, 2, 4, 8$ 或 16; $N1(2) = 1, 3$ 或 9; $N1(3) = 1$ 或 5; $N1(4) = 1$ 或 7. 示范程序中只列出了这些可能性中的一部分. IFLAG = 1 时，程序作反变换; IFLAG 为其他值时，程序作正变换.

```
      SUBROUTINE PFAF (X, Y, N, N1,
     +   IFLAG)
      DIMENSION X(N), Y(N), N1(4),
     +   N2(4), N3(4), I(16), J2(16),
     +   J9(9),J5(5),J7(7)
      EQUIVALENCE (I(1),I1),(I(2),I2),
     +   (I(3),I3),(I(4),I4),(I(5),I5),  . . .)
C
      DATA C3, S3/-0.5,        0.86602540/
      DATA C51, C52/-0.25,     0.55901699/
      DATA S51,S52/-0.58778525,1.53884177/
      DATA S53    /-0.36327126/
      . . . . . .
      DO 209 K = 1,4
        IF (N1(K).EQ.1) GOTO 209
        N2(K) = N/N1(K)
        L = N2(K)
200     IF(MOD(L,N1(K)).EQ.1) GOTO 201
        L = L + N2(K)
        GOTO 200
201     N3(K) = L/N2(K)
        IF(IFLAG.EQ.1) N3(K)= N1(K)-N3
     +  (K)
        IF(N1(K).LE.4) GOTO 209
C------ TO GENERATE OUTPUT POINTER--
        L = 0
        DO 208 K1 = 2, N1(K)
          L = L + N3(K)
          GOTO (202,203,204, 205) K
202       J2(K1) = MOD(L,N1(1)) + 1
          GOTO 208
203       J9(K1) = MOD(L,9) + 1
          GOTO 208
204       J5(K1) = MOD(L,5) + 1
          GOTO 208
205       J7(K1) = MOD(L,7) + 1
208     CONTINUE
209   CONTINUE
```

```
C--------------MAIN LOOP-------------------
      DO 110 K = 1,4
        IF (N1(K).EQ.1) GOTO 110
        DO 100 K1 = 1,N,N1(K)
          I(1) = K1
          DO 13 J1 = 2, N1(K)
            I(J1) = I(J1 - 1) + N2(K)
            IF(I(J1).GT.N) I(J1)=I(J1)-N
13            CONTINUE
            GOTO (100,20,30,40,50,···)N1(K)
C-------------MODULE FOR N = 2 ---------
20          R1 = X(I1)
            X(I1) = R1 + X(I2)
            X(I2) = R1 - X(I2)
            R1 = Y(I1)
            Y(I1) = R1 + Y(I2)
            Y(I2) = R1 - Y(I2)
            GOTO 100
C--------------- MODULE FOR N = 3--------
30          R1 = X(I2) + X(I3)
            R2 = X(I1) + R1*C3
            R3 = (X(I2) - X(I3))*S3
            X(I1) = X(I1) + R1
            T1 = Y(I2) + Y(I3)
            T2 = Y(I1) + T1*C3
            T3 = (Y(I2) - Y(I3))*S3
            Y(I1) = Y(I1) + T1
            GOTO (31,32) N3(K)
31          X(I2) = R2 + T3
            X(I3) = R2 - T3
            Y(I2) = T2 - R3
            Y(I3) = T2 + R3
            GOTO 100
32          X(I2) = R2 - T3
            X(I3) = R2 + T3
            Y(I2) = T2 + R3
            Y(I3) = T2 - R3
            GOTO 100
C ------------------MODULE FOR N = 4-----
40          R1 = X(I1) + X(I3)
            R3 = X(I2) + X(I4)
            R2 = X(I1) - X(I3)
            R4 = X(I2) - X(I4)
            X(I1) = R1 + R3
            X(I3) = R1 - R3
            T1 = Y(I1) + Y(I3)
            T3 = Y(I2) + Y(I4)
            T2 = Y(I1) - Y(I3)
            T4 = Y(I2) - Y(I4)
            Y(I1) = T1 + T3
            Y(I3) = T1 - T3
          GOTO (41,100,42) N3(K)
41          X(I2) = R2 + T4
            X(I4) = R2 - T4
            Y(I2) = T2 - R4
            Y(I4) = T2 + R4
            GOTO 100
42          X(I2) = R2 - T4
            X(I4) = R2 + T4
            Y(I2) = T2 + R4
            Y(I4) = T2 - R4
            GOTO 100
C-------------MODULE FOR N = 5------------
50          R1 = X(I2) + X(I5)
            R2 = X(I3) + X(I4)
            R5 = X(I2) - X(I5)
            R6 = X(I3) - X(I4)
            S3 = R1 + R2
            R4 = (R1 - R2)*C52
            R1 = X(I1) + R3*C51
            X(I1) = X(I1) + R3
            R2 = R1 - R4
            R1 = R1 + R4
            R3 = (R5 - R6)*S51
            R4 = R5*S52 + R3
            R3 = R6*S53 - R3
            T1 = Y(I2) + Y(I5)
            T2 = Y(I3) + Y(I4)
            T5 = Y(I2) - Y(I5)
            T6 = Y(I3) - Y(I4)
            T3 = T1 + T2
            T4 = (T1 - T2)*C52
            T1 = Y(I1) + T3*C51
            Y(I1) = Y(I1) + T3
            T2 = T1 - T4
            T1 = T1 + T4
            T3 = (T5 - T6)*S51
            T4 = T5*S52 + T3
            T3 = T6*S53 - T3
            X(I(J5(2))) = R1 + T4
            X(I(J5(3))) = R2 + T3
            X(I(J5(4))) = R2 - T3
            X(I(J5(5))) = R1 - T4
            Y(I(J5(2))) = T1 - R4
            Y(I(J5(3))) = T2 - R3
            Y(I(J5(4))) = T2 + R3
            Y(I(J5(5))) = T1 + R4
C-----------------MODULE FOR N = 7-------
      . . . . . .
100         CONTINUE
110       CONTINUE
        RETURN
        END
```

参 考 文 献

[1] J. Cooley, J. Tukey, *Math. Compus.* 19(1965), 297—301.

[2] P. C. Singerton, *IEEE Trans. on AU*, **AU-17** (1969), 99—103.
[3] P. Duhamel, H. Hollmann, *Elec. Letts.*, **20**(1984), 14—16.
[4] C. M. Rader, N. M. Brenner, *IEEE Trans. on ASSP*, **ASSP-24** (1976), 264—265.
[5] Zhongde Wang (王中德), *IEEE Trans. on ASSP*, **ASSP-32**(1984), 803—816.
[6] S. Winograd, *Math. Comput.*, **32**(1978), 175—199.
[7] D. P. Kolba, T. W. Parks, *IEEE Trans. on ASSP* **ASSP-25** (1977), 281—294.
[8] L. R. Morris, *IEEE Trans. on ASSP*, **ASSP-26** (1978), 141—150.
[9] C. S. Burrus, P. W. Eschenbacher, *IEEE Trans. on ASSP*, **ASSP-29** (1981), 806—817.
[10] I J. Good, *J. Royal. Stat. Soc.*, Ser. B, **20**(1958), 361—372.
[11] I. J. Good, *IEEE Trans. on Computer*, **C-20** (1971), 310—317.
[12] J. H. Rothweiler, *IEEE Trans. on ASSP*, **ASSP-30**(1982), 105—107.
[13] Johnson, C. S. Burrus, *IEEE Trans. on ASSP*, **ASSP-33**(1985), 248—253.
[14] F. Perez, T. *Takaoka, IEEE Trans. on ASSP*, **ASSP-35**(1987), 1221—1223.
[15] C. S. Burrus, *IEEE Trans. on ASSP*, **ASSP-25**(1977), 239—242.
[16] 钱惠生、朱起秀，电子学报，1981 年，第 5 期，第 12—21 页.
[17] Zhongde Wang (王中德), *Elec. Letts.*, **25**(1989), 12, 781—782.
[18] H. V. Sorensen, et al., *IEEE Trans. on ASSP*, **ASSP-34** (1986), 152—156

GENERAL IN-PLACE AND IN-ORDER PRIME FACTOR FFT ALGORITHM

Wang Zhongde

(Beijing University of Posts & Telecommunications, Beijing)

Abstract Starting from an index mapping for one to multi-dimensions, a general in-place and in-order prime factor FFT algorithm is proposed. Comparing with existing prime factor FFT algorithms, this algorithm saves about half of the storage and possesses a higher efficiency. In addition, this algorithm can easily implement the DFT and IDFT in a single subroutine.

Key words Fast algorithm; Discrete Fourier transform; FFT; Prime factor algorithm