JOURNAL OF COMPUTATIONAL PHYSICS **82**, 247–268 (1989)

Nesting Strategies for Prime Factor FFT Algorithms

CLIVE TEMPERTON*

Division de Recherche en Prévision Numérique, Service de l'Environnement Atmosphérique, 2121 Trans-Canada Highway, Dorval, Québec, Canada H9P 1J3

Received February 17, 1988

Several nesting techniques for the prime factor FFT algorithm are examined. These include the full nesting strategy of Winograd's algorithm, which gives a very low multiplication count; a split nesting technique, which requires the same number of multiplications as Winograd's FFT, but fewer additions; and a partial nesting technique, which further reduces the addition count at the cost of some extra multiplications and requires the lowest total number of floating-point operations. Applications to large multidimensional transforms, as well as to 1-dimensional transforms, are discussed. On large scientific computers where the multiplications can be overlapped with the additions, the prime factor algorithm without nesting remains the fastest and is the easiest to implement. © 1989 Academic Press, Inc.

1. INTRODUCTION

Since its introduction in 1965 by Cooley and Tukey [2], the fast Fourier transform (FFT) algorithm has become an indispensable tool of computational physics. In previous papers, the author has presented a unified derivation of the Cooley–Tukey family of algorithms [10], applications to real/half-complex transforms [12], and implementations on vector computers [9, 13].

Meanwhile, there have been interesting developments in another direction. Based on the prime factor FFT algorithm due to Good [5], Winograd [18] showed that for certain values of the transform length N the discrete Fourier transform (DFT) could be computed with many fewer multiplications than required by the Cooley–Tukey algorithm. The key idea of Winograd's algorithm is a *nesting* strategy. This represented a breakthrough from the point of view of computational complexity theory and has led to numerous applications in fields such as real-time signal processing using special-purpose hardware, where the number of multiplications is the dominant factor in the overall cost of computing the transform.

However, Winograd's algorithm has had much less impact in large-scale scientific computing. Several practical reasons may be invoked: a lack of suitable readily available software, the fact that descriptions of the algorithm are often couched in unfamiliar language, and the apparently complicated programming required.

* Present address: ECMWF, Shinfield Park, Reading, Berkshire RG2 9AX, England.

Moreover, Kolba and Parks [7] showed that, on machines where the cost of a floating-point addition is a significant fraction of that for a floating-point multiplication, it might be more efficient to use instead a form of Good's prime factor algorithm. The algorithm suggested by Kolba and Parks did, however, make use of Winograd's "small-*n* DFT modules." Temperton [11] pointed out that the analysis of Kolba and Parks applied with even greater force on machines such as the Cray-1 and Cyber 205, where the multiplications in the various FFT algorithms could be overlapped with the additions and were thus effectively free of charge. In [11] it was also shown that Winograd's algorithm would in fact be *slower* than the Cooley–Tukey algorithm on the Cray-1, since the computation time would be dominated by data transfers between central memory and vector registers. An alternative to the procedure of Kolba and Parks was proposed in [11], again based on Good's prime factor algorithm but using "minimum-add" small-*n* DFT modules in place of Winograd's. However, it was suggested that any gain over the Cooley–Tukey algorithm would be very modest.

The conclusions reached in [11] later turned out to be rather too pessimistic. Taking into account the work of Burrus and Eschenbacher [1], as refined in [14], the prime factor FFT algorithm was shown to provide significant gains on the Cray-1 [15] and Cray X-MP [17].

In this paper, we reconsider Winograd's algorithm in the light of these developments. We also consider alternative nesting strategies; bearing in mind the analysis presented in [11], the aim will be to reduce the number of additions rather than the number of multiplications. It turns out that the prime factor algorithm (without nesting) is still likely to be faster than Winograd's on machines such as the Cray-1, Cyber 205, or Cray X-MP; but several interesting new results, which may have more impact on other computers, are obtained along the way. These include the following:

(1) A variant of Winograd's nesting strategy systematically reduces the number of additions required, while maintaining the same very low multiplication count. Moreover, this variant permits a self-sorting "almost-in-place" implementation, in contrast to Winograd's formulation which requires a considerable amount of work space.

(2) An alternative nesting strategy further reduces the addition count at the expense of some extra multiplications and allows a more straightforward self-sorting in-place implementation. This algorithm ("partial nesting") leads to the smallest *total* operation count.

(3) For certain values of the transform length N, algorithms exist which require even fewer additions than the prime factor algorithm using "minimum-add" DFT modules.

These results are derived using straightforward matrix algebra; some readers may be relieved to know that no use whatsoever is made, for example, of the theory of irreducible cyclotomic polynomials [8].

The organization of this paper is as follows. In Section 2 we establish notation and briefly compare the family of "conventional" FFT algorithms, based on that of Cooley and Tukey, with the prime factor algorithm. Section 3 derives Winograd's algorithm and discusses its implementation. In Section 4 we use an approach due to Johnson and Burrus [6] to reduce the number of additions required by this algorithm. It is also shown that the modified algorithm can be implemented essentially in-place. Section 5 introduces another nesting strategy which trades a further reduction in the addition count for some extra multiplications and permits a straightforward self-sorting in-place implementation. At this juncture, we compare operation counts and show that the prime factor algorithm still requires the lowest number of additions amongst the algorithms so far considered. In Section 6 it is shown that, for certain values of the transform length, other algorithms do exist which require even fewer additions. Section 7 discusses some practical considerations in the choice of algorithms. Applications to multidimensional transforms are presented in Section 8, and finally Section 9 contains a brief summary.

2. Conventional and Prime Factor Algorithms

The discrete Fourier transform (DFT) is defined by

$$x_n = \sum_{k=0}^{N-1} c_k \omega_N^{kn}, \qquad 0 \leqq n \leqq N-1, \tag{1}$$

where $\omega_N = \exp(\pm 2i\pi/N)$. (Either sign may be chosen in the definition of ω_N). Throughout this paper it will be assumed that both x_n and c_k are complex. To develop fast algorithms for Eq. (1), we rewrite the transform as

$$\mathbf{x} = W_N \mathbf{c}, \tag{2}$$

where W_N is the DFT matrix of order N, with element (j, k) given by ω^{jk} where the rows and columns of W_N are indexed from 0 to $N-1$. Fast transform algorithms can be defined whenever N can be factorized as a product of smaller integers,

$$N = N_1 N_2 \cdots N_k.$$

Conventional Algorithms

A unified derivation of the many variants of the Cooley–Tukey FFT algorithm was given in [10], in terms of a factorization of the DFT matrix W_N. For this algorithm, there is no restriction on the factors N_i. It was shown for example that if $N = N_1 N_2$, then

$$W_N = (W_{N_2} \times I_{N_1})\, P_{N_2}^{N_1} D_{N_2}^{N_1} (W_{N_1} \times I_{N_2}), \tag{3}$$

where $P_{N_2}^{N_1}$ is a permutation matrix, $D_{N_2}^{N_1}$ is a diagonal matrix of "twiddle factors," and $\times$ denotes Kronecker (tensor product) multiplication.

The FFT is fast for two reasons. First, the transform of length N is decomposed into sets of shorter transforms of length N_i. Second, fast "small-n" algorithms ("DFT modules") are available for implementing the component transforms. A mixed-radix FFT code might, for example [10], include such modules for $N_i = 2, 3, 4, 5$, and 6. Efficient algorithms for $N_i = 2, 3, 4$ can easily be found by inspection. Efficient algorithms for $N_i = 5, 6$ were given in [10], together with formulae for calculating the overall operation count for a transform of length N.

Prime Factor Algorithms

Several years before the publication of the Cooley–Tukey algorithm, Good [5] developed a factorization of the DFT matrix for the case where the factors N_i of N are mutually prime (having no common factors). Good's factorization may be written as

$$W_N = Q^{-1}(W_{N_1} \times W_{N_2} \times \cdots \times W_{N_k})P, \tag{4}$$

where P and Q are special permutation matrices. Combining Eqs. (2) and (4), the transform becomes

$$\mathbf{x}' = (W_{N_1} \times W_{N_2} \times \cdots \times W_{N_k})\mathbf{c}', \tag{5}$$

where $\mathbf{c}' = P\mathbf{c}$ and $\mathbf{x}' = Q\mathbf{x}$ represent permutations of the input and output data.

In this paper we will not be especially concerned with the details of these permutations, which have been discussed previously in [1, 14]. It will suffice to make the following observations:

(a) If the transform is followed by its inverse, with some intervening operations in transform space which can use the coefficients in "scrambled" order, then the permutation $\mathbf{x} = Q^{-1}\mathbf{x}'$ can be dispensed with.

(b) By simple modifications to the component transform matrices W_{N_i}, and to the corresponding DFT modules, the permutation matrices P and Q can be made the same [1, 14].

(c) In neither case is any explicit physical rearrangement of the data required; the permutation can be handled implicitly by the indexing logic. A simple and elegant indexing scheme, which achieves this and permits in-place implementation, was presented in [14]. In case (b) above, the algorithm becomes self-sorting as well as in-place.

In view of the foregoing remarks, we will rewrite Eq. (4) as

$$\tilde{W}_N = W_{N_1} \times W_{N_2} \times \cdots \times W_{N_k}, \tag{6}$$

where $\tilde{W}_N = QW_NP^{-1}$ is simply the DFT matrix with its rows and columns permuted. The importance of Eq. (6) is that the original 1-dimensional transform of length N has been reduced to a k-dimensional $N_1 \times N_2 \times \cdots \times N_k$ transform, with no intervening "twiddle factors." This can be carried out simply by performing a transform of length N_1 along each of the N/N_1 "rows" (the first dimension), then a transform of length N_2 along each of the N/N_2 "columns" (the second dimension), and so on. In the remainder of this paper, it will be very helpful to visualize the implementation of Kronecker (tensor) products of matrices in this geometrical way. (A similar viewpoint can be taken to explain the Cooley–Tukey family of algorithms; see de Boor [3] for a notable example).

The prime factor algorithm (PFA) depends crucially on the factors N_i being mutually prime. With just the "small-n" algorithms as used in the conventional mixed-radix FFT, the range of possible values of N would be extremely restricted. We are thus driven to look for "small-n" algorithms for somewhat larger factors. For example, $N = 144$ can be factorized as $N = 3 \times 3 \times 4 \times 4$ or $N = 4 \times 6 \times 6$ for the conventional FFT, but it must be factorized as $N = 9 \times 16$ for the prime factor algorithm. Clearly we can build up the required algorithms from those for $N_i = 3$ and $N_i = 4$, using Eq. (3). This approach leads to DFT modules which seem (although there is no formal proof) to minimize the number of *additions*. It turns out, however, that we have a choice for some values of N_i; Winograd [18] developed an alternative set of DFT modules which minimize the number of *multiplications*. (As we shall see in the next section, Winograd's modules also have other important properties). Thus we can implement prime factor algorithms defined by Eq. (6) using either the "minimum-add" or "minimum-multiply" (Winograd) DFT modules as a basis. For the PFA and for Winograd's algorithm as described in Section 3, modules are usually provided for the following set of factors: {2, 3, 4, 5, 7, 8, 9, 16}. The operation counts for the individual DFT modules are summarized in Table I. The minimum-add modules are detailed in [16], while the Winograd modules may be found for example in [8 or 18]. Any of these modules could, of course, be used in the context of the conventional FFT.

The overall operation count for the PFA may be computed as follows. Suppose $N = N_1 N_2 \cdots N_k$, and let $A(N_i)$, $M(N_i)$ be respectively the number of real additions

TABLE I

Real Operation Counts for Small-n DFT Modules

	Minimum-add		Winograd	
n	Adds	Mults	Adds	Mults
2	4	0	4	0
3	12	4	12	4
4	16	0	16	0
5	32	12	34	10
7	60	32	72	16
8	52	4	52	4
9	80	28	88	20
16	144	24	148	20

and multiplications for the DFT module of order N_i, as given in Table I for the chosen variant. Then the total operation count is given by

$$A(N) = \sum_{i=1}^{k} (N/N_i)\, A(N_i),$$

$$M(N) = \sum_{i=1}^{k} (N/N_i)\, M(N_i).$$

3. The Winograd FFT Algorithm (WFTA)

Winograd's algorithm [18] depends on the following observation. Each of the "minimum-multiply" small-n DFT modules discussed in the previous section can be written in the form

$$W_n = B_n M_n A_n, \tag{7}$$

where A_n, B_n are matrices whose non-zero entries are all ± 1. Thus, multiplication by these matrices requires only additions and subtractions. M_n is a diagonal matrix whose entries are all pure real or pure imaginary numbers. In general A_n and B_n may be rectangular; e.g., A_n is an $\bar{n} \times n$ matrix, M_n is $\bar{n} \times \bar{n}$, and B_n is $n \times \bar{n}$, with $\bar{n} \geqq n$. However, in the algorithms given by Winograd, $\bar{n}$ is at most $n+2$. In the terminology of Johnson and Burrus [6], A_n and B_n are the "pre-weave" and "post-weave" matrices. An example for $n = 5$ is detailed in the Appendix.

The key to Winograd's algorithm is now the way the small-n DFT modules are combined. Using a factorization of the form (7) for each N_i, substituting in the prime factor algorithm defined by Eq. (6) and using the algebra of Kronecker products, we obtain

$$\tilde{W}_N = B_N M_N A_N, \tag{8}$$

where

$$B_N = B_{N_1} \times B_{N_2} \times \cdots \times B_{N_k}, \tag{9}$$

$$M_N = M_{N_1} \times M_{N_2} \times \cdots \times M_{N_k}, \tag{10}$$

$$A_N = A_{N_1} \times A_{N_2} \times \cdots \times A_{N_k}. \tag{11}$$

Note that Eq. (8) still has the same form as Eq. (7). The non-zero entries of the pre-weave and post-weave matrices A_N and B_N are still all ± 1, while M_N is still a diagonal matrix whose entries are all pure real or pure imaginary numbers.

From the expressions given in Eqs. (9) and (11) above, we see that multiplication by A_N is equivalent to applying, for each i $(1 \leqq i \leqq k)$, the pre-weave part of the transform of length N_i along the appropriate "row" or "column" of a k-dimensional array, and similarly for multiplication by the post-weave matrix B_N. These operations require only additions and subtractions. Multiplication by M_N is simply pointwise multiplication of every element in the array by a real or imaginary constant. Thus all the multiplications required in the algorithm have been nested together, and each takes just two real multiplications.

The difficulty with this algorithm, which complicates the implementation and in particular precludes in-place working, is the fact that the pre-weave and post-weave matrices are in general rectangular. Thus the k-dimensional array tends to expand along each dimension in turn as the pre-weave operations are applied, and similarly to contract during the post-weave phase.

One result is that the number of additions required is a function of the order in which the factors are used, in contrast to the conventional and prime factor algorithms for which the operation count is independent of the order of the factors.

To demonstrate this, we first summarize in Table II the number of real pre-weave additions $A(n)$ and post-weave additions $B(n)$ for each value of n. Also shown is the order $\bar{M}(n)$ of M_n, and the number $m(n)$ of entries of M_n which are ± 1 or $\pm i$, leading to "trivial" multiplications. The counts are taken from the Winograd modules as given by Nussbaumer [8].

Let $N = N_1 N_2 \cdots N_k$. Regardless of the order in which the factors are used, all the multiplications are nested together in the multiplication by M_N, and the number of non-trivial real multiplications is given by

$$M(N) = 2 \prod_{i=1}^{k} \bar{M}(N_i) - 2 \prod_{i=1}^{k} m(N_i).$$

Now suppose the factors are used in the order $N_1, N_2, \ldots, N_k$ in the pre-weave phase and in the reverse order during the post-weave phase. Let $\bar{N}_i$ be the size of the data array after i substages of the pre-weave phase; thus, $\bar{N}_i$ may be found recursively by setting

$$\bar{N}_0 = N, \qquad \bar{N}_i = \bar{N}_{i-1}(\bar{M}(N_i)/N_i) \qquad \text{for} \quad 1 \leq i \leq k. \tag{12}$$

TABLE II

Values of $A(n)$, $B(n)$, $\bar{M}(n)$, and $m(n)$ for Winograd's DFT Modules

n	$A(n)$	$B(n)$	$\bar{M}(n)$	$m(n)$
2	4	0	2	2
3	6	6	3	1
4	12	4	4	4
5	16	18	6	1
7	34	38	9	1
8	32	20	8	6
9	40	48	11	1
16	80	68	18	8

TABLE III

Operation Counts for Complex Transform, $N=720$

Algorithm	Real Adds	Real Mults	Total
Conventional, $N=2^4\cdot 3^2\cdot 5$	20642	12676	33318
Conventional, $N=3^2\cdot 4^2\cdot 5$	19922	11236	31158
Conventional, $N=4\cdot 5\cdot 6^2$	19322	10036	29358
PFA, min-add modules	17488	5048	22536
PFA, Winograd modules	18596	3940	22536
Winograd, best ordering	21312	2360	23672
Winograd, worst ordering	23112	2360	25472

Then the total number real additions is given by

$$A(N)=\sum_{i=1}^{k}(\bar{N}_{i-1}/N_i)(A(N_i)+B(N_i)).$$

To minimize the number of additions, the factors N_i should be used in a particular order. It is quite easy to show that $N_i=2, 3, 4, 8$ can be used in any order as "outer" factors (i.e., used before other factors in the pre-weave phase, and after other factors in the post-weave phase); the modules for these factors have square pre-weave and post-weave matrices, so they do not change the size of the array. The remaining factors should be used in the order 16 (outermost), 9, 7, 5 (innermost), in a symmetric manner for the pre-weave and post-weave phases.

We conclude this section with a comparison of operation counts for a complex transform of length $N=720$, as summarized in Table III. For the conventional algorithm, operation counts were computed as in [10] for three different factorizations, showing the improvements obtained by first grouping the factors of 2 in pairs (radix-4 vs. radix-2), and then including a radix-6 module in the algorithm. For the PFA and Winograd algorithms, the factorization is always $N=5\times 9\times 16$. The PFA based on minimum-add modules requires the fewest additions of all the algorithms in Table III, but the PFA based on Winograd DFT modules requires significantly fewer multiplications. These two versions of the PFA require the same total number of operations. Winograd's (nested) algorithm requires by far the smallest number of multiplications, but even with the factors optimally ordered it requires more additions than any of the other algorithms and more operations in total than the PFA.

4. Split Nesting

Johnson and Burrus [6] pointed out that the pre-weave and post-weave matrices A_n and B_n in Eq. (7) may be further factorized; thus each of the Winograd small-n DFT modules may be written in the form

$$W_n=F_nE_nM_nD_nC_n \tag{13}$$

with the pre-weave and post-weave phases each split into two parts. Inserting the factorization (13) into Eq. (6) and using the algebra of Kronecker products generates a large class of DFT algorithms which includes the PFA and the fully nested Winograd algorithm as special cases. To see how large this class is, suppose $N=N_1N_2N_3N_4$, where the N_i are mutually prime, and consider a given factorization of the form (13) for each of the four DFT modules involved. Using the formula given in [6], the number of distinct DFT algorithms which can be generated in this way is $20!/(5!)^4 \sim 1.17\times 10^{10}$. A dynamic programming procedure is presented in [6] which finds the best algorithm within this class, with respect to a user-specified cost function which may, for example, depend on the number of additions, multiplications, data transfers, and subroutine calls. A simple but illuminating example, for $N=35=5\times 7$, is presented in [6] to illustrate possible trade-offs between additions and multiplications. The results are reproduced in Table IV.

Thus the new algorithm generated by the procedure of Johnson and Burrus requires the same number of multiplications as the WFTA, while reducing the number of additions close to the level of the PFA and requiring fewer operations in total than either. This is a very interesting result; if the cost of additions and multiplications is comparable, then the new algorithm is better than either the PFA or the WFTA. Even in the case where the multiplications can be overlapped with the additions and are thus essentially free, the new algorithm is a close rival to the PFA.

Using the basic idea proposed by Johnson and Burrus in [6], it turns out to be straightforward to develop a general procedure for splitting and recombining the Winograd DFT modules in such a way as to maintain the multiplication count of the WFTA while reducing the addition count close to that of the PFA.

First, the way in which the factorized modules are recombined will be analogous to that used in Eqs. (8)–(11). Suppose $N=N_1N_2\cdots N_k$, and that a factorization of the form (13) is given for each N_i. Substituting in the prime factor algorithm defined by Eq. (6) and using the algebra of Kronecker products, we obtain

$$\tilde{W}_N=F_NE_NM_ND_NC_N, \tag{14}$$

where

$$F_N=F_{N_1}\times F_{N_2}\times \cdots \times F_{N_k},$$
$$E_N=E_{N_1}\times E_{N_2}\times \cdots \times E_{N_k},$$
$$M_N=M_{N_1}\times M_{N_2}\times \cdots \times M_{N_k},$$
$$D_N=D_{N_1}\times D_{N_2}\times \cdots \times D_{N_k},$$
$$C_N=C_{N_1}\times C_{N_2}\times \cdots \times C_{N_k}.$$

Second, the factorizations $A_n=D_nC_n$ and $B_n=F_nE_n$, for the pre-weave and post-weave matrices, respectively, will be done in such a way that C_n and F_n are square, while D_n and E_n contain as few operations as possible. Thus in Eq. (14), the first part of the pre-weave phase, represented by C_n, will not increase the size of the original k-dimensional array. Expansion of the array will only occur during the

TABLE IV

Operation Counts for Complex Transform, $N = 35$

Algorithm	Real Adds	Real Mults	Total
PFA, Winograd modules	598	150	748
WFTA	666	106	772
Johnson and Burrus	610	106	716

second part of the pre-weave, and as few additions as possible will be performed on the expanded array. The splitting of the post-weave phase is analogous, with as many additions as possible being delayed until the last stage, represented by F_N, by which time the array will have contracted back to its original size.

The required factorizations $A_n = D_n C_n$ and $B_n = F_n E_n$ are trivial in the cases $n = 2, 3, 4, 8$; since A_n and B_n are already square, we simply set $D_n = E_n = I_n$ (the identity matrix), $C_n = A_n$ and $F_n = B_n$. Examination of the Winograd DFT modules as specified, for example, by Nussbaumer [8] shows that the factorization can be written in such a way that in the case $n = 5$, D_5 and E_5 respectively contain just one and two complex additions. In the cases $n = 7, 9, 16$ (for which $\bar{n} = n + 2$), multiplications by D_n and E_n correspond respectively to 2 and 4 complex additions. The factorization for $n = 5$ is demonstrated in the Appendix.

Thus, in general, multiplication by the $\bar{n} \times n$ matrix A_n in Eq. (7) is equivalent to multiplying by an $n \times n$ matrix C_n, followed by $(\bar{n} - n)$ extra complex additions to produce the required vector of length $\bar{n}$. Similarly, multiplication by the $n \times \bar{n}$ matrix B_n is equivalent to performing $2(\bar{n} - n)$ complex additions before multiplying by the $n \times n$ matrix F_n. Within each individual DFT module, the operation count is unchanged. However, when the modules are nested as in Eq. (14) there is a significant decrease in the addition count compared with the "fully nested" algorithm of Eq. (8).

Table V summarizes, for each n, the number of real additions $A_1(n)$ in the algorithms for multiplications by C_n and F_n (the total is for the two matrices combined.) $A_2(n)$ is the corresponding total for multiplication by D_n and E_n.

The overall operation count for the split nested algorithm is found as follows, for $N = N_1 N_2 \cdots N_k$. First, the number of multiplications is exactly the same as for the WFTA. The number of real additions for the "non-expanding" parts of the pre-weave and post-weave stages is

$$A_1(N) = \sum_{i=1}^{k} (N/N_i)\, A_1(N_i)$$

and is clearly independent of the order in which the factors are used. Defining $\bar{N}_i$ as in Eq. (12), the number of real additions for the "expanding" parts is

$$A_2(N) = \sum_{i=1}^{k} (\bar{N}_{i-1}/N_i)\, A_2(N_i)$$

TABLE V

Values of $A_1(n)$ and $A_2(n)$ for Split Nested DFT Modules

n	$A_1(n)$	$A_2(n)$
2	4	0
3	12	0
4	16	0
5	28	6
7	60	12
8	52	0
9	76	12
16	136	12

and it turns out that this sum too is independent of the order in which the factors are used. Hence for the split nested algorithm, the operation count does not depend on the order of the factors.

Table VI summarizes, for $N = 720$, the operation counts for this "split nested" algorithm compared with those for the PFA based on Winograd's DFT modules and for the fully nested WFTA. The multiplication count is the same as for the WFTA, while the addition count has been reduced to a level close to that for the PFA, and the total operation count is lower than that for either the WFTA or the PFA. The operation counts for the split nested algorithm are the same as those [illegible] their $N = 35$ algorithm referred to earlier.

As described above, the split nested algorithm still suffers from the same disadvantage as Winograd's fully nested algorithm, namely an expanding work array. This could in principle be avoided by considering the structure of the matrices in the factorization given by Eq. (14). Since C_N and F_N are square and simply represent sequences of "row" or "column" operations on the k-dimensional array, multiplication by these matrices can be done in-place. In addition, the matrix product $E_N M_N D_N$ is square and very sparse, with many of its rows having a non-zero entry only on the diagonal. In general, a row may have 2^p non-zero entries where

TABLE VI

Operation Counts for Complex Transform, $N = 720$

Algorithm	Real adds	Real mults	Total
WFTA	21312	2360	23672
Split nested	19040	2360	21400
PFA, Winograd modules	18596	3940	22536
PFA, min-add modules	17488	5048	22536

$0 \leq p \leq k$; if $p > 0$, then there will in fact be $2^p - 1$ other rows with non-zero entries in the same columns. From a geometrical viewpoint, $E_N M_N D_N$ operates on small disjoint "blocks" of the k-dimensional array. Each such block consists of 2^p points which form the vertices of a p-dimensional "cuboid" within the array. If the multiplication by $E_N M_N D_N$ is done block by block, then only $2^k - 1$ (complex) words of work space are required. Such an implementation could be programmed in a manner analogous to that for the "X"-stage of the real/half-complex prime factor algorithm described in [17].

As an example of the usefulness of this device, consider the transform of length $N = 5040 = 5 \times 7 \times 9 \times 16$. Using either the WFTA or a straightforward implementation of the split nested algorithm, the data array expands to a length of $\bar{N} = 10692$, more than twice the original size. Using the modified algorithm just described, the computation can be done using just the original array plus 15 complex words of work space. If the rotated forms of the DFT modules are used, then the algorithm becomes self-sorting as well as effectively in-place.

5. Partial Nesting

In the previous section, the ideas of Johnson and Burrus [6] were used to develop an algorithm which required the same number of multiplications as the WFTA, while reducing the addition count. We will now show that similar ideas may be used to achieve further reductions in the number of additions, at the expense of some extra multiplications. However, the *total* operation count is reduced compared with split nesting, and the implementation is more straightforward.

As in the WFTA of Section 3, each of the small-n DFT modules will be written as the product of three factors:

$$W_n = B'_n M'_n A'_n \tag{15}$$

and the modules will be combined in exactly the same way as for the WFTA,

$$\tilde{W}_N = B'_N M'_N A'_N, \tag{16}$$

where

$$B'_N = B'_{N_1} \times B'_{N_2} \times \cdots \times B'_{N_k}, \tag{17}$$

$$M'_N = M'_{N_1} \times M'_{N_2} \times \cdots \times M'_{N_k}, \tag{18}$$

$$A'_N = A'_{N_1} \times A'_{N_2} \times \cdots \times A'_{N_k}. \tag{19}$$

The difference from the WFTA lies in the factorization (15). The non-zero entries of A'_n are all ± 1, but this time the matrix is square $(n \times n)$ rather than rectangular. M'_n is also $(n \times n)$ and diagonal with pure real or pure imaginary entries. Finally, B'_n is square $(n \times n)$ and is allowed to have (pure real) non-zero entries other than ± 1.

For $n = 2, 3, 4, 8$ the Winograd factorization (7) is already in this form. The necessary modifications to the Winograd algorithm for $n = 5$ are detailed in the Appendix, where the modified version is shown to be closely related to the minimum-add algorithm for $n = 5$. Inspection of the Winograd modules for $n = 7, 9, 16$ shows that similar modifications can be made in each case.

The algorithm defined by Eqs. (16)–(19) can easily be implemented in-place, since the pre-weave and post-weave matrices A'_N and B'_N are square and represent sequences of "row" or "column" operations on a k-dimensional array, while M'_N is diagonal and simply represents pointwise multiplication of the array by a field of predetermined constants. The size of the k-dimensional array is constant throughout the algorithm just as in the PFA, but most of the multiplications are nested as in the WFTA.

To determine the operation count for this "partially nested" algorithm, we specify for each DFT module the number of real additions $A(n)$, the number $m_0(n)$ of "trivial" (± 1, $\pm i$) diagonal entries in M'_n, and the number $\mu(n)$ of real multiplications required in the algorithm for multiplying by B'_n. This information is summarized in Table VII.

If $N = N_1 N_2 \cdots N_k$, then the total number of real additions is given by

$$A(N) = \sum_{i=1}^{k} (N/N_i)\, A(N_i)$$

while the total number of real multiplications is given by

$$M(N) = 2N - 2 \prod_{i=1}^{k} m_0(N_i) + \sum_{i=1}^{k} (N/N_i)\, \mu(N_i).$$

Some examples are included in Table VIII, for various values of N. In each case the number of additions lies between the counts for the two forms of the PFA. The number of multiplications is greater than for the WFTA or split nesting, but less

TABLE VII

Values of $A(n)$, $m_0(n)$, and $\mu(n)$ for the DFT Modules Used in the "Partially Nested" Algorithm

n	$A(n)$	$m_0(n)$	$\mu(n)$
2	4	2	0
3	12	1	0
4	16	4	0
5	32	1	4
7	68	1	8
8	52	6	0
9	84	1	8
16	144	8	8

TABLE VIII

Real Operation Counts for Complex Transforms of Length N

	Adds	Mults	Total
$N = 144$			
WFTA	2916	380	3296
Split nested	2764	380	3144
PFA (min-mult)	2740	500	3240
Partially nested	2640	472	3112
PFA (min-add)	2576	664	3240
$N = 720$			
WFTA	21312	2360	23672
Split nested	19040	2360	21400
PFA (min-mult)	18596	3940	22536
Partially nested	17808	3000	20808
PFA (min-add)	17488	5048	22536
$N = 1008$			
WFTA	34668	3548	38216
Split nested	30364	3548	33912
PFA (min-mult)	29548	5804	35352
Partially nested	28272	4552	32824
PFA (min-add)	26672	9256	35928
$N = 5040$			
WFTA	233928	21368	255296
Split nested	190736	21368	212104
PFA (min-mult)	182012	39100	221112
Partially nested	173616	26856	200472
PFA (min-add)	165616	58376	223992

than for either form of the PFA. The *total* number of operations is lower than for any of the other algorithms. Partial nesting might therefore be attractive for machines on which additions and multiplications contribute equally to the transform cost.

6. Further Reductions in the Addition Count

In the preceding sections, we have explored several FFT algorithms which display various trade-offs between additions and multiplications. The prime factor algorithm (PFA) based on the "minimum-add" DFT modules requires the fewest additions, but even for this algorithm the number of additions is several times the number of multiplications. On computers which can overlap the additions and the multiplications, a further reduction in the addition count would be profitable even at the expense of extra multiplications. It is thus natural to ask whether algorithms exist which require fewer additions than the PFA based on minimum-add DFT modules.

For certain values of N such algorithms do exist, and in this section we sketch some possibilities. Consider for example $N=144$. The PFA in this case, following Eq. (6), is based on the decomposition

$$\tilde{W}_{144} = W_9 \times W_{16}. \tag{20}$$

Alternatively we could have factorized 144 as 12×12 and used the "conventional" decomposition, following Eq. (3):

$$W_{144} = (W_{12} \times I_{12})\, P_{12}^{12} D_{12}^{12} (W_{12} \times I_{12}). \tag{21}$$

Equation (21) splits the transform into a sequence of transforms of length 12, each of which can be computed as a 3×4 transform via the PFA. The diagonal matrix D_{12}^{12} contains the "twiddle factors" resulting from the conventional factorization. Allowing special treatment of the twiddle factors of the form $\exp(i\theta)$, where θ is an integer multiple of $\pi/4$, the operation count for the algorithm defined by Eq. (21) is 2536 real additions, 832 real multiplications. Comparison with Table VIII shows that a small saving in the number of additions has been achieved.

Another possibility is to start from Eq. (20), but instead of using the minimum-add DFT modules of length 9 and 16 we further decompose W_9 and W_{16} via a conventional factorization to yield

$$\tilde{W}_{144} = [(W_3 \times I_3)\, P_3^3 D_3^3 (W_3 \times I_3)] \times [(W_4 \times I_4)\, P_4^4 D_4^4 (W_4 \times I_4)]. \tag{22}$$

The algebra of Kronecker products can then be used to convert Eq. (22) to the form

$$\tilde{W}_{144} = (W_3 \times I_3 \times W_4 \times I_4)(P_3^3 \times P_4^4)(D_3^3 \times D_4^4)(W_3 \times I_3 \times W_4 \times I_4). \tag{23}$$

In Eq. (23) we have succeeded in nesting the twiddle factors which were implicitly contained within the DFT modules of length 9 and 16, just as in the multi-dimensional FFT algorithm described in [10]. Again allowing special treatment of twiddle factors corresponding to angles which are multiples of $\pi/4$, the algorithm defined by Eq. (23) requires 2512 real additions and 760 real multiplications. Compared with that defined by Eq. (20), we have saved 64 additions at the cost of 96 extra multiplications. This example shows that *additions* can be nested—provided that they are embedded in complex multiplications.

This algorithm for $N=144$ can be linked via the PFA with the DFT modules of length 5 and 7, to generate algorithms for the other values of N shown in Table VIII. However, it is evident from the operation counts for $N=144$ that the savings over the basic PFA are hardly worthwhile, especially in view of the increased complexity in program structure and indexing. Although we have shown that the PFA does not always attain the minimum possible number of additions, it remains an open question whether a significant reduction can be achieved.

7. Practical Considerations

Sections 2–5 described several alternative algorithms for computing FFT's, all based on the factorization of the transform length N into mutually prime factors. The fully nested Winograd algorithm (WFTA) and the split nested algorithm require the fewest multiplications. The "partially nested" algorithm requires the [illegible]

In practice there are other important considerations involved in choosing an algorithm. On many computers, from the smallest up to the Cray-1 and beyond, the number of memory references may determine the running time of an algorithm. In the case of the algorithms described here, the number of memory references is closely related to the number of passes required through the data. If N has k mutually prime factors, then the prime factor algorithm (in either version) requires just k passes, one for each factor. Winograd's fully nested algorithm requires $2k$ passes, a "pre-weave" stage and a "post-weave" stage for each factor; this was pointed out in [11] as being a disadvantage on the Cray-1. The partially nested algorithm of Section 5 also requires $2k$ passes, while the split nested algorithm of Section 4 requires $4k$ passes (though some of these may be null, depending on the factors). From the point of view of memory references (or data transfers), the PFA has a clear advantage over the other algorithms.

If in-place capability is important, then the PFA again has the advantage [14], shared with the partially nested algorithm. The split nested algorithm can also be used in-place (Section 4) at the expense of more complicated code. The fully nested WFTA cannot be implemented in-place because of the varying size of the array occupied by the data. All of these algorithms can be made self-sorting by using the "rotations" described in [14] for the underlying prime factor algorithm decomposition.

Another advantage of the PFA is that it does not need a precalculated table of multiplier constants. The partially nested algorithm requires such a table (of length N), as do the WFTA and the split nested algorithm (of length $\bar{N} > N$, where $\bar{N}$ is the maximum size of the expanded array).

Finally, the PFA seems to have an advantage over the other algorithms in terms of code simplicity. As demonstrated in [14], the PFA code is very straightforward.

8. Multi Dimensional Transforms

So far in this paper we have been concerned only with 1-dimensional transforms. If the transform length N is the product of k mutually prime factors,

$N = N_1 N_2 ... N_k$, then the basis of the prime factor algorithm (and all other algorithms derived from it) is the conversion of the 1-dimensional transform to a k-dimensional transform, as in Eq. (4). Now, the matrix for a 2-dimensional $N \times N$ transform can be written immediately as a tensor product:

$$W_{(N \times N)} = W_N \times W_N. \tag{24}$$

Using Eq. (4), Eq. (24) becomes

$$W_{(N \times N)} = (Q^{-1} \times Q^{-1})(W_{N_1} \times \cdots \times W_{N_k} \times W_{N_1} \times \cdots \times W_{N_k})(P \times P) \tag{25}$$

which (apart from the permutations) is in the form of a $2k$-dimensional transform. The extension to three dimensions ($N \times N \times N$) is immediate. Moreover, any of the nesting strategies of Sections 3–5 can be used in constructing an algorithm for such a transform.

In Table IX we list the operation counts for a 3-dimensional transform of size $720 \times 720 \times 720$. Transforms of this size are of practical interest and have indeed already been achieved [4]. The PFA based on minimum-add DFT modules again requires the smallest number of additions. The partly nested algorithm saves over half the multiplications compared with the PFA and again requires the lowest total number of operations. The most striking result is for the fully nested Winograd algorithm, which now requires more than twice as many additions as the PFA. Moreover, the data array expands to almost $4\frac{1}{2}$ times its initial size, and this penalty has now made the full nesting strategy counterproductive for the multiplication count: the WFTA (and similarly the split nested algorithm) actually requires more multiplications than the partially nested algorithm.

The data for a complex $720 \times 720 \times 720$ 3-dimensional transform would occupy about three-quarters of the central memory of a gigaword machine. The prime factor algorithm could easily be implemented in its self-sorting in-place form, is the only one of the algorithms considered here which would not require a precomputed multiplier table at least as large as the data itself, and would be the fastest algorithm on any machine where the multiplications can be overlapped with the additions.

TABLE IX

Real Operation Counts for 3-Dimensional Complex Transform, $720 \times 720 \times 720$

Algorithm	Adds	Mults	Total
Fully nested WFTA	55 032 874 848	3 353 352 320	58 386 227 168
Split nested	33 064 578 432	3 353 352 320	36 417 930 752
PFA, Winograd modules	28 920 499 200	6 127 488 000	35 047 987 200
Partially nested	27 695 001 600	3 197 490 176	30 892 491 776
PFA, min-add modules	27 197 337 600	7 850 649 600	35 047 987 200

9. Summary

In this paper we have explored several nesting strategies which can be applied to the prime factor FFT algorithm. Winograd's technique [18] requires the lowest number of multiplications, but this is achieved at the expense of more additions and extra work space. Using a modification due to Johnson and Burrus [6], some of these extra additions can be eliminated and the algorithm can be implemented in-place. A "partial" nesting strategy is proposed to reduce the addition count further; this algorithm can be performed in-place and requires the smallest total number of operations. For large multi-dimensional transforms, Winograd's algorithm is no longer optimal for the number of multiplications, while the addition count and the work space required both increase sharply.

In the context of large-scale scientific computing on machines where the multiplications can be overlapped with the additions, it is clear that the prime factor algorithm (PFA) *without* nesting remains the best algorithm; fortunately it is also the simplest to implement. In most other contexts the relative simplicity of the PFA will also act in its favour; but if the number of multiplications or the total number of operations is the overriding consideration, then the "split nested" or "partially nested" variants of the algorithm might provide useful alternative formulations.

Appendix

Here we set out the details of the matrix factorizations used in the various nested algorithms, for the DFT module of length 5. Suppose we wish to compute the complex transform

$$x_j = \sum_{k=0}^{4} z_k \omega^{jk}, \qquad 0 \leqq j \leqq 4, \tag{26}$$

where $\omega = \exp(2\pi i/5)$. The "multiplier constants" used in Winograd's algorithm are as follows, where $\theta = 2\pi/5$:

$$\begin{aligned}
c_1 &= \tfrac{1}{2}(\cos\theta + \cos 2\theta) - 1 = -1.25\\
c_2 &= \tfrac{1}{2}(\cos\theta - \cos 2\theta) = \sqrt{5}/4\\
c_3 &= \sin\theta - \sin 2\theta\\
c_4 &= \sin\theta + \sin 2\theta\\
c_5 &= \sin\theta.
\end{aligned}$$

The Winograd algorithm for calculating the transform (26) may be found in Winograd [18] or Nussbaumer [8] (though in the latter case the algorithm requires slight modification to change the sign of the exponent of ω). With some convenient changes in notation, the algorithm is as follows, where the computation has been split into five stages for future reference:

(a) $u_1 = z_1 + z_4$; $u_2 = z_2 + z_3$; $t_3 = z_1 - z_4$; $t_4 = z_2 - z_3$; $t_1 = u_1 + u_2$; $t_2 = u_1 - u_2$; $t_0 = z_0 + t_1$;

(b) $t_5 = t_3 - t_4$;

(c) $m_0 = t_0$; $m_1 = c_1 * t_1$; $m_2 = c_2 * t_2$; $m_3 = ic_3 * t_3$; $m_4 = ic_4 * t_4$; $m_5 = ic_5 * t_5$;

(d) $s_3 = m_5 + m_4$; $s_5 = m_5 - m_3$;

(e) $s_1 = m_0 + m_1$; $s_2 = s_1 + m_2$; $s_4 = s_1 - m_2$; $x_0 = m_0$; $x_1 = s_2 + s_3$; $x_2 = s_4 + s_5$; $x_3 = s_4 - s_5$; $x_4 = s_2 - s_3$.

Stages (a) and (b) together, requiring only additions, form the "pre-weave" part of the algorithm; similarly, stages (d) and (e) together form the "post-weave" part. For purposes of actually *coding* this algorithm, it is the above description which is needed; but for deriving the algorithm for composite N, a matrix description, as in Eqs. (8)–(11), is more helpful. If we define the vectors

$$\mathbf{x} = (x_0, x_1, x_2, x_3, x_4)^{\mathrm{T}}$$
$$\mathbf{z} = (z_0, z_1, z_2, z_3, z_4)^{\mathrm{T}}$$

then the transform (26) becomes

$$\mathbf{x} = W_5 \mathbf{z}.$$

The algorithm above then becomes equivalent to a factorization of the matrix W_5, as in Eq. (7):

$$W_5 = B_5 M_5 A_5.$$

Defining the temporary vectors

$$\mathbf{t} = (t_0, t_1, t_2, t_3, t_4, t_5)^{\mathrm{T}}$$
$$\mathbf{m} = (m_0, m_1, m_2, m_3, m_4, m_5)^{\mathrm{T}}$$

the algorithm becomes

$$\mathbf{t} = A_5 \mathbf{z}; \qquad \mathbf{m} = M_5 \mathbf{t}; \qquad \mathbf{x} = B_5 \mathbf{m}.$$

Following through the operations in the algorithmic description,

$$A_5 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & -1 & 1 & -1 \end{bmatrix},$$

$M_5 = \mathrm{diag}(1, c_1, c_2, ic_3, ic_4, ic_5)$,

$$B_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & -1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 \end{bmatrix}.$$

This completes the matrix specification of Winograd's algorithm as used in Section 3.

For the split nesting algorithm of Section 4, we use a factorization of the form

$$W_5 = F_5 E_5 M_5 D_5 C_5$$

such that C_5 and F_5 are square, while E_5 and D_5 contain as few additions as possible. It is easily seen that at the end of stage (a) above, we have computed the 5-component vector $(t_0, t_1, t_2, t_3, t_4)^{\mathrm{T}}$ and that the original vector **z** is no longer needed, while only one further complex addition is required to compute t_5 and complete the vector **t**. Hence C_5 is obtained by deleting the last row of A_5, and corresponds to stage (a). D_5 corresponds to stage (b) and is defined by

$$D_5 = \left[\begin{array}{ccccc} \multicolumn{5}{c}{I_5} \\ \hline 0 & 0 & 0 & 1 & -1 \end{array}\right],$$

where I_5 is the identity matrix of order 5.

In the post-weave stage, it is again easily seen that after stage (d), only the 5-component vector $(m_0, m_1, m_2, s_3, s_5)^{\mathrm{T}}$ is needed for the remainder of the computation. Our "design criteria" are thus satisfied by defining

$$E_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

which corresponds to stage (d), leaving

$$F_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 0 & -1 \\ 1 & 1 & 1 & -1 & 0 \end{bmatrix},$$

corresponding to stage (e).

For the partial nesting algorithm of Section 5, we use a factorization of the form

$$W_5 = B_5' M_5' A_5', \tag{27}$$

where all the matrices are square. To obtain this form, the algorithm requires certain modifications; we first express s_3 and s_5 (from stage (d)) in terms of quantities computed in stage (a). Thus

$$\begin{aligned} s_3 = m_5 + m_4 &= i\{\sin\theta(t_3 - t_4) + (\sin\theta + \sin 2\theta)\, t_4\} \\ &= i\{\sin\theta \cdot t_3 + \sin 2\theta \cdot t_4\} \end{aligned}$$

while

$$\begin{aligned} s_5 = m_5 - m_3 &= i\{\sin\theta(t_3 - t_4) - (\sin\theta - \sin 2\theta)\, t_3\} \\ &= i\{\sin 2\theta \cdot t_3 - \sin\theta \cdot t_4\}. \end{aligned}$$

A modified algorithm is thus obtained by keeping stage (a), deleting stage (b), and replacing stage (c) by:

(c)′ $m_0 = t_0$; $m_1 = c_1 \cdot t_1$; $m_2 = c_2 \cdot t_2$; $m_3' = i \sin\theta \cdot t_3$; $m_4' = i \sin\theta \cdot t_4$

Stage (d) becomes:

(d)′ $s_3 = m_3' + \alpha m_4'$; $s_5 = \alpha m_3' - m_4'$,

where $\alpha = \sin 2\theta / \sin\theta$, and stage (e) remains the same. The resulting algorithm, derived above from Winograd's formulation, saves one complex addition at the cost of one more (real $\times$ complex) multiplication. Thus the operation count is the same as for the "minimum-add" algorithm for $n = 5$, and indeed it could have been derived easily from the "triadic" form of the minimum-add algorithm presented in [10] as being suitable for implementation on the Cyber 205.

To complete the factorization of Eq. (27), A_5' corresponds to stage (a) and is the same as C_5 of the split nested algorithm,

$$M_5' = \operatorname{diag}(1, c_1, c_2, i\sin\theta, i\sin\theta)$$

and

$$B'_5 = F_5 \cdot \left[\begin{array}{c|cc} I_3 & 0 & \\ \hline 0 & 1 & \alpha \\ & \alpha & -1 \end{array}\right],$$

where F_5 was defined earlier for the split nested algorithm.

In the above, the details for $n = 5$ have been explicitly presented in order to demonstrate the principles involved, and to illustrate the relationship between the small-n DFT algorithms and the corresponding matrix representations used in deriving the various algorithms for composite N. A similar exercise could be carried out for the other values of n, but for the purposes of this paper it is (fortunately) not necessary to find the explicit forms of all the matrices involved; all the information needed can be extracted fairly easily by inspecting the computational structures of the DFT-modules in algorithmic form.

Acknowledgments

The author thanks Andrew Staniforth for his helpful comments on this paper and Diane Lespérance for typing the manuscript.

References

1. C. S. Burrus and P. W. Eschenbacher, *IEEE Trans. Acoust. Speech Signal Process.* **29**, 806 (1981).
2. J. W. Cooley and J. W. Tukey, *Math. Comput.* **19**, 297 (1965).
3. C. de Boor, *SIAM J. Sci. Stat. Comput.* **1**, 173 (1980).
4. M. Edwards, *Cray Channels* (Spring 1987), p. 22.
5. I. J. Good, *J. Roy. Statist. Soc. Ser. B* **20**, 361 (1958).
6. H. W. Johnson and C. S. Burrus, *IEEE Trans. Acoust. Speech Signal Process.* **31**, 378 (1983).
7. D. P. Kolba and T. W. Parks, *IEEE Trans. Acoust. Speech Signal Process.* **25**, 281 (1987).
8. H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*, 2nd ed. (Springer-Verlag, Berlin, 1982).
9. C. Temperton, "Fast Fourier Transforms and Poisson-solvers on Cray-1," in *Supercomputers*, edited by C. R. Jesshope and R. W. Hockney (Infotech International Ltd., Maidenhead, UK, 1979), p. 359.
10. C. Temperton, *J. Comput. Phys.* **52**, 1 (1983).
11. C. Temperton, *J. Comput. Phys.* **52**, 198 (1983).
12. C. Temperton, *J. Comput. Phys.* **52**, 340 (1983).
13. C. Temperton, "Fast Fourier Transforms on the Cyber 205," in *High-Speed Computation*, edited by J. S. Kowalik (Springer-Verlag, Berlin, 1984), p. 403.
14. C. Temperton, *J. Comput. Phys.* **58**, 283 (1985).
15. C. Temperton, *Parallel Comput.* **6**, 99 (1988).
16. C. Temperton, *J. Comput. Phys.* **75**, 190 (1988).
17. C. Temperton, *J. Comput. Phys.* **75**, 199 (1988).
18. S. Winograd, *Math. Comput.* **32**, 175 (1978).