JOURNAL OF COMPUTATIONAL PHYSICS 58, 283–299 (1985)

Implementation of a Self-Sorting In-Place Prime Factor FFT Algorithm

CLIVE TEMPERTON

Division de recherche en prévision numérique,
Service de l'Environnement atmosphérique, Dorval, Québec H9P 1J3, Canada

Received February 21, 1984

A "prime factor" Fast Fourier Transform algorithm is described which is self-sorting and computes the transform in place. With a view to implementation on a Cray-1 or Cyber 205, the form of the algorithm is chosen to minimize the number of additions. With an appropriate choice of index mapping in the derivation, we obtain the unexpected result that the required indexing is actually simpler than that for a conventional FFT. The construction of the necessary "rotated" DFT modules is described, and comparisons are presented between the new algorithm and the conventional FFT in terms of operation counts and timings on an IBM 3081; on this machine, the new transform algorithm takes about 60% of the time for the conventional FFT. A Fortran routine for the new algorithm is outlined.

1. INTRODUCTION

In a recent review paper [16], the author developed a unified derivation of the numerous forms of the "conventional" Fast Fourier Transform (FFT) algorithm introduced by Cooley and Tukey [3]. Special emphasis was placed on self-sorting variants for which both the input and the output are naturally ordered, the only penalty being the need for a work array of the same size as the data array. Derivations were also given of four forms of the algorithm which require a permutation of the data either before or after the transform: the variants of Cooley and Tukey [3] and Gentleman and Sande [4] which compute the transform in place, and the Pease [13] and "transposed Pease" forms which require a work array but have a simpler structure. The implementation of the conventional FFT algorithm on vector computers such as the Cray-1 and the Cyber 205 was also discussed in [16].

In a second paper [17], consideration was given to the implementation on these machines of the so-called "prime factor" algorithm (PFA) first introduced by Good [5], and subsequently developed by Kolba and Parks [10] and especially by Winograd [19]; further discussion of these algorithms can be found in the recent books of McClellan and Rader [11] and Nussbaumer [12]. The important characteristic of the Cray-1 and the Cyber 205 in this context is not that they are vector

0021-9991/85 $3.00

machines, but that additions and multiplications can be performed simultaneously. It was shown in [17] that the best strategy would be to minimize the number of additions, and that this would be achieved by using Good's algorithm with the "small-n" transforms computed in a rather conventional manner rather than by the small-n transforms of Winograd [19] which minimize the number of multiplications. As the potential gain seemed to be very modest, and since at first sight the prime factor algorithms appear to require very complicated indexing, the idea of implementing these algorithms on the Cray-1 and Cyber 205 was not pursued further.

Johnson and Burrus [8] have recently taken the development of the prime factor algorithms further by constructing combinations of Good's and Winograd's algorithms which minimize a cost function which the user can specify, but their algorithms still require more additions than the basic prime factor algorithm advocated in [17].

Shortly after submitting [17] for publication, the author came across a paper by Burrus and Eschenbacher [2] in which it was shown that the prime factor FFT algorithm can be implemented in such a way that it is both self-sorting and in-place. In view of this very useful property, the use of the prime factor algorithm on vector computers deserves reconsideration, since it offers the prospect of economies in both time and storage.

In fact, the versions of the prime factor algorithm given most attention in [2] were in-place but not self-sorting. The authors also showed how to construct a variant which would be self-sorting as well as in-place, but suggested that an efficient implementation might require special coding for each value of N, the length of the transform. Rothweiler [14] presented an indexing scheme for the algorithm which permitted its use for general N (the only restriction being, as in all prime factor FFT algorithms, that N can be decomposed into mutually prime factors N_i such that an explicit algorithm is available for a DFT (discrete Fourier transform) of length N_i).

In this paper we develop a version of the algorithm similar in spirit to that described by Rothweiler [14], but with three important differences. First, a different index mapping is used to convert a one-dimensional transform of length N into a k-dimensional transform of size $N_1 \times N_2 \times \cdots \times N_k$, where $N = N_1 N_2 \ldots N_k$. Second, the short transforms of length N_i are carried out using algorithms which minimize the number of *additions*, as recommended in [17]. Third, the internal indexing permutations are performed quite differently. The resulting algorithm has a simpler indexing structure than those presented in [2] and [14]; rather remarkably, it is actually simpler than that required for the conventional FFT algorithms in [16].

The rest of this paper is organized as follows. In Section 2 we show how the prime factor FFT algorithm is derived. Section 3 discusses the choice of index mappings. Section 4 is concerned with the construction of the "rotated DFT modules" which form an essential part of the algorithm described here. In Section 5 we present operation counts and timing comparisons between the conventional and

prime factor algorithms implemented on a scalar computer (IBM 3081). Finally, Section 6 contains a summary and suggestions for further work. A Fortran subroutine which implements the proposed algorithm is outlined in the Appendix.

2. Derivation of the PFA

Burrus [1, 2] has presented a thorough derivation of the prime factor FFT algorithm. For the sake of completeness, we repeat the essentials here.

The discrete Fourier transform (DFT) is defined by

$$x(n) = \sum_{k=0}^{N-1} c(k)\, \omega_N^{kn}, \qquad 0 \leqslant n \leqslant N-1 \tag{1}$$

where we use the notation

$$\omega_N = \exp(2i\pi/N). \tag{2}$$

Since $\omega_N^N = 1$, the indices n, k, in Eq. (1) may be interpreted modulo N (i.e., x and c may both be regarded as periodic with period N). For compactness we will use the notation

$$\langle x \rangle_N \equiv x \text{ modulo } N. \tag{3}$$

In [2] and [14], ω_N is defined to be $\exp(-2i\pi/N)$, but this requires only minor changes in the derivation; we use Eq. (2) here to be consistent with the notation in [16].

The first idea used in deriving the PFA (which can also be used to derive the conventional FFT) is that of mapping the one-dimensional arrays x, c into multi-dimensional arrays. For simplicity we consider the case $N = N_1 N_2$. Two possible mappings are

$$n = N_2 n_1 + n_2 \tag{4}$$

and

$$n = n_1 + N_1 n_2 \tag{5}$$

where $0 \leqslant n_1 \leqslant N_1 - 1$, $0 \leqslant n_2 \leqslant N_2 - 1$. These mappings are illustrated in Table I for $N_1 = 3$, $N_2 = 5$. Using either of these maps, we can identify an element $x(n)$ of the original array with an element $\hat{x}(n_1, n_2)$ of a corresponding two-dimensional array.

In deriving the PFA we use the more general index maps:

$$k = \langle K_1 k_1 + K_2 k_2 \rangle_N \tag{6}$$

$$n = \langle K_3 n_1 + K_4 n_2 \rangle_N \tag{7}$$

TABLE I

Index Maps for $N_1 = 3$, $N_2 = 5$

	$n = N_2 n_1 + n_2$			
		n_1		
		0	1	2
n_2	0	0	5	10
	1	1	6	11
	2	2	7	12
	3	3	8	13
	4	4	9	14

	$n = n_1 + N_1 n_2$			
		n_1		
		0	1	2
n_2	0	0	1	2
	1	3	4	5
	2	6	7	8
	3	9	10	11
	4	12	13	14

where k_1 and n_1 run from 0 to $N_1 - 1$, and k_2 and n_2 run from 0 to $N_2 - 1$. We can then map the one-dimensional arrays $c(k)$, $x(n)$ into the corresponding two-dimensional arrays $\hat{c}(k_1, k_2)$, $\hat{x}(n_1, n_2)$. Burrus [1] gives the conditions under which the mappings defined by Eqs. (6) and (7) are unique.

The second idea used in the PFA comes from an application of the Chinese Remainder Theorem [11]. If N_1, N_2 are mutually prime (i.e., have no common factors) then we can find integers p, q, r, s $(0 < p, s < N_1, 0 < q, r < N_2)$ such that

$$pN_2 = rN_1 + 1 \tag{8}$$

$$qN_1 = sN_2 + 1. \tag{9}$$

If we use $K_3 = pN_2$, $K_4 = qN_1$ in Eq. (7), then it is easily verified that

$$n_1 = \langle n \rangle_{N_1}, \qquad n_2 = \langle n \rangle_{N_2}. \tag{10}$$

This is referred to as the Chinese Remainder Theorem (CRT) mapping.

Another possibility, referred to as the "Ruritanian" mapping [6], is simply to set $K_1 = N_2$, $K_2 = N_1$. The expression for k in Eq. (6) then becomes

$$k = \langle N_2 k_1 + N_1 k_2 \rangle_N. \tag{11}$$

It can be verified that the implied values of k_1, k_2 are

$$k_1 = \langle pk \rangle_{N_1}, \qquad k_2 = \langle qk \rangle_{N_2} \tag{12}$$

where p, q are defined by Eqs. (8) and (9).

A third idea which is useful in the PFA is that of a *rotated* transform. Using the notation of [16], let W_N be the DFT matrix of order N; element (j, k) of W_N is ω_N^{jk}, where the rows and columns of W_N are indexed from 0 to $N-1$. Equation (1) can then be written as

$$\mathbf{x} = W_N \mathbf{c}. \tag{13}$$

Now define $W_N^{[r]}$ to be the matrix with element (j, k) given by ω_N^{jkr}, i.e., each element of W_N is raised to the power r. It can be shown [11] that if r is mutually prime to N and we compute

$$\mathbf{x}' = W_N^{[r]}\mathbf{c} \tag{14}$$

then $\mathbf{x}'$ can be obtained simply by permuting the elements of $\mathbf{x}$. If

$$\mathbf{x} = (x_0, x_1, x_2, ..., x_{N-1})^{\mathrm{T}} \tag{15}$$

then

$$\mathbf{x}' = (x_0, x_r, x_{2r}, ...)^{\mathrm{T}} \tag{16}$$

where the subscripts in Eq. (16) are interpreted modulo N. Equation (14) is called a *rotated* DFT. For example, if $N = 5$ and

$$\mathbf{x} = W_5\mathbf{c} = (x_0, x_1, x_2, x_3, x_4)^{\mathrm{T}} \tag{17}$$

then

$$\mathbf{x}' = W_5^{[2]}\mathbf{c} = (x_0, x_2, x_4, x_1, x_3)^{\mathrm{T}} \tag{18}$$

as can easily be verified by comparing the explicit forms of W_5 and $W_5^{[2]}$.

The following relationship will be useful later:

$$W_N^{[N-1]} = W_N^* \tag{19}$$

where the asterisk denotes the complex conjugate.

Burrus and Eschenbacher [2] use the Ruritanian map for k and the CRT map for n; thus $K_1 = N_2$, $K_2 = N_1$, $K_3 = pN_2$, $K_4 = qN_1$ where p, q are defined by Eqs. (8) and (9). It can easily be shown that

$$\langle K_1K_3\rangle_N = N_2, \qquad \langle K_2K_4\rangle_N = N_1 \tag{20}$$

and it is obvious (since $N_1N_2 = N$) that

$$\langle K_1K_4\rangle_N = \langle K_2K_3\rangle_N = 0. \tag{21}$$

Substituting the maps of Eqs. (6) and (7) into Eq. (1), we obtain

$$\hat{x}(n_1, n_2) = \sum_{k_2=0}^{N_2-1}\sum_{k_1=0}^{N_1-1} \hat{c}(k_1, k_2)\,\omega_N^{(K_1k_1 + K_2k_2)(K_3n_1 + K_4n_2)}. \tag{22}$$

Using the results of Eqs. (20) and (21), Eq. (22) reduces to

$$\hat{x}(n_1, n_2) = \sum_{k_2=0}^{N_2-1}\sum_{k_1=0}^{N_1-1} \hat{c}(k_1, k_2)\,\omega_N^{N_2k_1n_1}\,\omega_N^{N_1k_2n_2} \tag{23}$$

and since $\omega_N^{N_2} = \omega_{N_1}$, $\omega_N^{N_1} = \omega_{N_2}$, this is in the form of a simple $N_1 \times N_2$ two-dimensional DFT:

$$\hat{x}(n_1, n_2) = \sum_{k_2=0}^{N_2-1} \left[\sum_{k_1=0}^{N_1-1} \hat{c}(k_1, k_2)\, \omega_{N_1}^{k_1 n_1} \right] \omega_{N_2}^{k_2 n_2}. \tag{24}$$

The important difference between Eq. (24) and the conventional FFT is that there are no phase ("twiddle") factors to be applied between the N_2 transforms of length N_1 and the subsequent N_1 transforms of length N_2.

If the two-dimensional transform, Eq. (24), is performed in place, then the input $\hat{c}(k_1, k_2)$ is replaced by the result $\hat{x}(k_1, k_2)$, i.e., the *input* index map $k \to (k_1, k_2)$ given by Eq. (12) remains intact. Since the derivation of the transform requires a different index map for the output $n \to (n_1, n_2)$ as given by Eq. (10), a reordering or unscrambling step is necessary to produce $\hat{x}(n_1, n_2)$, just as in the case of the in-place conventional FFT of Gentleman and Sande [4]. Alternatively the input data can be scrambled, as in the algorithm of Cooley and Tukey [3], and the transform can be performed in place using the output index map throughout.

To obtain an in-place algorithm which is also self-sorting (i.e., requires no reordering of the data before or after the transform), a variation of the PFA was suggested by Burrus and Eschenbacher [2] and implemented by Rothweiler [14]. Suppose we use the Ruritanian map of Eq. (11) for both k and n, i.e., we set

$$k = \langle N_2 k_1 + N_1 k_2 \rangle_N \tag{25}$$

$$n = \langle N_2 n_1 + N_1 n_2 \rangle_N. \tag{26}$$

Substituting (25) and (26) in Eq. (1), we obtain

$$\begin{aligned} \hat{x}(n_1, n_2) &= \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} \hat{c}(k_1, k_2)\, \omega_N^{(N_2 k_1 + N_1 k_2)(N_2 n_1 + N_1 n_2)} \\ &= \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} \hat{c}(k_1, k_2)\, \omega_N^{N_2 k_1 N_2 n_1}\, \omega_N^{N_1 k_2 N_1 n_2} \\ &= \sum_{k_2=0}^{N_2-1} \left[\sum_{k_1=0}^{N_1-1} \hat{c}(k_1, k_2)\, \omega_{N_1}^{N_2 k_1 n_1} \right] \omega_{N_2}^{N_1 k_2 n_2}. \end{aligned} \tag{27}$$

This is in the same form as Eq. (24) except that ω_{N_1}, ω_{N_2} are replaced by $\omega_{N_1}^{N_2}$, $\omega_{N_2}^{N_1}$. Since N_1, N_2 are mutually prime, this simply implies a permutation of the results of each of the transforms of length N_1 or N_2; we have rotated the DFT's as in Eq. (14).

For reasons to be explained in the next section, we choose here instead to use the CRT map for both input and output, i.e.,

$$k = \langle p N_2 k_1 + q N_1 k_2 \rangle_N \tag{28}$$

$$n = \langle p N_2 n_1 + q N_1 n_2 \rangle_N \tag{29}$$

where p, q are defined by Eqs. (8) and (9). Substituting (28) and (29) into Eq. (1), we obtain

$$\hat{x}(n_1, n_2) = \sum_{k_2=0}^{N_2-1} \left[\sum_{k_1=0}^{N_1-1} \hat{c}(k_1, k_2)\, \omega_{N_1}^{p^2 N_2 k_1 n_1} \right] \omega_{N_2}^{q^2 N_1 k_2 n_2}. \tag{30}$$

Since the exponents in Eq. (30) can be regarded modulo N_1 and N_2, respectively, they can be simplified. Using Eq. (8) we have

$$\langle p^2 N_2 \rangle_{N_1} = \langle p \cdot pN_2 \rangle_{N_1} = \langle p(rN_1 + 1) \rangle_{N_1} = p$$

and similarly from Eq. (9) we have $\langle q^2 N_1 \rangle_{N_2} = q$. Equation (30) thus becomes

$$\hat{x}(n_1, n_2) = \sum_{k_2=0}^{N_2-1} \left[\sum_{k_1=0}^{N_1-1} \hat{c}(k_1, k_2)\, \omega_{N_1}^{pk_1 n_1} \right] \omega_{N_2}^{qk_2 n_2}. \tag{31}$$

Equation (31) is similar to Eq. (27), but implies a different permutation of the results of the transforms of length N_1 and N_2. Equations (8) and (9) guarantee that p is mutually prime to N_1 and q is mutually prime to N_2.

3. Mapping and Indexing

We now demonstrate the advantage of using the CRT map instead of the Ruritanian map for the in-place self-sorting PFA. Consider the case $N_1 = 3$, $N_2 = 5$. The solution of Eqs. (8) and (9) is $p = 2$, $q = 2$, $r = 3$, $s = 1$. The Ruritanian map is given by

$$n = \langle N_2 n_1 + N_1 n_2 \rangle_N \tag{32}$$

where $n_1 = \langle pn \rangle_{N_1} = \langle 2n \rangle_3$, $n_2 = \langle qn \rangle_{N_2} = \langle 2n \rangle_5$. The CRT map is given by

$$n = \langle pN_2 n_1 + qN_1 n_2 \rangle_N$$

where $n_1 = \langle n \rangle_{N_1} = \langle n \rangle_3$, $n_2 = \langle n \rangle_{N_2} = \langle n \rangle_5$. The two maps are illustrated in Table II.

In the first pass of the two-dimensional transform given by Eq. (27) or Eq. (31), we need to perform $N_2 = 5$ (rotated) DFT's of length $N_1 = 3$. Following a similar principle to that used to index the conventional FFT in [16], we set up addresses *IA*, *IB*, *IC* outside the loop and then perform a computation of the form:

DO 10 $L = 1, N_2$

$$\begin{bmatrix} A(IA) \\ A(IB) \\ A(IC) \end{bmatrix} = W_3^{[r]} * \begin{bmatrix} C(IA) \\ C(IB) \\ C(IC) \end{bmatrix}$$

C Now update *IA*, *IB*, *IC*

10 CONTINUE

TABLE II

Index Maps for $N_1 = 3$, $N_2 = 5$

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

where W_3 is the DFT matrix of order 3, and the output vector can overwrite the input vector for in-place computation.

Referring to the Ruritanian map in Table II, we see that the initial values of IA, IB, IC are 0, 5, 10, and the code for updating these addresses is of the form

$$\begin{aligned} IA &= IA + 3 \\ IB &= \langle IB + 3 \rangle_N \\ IC &= \langle IC + 3 \rangle_N \end{aligned} \qquad \text{or} \qquad \begin{aligned} IA &= IA + 3 \\ IB &= \langle IA + 5 \rangle_N \\ IC &= \langle IB + 5 \rangle_N . \end{aligned}$$

Either form requires IB and IC to be evaluated modulo N, or the insertion of equivalent IF statements as in the codes given by Burrus and Eschenbacher [2] and Rothweiler [14]. From the expression for the Ruritanian map in Eq. (32), it is clear that the general case is obtained by replacing 3 by N_1 and 5 by N_2 in the code above (generalized to N_1 base addresses).

If on the other hand we use the CRT map in Table II, the initial values of IA, IB, IC are 0, 10, 5; updating the values for each subsequent transform can be achieved simply by the code

$$\begin{aligned} IX &= IC + 1 \\ IC &= IB + 1 \\ IB &= IA + 1 \\ IA &= IX \end{aligned}$$

requiring no IF statements or address computations modulo N. This code makes use of the fact, evident from the CRT map in Table II, that if (n_1, n_2) corresponds to n then $(\langle n_1 + 1 \rangle_{N_1}, \langle n_2 + 1 \rangle_{N_2})$ corresponds to $n + 1$. This is obvious from Eq. (10), and is equivalent to the observation that in the CRT map of Table II, the sequence 0, 1, 2,..., $N - 1$ appears along a continuous diagonal if the table is regarded as periodic in both dimensions.

In Rothweiler's implementation of the PFA [14], the permutation within each

transform of length N_i is handled by defining a second set of addresses for the output, e.g., in the case $N_i = 3$ the code is of the form

DO 10 $L = 1, N_2$

$$\begin{bmatrix} A(JA) \\ A(JB) \\ A(JC) \end{bmatrix} = W_3 * \begin{bmatrix} C(IA) \\ C(IB) \\ C(IC) \end{bmatrix}$$

C update IA, IB, IC

C update JA, JB, JC

10 CONTINUE

The output vector can still overwrite the input vector for in-place computation. There are several ways of handling the addressing for JA, JB, JC, but none of them is particularly elegant or efficient.

The approach taken in this paper is to modify the transform algorithms themselves so that they perform the required rotated transforms; the addresses IA, IB, IC in the above example then serve to index both the input vector and the output vector. The algorithmic form of the rotated transforms will be discussed in the next section.

Since Eq. (31) can equally well be written as

$$\hat{x}(n_1, n_2) = \sum_{k_1=0}^{N_1-1} \left[\sum_{k_2=0}^{N_2-1} \hat{c}(k_1, k_2)\, \omega_{N_2}^{qk_2n_2} \right] \omega_{N_1}^{pk_1n_1}, \tag{33}$$

it makes no difference to the indexing if we perform the N_1 transforms of length N_2 before the N_2 transforms of length N_1, rather than vice versa. It follows by induction that in the general case the indexing during the transforms of length N_i depends only on N/N_i and not on the order in which the factors N_i are used. This is a further simplification over the case of the conventional FFT [16] in which a nested loop structure is required to handle the dependence of the indexing on the order in which the factors are used.

4. Rotated DFT Modules

In this section we show how to construct rotated DFT modules by changing the multiplier constants in the original modules. This idea is not new; it is briefly implied in Winograd's paper [19], and discussed in some detail by Johnson and Burrus [9]. However, in both cases the application is to DFT modules of the form

$$W_{N_i} = AMB \tag{34}$$

where A and B are (generally rectangular) "incidence" matrices with entries $+1$, 0,

-1, and all the multiplications are contained in the diagonal matrix M. This form is chosen to minimize the number of *multiplications*. Here, following the recommendations in [17], we are using DFT modules which minimize the number of *additions*, and which cannot (except in the simplest cases) be written in the form of Eq. (34). Nevertheless, the same idea can be applied: the corresponding rotated DFT modules can be constructed simply by changing the values of certain constants which appear in the algorithm.

We will consider each case in turn. Given the algorithm in [16] to compute $\mathbf{x} = W_{N_i}\mathbf{z}$, we show how to generalize the algorithm to compute the rotated transform $\mathbf{x}' = W_{N_i}^{[r]}\mathbf{z}$ for any r mutually prime to N_i.

(a) $N_i = 2$: Since r must be non-zero and mutually prime to N_i, the only possibility is the usual case $W_2^{[1]}$; no other rotation is possible.

(b) $N_i = 3$: For the case $r = 1$, the algorithm as given in [16] is:

$$t_1 = z_1 + z_2; \qquad t_2 = z_0 - \tfrac{1}{2}t_1; \qquad t_3 = \sin 60 * (z_1 - z_2);$$
$$x_0 = z_0 + t_1; \qquad x_1 = t_2 + it_3; \qquad x_2 = t_2 - it_3.$$

The case $r = 2$ is equivalent to exchanging the output locations of x_1 and x_2. From the above expression of the algorithm, this is equivalent to changing the sign of t_3, and can be achieved simply by replacing the multiplier constant sin 60 by $-\sin 60$.

(c) $N_i = 5$: The algorithm can be written

$$t_1 = z_1 + z_4; \quad t_2 = z_2 + z_3; \quad t_3 = z_1 - z_4; \quad t_4 = z_2 - z_3; \quad t_5 = t_1 + t_2;$$
$$t_6 = c_1 * (t_1 - t_2); \quad t_7 = z_0 - \tfrac{1}{4}t_5; \qquad t_8 = t_7 + t_6; \qquad t_9 = t_7 - t_6;$$
$$t_{10} = c_2 * t_3 + c_3 * t_4; \quad t_{11} = c_3 * t_3 - c_2 * t_4;$$
$$x_0 = z_0 + t_5; \quad x_1 = t_8 + it_{10}; \quad x_2 = t_9 + it_{11}; \quad x_3 = t_9 - it_{11}; \quad x_4 = t_8 - it_{10}$$

where, in the standard case $r = 1$, we have from [16]:

$$c_1 = \sqrt{5}/4; \qquad c_2 = \sin 72; \qquad c_3 = \sin 36.$$

Possible rotations are given by $r = 2, 3, 4$. The easiest modification is for $r = 4$; the original results emerge in the order 0, 4, 3, 2, 1, and we need to exchange x_1 with x_4 and x_2 with x_3. This can be done by changing the signs of t_{10} and t_{11}, which is achieved by changing c_2 to $-c_2$ and c_3 to $-c_3$.

In the case $r = 2$, the original results emerge in the order 0, 2, 4, 1, 3. This can be achieved by exchanging t_8 and t_9, replacing t_{10} by t_{11} and t_{11} by $-t_{10}$. In turn, this can be done by changing the sign of c_1, replacing c_2 by c_3 and c_3 by $-c_2$. Similarly in the case $r = 3$, the original results emerge in the order 0, 3, 1, 4, 2: we exchange t_8 and t_9, replace t_{10} by $-t_{11}$ and t_{11} by t_{10}, by changing the sign of c_1, replacing c_2 by $-c_3$ and c_3 by c_2.

To summarize, the algorithm above computes $\mathbf{x} = W_5^{[r]}\mathbf{z}$ for $r = 1, 2, 3, 4$ by specifying the following values for the multiplier constants:

$$\begin{aligned}
r=1:\ c_1 &= \sqrt{5}/4; \quad & c_2 &= \sin 72; \quad & c_3 &= \sin 36;\\
r=2:\ c_1 &= -\sqrt{5}/4; \quad & c_2 &= \sin 36; \quad & c_3 &= -\sin 72;\\
r=3:\ c_1 &= -\sqrt{5}/4; \quad & c_2 &= -\sin 36; \quad & c_3 &= \sin 72;\\
r=4:\ c_1 &= \sqrt{5}/4; \quad & c_2 &= -\sin 72; \quad & c_3 &= -\sin 36.
\end{aligned}$$

(d) $N_i = 7$ (and higher prime values of N_i): In the normal unrotated case, it was recommended in [17] that the algorithm given by Singleton [15] be used to minimize the number of additions. The complex transform of odd order N_i may be written

$$\left.\begin{aligned} x_0 &= x_0^+ \\ x_j &= x_j^+ + ix_j^- \\ x_{N_i-j} &= x_j^+ - ix_j^- \end{aligned}\right\} \quad 1 \leqslant j \leqslant (N_i-1)/2 \tag{35}$$

where

$$x_j^+ = z_0 + \sum_{k=1}^{(N_i-1)/2} (z_k + z_{N_i-k}) \cos(2jk\pi/N_i) \tag{36}$$

and

$$x_j^- = \sum_{k=1}^{(N_i-1)/2} (z_k - z_{N_i-k}) \sin(2jk\pi/N_i). \tag{37}$$

The summations in Eqs. (36) and (37) are computed explicitly, and the whole algorithm requires $(N_i-1)(N_i+3)$ real additions and $(N_i-1)^2$ real multiplications. It is easily seen how to generalize this algorithm to compute the rotated DFT; for any rotation value r, simply replace the angle $\theta = 2\pi/N_i$ which appears in Eqs. (36) and (37) by the corresponding angle $r\theta$. Any coding designed to implement Singleton's algorithm for general N_i could easily be modified to handle a general rotation value r.

(e) $N_i = 9$: The algorithm recommended in [17] for this case was simply the conventional FFT with N_i factored as 3×3. The development of a corresponding rotated algorithm leads to a result for the general case $N_i = pq$. In [16] it was shown that

$$W_{pq} = (W_q \times I_p)\, P_q^p D_q^p (W_p \times I_q) \tag{38}$$

where W_p, W_q are the DFT matrices of order p, q; I_p, I_q are the corresponding identity matrices; P_q^p is a permutation matrix; D_q^p is a diagonal matrix of complex

phase (twiddle) factors; and $\times$ denotes the Kronecker product. A simple modification of the proof given in [16] for Eq. (38) shows that

$$W_{pq}^{[r]} = (W_q^{[r]} \times I_p)\, P_q^p (D_q^p)^r (W_p^{[r]} \times I_q). \tag{39}$$

In other words, the rotated algorithm for the case $N_i = pq$ is obtained from the original algorithm of Eq. (38) by applying the rotation r to the short transforms of length p and q, and raising the phase factors to the power r. In applying the rotation to the short transforms, r may be interpreted modulo p or q as necessary. Equation (39) can readily be generalized to the case of more than two factors.

(f) $N_i = 4, 8, 16$: The case $N_i = 2^p$ has deliberately been left until now, since it requires a slightly different technique. Consider the unrotated ($r = 1$) algorithm for $N_i = 4$:

$$t_1 = z_0 + z_2; \qquad t_2 = z_1 + z_3; \qquad t_3 = z_0 - z_2; \qquad t_4 = z_1 - z_3;$$

$$x_0 = t_1 + t_2; \qquad x_1 = t_3 + it_4; \qquad x_2 = t_1 - t_2; \qquad x_3 = t_3 - it_4.$$

The only other rotation allowed is $r = 3$, which interchanges x_1 and x_3, requiring that the sign of t_4 be changed. One possibility is to insert a multiplier in the expresson for t_4, which can be set equal to $+1$ for $r = 1$ or -1 for $r = 3$. In the context of FFT algorithms on the Cray-1 or Cyber 205, multiplications are "free" and there is no penalty involved [17]. Alternatively the sign bit of t_4 could be controlled by a logical operation, or separate code could be provided for the two cases.

The rotated modules for $N_i = 8, 16$ can be built up using Eq. (39) and the DFT modules for $N_i = 2, 4$. After some rearrangement of the computation, rotated modules were constructed which required 6 extra real multiplications (or logical operations) in the case $N_i = 8$, and 16 in the case $N_i = 16$. In either case, the same result could be achieved by providing just two separate sequences of code.

We have now shown how to construct rotated DFT modules for all the values of N_i normally used in prime factor algorithms. Corresponding modules for N_i a power of a prime (e.g., $N_i = 25, 27, 32...$) can be constructed using Eq. (39), while for primes $N_i > 7$ the same modification of Singleton's algorithm [15] can be used as for the case $N_i = 7$. (However, Johnson and Burrus [7] have constructed DFT modules for $N_i = 13, 17, 19$ which require slightly fewer additions and many fewer multiplications than Singleton's algorithm.)

5. Operation Counts and Timing Results

Although the prime factor algorithm described in this paper was designed for a vector machine such as the Cray-1 or Cyber 205 on which the multiplications are essentially free, it seemed appropriate to investigate its properties first on a scalar machine (IBM 3081). Since the results were rather interesting, they are reported here.

A Fortran routine was written to implement the self-sorting in-place PFA for N composed of mutually prime factors from the set (2, 3, 4, 5, 7, 8, 9, 16). A shortened and simplified version of this routine is given in the Appendix. For comparison purposes, a self-sorting conventional mixed-radix FFT routine was also written, following the algorithm design in [16], with coding for factors 2, 3, 4, 5, 7. Both routines were compiled at the highest level of optimization, and timed for numerous values of N, suitable for the PFA; the conventional routine was also timed for $N = 2^p$, using the radix $4+2$ formulation.

Table III first shows the number of real additions and multiplications for a selection of values of N, for the conventional FFT and for the PFA. For a given value of

TABLE III

Operation Counts and Times for FFT and PFA

	Adds/Mults		Time (msec)	
N	FFT	PFA	FFT	PFA
30	432/232	372/112	0.64	0.38
32	386/132	—	0.60	—
35	572/360	524/264	0.64	0.46
36	512/256	464/160	0.67	0.41
60	1012/520	864/224	1.15	0.69
63	1196/796	1100/604	1.29	0.85
64	930/324	—	1.06	—
70	1352/856	1188/528	1.52	0.96
120	2382/1276	2028/508	2.54	1.45
126	2768/1840	2452/1208	2.99	1.81
128	2242/900	—	2.51	—
140	3052/1848	2656/1056	2.96	1.92
240	5362/2788	4656/1256	5.22	3.11
252	6164/3928	5408/2416	5.87	3.71
256	5122/2052	—	4.90	—
280	6942/4252	6012/2252	6.57	4.09
504	13838/8860	12076/5084	13.03	8.09
512	11778/5124	—	11.67	—
560	15282/9060	13424/5064	13.73	8.89
1008	30194/18724	26672/11176	26.98	17.32
1024	26114/11268	—	23.48	—
1260	40892/26680	35104/15104	36.55	23.30
1680	54802/33892	46992/17432	49.31	30.56
2048	58370/26628	—	55.76	—
2520	89342/58396	76508/31468	79.75	50.50
4096	126978/57348	—	113.72	—
5040	191282/121828	165616/67976	167.13	108.54

N, as noted in [17], use of the PFA reduces the number of additions by typically 10%, while the number of multiplications is reduced by more than 50% in some cases. (The "extra" multiplications in the rotated DFT modules for $N_i = 2^p$ are not included in the PFA total; although the routine described here included the extra multiplications, they can be avoided as suggested in Section 4.)

Table III also shows the time taken on the IBM 3081 to perform a transform of length N using the FFT and PFA routines. For a given value of N, the time for the PFA is typically only 60% of the time for the FFT. More surprisingly, it appears to be distinctly advantageous to choose a value of N suitable for the PFA rather than a nearby value of the form $N = 2^p$ suitable for the FFT, even if the operation count for the PFA is slightly higher; compare, for example, the times and operation counts for $N = 126$ (PFA) and $N = 128$ (FFT). The reason presumably lies in the unexpectedly simpler structure of the PFA.

6. Summary and Prospects

We have shown in this paper how to design a self-sorting in-place prime factor FFT algorithm which minimizes the number of additions. Contrary to expectations, the indexing structure turns out to be simpler than that for the conventional FFT. The timing results obtained on an IBM 3081 are very encouraging.

It is intended that the algorithm be implemented on a Cray-1; since the reduction in the number of additions is relatively modest [17], the improvement in speed is unlikely to be as impressive as on the IBM 3081. Nevertheless, the economy in both time and storage will be useful. A further question worth investigating concerns the specialization of the algorithm presented here to the case of real/half-complex transforms [18].

Appendix

The following routine implements the prime factor algorithm developed in this paper; it should be clear how to extend the code to include higher factors. Note that to compute $\mathbf{x} = W_N^* \mathbf{z}$ rather than $\mathbf{x} = W_N \mathbf{z}$, one statement has to be added; this follows from Eq. (19).

```
C       SELF-SORTING IN-PLACE PRIME FACTOR FFT ALGORITHM
C       ------------------------------------------------
C
C       COMPLEX ARRAY A CONTAINS INPUT AND OUTPUT DATA
C
C       N = IFAX(1) * IFAX(2) * ... * IFAX(NFAX)
C
```

```
C       FOR GREATER EFFICIENCY, USE REAL ARITHMETIC
C       ------------------------------------------
C
C
      SUBROUTINE PFA(A, N, IFAX, NFAX)
      DIMENSION A(N), IFAX(NFAX)
      COMPLEX A, T1, T2, T3, T4, I
      DATA SIN60/0.86602540378/, I/(0.0, 1.0)/
C
      DO 1000 K = 1, NFAX
C
      IFAC = IFAX(K)
      M = N/IFAC
      DO 100 J = 1, IFAC
      MU = J
      MM = J * M
      IF (MOD(MM, IFAC).EQ. 1) GO TO 110
  100 CONTINUE
  110 CONTINUE
C
C     MU IS THE REQUIRED ROTATION FOR THE DFT MODULE
C     OF ORDER IFAC. TO IMPLEMENT THE INVERSE
C     TRANSFORM, INSERT THE STATEMENT:
C     MU = IFAC - MU
C
C     NOW COMPUTE THE ADDRESSES IA, IB ETC. AND SELECT
C     THE CODING FOR THE CURRENT FACTOR
C
      IA = 1
      IB = IA + MM
      IF (IFAC. EQ. 2) GO TO 200
      IC = IB + MM
      IF (IC. GT. N) IC = IC - N
      IF (IFAC. EQ. 3) GO TO 300
      ID = IC + MM
      IF (ID. GT. N) ID = ID - N
      IF (IFAC. EQ. 4) GO TO 400
C
C     CONTINUE THIS SEQUENCE FOR HIGHER FACTORS
C
C
C     CODING FOR FACTOR 2
C     -------------------
```

```
  200 CONTINUE
      DO 210 L = 1, M
      T1 = A(IA) - A(IB)
      A(IA) = A(IA) + A(IB)
      A(IB) = T1
      IX = IB + 1
      IB = IA + 1
      IA = IX
  210 CONTINUE
      GO TO 1000
C
C     CODING FOR FACTOR 3
C     - - - - - - - - - - - - - - - -
  300 CONTINUE
      Z3 = SIN60
      IF (MU. EQ. 2) Z3 = - Z3
      DO 310 L = 1, M
      T1 = A(IB) + A(IC)
      T2 = A(IA) - 0.5 * T1
      T3 = Z3 * (A(IB) - A(IC))
      A(IA) = A(IA) + T1
      A(IB) = T2 + I * T3
      A(IC) = T2 - I * T3
      IX = IC + 1
      IC = IB + 1
      IB = IA + 1
      IA = IX
  310 CONTINUE
      GO TO 1000
C
C     CODING FOR FACTOR 4
C     - - - - - - - - - - - - - - - - -
  400 CONTINUE
      Z4 = 1.0
      IF (MU. EQ. 3) Z4 = -Z4
      DO 410 L = 1, M
      T1 = A(IA) + A(IC)
      T2 = A(IB) + A(ID)
      T3 = A(IA) - A(IC)
      T4 = Z4 * (A(IB) - A(ID))
      A(IA) = T1 + T2
      A(IB) = T3 + I * T4
      A(IC) = T1 - T2
```

```
      A(ID) = T3 - I * T4
      IX = ID + 1
      ID = IC + 1
      IC = IB + 1
      IB = IA + 1
      IA = IX
  410 CONTINUE
      GO TO 1000
C
C     ADD MORE FACTORS AS REQUIRED
C
 1000 CONTINUE
      RETURN
      END
```

ACKNOWLEDGMENTS

Most of this work was performed while the author was with the U. K. Meteorological Office. I am grateful to Dr. Hal Ritchie of DRPN for his review of the first draft of this paper, and for suggesting some clarifications.

REFERENCES

1. C. S. BURRUS, *IEEE Trans. Acoust. Speech Signal Process.* **25** (1977), 239–242.
2. C. S. BURRUS AND P. W. ESCHENBACHER, *IEEE Trans. Acoust. Speech Signal Process.* **29** (1981), 806–817.
3. J. W. COOLEY AND J. W. TUKEY, *Math. Comp.* **19** (1965), 297–301.
4. W. M. GENTLEMAN AND G. SANDE, *Proc. AFIPS Joint Computer Conference* **29** (1966), 563–578.
5. I. J. GOOD, *J. Roy. Statist. Soc. Ser. B* **20** (1958), 361–372.
6. I. J. GOOD, *IEEE Trans. Comput.* **20** (1971), 310–317.
7. H. W. JOHNSON AND C. S. BURRUS, "Large DFT Modules: 11, 13, 17, 19 and 25," Technical Report No. 8105, Department of Electrical Engineering, Rice University, Houston, Texas, 1981.
8. H. W. JOHNSON AND C. S. BURRUS, *IEEE Trans. Acoust. Speech Signal Process.* **31** (1983), 378–387.
9. H. W. JOHNSON AND C. S. BURRUS, On the structure of DFT algorithms, submitted to IEEE Trans. Acoust. Speech Signal Process.
10. D. P. KOLBA AND T. W. PARKS, *IEEE Trans. Acoust. Speech Signal Process.* **25** (1977), 281–294.
11. J. H. MCCLELLAN AND C. M. RADER, "Number Theory in Digital Signal Processing," Prentice–Hall, Englewood Cliffs, N.J., 1979.
12. H. J. NUSSBAUMER, "Fast Fourier Transform and Convolution Algorithms," 2nd ed., Springer-Verlag, Berlin, 1982.
13. M. C. PEASE, *J. Assoc. Comput. Mach.* **15** (1968), 252–264.
14. J. H. ROTHWEILER, *IEEE Trans. Acoust. Speech Signal Process.* **30** (1982), 105–107.
15. R. C. SINGLETON, *IEEE Trans. Audio. Electroacoust.* **17** (1969), 93–103.
16. C. TEMPERTON, *J. Comput. Phys.* **52** (1983), 1–23.
17. C. TEMPERTON, *J. Comput. Phys.* **52** (1983), 198–204.
18. C. TEMPERTON, *J. Comput. Phys.* **52** (1983), 340–350.
19. S. WINOGRAD, *Math. Comp.* **32** (1978), 175–199.