NASA Technical Memorandum 103914

A Discrete Fourier Transform for Virtual Memory Machines

David C. Galant, Ames Research Center, Moffett Field, California

March 1992

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035 -1000

Summary

The discrete Fourier transform (DFT) is a pervasive tool in scientific and engineering calculation. It lies at the heart of many methods for solving partial differential equations, data analysis, statistical calculations, and of course Fourier Analysis (see ref 1). More than one hundred papers on fast algorithms have been published! So, why another? Most DFT algorithms assume use of high speed computer memory that is local and contiguous; others assume the data is stored in blocks of external – remote – memory. In contrast, the efficient DFT algorithm provided in this paper assumes neither case; instead, it works efficiently on computers with virtual or cache memory where the location of the data is mapped automatically onto a small high speed memory by the computer's operating system. Important features of the algorithm are:

- no details of page mechanism to be used
- the code should be compact and clear
- no auxiliary storage should be required
- efficient execution when the data resides entirely in high speed memory.

Additionally, a matrix theory of the DFT is given. This theory provides the basis of the "fast" calculation of the transforms and the proofs give the insight to efficient organization of the calculation under different assumptions.

Implementing DFT's on Virtual Memory Machines

The DFT is most directly expressed as the product of a matrix with a vector. If $(x_1, x_2, \ldots, x_n)^T$ denotes the data to be transformed, $\omega_n = \exp(2\pi i/n)$, and

$$\mathcal{W}_n = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{1-n} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{2\cdot(1-n)} \\ \vdots & & & & \\ 1 & \omega_n^{1-n} & & \cdots & \omega_n^{(1-n)\cdot(1-n)} \end{bmatrix} \qquad (1)$$

then $a = \mathcal{W}_n \cdot x$ is the discrete Fourier transform of x. If a super bar is used to denote complex conjugate, $\frac{1}{n}\overline{\mathcal{W}}_n^{-1} \cdot x$ is the inverse discrete Fourier transform of x. $\mathcal{W}_n$ is called the discrete Fourier operator.

Appendix B provides decompositions of the matrix $\mathcal{W}_n$ as a product of sparse matrices. These decompositions provide a framework for implementing versions of fast DFTs. The results are not new, but the details of the proofs provide computational insight.

DFT algorithms to evaluate (1) are based on writing $\mathcal{W}_n$ as a product of sparse matrices which are then sequentially applied to x (ref 2, ref 3, ref 4.). The decomposition is not unique, so we may choose one that has properties that we find important.

The salient feature of computers with hierarchies of memory is that transfers of data between different levels is by blocks rather than individual words. Therefore, more complete use of data in a block yields a premium in inter-memory traffic. The features that we wish to emphasize, then, flow from a high locality of data. One simple approach is to access data in sequentially increasing order and to use every datum as much as possible before a block is discarded. Unfortunately, the latter is difficult to achieve without detailed knowledge of block sizes, but, careful use of the former yields almost all of the efficiency gain.

The following discussion assumes that the calculation overwrites the data to be transformed and the vector length is a power of two. Appendix B decomposes the DFT into a two phases. One scrambles the data; the other converts the data into its Fourier coefficients.

In the scrambling phase, usually the binary reversal of an index is calculated directly and the two elements of the vector are exchanged. A number and its binary reversal usually occur in different parts of the vector. The binary reversal of two successive integers are in different halves of the vector; even integers have binary reversals in the lower half, odd integers have their binary reversal in the upper half. So potentially almost as many block transfers occur as half the number of elements in the vector. Direct application of the unscrambling matrix decomposition requires one sequential pass through the data for each factor. Thus, total data motion is considerably reduced. When the vectors are long and real memory is small, the overhead from data motion is reduced considerably.

During the other phase of the DFT, the computation can be organized in several ways, one is to apply a factor of the decomposition to the same element in each block. This minimizes the computation of trigonometric function values (which are mostly calculated by recurrence relations), but does not address the data in a strictly sequential fashion. This organization is advantageous for machines with vector processing capabilities. Applying a factor of the decomposition on a complete block by complete block basis leads to strictly sequential access of data at the cost of repeated calculation of trigonometric function values. However, the decrease in overhead from data motion can often more than compensate this cost. This organization is also more suitable for machines with multiple processors than the one above.

The algorithm given in Appendix A is based upon the factorization in equation (15b). The transformation phase is based upon base four transforms rather than purely base two transforms because there is an increase in computational efficiency without complication to the coding of the algorithm.

References

1. Henrici, Peter, "Fast Fourier Methods in Compuational Complex Analysis," SIAM Review **21**, No.4, October 1979, pp. 481 - 527.
2. Kahaner, David K., "Matrix Description of the Fas t Fourier Transform," IEEE Transactions on audio and Electroacoustics, **AU-18**, No. 4, December 1970, pp. 442-450.
3. Rose, Donald J., "Matrix Identities of the Fast Fourier Transform," Linear Algebra and Its Applications, **29**, 1980, pp. 423-443.
4. Nicholson, Peter J., "Algebraic Theory of Finite Fourier Transforms," Journal of Computer and System Sciences, **5**, 1971, pp. 524-547.
5. Polge, Robert J., NB. K. Bhagavan, and James M. Carswell, "Fast Computational Algorithms for Bit Reversal," IEEE Transactons on Computers, **C-23**, No. 1, January 1974, pp. 1-9.
6. Brenner, Normal M., "Fast Fourier Transform of Externally Stored Data," IEEE Transactions on Audio and Electroacoustics, **AU-17**, No. 2, June 1969, pp. 128-132.
7. Singleton, Richard C., "A Method for Computing the Fast Fourier Transform with Auxiliary Memory and Limited High-Speed Storage," IEEE Transactions on Audio and Electroacoustics, AU-15, No. 2, June 1967, pp. 91-98.
8. Temperton, Clive, "Self-Sorting Mixed-Radix Fast Fourier Transforms," J. of Comp. Phys., **52**, (1983), pp. 1-23.

APPENDIX A - Fortran Code

```
      SUBROUTINE VMDFT(N, A, IER)
C-----------------------------------------------------------------------
C                    VIRTUAL MEMORY DISCRETE FOURIER TRANSFORM         I
C-----------------------------------------------------------------------
C     THIS ROUTINE EVALUATES                                           I
C                                                                      I
C            N-1         JK                                            I
C     X(J) = SUM  (A(K)W  ), J = 0(1)N-1                               I
C            K=0                                                       I
C                                                                      I
C     WHERE W = EXP(2*PI*I/N), I = SQRT(-1), N IS A POWER OF 2         I
C-----------------------------------------------------------------------
C     FORMAL PARAMETERS:                                               I
C                                                                      I
C          N     INTEGER, IN. LENGTH OF A                              I
C                                                                      I
C          A     COMPLEX VECTOR OF LENGTH N.  AT ENTRY, A CONTAINS THE I
C                DATA TO BE TRANSFORMED.  AT EXIT, IT CONTAINS THE     I
C                TRANSFORMED DATA.                                     I
C                                                                      I
C          IER   INTEGER, OUT. ERROR INDICATOR                         I
C                IER = 1 IF NO ERROR HAS BEEN DETECTED                 I
C                IER = 2 IF N .LE. 0 ON ENTRY                          I
C                IER = 2 IF N IS NOT A POWER OF 2                      I
C-----------------------------------------------------------------------
C     CACM CATOGORY: J1                                                I
C-----------------------------------------------------------------------
C     KEYWORDS: FOURIER TRANSFORM, VIRTUAL MEMORY, BINARY REVERSAL     I
C-----------------------------------------------------------------------
C     THE ALGORITHM IS A VARIATION OF THE COOLEY-TUKEY ALGORITHM.      I
C                                                                      I
C     1) THE TRANSFORMATION IS DONE IN PLACE                           I
C     2) NO AUXILIARY TABLES ARE REQUIRED                              I
C     3) REQUIRED TRIGONOMETRIC VALUES ARE USED IN NORMAL ORDER        I
C          AND ARE GENERATED FROM STABILIZED RECURRENCE RELATIONS      I
C     4) NO BINARY REVERSE COUNTER IS REQUIRED                         I
C     5) THE DATA IS ALWAYS ACCESSED IN MONOTONE INCREASING ORDER      I
C     6) THE ORGANIZATION HAS A HIGH LOCALITY OF DATA                  I
C     7) THE MINIMUM NUMBER OF PASSES OVER THE DATA IS REQURIED        I
C     8) THE ALGORITHM NEEDS NO PAGE SIZE INFORMATION, BUT THERE       I
C          MUST BE AT LEAST 4 DATA PAGES PLUS ROOM FOR THE CODE FOR    I
C          IMPROVEMENT TO SHOW                                         I
C     9) THE CODE IS CLEAR AND COMPACT                                 I
C    10) THE COST IN CPU TIME OVER THE MOST EFFICIENT DFT ALGORITH     I
C          IS SMALL                                                    I
C-----------------------------------------------------------------------
C     USAGE:                                                           I
C                                                                      I
C     TO EVALUATE THE FOURIER TRANSFORM OF THE N = 2**M DATA STORED    I
C       IN THE COMPLEX VECTOR A:                                       I
C                                                                      I
C     CALL VMDFT(N, A, IER)                                            I
C-----------------------------------------------------------------------
```

```
C-----------------------------------------------------------------------
C                      PARAMETER DECLARATIONS
C-----------------------------------------------------------------------
      INTEGER N, IER
      COMPLEX A(N)
C-----------------------------------------------------------------------
C                      INTERNAL DECLARATIONS
C-----------------------------------------------------------------------
      COMPLEX A0, A1, A2, A3, S, T
      REAL C1, C2, C3, DC, DS, FOUR, HALF, ONE, QRTR, RAD
      REAL S1, S2, S3, THRHLF, TI, TR, TWO, ZERO
      INTEGER K, KB, KK, KP, KS, KSH, KSP, KSPM, KSPP, KST
      INTEGER K0, K1, K1MX, K2, K2MX, K3, M, MM, MP, NL
C-----------------------------------------------------------------------
C                       CONSTANTS
C-----------------------------------------------------------------------
      DATA ZERO /0.0/, ONE /1.0/,  TWO /2.0/
      DATA FOUR /4.0/, HALF /0.5/, QRTR /0.25/, THRHLF /1.5/
      RAD = FOUR * ATAN(ONE)
C-----------------------------------------------------------------------
C                STAGE 0 - TEST INPUT AND INITIALIZE                   I
C-----------------------------------------------------------------------
      IER = 1
      NL = N
      IF (NL .LE. 0)            RETURN
      IER = 2
C--------------------------------------------->>> COMPUTE LOG BASE 2 OF N
      M = 0
      MM = 1
10    IF ( MM .GE. NL)          GOTO 20
      MM = MM + MM
      M = M + 1
      GOTO 10
20    IF( MM .GT. NL)           RETURN
      IER = 0
      MP = M / 2
```

```
C-----------------------------------------------------------------------
C                STAGE 1 - SCRAMBLING THE DATA                         I
C-----------------------------------------------------------------------
C     BINARY REVERSAL BY SEQUENTIAL EXCHANGE OF SYMMETRICALLY LOCATED  I
C     BITS REQURIED M/2 SEQUENTIAL PASSES OVER THE DATA AND HAS MUCH   I
C     LOWER OVERHEAD FOR MACHINES WITH PAGED MEMORY THAN USING A       I
C     BINARY REVERSAL COUNTER                                          I
C-----------------------------------------------------------------------
C     THE CODE IS A VERY SLIGHTLY MODIFIED VERSION OF THE SUBROUTINE   I
C     "UNSBIN" GIVEN IN                                                I
C     POLGE, R. J., B. K. BHAGAVAN, AND J. M. CARSWELL, "FAST          I
C     COMPUTATIONAL ALGORITHMS FOR BIT REVERSAL," IEEE TRANSACTIONS    I
C     ON COMPUTERS, C-23(1), PP. 1-9.                                  I
C-----------------------------------------------------------------------
      KSP = 1
      KSH = NL
C------------------------------------- >>> DO (M/2) PASSES OVER THE DATA
      DO 140 KP = 1, MP
            KSPP = KSP + 1
            KSPM = KSP - 1
            KS = KSH
            KSH = KSH / 2
            KST = KSH - KSP
            KSP = KSP + KSP
C-------------------------------------->>>STEP THROUGH THE FILE SEGMENTS
            DO 130 K0 = KSPP, NL, KS
                  K1MX = K0 + KST
C---------------------------->>>STEP THROUGH BLOCKS IN EACH FILE SEGMENT
                  DO 120 K1 = K0, K1MX, KSP
                  K2MX = K1 + KSPM
C--------------------------------------------->>>SWAP TWO BLOCKS OF DATA
                        DO 110 K2 = K1, K2MX
                              K3 = K2 + KST
                              S = A(K2)
                              A(K2) = A(K3)
                              A(K3) = S
110                      CONTINUE
120                CONTINUE
130          CONTINUE
140    CONTINUE
```

```
C---------------------------------------------------------------------
C           STAGE 2 - THE TRANSFORMATIONS                              I
C---------------------------------------------------------------------
C    RADIX 4 + 2 FAST FOURIER TRANSFORM ROUTINE USING A COOLEY-TUKEY   I
C    LIKE ALGORITHM FOR BINARY REVERSED DATA. THE ORGANIZATION USED    I
C    REQUIRES THE MINIMUM NUMBER OF SEQUENTIAL PASSES THROUGH THE      I
C    DATA AND ALLOWS THE SINE-COSINE VALUES TO BE GENERATED IN         I
C    SEQUENTIAL ORDER AS WELL.                                         I
C---------------------------------------------------------------------
C                                                                      I
C     THE CODE IS ADAPTED FROM THE ALGOL PROCEDURE "REVFFT4" FROM      I
C     CACM ALGORITHM 345, "ALGORITHM 345, AN ALGOL CONVOLUTION         I
C     PROCEDURE BASED ON THE FAST FOURIER TRANSFORM," BY RICHARD C,    I
C     SINGLETON.                                                       I
C---------------------------------------------------------------------
C              IF M IS ODD THEN DO A RADIX 2 TRANSFORM
C---------------------------------------------------------------------
      KSP = 1
      IF ( 2 * MP .EQ. M)  GOTO 215
      DO 210 K0 = 1, NL, 2
            K2 = K0 + 1
            A0 = A(K2)
            A(K2) = A(K0) - A0
            A(K0) = A(K0) + A0
210   CONTINUE
      KSP = KSP + KSP
      RAD = HALF * RAD
C---------------------------------------------------------------------
                  NOW DO M/2 RADIX 4 TRANSFORMS
C---------------------------------------------------------------------
215   IF ( MP .LT. 1 )  RETURN
      DO 250 KP = 1, MP
            RAD = QRTR * RAD
C---------------------------------------------------------------------
C                             NOTE: SIN-COS REFERENCES
C                             CAN BE REPLACED BY
C                             INLINE APPROXIMATIONS
C---------------------------------------------------------------------
            DC = TWO * SIN(RAD) ** 2
            DS = SIN(TWO * RAD)
            KST = 4 * KSP
```

```
C--------------------------------------------------BLOCK BEING TRANSFORMED
            DO 240 KB = 1, NL, KST
                  C1 = ONE
                  S1 = ZERO
C-----------------------------------------------------TRANSFORM THE BLOCK
                  DO 230 KK = 1, KSP
                        K = KK - 1
                        K0 = KB + K
                        K1 = K0 + KSP
                        K2 = K1 + KSP
                        K3 = K2 + KSP
                        A0 = A(K0)
                        A2 = A(K1)
                        A1 = A(K2)
                        A3 = A(K3)
                        IF ( K .EQ. 0 )          GOTO 225
C                                              ----------------------------
C                                              ADVANCE TRIG FUNCTION VALUES
C                                              ----------------------------
                        C2 = C1 - (DC * C1 + DS * S1)
                        S1 = S1 + (DS * C1 - DC * S1)
C                                     -------------------------------------
C                                      THE FOLLOWING 3 STATEMENTS COMPENSATE
C                                     FOR TRUNCATION ERROR.  IF ROUNDED
C                                      ARITHMETIC IS USED, REPLACE THEM WITH
C                                                   C1 = C2
C                                     -------------------------------------
                        C1 = THRHLF - HALF * (C2 * C2 + S1 * S1)
                        S1 = C1 * S1
                        C1 = C1 * C2
                        C2 = C1 * C1 - S1 * S1
                        S2 = TWO * S1 * C1
                        C3 = C2 * C1 - S2 * S1
                        S3 = C2 * S1 + S2 * C1
C-------------------------------------------->>>APPLY RADIX 4 TRANSFORM
                        A2 = A2 * CMPLX(C2, S2)
                        A1 = A1 * CMPLX(C1, S1)
                        A3 = A3 * CMPLX(C3, S3)
225                     S  = (A0 + A2)
                        T  = (A1 + A3)
                        A(K0) = S + T
                        A(K2) = S - T
                        S =  (A0 - A2)
                        T  = (A1 - A3)
                        TR = REAL(T)
                        TI = AIMAG(T)
                        T = CMPLX(-TI, TR)
                        A(K1) = S + T
                        A(K3) = S - T
230               CONTINUE
240         CONTINUE
            KSP = KST
250   CONTINUE
                              RETURN
      END
```

APPENDIX B - ALGEBRAIC THEORY

DFT Identities

This appendix gives the mathematical background and required identities for the DFT viewed as a matrix operator. We begin with some definitions. The results have been previously published, but the development is more complete than published results and the actual development is more directly useful in teasing out details for actual implementation.

A **permutation matrix** is any matrix which can be obtained from an identity matrix by permuting its rows or columns. Let the set of numbers $\{c(j)\}$, $0 \le j \le p-1$, be a permutation of the integers $0 \ldots p-1$, and let $\{r(j)\}$ be the inverse permutation:

$$r(c(j)) = c(r(j)) = j$$

The permutation matrix P, of order p, is defined by

$$P_{jk} = \delta(j,r(k)) = \delta(c(j),k),\ 0 \le j,\ k \le p-1.$$

The functions r(j) and c(j) are called the **row** and **column** functions, respectively. Thus the matrix has just one non-zero element (equal to unity) in each row and column. The unit element in row j is c(j), and the unit element in column k is in row r(k).

The following properties of permutation matrices follow directly from the definition:

(1) The transpose is the inverse: $P^T = P^{-1}$

(2) If P_1 and P_2 are permutation matrices of the same order with row functions r_1 and r_2 and columns c_1 and c_2, then their product $P = P_1P_2$ is a permutation matrix with row and column functions given by the transitive formulas

$$r(j) = r_1[r_2(j)]$$
$$c(j) = c_2[c_1(j)]$$

(3) IF X is an arbitrary matrix, the effect of premultiplying by a permutation matrix is to permute the rows, while post multiplying permutes the columns:

Let

$$Y = PX$$
$$Z = XP$$

Then

$$Y_{jk} = X_{c(j),k}$$
$$Z_{jk} = X_{j,r(k)}.$$

In particular, if

$$U = XP^T$$

then

$$U_{jk} = X_{j,c(k)}$$

That is, premultiplying by P produces a permutation of the rows, postmultiplying by the transpose P^T produces the *same* permutation of the columns.

Two important permutation matrices are the **square deal** $S_{p,q}$ and its transpose (and inverse) the **perfect shuffle**, $T_{q,p}$, each of order pq, defined by the the row and column functions

$$c_S(j + lq) = jp + l = r_T(j + lq)$$
$$r_S(jp + l) = j + lq = c_T(jp + l)$$

with $0 \le j \le q - 1, 0 \le k \le p - 1$.

Premultiplying by S represents dealing a deck of pq cards to p players, while premultiplying by T represents shuffling (merging) p decks of q cards each. The terminology is actually somewhat ambiguous, since the roles are reversed in postmultiplication. Note the identities

$$S_{q,p} = T_{p,q} = S^T_{p,q} = S^{-1}_{p,q}$$
$$S_{p,p} = T_{p,p}$$
$$S_{p,1} = S_{1,p} = T_{p,1} = T_{p,1} = I_p$$

The **tensor product** of two matrices, A and B, denoted A⊗B, is the partitioned block matrix obtained by multiplying each element of A by the matrix B

$$A \otimes B = (A_{l,m} B)$$

more specifically, if A is of order p and B is of order q, and $C = A \otimes B$, then C is of order pq, with elements given by

$$C_{j+lq,\, k+mq} = A_{l,m} B_{j,k}\ ,\ 0 \le j, k \le q - 1, 0 \le l, m \le p - 1.$$

The following properties follow directly from the definition:

Associative Law:

$$(A \otimes B) \otimes C = A \otimes (B \otimes C) = A \otimes B \otimes C \tag{2}$$

Similarity Law: If A is of order p and B is of order q, then

$$B \otimes A = T_{p,q} (A \otimes B) S_{p,q} \tag{3}$$

Transpose:

$$(A \otimes B)^T = A^T \otimes B^T \tag{4}$$

Inverse:

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{5}$$

Identity:

$$\left.\begin{array}{r} A \otimes I_1 = I_1 \otimes A = A \\ I_p \otimes I_q = I_{pq} \end{array}\right| \tag{6}$$

Multiplicative Law: $$(AB)\otimes(CD) = (A\otimes C)(B\otimes D) \tag{7}$$

Distributive Law: $$I \otimes \prod_j A_j = \prod_j (I\otimes A_j), \left(\prod_j A_j\right) \otimes I = \prod_j (A_j\otimes I) \tag{8}$$

where the Aj are all of the same order, I is an identity matrix of arbitrary order, and $\prod_j$ denotes the ordinary product of a finite number of factors.

Basic Factor Theorem If P is of order p and Q is of order Q. Then

$$\begin{aligned} P\otimes Q &= (PI_p) \otimes (I_qQ) \\ &= (P \otimes I_q)(I_p \otimes Q) \\ &= (I_pP) \otimes (QI_q) \\ &= (I_p \otimes Q)(P \otimes I_q) \end{aligned} \tag{9}$$

Thus, the tensor product can be expressed as an ordinary (commutative) product), each factor being a tensor product with an identity matrix.

If P and Q are permutation matrices, then so is their tensor product $A = P\otimes Q$, and its column function is

$$c_A(j + lq) = c_Q(j) + q\, c_P(l),\ 0 \le j < q,\ 0 \le l < p$$

and its row function is

$$r_A(k + mq) = r_Q(k) + qr_P(m),\ 0 \le k < q,\ 0 \le m < p$$

since

$$A_{j + lq,\, k + mq} = P_{l,\, m}Q_{j,\, k}$$

The tensor product has a simple representation when one factor is a permutation matrix and the other is an identity matrix. Let P be a permutation matrix of order n with column function c(j), and let I_s be the identity matrix of order s. The matrices

$$Q = P\otimes I_s$$
$$R = I_s\otimes P$$

are permutation matrices with column functions

$$\left.\begin{aligned} c_Q(js+l) &= sc(j)+l \\ c_R &= ln + c(j) \end{aligned}\right\} \begin{aligned} 0 \le j < n \\ 0 \le l < s \end{aligned}$$

So, if P is regarded as the operator that shuffles (permutes) a deck of cards, then the effect of Q is to perform the same permutation on a deck of "packets", each packet containing s cards that are stuck together. Thus, for example, the matrices

$$\begin{gathered} S_{p,q} \otimes I_s \\ T_{p,q} \otimes I_s \end{gathered} \tag{10}$$

can be described as the "dirty deal" and the "sticky shuffle" respectively. On the other hand, the effect of R is to shuffle s decks in parallel and then stack them together. Combining these gives the formula for the triple product:

Let $B = I_p \otimes P_q \otimes I_r$ where P_q is a permutation matrix of order q with column function $c_P(j)$. Then B is a permutation matrix of order pqr with column function $c_B(j)$ given by

$$c_B(\alpha qr + \beta r + \gamma) = \alpha qr + rc_P(\beta) + \gamma$$

$$\text{for } 0 \le \alpha < p,\ 0 \le \beta < q,\ 0 \le \gamma < r$$

More generally, if A, B, C are arbitrary matrices of order p, q, r respectively, then the elements of the triple product

$$D = A \otimes B \otimes C$$

are given by

$$D_{iqr+kr+m,\, jqr+lr+n} = A_{i,j}B_{k,l}C_{m,n}$$

$$0 \le i, j < p;\ 0 \le k, l < q;\ 0 \le m, n < r$$

Even Order DFT

The reduction of the Fourier matrix of even order M = 2q to a product of simpler matrices depends upon the simple observation

$$\left.\begin{aligned} \omega_{2q}^{2jk} &= \omega_q^{jk} \\ \omega_{2q}^{(2j+1)k} &= \omega_q^{jk}\omega_{2q}^{k} \end{aligned}\right\} \quad \begin{aligned} &0 \le j < q \\ &0 \le k < 2q \end{aligned}$$

That is, the even numbered rows of $\mathcal{W}_{2q}$ contain the elements of $\mathcal{W}_q$, while the elements in the odd numbered rows are products of elements from both $\mathcal{W}_q$ and $\mathcal{W}_{2q}$. This suggests permuting the rows of W_{2q} to bring the even numbered rows together in one block and the odd numbered rows into another. This can be done by using $S_{2,q}$. Define $V = S_{2,q}\mathcal{W}_{2q}$. The elements of V are

$$V_{j,k} = \omega_q^{jk}$$
$$V_{j+q,k} = \omega_q^{jk}\omega_{2q}^{k}$$

for $0 \le j, k < q$. The columns can be partitioned by noting

$$\omega_q^{j(k+q)} = \omega_q^{jk}$$
$$\omega_{2q}^{(k+q)} = -\omega_{2q}^{k}$$

for $0 \le j, k < q$. Thus the elements of V are

$$V_{j,k} = \omega_q^{jk}, \quad V_{j,k+q} = \omega_q^{jk}$$
$$V_{j+q,k} = \omega_q^{jk}\omega_{2q}^{k}, \quad V_{j+q,k+q} = -\omega_q^{jk}\omega_{2q}^{k}$$

for $0 \le j, k < q$. Define

$$D_q(x) = \begin{bmatrix} 1 & & & & \\ & x & & 0 & \\ & & x^2 & & \\ & 0 & & \ddots & \\ & & & & x^{q-1} \end{bmatrix}, \; D_q(1) = I_q, D_1(x) = I_1$$

Then

$$V = \begin{bmatrix} \mathcal{W}_q & \mathcal{W}_q \\ \mathcal{W}_q D_q(\omega_{2q}) & -\mathcal{W}_q D_q(\omega_{2q}) \end{bmatrix}$$

$$= \begin{bmatrix} \mathcal{W}_q & 0 \\ 0 & \mathcal{W}_q \end{bmatrix} \begin{bmatrix} I_q & 0 \\ 0 & D_q(\omega_{2q}) \end{bmatrix} \begin{bmatrix} I_q & I_q \\ I_q & -I_q \end{bmatrix}$$

which using the results from the previous section can be written compactly as

$$W_{2q} = T_{2,q}\,(I_2 \otimes \mathcal{W}_q)\,\Delta_{2,q}\,(\mathcal{W}_2 \otimes I_q) \tag{11}$$

Where

$$\Delta_{2,q} = \begin{bmatrix} I_q & 0 \\ 0 & D_q(\omega_{2q}) \end{bmatrix} \text{and } W_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The Fundamental Factor Theorem

From the above, the DFT of M = 2q points can be written as a product of sparse matrices. If the integer q is, itself, a power of two, then the above can be iterated to yield the complete Cooley-Tukey algorithm. However, the process generalizes to general factors.

Let

$$\Delta_{p,q} = \begin{bmatrix} I_q & & & & \\ & D_q(\omega_{pq}) & & 0 & \\ & & D_q(\omega_{pq}^2) & & \\ & 0 & & \ddots & \\ & & & & D_q(\omega_{pq}^{p-1}) \end{bmatrix}.$$

Note that

$$\Delta_{q,p} = T_{p,q}\,\Delta_{p,q}\,S_{p,q} \tag{12}$$

and reduces to the identity if either subscript is unity

$$\Delta_{p,1} = \Delta_{1,p} = I_p$$

Fundamental Factor Theorem

$$\mathcal{W}_{pq} = T_{p,q}\left(I_p \otimes \mathcal{W}_q\right)\Delta_{p,q}\left(\mathcal{W}_p \otimes I_q\right) \tag{13a}$$

alternatively,

$$\mathcal{W}_{pq} = \left(\mathcal{W}_q \otimes I_p\right)\Delta_{q,p}\left(I_q \otimes \mathcal{W}_p\right)T_{p,q} \tag{13b}$$

Proof: We first prove (13a). By analogy with the even order case. Start by applying the square deal to $\mathcal{W}$

$$V = S_{p,q}\ \mathcal{W}_{pq}$$

The elements of V are

$$V_{j+lq,\,k+mq} = \sum_{r=0}^{pq-1} S_{p,q_{j+lq,\,r}}\ \mathcal{W}_{r,\,k+mq},\ 0 \le j,\ k \le q,\ 0 \le l,\ m < p.$$

But

$$\begin{aligned} S_{p,q_{j+lq,\,n}} &= \delta[n, cs(j+lq)] \\ &= \delta[n, jp+l] \end{aligned}$$

so that

$$\begin{aligned} V_{j+lq,\,k+mq} &= W_{jp+l,\,k+mq} \\ &= \omega_{pq}^{(jp+l)(k+mq)} \end{aligned}$$

but recalling that

$$\omega_{pq} = e^{2\pi i/pq},$$

the identities

$$\begin{aligned} \omega_{pq}^{p} &= \omega_q \\ \omega_{pq}^{q} &= \omega_q \\ \omega_{pq}^{pq} &= 1 \end{aligned}$$

give

$$V_{j+lq,\,k+mq} = \omega_q^{jk}\omega_{pq}^{lk}\omega_p^{lm}$$

Thus, V is a partitioned block matrix, with (l, m) denoting the row and column position of each block and (j, k) denoting the local row and column position of each element within the block $V = (V_{l,m})$, $0 \le l, m < q$. Hence,

$$V_{l,m} = \mathcal{W}_q D_q(\omega_{pq}^l)\omega_p^{lm} I_q$$

Since $\mathcal{W}_q$ is a common factor of every block and $D_q(\omega_{pq}^l)$ is a common factor of every block in row l, V can be factored:

$$V = \begin{bmatrix} \mathcal{W}_q & & & \\ & \mathcal{W}_q & & 0 \\ & & \mathcal{W}_q & \\ 0 & & & \ddots \\ & & & & \mathcal{W}_q \end{bmatrix} \begin{bmatrix} I_q & & & & \\ & D_q(\omega_{pq}) & & 0 & \\ & & D_q(\omega_{pq}^2) & & \\ & 0 & & \ddots & \\ & & & & D_q(\omega_{pq}^{p-1}) \end{bmatrix} (\mathcal{W}_p \otimes I_q)$$

that is,

$$V = (I_p \otimes \mathcal{W}_q)\,\Delta_{p,q}\,(\mathcal{W}_p \otimes I_q)$$

Since, $V = S_{p,q}\,\mathcal{W}_{pq}$ and $T_{p,q}$ is the inverse of $S_{p,q}$, the result follows immediately.

(13b) follows by writing

$$\begin{aligned}\mathcal{W}_{pq} &= T_{p,q}(I_p \otimes \mathcal{W}_q)S_{p,q}\,T_{p,q}\,\Delta_{p,q}\,S_{p,q}\,T_{p,q}(\mathcal{W}_p \otimes I_q)S_{p,q}\,T_{p,q} \\ &= (T_{p,q}(I_p \otimes \mathcal{W}_q)S_{p,q})(T_{p,q}\,\Delta_{p,q}\,S_{p,q})(T_{p,q}(\mathcal{W}_p \otimes I_q)S_{p,q})T_{p,q}\end{aligned}$$

and then applying (3) and (12).

Complete Factor Theorem

Clearly the fundamental factor theorem can be iterated if either of the integers p, q is not prime. For example, take $p = p_2$, $q = p_1$, then

$$\mathcal{W}_{p_2p_1} = T_{p_2,\, p_1} \left(I_{p_2} \otimes \mathcal{W}_{p_1}\right) \Delta_{p_2,\, p_1} \left(\mathcal{W}_{p_2} \otimes I_{p_1}\right)$$

Now take $p = p_3$, $q = p_2p_1$ in the fundamental theorem giving

$$\mathcal{W}_{p_3p_2p_1} = T_{p_3,\, p_2p_1} \left(I_{p_3} \otimes \mathcal{W}_{p_2p_1}\right) \Delta_{p_3,\, p_2p_1} \left(\mathcal{W}_{p_3} \otimes I_{p_2p_1}\right)$$

Inserting the expression given above for $\mathcal{W}_{p_2p_1}$ and using the distributive law,

$$\mathcal{W}_{p_3p_2p_1} = T_{p_3,\, p_2p_1} \left(I_{p_3} \otimes T_{p_2,\, p_1}\right)\left(I_{p_3p_2} \otimes \mathcal{W}_{p_1}\right)\left(I_{p_3} \otimes \Delta_{p_2,\, p_1}\right)\left(I_{p_3} \otimes \mathcal{W}_{p_2} \otimes I_{p_1}\right) \Delta_{p_3,\, p_2p_1} \left(\mathcal{W}_{p_3} \otimes I_{p_2p_1}\right)$$

To continue the process, only a proper notation is necessary. To this end, consider an indefinite sequence of positive integers, $\{p_j\}$, each $p_j > 1$. Define arrays r_j and $q_{n,j}$ by

$$r_1 = 1$$
$$r_{j+1} = p_j\, r_j,\ j \geq 1$$

so that

$$r_j = \prod_{k=1}^{j-1} p_k,\ j \geq 2$$

$$q_{n,n} = 1,\ n \geq 0$$
$$q_{n,j} = p_{j+1} q_{n,j+1},\ 0 \leq j < n$$

so that

$$q_{n,j} = \prod_{k=j+1}^{n} p_k,\ n \geq 1$$

Further, define

$$M_n = \prod_{k=1}^{n} p_k,\ n \geq 1$$

so that

$$r_{n+1} = q_{n,0} = M_n\ ,\ n \geq 1$$

Now note that

$$q_{n+1,j} = p_{n+1}\, q_n\, ,j,\ 0 \leq j \leq n$$
$$p_j\, q_{n,j}\, r_j = M_n,\ 1 \leq j \leq n$$

Then the procedure outlined above yields, by induction,

The Complete Factorization Theorem (CFT) Let

$$\left.\begin{aligned} B_{M_1} &= B_{p_1} = I_{p_1} \\ B_{M_{n+1}} &= T_{p_{n+1}, r_{n+1}}\left(I_{p_{n+1}} \otimes B_{M_n}\right),\ n \geq 1 \end{aligned}\right\} \tag{14}$$

then

$$\mathcal{W}_{M_n} = B_{M_n} \prod_{j=1}^{n} \left(I_{q_{n,j}} \otimes \Delta_{p_j, r_j}\right)\left(I_{q_{n,j}} \otimes \mathcal{W}_{p_j} \otimes I_{r_j}\right) \tag{15a}$$

alternatively,

$$\mathcal{W}_{M_n} = \prod_{j=n}^{1} \left(I_{q_{n,j}} \otimes \mathcal{W}_{p_j} \otimes I_{r_j}\right)\left(I_{q_{n,j}} \otimes \Delta_{p_j, r_j}\right) C_{M_n} \tag{15b}$$

where

$$\left.\begin{aligned} C_{M_1} &= C_{p_1} = I_{p_1} \\ C_{M_{n+1}} &= \left(I_{p_{n+1}} \otimes C_{M_n}\right) T_{r_{n+1}, p_{n+1}},\ n \geq 1 \end{aligned}\right\}$$

Notice that B_{M_n} is a permutation matrix of order M_n. Also notice that the notation is not complete because the matrix B_{M_n} depends on the **ordered** set of integers $p_1, p_2, \ldots, p_n$, and not simply on their product M_n. In fact, by induction,

$$B_{M_n} = \prod_{j=n}^{1} \left(I_{q_{n,j}} \otimes T_{p_j, r_j}\right) \tag{16}$$

Since matrix multiplication is not commutative, any permutation of the order of the p_j would change the matrix B_{M_n}. Also, the arrays r and q are defined with respect to the **ordered** sequence of p's.

Proof: First we prove (15a). First note that $r_1 = q_{n,n} = 1$, $\Delta_{p_1, r_1} = I_p$ so that the theorem is true for $n = 1, 2, 3$. Assume it is true for n. Then

$$\mathcal{W}_{M_{n+1}} = \mathcal{W}_{p_{n+1} M_n} = \mathcal{W}_{p_{n+1} r_{n+1}}$$

Setting $p = p_{n+1}$, $p = r_{n+1} = M_n$ in the fundamental factor theorem gives

$$\mathcal{W}_{M_{n+1}} = T_{p_{n+1}, r_{n+1}}\left(I_{p_{n+1}} \otimes \mathcal{W}_{M_n}\right) \Delta_{p_{n+1} r_{n+1}} \left(\mathcal{W}_{p_{n+1}} \otimes I_{r_{n+1}}\right)$$

By the induction hypothesis, $\mathcal{W}_{M_n}$ is given by the theorem. Hence, by the distributive law,

$$I_{p_{n+1}} \otimes \mathcal{W}_{M_n} = \left(I_{p_{n+1}} \otimes B_{M_n}\right) \prod_{j=1}^{n} \left(I_{p_{n+1} q_{n,j}} \otimes \Delta_{p_j, r_j}\right)\left(I_{p_{n+1} q_{n,j}} \otimes \mathcal{W}_{p_j} \otimes I_{r_j}\right)$$

But $p_{n+1} q_{n,j} = q_{n+1,j}$. Hence,

$$\mathcal{W}_{M_{n+1}} = T_{p_{n+1}, r_{n+1}}\left(I_{p_{n+1}} \otimes B_{M_n}\right) \prod_{j=1}^{n} \left(I_{q_{n+1,j}} \otimes \Delta_{p_j, r_j}\right)\left(I_{q_{n+1,j}} \otimes \mathcal{W}_{p_j} \otimes I_{r_j}\right) \Delta_{p_{n+1}, r_{n+1}} \left(\mathcal{W}_{p_{n+1}} \otimes I_{r_{n+1}}\right)$$

The last two factors are simply those for $j = n + 1$. Therefore, the (15a) is proved. Equation (15b) can be proved similarly.

Computing the DFT using the CFT

To compute the DFT of a vector, two basic matrix-vector algorithms are needed.

I. $\vec{b} = \left(I_q \otimes \mathcal{W}_p \otimes I_r\right) \vec{a}$

II. $\vec{c} = \left(I_q \otimes \Delta_{p,r}\right) \vec{b}$

The definitions yield:

$$\vec{b}_{kpr + lr + m} = \sum_{\alpha=0}^{p-1} \omega_p^{l\alpha} a_{kpr + \alpha r + m}$$

$$\vec{c}_{kpr + lr + m} = \omega_{pr}^{lm} \vec{b}_{kpr + lr + m}$$

for $0 \le k < q$, $0 \le l < p$, $0 \le m < r$, and, for any β, $\omega_\beta = \exp(2\pi i / \beta)$. In the binary case,

$$\left.\begin{array}{ll} \text{I:} & b_\alpha = a_\alpha + a_{\alpha + r} \\ & b_\beta = a_{\beta - r} - a_\beta \\ \text{II:} & c_\alpha = b_\alpha \\ & c_\beta = \omega_{pr}^{m} b_\beta \end{array}\right| \quad \begin{array}{l} \text{for } a = 2kr + m,\ b = (2k+1)r + m \\ \text{with } 0 \le k < q,\ 0 \le m < r \end{array}$$

Of course, there remains the the final multiplication of a vector by the permutation matrix B_{M_n}. This requires an expression for its column function. To obtain this, we briefly digress into number theory.

Theorem If j is any integer in the range $0 \le j < M_n$ then j has the unique representations

(1) $$j = \sum_{k=1}^{n} \alpha_k r_k$$

(2) $$j = \sum_{k=1}^{n} \beta_k q_{n,k}$$

with $0 \le \alpha_k, \beta_k < p_k$ where the a and β are given by:

(1) $$\begin{cases} R_1 = \left[\frac{j}{p_1}\right], \alpha_1 = j - p_1 R_1 \\ \text{for } k = 2, \cdots, n, R_k = \left[\frac{R_{k-1}}{p_k}\right], \alpha_k = R_{k-1} - p_k R_k \end{cases}$$

(2) $$\begin{cases} Q_1 = \left[\frac{j}{p_n}\right], \beta_n = j - p_n Q_n \\ \text{for } k = n, \cdots, 2, Q_{k-1} = \left[\frac{Q_k}{p_{k-1}}\right], \beta_{k-1} = Q_k - p_{k-1} Q_{k-1} \end{cases}$$

where [x] is the greatest integer less than or equal to x.

Proof: of (1). The definition of [x] gives

$$\frac{j}{p_1} - 1 < R_k \le \frac{j}{p_1}$$

giving $$0 \le \alpha_1 < p_1$$

and

$$\frac{R_{k-1}}{p_k} - 1 < R_k \le \frac{R_{k-1}}{p_k}$$

gives $$0 \le \alpha_k < p_k.$$

Next,

$$R_1 \leq \frac{j}{p_1} < \frac{M_n}{p_1} = q_{n,1}$$

and, by induction, $R_k < q_{n,k}$
Hence, $R_n < 1$, that is, $R_n = 0$.
Thus,

$$\begin{aligned}\sum_{k=1}^{n} \alpha_k r_k &= \alpha_1 r_1 + \sum_{k=2}^{n} \alpha_k r_k \\ &= j - p_1 R_1 + \sum_{k=2}^{n} (R_{k-1} - p_k R_k)\, r_k \\ &= j - p_1 R_1 + \sum_{k=2}^{n} (R_{k-1} r_k - R_k r_{k+1}) \\ &= j - p_1 R_1 + R_1 r_2 - R_n r_{n+1}\end{aligned}$$

but $r_1 = p_1$, $R_n = 0$, so $\sum_{k=1}^{n} \alpha_k r_k = j$.

(2) is proved similarly.

The Scrambling Matrix The column function for the scrambling matrix, B_{M_n}, can now be given. For $0 \leq j < M_n$, we have $j = \sum_{k=1}^{n} \beta_k q_{n,k}$. The column function is

$$C_{B_{M_n}}(j) = \sum_{k=1}^{n} \beta_k r_k$$

This is easily proved by induction on n.
For $n = 1$, $k = 1$, $q_{1,1} = r_1 = 1$ and $j = \beta_1 = c(j)$ which is correct since $B_{p_1} = I_{p_1}$, the identity matrix.
Assume the result is true for n. By the recursive definition,

$$B_{M_{n+1}} = T_{p_{n+1}, r_{n+1}}(I_{p_{n+1}} \otimes B_{M_n})$$

and for $0 \leq j < M_{n+1}$ use the representation $j = \sum_{k=1}^{n+1} \beta_k q_{n+1,k}$. Since $q_{n+1,k} = p_{n+1} q_{n,k}$ for

$1 \leq k \leq n$, we can write $j = \sum_{k=1}^{n} \beta_k q_{n,k} p_{n+1} + \beta_{n+1}$, or $j = l\, p_{n+1} + m$ with

$$l = \sum_{k=1}^{n} \beta_k q_{n,k},\ m = \beta_{n+1}$$

so that

$$0 \leq l < M_n = r_{m+1},\ 0 \leq m < p_{n+1}.$$

By definition of the perfect shuffle,

$$l = \sum_{k=1}^{n} \beta_k q_{n,k},\ m = \beta_{n+1}$$

Let $D = I_{p_{n+1}} \otimes B_{M_n}$. The column function for D is

$$c_D(l + mM_n) = mM_n + c_{B_n}(l) = \beta_{n+1} r_{n+1} + c_{B_n}(l).$$

By the inductive hypothesis

$$c_{B_n}(l) = \sum_{k=1}^{n} \beta_k r_k$$

Hence,

$$c_{B_{n+1}}(j) = c_D[c_T(j)] = \beta_{n+1} r_{n+1} + \sum_{k=1}^{n} \beta_k r_k = \sum_{k=1}^{n+1} \beta_k r_k$$

Calculation of $c_{B_{M_n}}(j)$ is easily implemented. If

$$j = \sum_{l=1}^{n} \beta_l q_{n,l}$$

then

$$k \equiv c(j) = \sum_{l=1}^{n} \beta_l r_l$$

Define

$$k_m = \sum_{l=1}^{m} \beta_l r_l \text{ , } 1 \le m \le n, k_0 = 0$$

then

$$k_m = k_{m-1} + \beta_m r_m \text{ , } 1 \le m \le n, k_n = k$$

Set j = 0

```
for $k_1 := 0$ to $r_2 - 1$ by $r_1$ do                       |
  for $k_2 := k_1$ to $r_3 - 1$ by $r_2$ do                   |
              :                                               |  "n" loops     (17)
    for $k_m := k_{m-1}$ to $r_{m+1} - 1$ by $r_m$ do         |
              :                                               |
      for $k_n := k_{n-1}$ to $M - 1$ by $r_n$ do             |
        begin
          $c(j) := k_n$;
          $j := j + 1$;
        end;
```

Alternatively, define $J_n = \sum_{l=m}^{n} \beta_l q_{l,\,n}$, $i \le m \le n$, so that $J_1 = j$ and $J_m = J_{m+1} + \beta_m q_{n,\,n}$, $1 \le m < n$.

Thus, set K = 0

```
for $j_n := 0$ to $q_{n,\,n-1} - 1$ by $q_{n,\,n}$ do                 |
  for $j_{n-1} := j_n$ to $q_{n,\,n-2} - 1$ by $q_{n,\,n-1}$ do       |
              :                                                       |  "n" loops
    for $j_m := j_{m+1}$ to $q_{n,\,m-1} - 1$ by $q_{n,\,m}$ do       |
              :                                                       |
      for $j_1 := j_2$ to $M - 1$ by $q_{n,\,1}$ do                   |
        begin                                                                        (18)
          $c(j_1) := k$;
          $k := k + 1$;
        end;
```

For the binary case, $p_j = 2$, $r_j = 2^{j-1}$, $q_{n,j} = 2^{n-j}$. So if $j = \sum_{k=1}^{n} \beta_k 2^{n-k}$, $0 \le \beta_k \le 1$, then

$$c_B(j) = \sum_{k=1}^{n} \beta_k 2^{k-1}, 0 \le \beta_k \le 1$$

is the column function, which can be written

$$c_B(j) = \sum_{k=1}^{n} \beta_{n+1-k} 2^{n-k}$$

so B is symmetric and hence $B^{-1} = B$. Thus when j is represented as an n-bit binary number, c(j) is obtained from j by simply reversing the order of its bits. For this reason, B is also called the **bit-reversal** matrix. In this case, a simpler notation can be used for the perfect shuffle and square deal matrices. Define

$$\left.\begin{array}{l} T_{2^n} \equiv T_{2,\,2^{n-1}} \\ S_{2^n} \equiv S_{2,\,2^{n-1}} \end{array}\right\} n \ge 1, T_1 = S_1 = I_1$$

Then with $j = \sum_{k=1}^{n} \beta_k 2^{n-k}$, the column functions for S and T are

$$c_T(j) = \sum_{k=1}^{n} \gamma_k 2^{n-k}$$

$$c_S(j) = \sum_{k=1}^{n} \sigma_k 2^{n-k}$$

where $\gamma_1 = \beta_n$, $\gamma_k = \beta_{k-1}$, $k > 1$, $\gamma_1 = \beta_n$, $\sigma_n = \beta_1$, $\sigma_k = \beta_{k+1}$, $k < n$. Thus writing j in binary notation: $j = \beta_1 \beta_2 \cdots \beta_{n-1} \beta_n$ gives

$$c_T(j) = \beta_n \beta_1 \beta_2 \cdots \beta_{n-1}$$

$$c_S(j) = \beta_2 \beta_3 \cdots \beta_n \beta_1$$

So $c_T(j)$ = is obtained by shifting the bits in j one place to the right, end around, and $c_S(j)$ is obtained by shifting the bits one place to the left, end around.

We now introduce one more permutation matrix, the **exchange matrix** X_{2^n}, defined by its column function $c_X(j)$:

$$j = \sum_{k=1}^{n} \beta_k 2^{n-k}$$

$$c_X(j) = \sum_{k=1}^{n} \chi_k 2^{n-k}, \chi_1 = \beta_n, \chi_n = \beta_1, \chi_k = \beta_k, 1 < k < n$$

Thus, the action of X is to exchange the first and last bits of j. So

$$\left.\begin{aligned} c_X(2k+1) &= 2k + \frac{M}{2} \\ c_X\left(2k + \frac{M}{2}\right) &= (2k+1) \end{aligned}\right\} 0 \le k \le \frac{M}{4}$$
$$c_X(j) = j \text{ for all other values of } j \tag{19}$$

Thus, if

$$\vec{b} = X\,\vec{a},$$

then $\vec{b}$ is obtained from $\vec{a}$ simply by moving the appropriate element $\pm(M/2 - 1)$. Note that X is symmetric. In other words, the odd numbered components in the first half of $\vec{a}$ are exchanged with the even components of the second half of $\vec{a}$. Directly from the definitions we have

$$B_2 = X_2 = T_2 = S_2 = I$$
$$B_4 = X_4 = T_4 = S_4$$
$$B_8 = X_8$$

$$\begin{aligned} T_{2^n} &= X_{2^n}(I_2 \otimes T_{2^{n-1}}) = (T_{2^{n-1}} \otimes I_2)\, X_{2^n} \\ S_{2^n} &= (I_2 \otimes S_{2^{n-1}})\, X_{2^n} = X_{2^n}(S_{2^{n-1}} \otimes I_2) \\ B_{2^n} &= T_{2^n}(I_2 \otimes B_{2^{n-1}}) = (B_{2^{n-1}} \otimes I_2)\, T_{2^n} \\ B_{2^n} &= T_{2^n}(I_2 \otimes B_{2^{n-1}}) = (B_{2^{n-1}} \otimes I_2)\, T_{2^n} \\ &= (I_2 \otimes B_{2^{n-1}})\, S_{2^n} = S_{2^n}(B_{2^{n-1}} \otimes I_2) \\ &= (I_2 \otimes B_{2^{n-2}} \otimes I_2)\, X_{2^n} = X_{2^n}(I_2 \otimes B_{2^{n-2}} \otimes I_2) \end{aligned} \tag{20}$$

By induction, these yield,

$$\left.\begin{aligned} T_{2^n} &= \prod_{j=0}^{n-2} \left(I_{2^j} \otimes X_{2^{j-n}}\right) = \prod_{j=n-2}^{0} \left(X_{2^{n-j}} \otimes I_{2^j}\right) \\ S_{2^n} &= \prod_{j=0}^{n-2} \left(X_{2^{n-j}} \otimes I_{2^j}\right) = \prod_{j=n-2}^{0} \left(I_{2^j} \otimes X_{2^{n-j}}\right) \end{aligned}\right\}, n \geq 2 \tag{21}$$

and

$$\left.\begin{aligned} B_{2^n} &= \prod_{j=0}^{\left\lfloor \frac{n}{2} \right\rfloor - 1} \left(I_{2^j} \otimes X_{2^{n-2j}} \otimes I_{2^j}\right) \\ &= \prod_{j=\left\lfloor \frac{n}{2} \right\rfloor - 1}^{0} \left(I_{2^j} \otimes X_{2^{n-2j}} \otimes I_{2^j}\right) \end{aligned}\right\}, n \geq 2 \tag{22}$$

So scrambling, shuffling, and dealing can be done purely in terms of exchange matrices.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1992	3. REPORT TYPE AND DATES COVERED Technical Memorandum

4. TITLE AND SUBTITLE A Discrete Fourier Transform for Virtual Memory Machines	5. FUNDING NUMBERS 506-59-31
6. AUTHOR(S) David C. Galant	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Ames Research Center Moffett Field, CA 94035-1000	8. PERFORMING ORGANIZATION REPORT NUMBER A-92048
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA TM-103914

11. SUPPLEMENTARY NOTES

Point of Contact: David C. Galant, Ames Research Center, MS 269-3, Moffett Field, CA 94035-1000
(415) 604-4851 or FTS 464-4851

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified-Unlimited Subject Category – 67	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

An algebraic theory of the Discrete Fourier Transform is developed in great detail. Examination of the details of the theory leads to a computationally efficient fast Fourier transform for use on computers with virtual memory. Such an algorithm is of great use on modern desk top machines. A Fortran coded version of the algorithm is given for the case when the sequence of numbers to be transformed is a power of two.

14. SUBJECT TERMS Discrete Fast Fourier Transform, Computer implementation, Numerical analysis	15. NUMBER OF PAGES 30
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT