An Adaptation of the Fast Fourier Transform for Parallel Processing

MARSHALL C. PEASE

Stanford Research Institute, Menlo Park, California*

ABSTRACT. A modified version of the Fast Fourier Transform is developed and described. This version is well adapted for use in a special-purpose computer designed for the purpose. It is shown that only three operators are needed. One operator replaces successive pairs of data points by their sums and differences. The second operator performs a fixed permutation which is an ideal shuffle of the data. The third operator permits the multiplication of a selected subset of the data by a common complex multiplier.

If, as seems reasonable, the slowest operation is the complex multiplications required, then, for reasonably sized data sets—e.g. 512 complex numbers—parallelization by the method developed should allow an increase of speed over the serial use of the Fast Fourier Transform by about two orders of magnitude.

It is suggested that a machine to realize the speed improvement indicated is quite feasible.

The analysis is based on the use of the Kronecker product of matrices. It is suggested that this form is of general use in the development and classification of various modifications and extensions of the algorithm.

KEY WORDS AND PHRASES: transforms, Fourier transforms, Fourier series, parallel processors, parallel execution, simultaneous processors, convolution, power spectrum, Fast Fourier Transform

CR CATEGORIES: 5.19

1. *Introduction*

The Fast Fourier Transform and its variations have become the principal methods for the machine calculation of the Fourier series of time-sampled signals or other sequences of data. The basic algorithm has a long complicated history, as described by Cooley, Lewis and Welch [1], although general recognition of its computational power dates from the paper of Cooley and Tukey [2]. Elaborations led to applications to other purposes, by Stockham[3], Bingham, Godfrey, and Tukey [4], Gentleman and Sande [5], Singleton [6], Cooley, Lewis and Welch [7], and Helms [8].

The present work is concerned with the development of a variation of the algorithm that is better adapted to parallel processing in a special purpose machine. Our primary objective was to see what capabilities would be needed in such a machine.

This work may also be of value as an aid in understanding the algorithm, at least for those who are used to thinking in terms of matrix operators, and as a formalism within which other variations can be described and investigated.

For the purpose of parallel processing in a special purpose machine, we require that the operation be organized in a few levels, each of which involves a set of

* Computer Techniques Laboratory. This work was sponsored in part by the Office of Naval Research, Information Systems Branch, under Contract Nonr 4833(00).

elementary operations that can be done simultaneously. Preferably each level should involve only a single type of elementary process, so that the entire level can be initiated by a single command. Also there should be as few as possible distinct types of elementary operations, so that the parallel capability required shall be as simple as possible. For example, in the original algorithm each "pass" involves a different grouping of the data into pairs. We would like, if possible, to use only a single pattern of pairings which could then be "wired in."

On the other hand, in a special purpose machine—for example, one designed to be part of an autocorrelator, or a digital filtering system—the number of data points can be taken as fixed. We can then reasonably assume that this number is a power of 2, $N = 2^n$. This is fortunate since clearly we cannot expect to be able to use only a small number of types of operations in the general case, when we can only assume N to be a composite number. We therefore assume throughout that the number of data to be processed is $N = 2^n$.

2. *Finite Fourier Transform*

First, let us very briefly review what is meant by the Finite Fourier Transform. A more complete description can be found in the report of the G-AE Subcommittee on Measurement Concepts [9].

We consider a possibly complex-valued curve $f(t)$ over some interval of time that we can normalize to the unit interval $0 \leq t < 1$. We sample the curve at N points equally spaced along the interval:

$$t_0 = 0,$$
$$t_i = i/N, \qquad 0 \leq i < (N-1),$$
$$f_i = f(t_i).$$

The Fourier coefficients are obtained as

$$g_r = \frac{1}{N} \sum_{s=0}^{N-1} (\exp 2\pi jrs/N) f_s. \tag{1}$$

If the time scale is normalized so that $f(t)$ is periodic with period 1, then g_r is the complex amplitude of the component at frequency r of the usual Fourier series. If outside the given interval $f(t)$ vanishes, then g_r may be taken as a discrete approximation of the complex-valued Fourier integral. In either case frequencies higher than $r = (N-1)$ cannot be computed meaningfully with the sampling rate specified.

The sets (f_s) and (g_s) can now be written as column vectors[1]:

$$\mathbf{f} = \text{col } (f_0, f_1 \cdots f_{N-1}), \qquad \mathbf{g} = \text{col } (g_0, g_1 \cdots g_{N-1}).$$

We define the matrix $\mathbf{T}$, whose coefficients are $(\mathbf{T})_{rs} = \exp(2\pi jrs/N)$. Then eq. (1) can be written as

$$\mathbf{g} = \frac{1}{N} \mathbf{T}\mathbf{f}. \tag{2}$$

[1] Lower-case boldface letters indicate vectors, and capital boldface letters indicate matrices.

The matrix **T** is called the Finite Fourier Transform. Writing w as the principal Nth root of unity, the coefficients of **T** are given by

$$(\mathbf{T})_{rs} = w^{rs}. \tag{3}$$

To simplify notation, we preserve only the exponent of w. That is, we write k in place of w^k. Then **T** can be written as

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 2 & 3 & \cdots & N-1 \\ 0 & 2 & 4 & 6 & \cdots & 2(N-1) \\ 0 & 3 & 6 & 9 & \cdots & 3(N-1) \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & N-1 & 2(N-1) & 3(N-1) & \cdots & (N-1)^2 \end{bmatrix}. \tag{4}$$

In this notation, multiplication of entries becomes addition. The expression for **T** can be further reduced by noting that each term may be replaced by its principal value (between 0 and $N-1$), modulo N.

3. *The Fast Fourier Transform*

The transformation used by Cooley and Tukey accomplishes the same thing as **T**, except that the output is obtained in permuted order—specifically in digit-reversed order, where the rows are numbered from 0 to $(N-1)$, as expressed in the base of the prime factor being used (in the present case, in binary notation). The transformation to this form is given, in general, by $\mathbf{T}' = \mathbf{QT}$, where **Q** is the appropriate permutation matrix.

We can obtain $\mathbf{T}'$ directly from **T** by permuting the rows of **T** as specified. For example, for $N = 8$, after reducing all terms modulo 8, we find that **T** in reduced exponent notation is given by

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 2 & 4 & 6 & 0 & 2 & 4 & 6 \\ 0 & 3 & 6 & 1 & 4 & 7 & 2 & 5 \\ 0 & 4 & 0 & 4 & 0 & 4 & 0 & 4 \\ 0 & 5 & 2 & 7 & 4 & 1 & 6 & 3 \\ 0 & 6 & 4 & 2 & 0 & 6 & 4 & 2 \\ 0 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} \quad \begin{matrix} 0\,0\,0 \\ 0\,0\,1 \\ 0\,1\,0 \\ 0\,1\,1 \\ 1\,0\,0 \\ 1\,0\,1 \\ 1\,1\,0 \\ 1\,1\,1 \end{matrix}$$

where the binary row designation appears on the right. Then the modified transform is given by $\mathbf{T}'$ as follows:

$$\mathbf{T}' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 4 & 0 & 4 & 0 & 4 \\ 0 & 2 & 4 & 6 & 0 & 2 & 4 & 6 \\ 0 & 6 & 4 & 2 & 0 & 6 & 4 & 2 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 5 & 2 & 7 & 4 & 1 & 6 & 3 \\ 0 & 3 & 6 & 1 & 4 & 7 & 2 & 5 \\ 0 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} \quad \begin{matrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \tag{5}$$

The key to the Fast Fourier Transform algorithm lies in the fact that $\mathbf{T}'_N$ (when $N = 2^n$) can be first partitioned and then factored:

$$\begin{aligned} \mathbf{T}'_N &= \begin{bmatrix} \mathbf{T}'_{N/2} & \mathbf{T}'_{N/2} \\ (\mathbf{T}'_{N/2})\mathbf{K} & -(\mathbf{T}'_{N/2})\mathbf{K} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{T}'_{N/2} & \phi \\ \phi & \mathbf{T}'_{N/2} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{K} & -\mathbf{K} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{T}'_{N/2} & \phi \\ \phi & \mathbf{T}'_{N/2} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I} & \phi \\ \phi & \mathbf{K} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix}, \end{aligned} \tag{6}$$

where, in exponent notation, $\mathbf{K} = \text{diag}\ (0\,1\,2\,3\,\cdots)$ and ϕ indicates the null matrix of appropriate dimension.

The minus sign that occurs in eq. (6) applies to the *values* of the submatrices that follow it, and not to the exponent-coefficients that may actually be used. Since w is the Nth root of unity, $w^{N/2}$ is the square root of unity or the value (-1). Hence the minus sign can be translated to exponent notation by adding $N/2$ to the exponent-coefficients involved.

It should also be noted that $\mathbf{T}'_N$ is written in terms of w, the principle Nth root of unity. $\mathbf{T}'_{N/2}$ would normally be written in terms of the principle $(N/2)$-nd root, which is w^2. Hence, if eq. (6) is actually written out, the entries in $\mathbf{T}'_{N/2}$ will be twice those that would occur were it written in terms of the $(N/2)$-nd root.

The algorithm can now be iterated by applying the factorization of eq. (6) to each occurrence of $\mathbf{T}'_{N/2}$, and so on. If we carry the process to completion, we obtain the Fast Fourier Transform.

For the purposes of parallelization, the process given suffers from the defect that a coefficient in a given location is combined with coefficients in different loca-

tions each time the sum and difference are formed. For example, if we carry eq. (6) to the second stage of iteration, we get

$$\mathbf{T}'_N = \begin{bmatrix} \mathbf{T}'_{N/4} & \phi & \phi & \phi \\ \phi & \mathbf{T}'_{N/4} & \phi & \phi \\ \phi & \phi & \mathbf{T}'_{N/4} & \phi \\ \phi & \phi & \phi & \mathbf{T}'_{N/4} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{I} & \phi & \phi & \phi \\ \phi & \mathbf{K}' & \phi & \phi \\ \phi & \phi & \mathbf{I} & \phi \\ \phi & \phi & \phi & \mathbf{K}' \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{I} & \phi & \phi \\ \mathbf{I} & -\mathbf{I} & \phi & \phi \\ \phi & \phi & \mathbf{I} & \mathbf{I} \\ \phi & \phi & \mathbf{I} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \phi \\ \phi & \mathbf{K} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix}.$$

The last factor, which is the first operator applied to the data, forms the sums and differences of coefficients in the first half with corresponding terms in the second. The third factor forms the sums and differences of the coefficients in the first quarter with corresponding terms in the second, and of the coefficients in the third quarter with those in the fourth. This prevents use of "wired-in" data paths, or else it requires an excessive number of such paths.

We therefore ask if the algorithm can be modified so as to avoid this difficulty.

4. *Modified Fast Fourier Transform*

We observe that the recursion formula of eq. (6) can be written in the form

$$\mathbf{T}'_N = (\mathbf{T}'_{N/2} \times \mathbf{I}_2)\, \mathbf{D}_N (\mathbf{I}_{N/2} \times \mathbf{T}'_2), \tag{7}$$

where $\mathbf{D}_N$ is the $N \times N$ diagonal matrix, quasidiag $(\mathbf{I}\ \mathbf{K})$, and $\mathbf{T}'_2$ is given by

$$\mathbf{T}'_2 = \begin{pmatrix} 0 & 0 \\ 0 & \frac{1}{2}N \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{8}$$

where in the latter matrix the coefficients are the actual values, not exponents.

The multiplication sign in eq. (7) indicates the direct, or Kronecker, product [10] of the submatrices, indexed first on the indices of the first component and then on the indices of the second. For example, if $\mathbf{A} = (a_{ij})$, $\mathbf{B} = (b_{kh})$, with all indices being either 0 or 1, then

$$\mathbf{A} \times \mathbf{B} = \begin{pmatrix} a_{00}b_{00} & a_{01}b_{00} & a_{00}b_{01} & a_{01}b_{01} \\ a_{10}b_{00} & a_{11}b_{00} & a_{10}b_{01} & a_{11}b_{01} \\ a_{00}b_{10} & a_{01}b_{10} & a_{00}b_{11} & a_{01}b_{11} \\ a_{10}b_{10} & a_{11}b_{10} & a_{10}b_{11} & a_{11}b_{11} \end{pmatrix}.$$

The Kronecker product can be combined with matrix multiplication through the formula,

$$(\mathbf{A} \times \mathbf{B})(\mathbf{C} \times \mathbf{D}) = (\mathbf{AC}) \times (\mathbf{BD}),$$

provided that the dimensions of the matrices are compatible. In particular, we can write

$$(\mathbf{AB} \times \mathbf{I}) = (\mathbf{AB}) \times \mathbf{I}^2 = (\mathbf{A} \times \mathbf{I})(\mathbf{B} \times \mathbf{I})$$

or, more generally, if $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\cdots$ are all $k \times k$ matrices, then

$$(\mathbf{ABC} \cdots) \times \mathbf{I} = (\mathbf{A} \times \mathbf{I})(\mathbf{B} \times \mathbf{I})(\mathbf{C} \times \mathbf{I}) \cdots .$$

Using this formula, we can apply eq. (7) iteratively and obtain, when $N = 2^n$,

$$\begin{aligned} \mathbf{T}'_N = {} & (\mathbf{T}'_2 \times \mathbf{I}_2 \times \cdots \times \mathbf{I}_2)\cdot\mathbf{Q}_1 \\ & \cdot(\mathbf{I}_2 \times \mathbf{T}'_2 \times \mathbf{I}_2 \times \cdots \times \mathbf{I}_2)\cdot\mathbf{Q}_2 \\ & \cdot(\mathbf{I}_2 \times \mathbf{I}_2 \times \mathbf{T}'_2 \times \cdots \times \mathbf{I}_2)\cdot\mathbf{Q}_3 \\ & \vdots \\ & \cdot(\mathbf{I}_2 \times \mathbf{I}_2 \times \cdots \times \mathbf{T}'_2), \end{aligned} \tag{9}$$

where each of the cross-products includes n factors, of which $(n-1)$ are the 2×2 identity and one is $\mathbf{T}'_2$, and the $\mathbf{Q}_i$ are diagonal matrices. For example, for $N = 8$, $\mathbf{T}'_8 = (\mathbf{T}'_4 \times \mathbf{I}_2)\,\mathbf{D}_8(\mathbf{I}_4 \times \mathbf{T}'_2)$, and, since $\mathbf{I}_4 = \mathbf{I}_2 \times \mathbf{I}_2$ and

$$\mathbf{T}'_4 = (\mathbf{T}'_2 \times \mathbf{I}_2)\,\mathbf{D}_4(\mathbf{I}_2 \times \mathbf{T}'_2),$$

we find that

$$\begin{aligned} \mathbf{T}'_8 &= [\{(\mathbf{T}'_2 \times \mathbf{I}_2)\,\mathbf{D}_4(\mathbf{I}_2 \times \mathbf{T}'_2)\} \times \mathbf{I}_2]\,\mathbf{D}_8(\mathbf{I}_2 \times \mathbf{I}_2 \times \mathbf{T}'_2) \\ &= \{(\mathbf{T}'_2 \times \mathbf{I}_2 \times \mathbf{I}_2)(\mathbf{D}_4 \times \mathbf{I}_2)(\mathbf{I}_2 \times \mathbf{T}'_2 \times \mathbf{I}_2)\}\,\mathbf{D}_8(\mathbf{I}_2 \times \mathbf{I}_2 \times \mathbf{T}'_2) \\ &= (\mathbf{T}'_2 \times \mathbf{I}_2 \times \mathbf{I}_2)\,\mathbf{Q}_1(\mathbf{I}_2 \times \mathbf{T}'_2 \times \mathbf{I}_2)\,\mathbf{Q}_2(\mathbf{I}_2 \times \mathbf{I}_2 \times \mathbf{T}'_2), \end{aligned}$$

where $\mathbf{Q}_1 = \mathbf{D}_4 \times \mathbf{I}_2$, $\mathbf{Q}_2 = \mathbf{D}_8$.

For the purpose of parallelization the difficulty now with eq. (9) is that $\mathbf{T}'_2$ occurs in different locations in the various factors. We therefore ask if there is a permutation operator, $\mathbf{P}$, such that

$$\mathbf{P}(\mathbf{T}'_2 \times \mathbf{A}_{N/2})\mathbf{P}^{-1} = \mathbf{A}_{N/2} \times \mathbf{T}'_2. \tag{10}$$

If so, we can write each of the product terms in eq. (9) in terms of a single one of them, say the first:

$$\begin{aligned} \mathbf{C} &= \mathbf{T}'_2 \times \mathbf{I}_2 \times \mathbf{I}_2 \times \cdots \times \mathbf{I}_2 \\ &= \mathbf{T}'_2 \times \mathbf{I}_{N/2}\,. \end{aligned} \tag{11}$$

By considering the behavior of the product operators on an arbitrary vector, we can find that a possible $\mathbf{P}$ is the "ideal shuffle," in which

$$\begin{aligned} \mathbf{P}\cdot\mathrm{col}\,(x_0, x_1, \cdots, x_{\frac{1}{2}N-1}, x_{N/2} \cdots x_{N-1}) \\ = \mathrm{col}\,(x_0, x_{N/2}, x_1, x_{\frac{1}{2}N+1}, \cdots, x_{\frac{1}{2}N-1}, x_{N-1}) \end{aligned} \tag{12}$$

or, for $N = 8$, for example,

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{13}$$

where the entries, in this case, are the actual values, not exponents.

(We could also use the ideal shuffle which does not keep the first coefficient fixed. In special circumstances other permutations are also possible. Our concern here, however, is to find one such **P** operator, not to define the range of possible permutations.)

With this **P**,

$$\mathbf{I}_2 \times \mathbf{T}_2' \times \mathbf{I}_{N/4} = \mathbf{P}(\mathbf{T}_2' \times \mathbf{I}_{n/2})\,\mathbf{P}^{-1} = \mathbf{PCP}^{-1},$$

$$\mathbf{I}_4 \times \mathbf{T}_2' \times \mathbf{I}_{N/8} = \mathbf{P}^2\mathbf{CP}^{-2},$$

etc.,

where **C** is defined in eq. (11). Then eq. (9) becomes

$$\mathbf{T}_N' = \mathbf{CE}_1\mathbf{PCP}^{-1}\mathbf{E}_2\mathbf{P}^2\mathbf{CP}^{-2}\mathbf{E}_3 \cdots \mathbf{P}^{n-1}\mathbf{CP}^{-(n-1)}. \tag{14}$$

$\mathbf{E}_i'$ and $\mathbf{E}_i''$ can now be defined by

$$\mathbf{E}_i = \mathbf{P}^{(i-1)}\mathbf{E}_i'\mathbf{P}^{-(i-1)} = \mathbf{P}^i\mathbf{E}_i''\mathbf{P}^{-i},$$

which permit us to pass the diagonal operator through the permutations. The operators $\mathbf{E}_i'$ and $\mathbf{E}_i''$ will also be diagonal, but with the values on the diagonal permuted. Using these transformations and noting that, since $\mathbf{P}^n = \mathbf{I}$, $\mathbf{P}^{-(n-1)} = \mathbf{P}$, eq. (14) becomes either

$$\mathbf{T}_N' = \mathbf{CE}_1'\mathbf{PCE}_2'\mathbf{PCE}_3' \ldots \mathbf{PCP} \tag{15}$$

or

$$\mathbf{T}_N' = \mathbf{CPE}_1''\mathbf{CPE}_2''\mathbf{CP} \ldots \mathbf{CP}. \tag{16}$$

The $\mathbf{E}_i'$ and $\mathbf{E}_i''$ matrices are diagonal and have the effect of multiplying selected elements of the data set by various powers of w. We can, if we like, factor each of these matrices into diagonal operators, each of which multiplies a selected subset of the data set by a common multiplier which is a power of w. For example, for

$N = 16$, $\mathbf{E}_1'$ is given by

$$\mathbf{E}_1' = \text{diag } (0\,0\,0\,4\,0\,2\,0\,6\,0\,1\,0\,5\,0\,3\,0\,7) = \mathbf{F}_1' \mathbf{F}_2' \mathbf{F}_4',$$

where

$$\mathbf{F}_1' = \text{diag } (0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1),$$
$$\mathbf{F}_2' = \text{diag } (0\,0\,0\,0\,0\,2\,0\,2\,0\,0\,0\,0\,0\,2\,0\,2),$$
$$\mathbf{F}_4' = \text{diag } (0\,0\,0\,4\,0\,0\,0\,4\,0\,0\,0\,4\,0\,0\,0\,4),$$

in exponent notation. In particular, each $\mathbf{E}_i'$ or $\mathbf{E}_i''$ can be factored into the product of $\mathbf{F}_{(2^i)}'$ or $\mathbf{F}_{(2^i)}''$, where each $\mathbf{F}'$ or $\mathbf{F}''$ multiplies exactly $N/4$ of the data by w^{2^i}.

Whether or not this factorization of $\mathbf{E}_i'$ or $\mathbf{E}_i''$ is advantageous depends upon the particular capabilities available in the processor. If we can store $(n - 1)$ vectors internally and can multiply two vectors term by term in parallel, we can implement the $\mathbf{E}_i'$ or $\mathbf{E}_i''$ operations directly. If, on the other hand, internal storage is limited, so that the multiplying vector has to be loaded as used, it is probably better to use the factorization into the **F**-operators. Then we need only introduce the particular w^k and the enabling or disabling signals to designate the subset to be multiplied.

The alternative factorizations for $N = 16$ are given by

$$\mathbf{F}_{(4)}'' = \text{diag } (0\,0\,0\,0\,0\,0\,0\,0\,0\,4\,0\,4\,0\,4\,0\,4),$$
$$\mathbf{F}_{(2)}'' = \text{diag } (0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,2\,2\,0\,0\,2\,2),$$
$$\mathbf{F}_{(1)}'' = \text{diag } (0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1).$$

With these factorizations of the **E**-operators, eqs. (15) and (16) become

$$\mathbf{T}_{16}' = \mathbf{C}\mathbf{F}_{(4)}'\,\mathbf{F}_{(2)}'\,\mathbf{F}_{(1)}'\,\mathbf{P}\mathbf{C}\mathbf{F}_{(4)}'\,\mathbf{F}_{(2)}'\,\mathbf{P}\mathbf{C}\mathbf{F}_{(4)}'\,\mathbf{P}\mathbf{C}\mathbf{P}, \tag{17}$$

$$\mathbf{T}_{16}' = \mathbf{C}\mathbf{P}\mathbf{F}_{(4)}''\,\mathbf{F}_{(2)}''\,\mathbf{F}_{(1)}''\,\mathbf{C}\mathbf{P}\mathbf{F}_{(4)}''\,\mathbf{F}_{(2)}''\,\mathbf{C}\mathbf{P}\mathbf{F}_{(4)}''\,\mathbf{C}\mathbf{P}. \tag{18}$$

Since diagonal matrices commute, we can, of course, permute each subsequence of **F** operators as convenient.

Equation (18) was the first discovered. Equation (17) was found as a result of conversations with Jack Goldberg. It has the advantage, for some purposes, that adjacent data points are never involved in a given $\mathbf{F}_{(i)}'$.

The general rules for the **F**-matrices are as follows: We number the positions along the diagonal from 0 to $N - 1$, and express these numbers in binary form. Then, in $\mathbf{F}_{(2^m)}'$, the exponent-coefficients are 2^m in those positions in which the $(m + 1)$-st and the last digits are 1. All other exponent-coefficients are 0. For example, for $N = 16$, $\mathbf{F}_{(4)}'$ has the exponent-coefficient 4 in the 3rd, 7th, 11th, and 15th positions, for which the binary representations are 0 0 1 1, 0 1 1 1, 1 0 1 1, and 1 1 1 1, which are the numbers whose 3rd and last digits are 1. Similarly, in general, the $\mathbf{F}_{(2^m)}''$-matrix has the exponent-coefficient 2^m in the positions for which, in the binary representation, the first and $(m + 2)$-nd digits are 1.

The final results, given in eqs. (15) or (16), can be written as

$$\mathbf{T}_{N=2^n}' = \prod_{m=1}^{n} (\mathbf{C}\,\mathbf{E}_m'\,\mathbf{P}) \tag{19}$$

$$= \prod_{m=1}^{n} (\mathbf{C}\,\mathbf{P}\,\mathbf{E}_m''), \tag{20}$$

where $\mathbf{E}'_m$ and $\mathbf{E}''_m$ are the diagonal matrices defined before, and $\mathbf{E}'_n = \mathbf{E}''_n = \mathbf{I}$. In a parallel processor, each factor represents one "pass," a pass being defined as a single sequence of parallel operations involving, in eq. (19), first a permutation of the data set; second, the multiplication of selected elements of the data set by appropriate powers of w; and finally the replacement of the data set by the sums and differences of adjacent elements.

Equations (15) and (16), or (19) and (20), or, in the fully factored form, eqs. (17) and (18), and their generalization to arbitrary n, are the modified form of the Fast Fourier Transform that we seek.

5. *Generalizations and Variations*

There are some variations of the procedure that are worth noting.

(a) *Radix 4—Binary Reversal.* It is not necessary to factor $\mathbf{T}'$ all the way to factors involving $\mathbf{T}'_2$. It may, for example, be convenient to stop at $\mathbf{T}'_4$ since

$$\mathbf{T}'_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 8 \\ 0 & 4 & 8 & 12 \\ 0 & 12 & 8 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \\ 1 & -j & -1 & j \end{bmatrix},$$

the latter being in terms of actual values, not exponents. $\mathbf{T}'_4$ then involves quite simple linear combinations of quadruples of elements of the data set. If this can be conveniently implemented in a parallel processor, its use in place of the C-operator will halve the number of passes required. The diagonal operators must then be redetermined. We can find, for example, that

$$\mathbf{T}'_{16} = (\mathbf{T}'_4 \times \mathbf{I}_4)\,\mathbf{Q}'(\mathbf{I}_4 \times \mathbf{T}'_4),$$

where

$$\mathbf{Q}' = \text{diag}\,(0\,0\,0\,0\,0\,2\,4\,6\,0\,1\,2\,3\,0\,3\,6\,9).$$

We can then use a four-fold interleave permutation in place of $\mathbf{P}$ to permute $\mathbf{I}_4 \times \mathbf{T}'_4$ into $\mathbf{T}'_4 \times \mathbf{I}_4$, as before. Presumably, this process can be generalized for the case where N is any even power of 2, although we have not worked out the details. We would also expect to be able to perform a similar factorization in terms of $\mathbf{T}'_8$, $\mathbf{T}'_{16}$, etc., if this were needed.

(b) *Full Radix 4.* As an alternative to the procedure used in (a) we can label the rows of $\mathbf{T}$ to the base 4 instead of 2, reverse the digits as before, and compute a $\mathbf{T}''$ as the new transform operator. We then obtain, for example,

$$\mathbf{T}''_{16} = (\mathbf{T}''_4 \times \mathbf{I}_4)\,\mathbf{Q}''(\mathbf{I}_4 \times \mathbf{T}''_4),$$

where now

$$\mathbf{T}_4'' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 8 & 12 \\ 0 & 8 & 0 & 8 \\ 0 & 12 & 8 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}$$

and

$$\mathbf{Q}'' = \text{diag } (0\,0\,0\,0\,0\,1\,2\,3\,0\,2\,4\,6\,0\,3\,6\,9).$$

Again, we can use in place of $\mathbf{P}$ the four-fold interleaving operator to permute the factors in the Kronecker product.

(c) *Arbitrary Radix.* If $N = p^n$, where p is any prime number, a similar process can be used. Nevertheless, with the possible exception of $p = 3$ there appears to be little advantage for parallelization in using such a transform, since the basic operator $\mathbf{T}_p'$ or $\mathbf{T}_{p^k}'$ becomes quite complicated.

(d) *Composite N.* The process can be generalized for the case where N is any composite number. To illustrate by a simple example, consider $N = 6$. Then

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 2 & 4 & 0 & 2 & 4 \\ 0 & 3 & 0 & 3 & 0 & 3 \\ 0 & 4 & 2 & 0 & 4 & 2 \\ 0 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} \begin{matrix} 0\,0 & 0\,0 \\ 0\,1 & 0\,1 \\ 1\,0 & 0\,2 \\ 1\,1 & 1\,0 \\ 2\,0 & 1\,1 \\ 2\,1 & 1\,2 \end{matrix}.$$

Now we have two possible labelings of the rows. Either the pth row can be labeled by $p = 3i + j$, $i = 0, 1$; $j = 0, 1, 2$; or $p = 2i + j$, $i = 0, 1, 2$, $j = 0, 1$, as indicated alongside the matrix. Reversing the digits of the first designation, and making the pth row the qth, where $q = 2j + i$, we obtain

$$\mathbf{T}_6' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 4 & 0 & 2 & 4 \\ 0 & 4 & 2 & 0 & 4 & 2 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 3 & 0 & 3 & 0 & 3 \\ 0 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} = (\mathbf{T}_3' \times \mathbf{I}_2)\,\mathbf{Q}'(\mathbf{I}_3 \times \mathbf{T}_2'), \quad \mathbf{Q}' = \text{diag } (0\,0\,0\,0\,1\,2).$$

Using digit reversal on the second designation, we obtain

$$\mathbf{T}_6'' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 3 & 0 & 3 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 4 & 2 & 0 & 4 & 2 \\ 0 & 2 & 4 & 0 & 2 & 4 \\ 0 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} = (\mathbf{T}_2'' \times \mathbf{I}_3)\, \mathbf{Q}''\, (\mathbf{I}_2 \times \mathbf{T}_3''), \qquad \mathbf{Q}'' = (0\,0\,0\,1\,0\,2).$$

We can, in either case, again find a permutation operator that will permute the order of the factors in the Kronecker products. In this situation, however, it does not appear to be of advantage to do so, since we do not, by this means, obtain a repeated factor. For example, the shuffle operator $\mathbf{P}$ takes $(\mathbf{I}_3 \times \mathbf{T}_2')$ into $\mathbf{T}_2' \times \mathbf{I}_3$ in $\mathbf{T}_6'$, but this is not the same as the first factor, $(\mathbf{T}_3' \times \mathbf{I}_2)$. This variation therefore seems to be mainly of academic interest.

Where the factorization of N is into relatively prime factors, as in the example, the factorization of $\mathbf{T}_N'$ is the "prime factor algorithm," which is sometimes confused with the Fast Fourier Transform [9].

6. *Conclusions*

With eqs. (17) and (18), we have expressed the transform $\mathbf{T}_N'$ $(N = 2^n)$ using only the operators $\mathbf{C}$, $\mathbf{P}$, and a set of $(n - 1)$ operators $\mathbf{E}_{(i)}'$ or $\mathbf{E}_{(1)}''$, or $\mathbf{F}_{(2^i)}'$ or $\mathbf{F}_{(2^i)}''$. The operator $\mathbf{C}$ forms the sum and difference of pairs of successive components of a vector, and puts the computed values in place of the coefficients involved. The operator $\mathbf{P}$ permutes the components of the vector by a fixed permutation. The various $\mathbf{E}'$ or $\mathbf{E}''$ multiply each element of the data set by an appropriate power of w. The $\mathbf{F}_{(2^i)}'$ and $\mathbf{F}_{(2^i)}''$ multiply fixed subsets of the data set by the common multiplier w^{2^i}.

The transform, in the forms given in eqs. (17) or (18), could be implemented in a special purpose computer having the following capabilities:

(1) For i even:

$$a_i \leftarrow a_i + a_{i+1}, \qquad a_{i+1} \leftarrow a_i - a_{i+1},$$

the computations to be done in parallel for successive pairs of data and in parallel for the real and imaginary parts.

(2) The parallel shift of data in accordance with $\mathbf{P}$—i.e., an ideal shuffle. This is to be done separately but simultaneously for the real and imaginary parts, and can be conveniently done by a "wired-in" shift network involving bit-serial, word-parallel transfers.

(3) The ability to multiply in parallel a subset of the data set by a common multiplier. The subset can be selected by a stored "masking vector" as part of the control system.

Alternatively, if we use eqs. (15), (16), (19), or (20), in place of (3) we require

(3′) The ability to multiply in parallel each element of the data-set by a stored multiplier.

These capabilities should be relatively easy to implement in a special purpose machine. Hence it should be quite possible to build a special purpose computer using any of the given factorizations of the transform.

It is reasonable to suppose that the slowest operations of those required will be the complex multiplications required by the $\mathbf{E}'_{(i)}$ or $\mathbf{E}''_{(i)}$ operators. There are $\frac{1}{2}n(n-1)$ of these in the factorization of eqs. (17) or (18). If these can all be done in parallel, the resultant speed should be very high.

For example, consider a machine to form the Fourier transform of 512 complex data points. The direct computation of the Fourier transform requires $N^2 = 262{,}144$ complex multiplications. The serial implementation of the Fast Fourier Transform requires $N \log_2 N = nN = 4608$ complex multiplications. The machine suggested here requires only 36 sets of parallel complex multiplications with a common multiplier.

For data sets of the order indicated, the serial use of the Fast Fourier Transform is about two orders of magnitude faster than the direct transform. We would expect a special purpose machine implementing the factorization given here to be about another two orders of magnitude faster than the serial use of the Fast Fourier Transform.

We have also noted the possibility of other variations. For parallelization the most interesting of these is the possibility of using a factorization into $\mathbf{T}'_4$ or $\mathbf{T}''_4$, in the symbology of Section 5, i.e., a 4-step process. This cuts in half the number of passes required, but at some expense in terms of the operations needed per pass. Whether or not this would lead to a net gain in speed and, if so, at what cost in additional complexity in a special purpose machine, could be determined only as a result of detailed design work. If such a machine were built, this is a question that should be considered.

We conclude that if sufficient need exists it would be possible to build a special purpose parallel computer to calculate the Fourier transforms of large sets of data at extremely high speed.

ACKNOWLEDGMENT. The author is indebted to R. C. Singleton and W. H. Kautz of Stanford Research Institute for helpful discussions on this problem.

REFERENCES

1. COOLEY, J. W., LEWIS, P. A. W., AND WELCH, P. D. Historical notes on the Fast Fourier Transform. *IEEE Trans. AU-15* (June 1967), 76–79.
2. —— AND TUKEY, J. W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput. 19*, 90 (April 1965), 297–301.
3. STOCKHAM, T. G., JR. High-speed convolution and correlation. Proc. AFIPS 1966 Spring Joint Comput. Conf., Vol. 28, pp. 229–233.
4. BINGHAM, C., GODFREY, M. D., AND TUKEY, J. W. Modern techniques of power spectrum estimation. (Unpublished.)
5. GENTLEMAN, W. M., AND SANDE, G. Fast Fourier Transforms—for fun and profit. Proc. AFIPS 1966 Fall Joint Comput. Conf., Vol. 29, pp. 563–578.
6. SINGLETON, R. C. A method for computing the Fast Fourier Transform with auxiliary memory and limited high-speed storage. *IEEE Trans. AU-15* (June 1967), 91–97.
7. COOLEY, J. W., LEWIS, P. A. W., AND WELCH, P. D. Application of the Fast Fourier Trans-

form to computation of Fourier integrals, Fourier series, and convolution integrals. *IEEE Trans. AU-15* (June 1967), 79–84.
8. HELMS, H. D. Fast Fourier Transform method of computing difference equations and simulating filters. *IEEE Trans. AU-15* (June 1967), 85–90.
9. G-AE SUBCOMMITTEE ON MEASUREMENT CONCEPTS. What is the Fast Fourier Transform? *IEEE Trans. AU-15* (June 1967), 45–55.
10. PEASE, M. C. *Methods of Matrix Algebra*. Academic Press, New York, 1965, Ch. XIV.

RECEIVED JULY, 1967; REVISED SEPTEMBER, 1967