MATHEMATICS OF COMPUTATION
VOLUME 60, NUMBER 201
JANUARY 1993, PAGES 347–361

# MODIFIED FFTs FOR FUSED MULTIPLY-ADD ARCHITECTURES

ELLIOT LINZER AND EPHRAIM FEIG

ABSTRACT. We introduce fast Fourier transform algorithms (FFTs) designed for fused multiply-add architectures. We show how to compute a complex discrete Fourier transform (DFT) of length $n = 2^m$ with $\frac{8}{3}nm - \frac{16}{9}n + 2 - \frac{2}{9}(-1)^m$ real multiply-adds. For real input, this algorithm uses $\frac{4}{3}nm - \frac{17}{9}n + 3 - \frac{1}{9}(-1)^m$ real multiply-adds. We also describe efficient multidimensional FFTs. These algorithms can be used to compute the DFT of an $n \times n$ array of complex data using $\frac{14}{3}n^2m - \frac{4}{3}n^2 - \frac{4}{9}n(-1)^m + \frac{16}{9}$ real multiply-adds. For each problem studied, the number of multiply-adds that our algorithms use is a record upper bound for the number required.

## 1. INTRODUCTION

The discrete Fourier transform of length $n$ (DFT($n$)) of a complex vector $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})^t$ is $\mathbf{X} = (X_0, X_1, \ldots, X_{n-1})^t$, where

$$X_k = \sum_{r=0}^{n-1} W_n^{kr} x_r,$$

$$W_n^k = \exp(-i2\pi k/n) \quad \text{and} \quad i = \sqrt{-1}.$$

Since the publication by Cooley and Tukey in 1965 of a fast algorithm for computing the DFT [3], many researchers have studied efficient algorithms for computing the DFT [6, 2, 12, 17, 4, 14]. These algorithms have come to be collectively known as fast Fourier transforms (FFTs).

When determining the efficiency of an algorithm, one must have a model that specifies the cost of various arithmetic operations (and possibly of data movement as well). Winograd and others studied the DFT from a multiplicative complexity point of view, developed algorithms that used only $O(n)$ multiplications, and found lower bounds on the number of multiplications required to compute DFTs [17, 18, 19, 1]. Other researchers have sought out algorithms with reduced numbers of both additions and multiplications [16, 4, 14].

Many of today's advanced workstations and signal processors are designed for efficient fused multiply-add operations [10]. Here, the primitive operation is

Received by the editor December 20, 1990 and, in revised form, November 11, 1991 and February 11, 1992.

1991 *Mathematics Subject Classification.* Primary 65T10.

*Key words and phrases.* Fast Fourier transform, split-radix FFT, number of multiply-adds.


0025-5718/93 $1.00 + $.25 per page

a multiply-add, which computes $\pm a \pm bc$, where $a$, $b$, and $c$ are real numbers. On these architectures, a multiply-add, a simple addition, or a simple multiplication each requires one machine cycle. We will call any of these computations an $m/a-op$, and assign unit cost to each.

In this paper, we introduce FFT algorithms that use fewer $m/a-ops$ than any algorithm previously reported. We study the case when $n$ is a power of 2 in detail, but our techniques can be used to obtain efficient algorithms for other input lengths. We have already used these same techniques to obtain algorithms for computing discrete cosine transforms with fewer $m/a-ops$ than traditional algorithms [8]. We are aware of previous work by Ali Mechentel on FFTs optimized for multiply-add architectures [9].

Our algorithms are closely related to the Cooley-Tukey and split-radix algorithms, which we discuss in the next section. In §3, we describe a method for converting Cooley-Tukey and split-radix algorithms into algorithms that use as many $m/a-ops$ as the number of real additions used in the original algorithm, assuming the original algorithm uses a 2-addition, 4-multiplication algorithm for computing complex multiplications. Because it is common in the literature to calculate the number of additions used in an algorithm by assuming that a 3-addition, 3-multiplication algorithm is used for complex multiplications, the number of $m/a-ops$ used in the new algorithms will be smaller than some of the published numbers of real additions used in the original algorithms.

A feature of the algorithms that we introduce is that every multiplication appears as a genuine multiply-add. The Rader-Brenner FFT [12] shares this feature, but still uses more $m/a-ops$ than the most efficient version of our algorithms. The multiplicative constants used in our FFTs are all smaller than one and, for large $n$, the smallest multiplicative constants that our algorithms use are slightly larger than the smallest constants used in regular FFTs. By contrast, the Rader-Brenner FFT uses multiplicative constants that are larger than one, and, for large $n$, the multiplicative constants are so large as to cause numerical instabilities [12]. In §4, we extend our results and derive similar new algorithms with identical operation counts and give bounds on the magnitude of the coefficients our algorithms use.

In §5 we show how the ideas introduced in §3 can be used to develop efficient algorithms for DFTs with real inputs and for multidimensional DFTs.

## 2. Cooley-Tukey FFTs

If $n$ is a composite number, then index transforms can be used to simplify the computation of the DFT [3, 2]. For $n = n_1 n_2$, we can introduce new indices $k_1$, $k_2$, $r_1$, and $r_2$ $(0 \le k_j,\ r_j < n_j)$ by $k = k_1 + n_1 k_2$, $r = r_2 + n_2 r_1$. Then we can write

$$X_{k_1+n_1k_2} = \sum_{r_2=0}^{n_2-1} W_{n_2}^{r_2k_2} \left[ W_n^{r_2k_1} \left( \sum_{r_1=0}^{n_1-1} W_{n_1}^{r_1k_1} x_{r_2+n_2r_1} \right) \right]. \tag{1}$$

Equation (1) allows one to compute a DFT($n$) by first computing $n_2$DFT($n_1$)s, performing $n$ complex multiplications (by $W_n^{r_2k_1}$) and then computing

$n_1$DFT($n_2$)s. The constants $W_n^{r_2 k_1}$ are called *twiddle factors*. If $n_1$ and $n_2$ are coprime, then decompositions that are more efficient than (1) can be used [2]. FFT algorithms based on (1) are generally called Cooley-Tukey FFTs.

If $n$ is highly composite, then (1) can be applied recursively. If $n = q^m$ and we always take $n_1 = q$, then the resulting algorithm is called a *radix-q decimation-in-frequency* (DIF) algorithm. If we always take $n_2 = q$, then the algorithm is called a radix-$q$ *decimation-in-time* (DIT) algorithm.

For the remainder of the paper we consider the case $n = 2^m$ for some integer $m > 1$. The radix-2 DIF equations are

$$
\begin{aligned}
X_{2k} &= \sum_{r=0}^{(n/2)-1} W_{n/2}^{kr}[W_n^0(x_r + x_{r+n/2})], \\
X_{2k+1} &= \sum_{r=0}^{(n/2)-1} W_{n/2}^{kr}[W_n^r(x_r - x_{r+n/2})], \qquad 0 \le k \le \frac{n}{2} - 1.
\end{aligned} \tag{2}
$$

Thus, a DFT($n$) can be computed with $n$ complex additions (by additions, we mean additions or subtractions), $n$ twiddle factor multiplications, and two DFT($n/2$)s. If $k$ is a multiple of $n/4$, then $W_n^k$ is called a *trivial* twiddle factor. Multiplication by a trivial twiddle factor can be accomplished by (at most) changing the signs and permuting the real and imaginary parts of $b$. For the radix-2 algorithm, $n/2 + 2$ of the $n$ twiddle factors are trivial.

If $m > 2$, a radix-4 decomposition can be used. The radix-4 DIF equations are

$$
\begin{aligned}
X_{4k} &= \sum_{r=0}^{(n/4)-1} W_{n/4}^{kr}[W_n^0(x_r + x_{r+n/4} + x_{r+n/2} + x_{r+3n/4})], \\
X_{4k+1} &= \sum_{r=0}^{(n/4)-1} W_{n/4}^{kr}[W_n^r(x_r - ix_{r+n/4} - x_{r+n/2} + ix_{r+3n/4})], \\
X_{4k+2} &= \sum_{r=0}^{(n/4)-1} W_{n/4}^{kr}[W_n^{2r}(x_r - x_{r+n/4} + x_{r+n/2} - x_{r+3n/4})], \\
X_{4k+3} &= \sum_{r=0}^{(n/4)-1} W_{n/4}^{kr}[W_n^{3r}(x_r + ix_{r+n/4} - x_{r+n/2} - ix_{r+3n/4})], \\
&\qquad\qquad 0 \le k \le \frac{n}{4} - 1.
\end{aligned}
$$

If the 4-point DFTs are computed with a radix-2 algorithm, then, for $n \ge 16$, the radix-4 decomposition is more efficient than the radix-2 decomposition.

The DIF *split-radix* FFT computes the components of $\mathbf{X}$ with even indices using a radix-2 algorithm, and the components of $\mathbf{X}$ with odd indices using a radix-4 algorithm [4, 14]. The recursive equations are

$$
\begin{aligned}
X_{2k} &= \sum_{r=0}^{(n/2)-1} W_{n/2}^{kr}[W_n^0(x_r + x_{r+n/2})], \qquad 0 \le k \le \frac{n}{2} - 1, \\
X_{4k+1} &= \sum_{r=0}^{(n/4)-1} W_{n/4}^{kr}[W_n^r(x_r - ix_{r+n/4} - x_{r+n/2} + ix_{r+3n/4})], \\
X_{4k+3} &= \sum_{r=0}^{(n/4)-1} W_{n/4}^{kr}[W_n^{3r}(x_r + ix_{r+n/4} - x_{r+n/2} - ix_{r+3n/4})], \\
& \qquad\qquad 0 \le k \le \frac{n}{4} - 1.
\end{aligned} \tag{3}
$$

Thus, a DFT($n$) can be computed with $3n/2$ complex additions, $(n/2) - 2$ complex multiplications, a DFT($n/2$) and two DFT($n/4$)s. For $n \ge 32$ the split-radix decomposition is more efficient than either the radix-2 or radix-4 Cooley-Tukey FFTs. Among all reported algorithms for computing DFT($2^m$), the split-radix FFT uses the fewest real additions and has the smallest total number of real additions and real multiplications.

From (3), we see that the number of complex additions that the split-radix algorithm uses, $\alpha_c(n)$, satisfies the recursion

$$\alpha_c(n) = 3n/2 + \alpha_c(n/2) + 2\alpha_c(n/4).$$

The number of nontrivial complex multiplications, $\mu_c(n)$, satisfies

$$\mu_c(n) = n/2 - 2 + \mu_c(n/2) + 2\mu_c(n/4).$$

With the initial conditions

$$\alpha_c(2) = 2, \qquad \alpha_c(4) = 8, \qquad \mu_c(2) = \mu_c(4) = 0$$

we obtain

$$\alpha_c(n) = nm \tag{4}$$

and

$$\mu_c(n) = \tfrac{1}{3}nm - \tfrac{8}{9}n + 1 - \tfrac{1}{9}(-1)^m. \tag{5}$$

Each nontrivial complex multiplication requires four $m/a - ops$ (two multiplications and two multiply-adds). A complex sum (or difference) requires two $m/a - ops$. From (4) and (5), the total number of $m/a - ops$ used by a split-radix FFT, $\pi_{sr}(n)$, is equal to

$$\pi_{sr}(n) = \tfrac{10}{3}nm - \tfrac{32}{9}n + 4 - \tfrac{4}{9}(-1)^m. \tag{6}$$

A flow graph interpretation of the split-radix algorithm for $n = 32$ is shown in Figure 1.

Any Cooley-Tukey FFT or the split-radix FFT can be thought of as a factorization of the DFT($n$)-matrix, $\mathbf{F}^{(n)}$, defined by $\mathbf{F}^{(n)} = (W_n^{jk})_{j,k=0}^{n-1}$, into a sequence of sparse matrices. If $n = 2^m$ and the DFT($q$)s in a radix-$q$ FFT are themselves computed with (1), then a DIF algorithm can be written as the factorization

$$\mathbf{F}^{(n)} = \mathbf{P}^{(n)}\mathbf{A}^{m,m-1}\mathbf{D}^{m,m-1}\cdots\mathbf{A}^{m,1}\mathbf{D}^{m,1}\mathbf{A}^{m,0}. \tag{7}$$

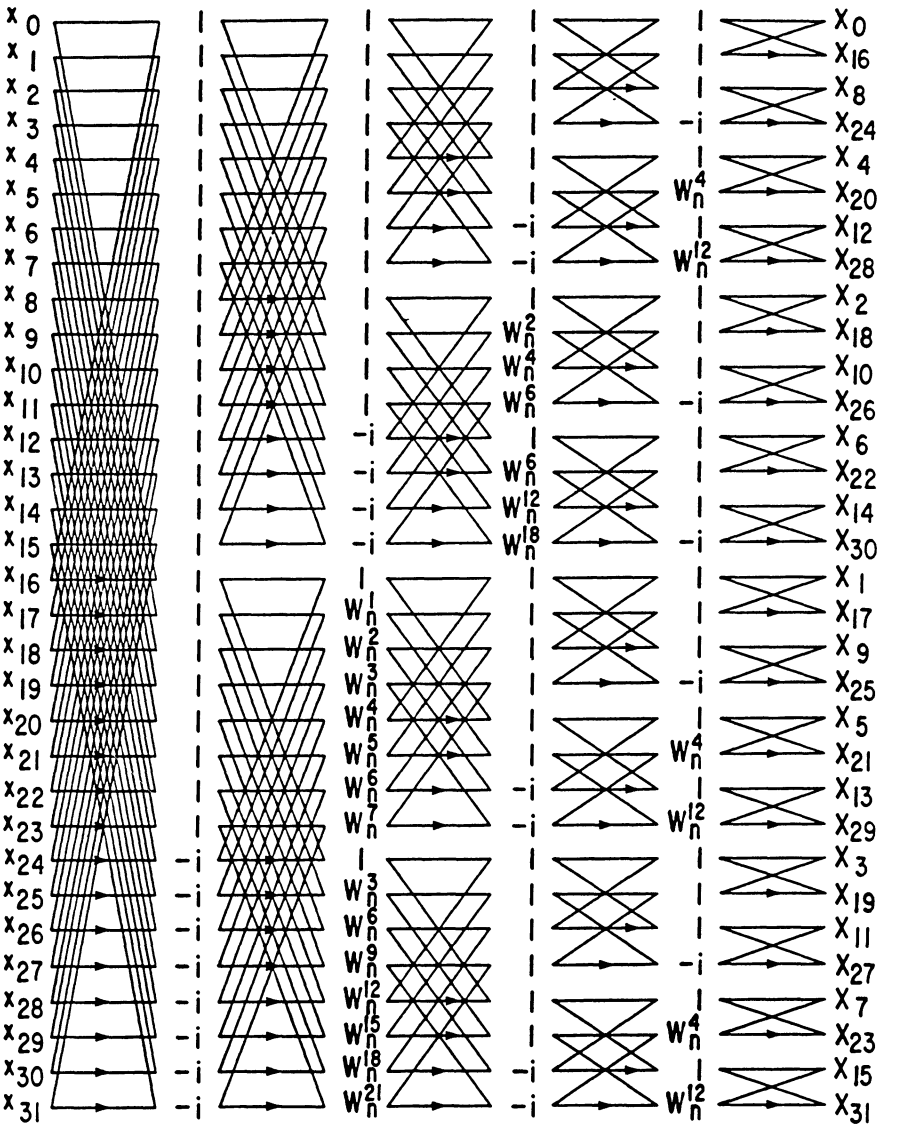


FIGURE 1. Flow graph for decimation-in-frequency split-radix FFT for computing DFT(32). Arrows indicate multiplication by $-1$.

The matrix $\mathbf{P}^{(n)}$ is the "bit-reversal" matrix; it is a symmetric permutation matrix. (See [15, p. 98] for a definition.) The $n \times n$ matrices $\mathbf{A}^{m,l}$ $(l = 0, \ldots, m-1)$ are block-diagonal addition matrices defined by

$$\mathbf{A}^{m,l} = \operatorname{diag}(\mathbf{N}^{(2^{m-1})}, \ldots, \mathbf{N}^{(2^{m-1})}), \tag{8}$$

where

$$\mathbf{N}^{2k} = \begin{pmatrix} \mathbf{I}^{(k)} & \mathbf{I}^{(k)} \\ \mathbf{I}^{(k)} & \mathbf{I}^{(k)} \end{pmatrix}, \qquad \mathbf{I}^{(k)} = (\delta_{j,p})_{j,p=0}^{k-1},$$

and $\delta_{j,p}$ is the Kronecker symbol. The $n\times n$ matrices $\mathbf{D}^{m,l}$ $(l=1,\ldots,m-1)$ are defined by

$$\mathbf{D}^{m,l}=\operatorname{diag}(W_n^{K(m,l,0)},W_n^{K(m,l,1)},\ldots,W_n^{K(m,l,n-1)}).$$

The particular algorithm used determines (and is determined by) the integer function $K(m,l,j)$. For example, for the radix-2 algorithm, the function $K_{r2}(m,l,j)$, defined by

$$K_{r2}(m,l,j)=\begin{cases}0 & \text{if } (j \bmod 2^{m-l+1})<2^{m-l},\\ 2^{l-1}(j \bmod 2^{m-l}) & \text{otherwise},\end{cases} \tag{9}$$

is used for $K(m,l,j)$. For the split-radix FFT, we use the function $K_{sr}(m,l,j)$, defined by

$$K_{sr}(m,l,j)=\begin{cases}1 & \text{if } r(l,r\oslash 2^{m-l+1})=0\\ & \quad\text{and } (j \bmod 2^{m-l+1})<\frac{3}{4}2^{m-l+1},\\ n/4 & \text{if } r(l,r\oslash 2^{m-l+1})=0\\ & \quad\text{and } (j \bmod 2^{m-l+1})\geq\frac{3}{4}2^{m-l+1},\\ 2^{l-1}(j \bmod 2^{m-l}) & \text{if } r(l,r\oslash 2^{m-l+1})=1\\ & \quad\text{and } (j \bmod 2^{m-l+1})<2^{m-l},\\ 3\cdot 2^{l-1}(j \bmod 2^{m-l}) & \text{if } r(l,r\oslash 2^{m-l+1})=1\\ & \quad\text{and } (j \bmod 2^{m-l+1})\geq 2^{m-l},\end{cases} \tag{10}$$

where $\oslash$ is integer division without remainder $(j\oslash k=(j-(j \bmod k))/k)$ and the function $r(l,p)$ $(l=1,2,\ldots,m-1$ and $p=0,1,\ldots,2^{l-1}-1)$ is defined by the recursion

$$r(l,0)=0,\qquad r(l,p)=\begin{cases}0 & \text{if } r(l-1,\ p\oslash 2)=1 \text{ or } (p \bmod 2)=0,\\ 1 & \text{otherwise}.\end{cases}$$

(When implementing the split-radix FFT, Duhamel [4] suggests precomputing and storing a representation of the function $r(l,p)$.)

Each time that (1) is used to break a DFT into smaller DFTs, the 0th twiddle factor is simply $W_n^0=1$. Thus, we will always have

$$K(m,l,h2^{m-l})=0,\qquad h=0,1,\ldots,2^l-1. \tag{11}$$

Multiplication of a vector by $\mathbf{A}^{m,l}$ requires $n$ complex additions. The complexity of multiplying a vector by $\mathbf{D}^{m,l}$ depends on the particular matrix $\mathbf{D}^{m,l}$. Thus, all of the Cooley-Tukey and related algorithms use exactly $nm$ complex additions but differ in the number nontrivial complex multiplications used.

## 3. A NEW ALGORITHM

Cooley-Tukey FFTs and the split-radix FFT use more real additions than real multiplications. It is not possible to schedule all of the multiplications in these FFTs so that they appear as genuine multiply-adds; some of the real multiplications used in the twiddle factor multiplications appear as simple real multiplications. Therefore, the number of $m/a-ops$ used will exceed the number of real additions.

In this section, we present a method for converting Cooley-Tukey and split-radix algorithms into algorithms with fewer $m/a-ops$. In the algorithms

that we describe, the number of real multiplications will be greater than the number of real multiplications used in the original Cooley-Tukey or split radix algorithm, but the number of real additions will remain the same. The improved efficiency of these algorithms results from the fact that *every multiplication will appear as a multiply-add*. Thus the total number of $m/a-ops$ used will equal the number of real additions used in the original Cooley-Tukey or split-radix algorithm.

Consider the multiplication of a twiddle factor, $W_n^k$, by a complex number. If $W_n^k$ is not a trivial twiddle factor, then this multiplication done in the obvious way uses two multiplies and two multiply-adds, for a total of four $m/a-ops$. Now let us define

$$\alpha_{n,k} = \max(|\cos(2\pi k/n)|,\ |\sin(2\pi k/n)|) \tag{12}$$

and

$$\widetilde{W_n^k} = W_n^k/\alpha_{n,k}. \tag{13}$$

Either the real or imaginary part of $\widetilde{W_n^K}$ has absolute value 1, and therefore we can multiply a complex number by $\widetilde{W_n^k}$ using only two $m/a-ops$.

We will begin with the modified radix-2 algorithm for computing DFT$(n)$. Our derivation is based on (2). Using (13), we can write

$$\begin{aligned} X_{2k} &= \sum_{r=0}^{n/2-1} (x_r + x_{r+n/2}) W_{n/2}^{kr}, \\ X_{2k+1} &= \sum_{r=0}^{n/2-1} [(x_r - x_{r+n/2})\widetilde{W_n^r}]\alpha_{n,r} W_{n/2}^{kr}. \end{aligned} \tag{14}$$

Thus, the even outputs are computed with $n/2$ complex additions and one DFT$(n)$, whereas the odd outputs are computed with $n/2$ complex additions, multiplications by $n/2-2$ nontrivial scaled twiddle factors, $\widetilde{W_n^r}$, and a problem of size $n/2$. The problem of size $n/2$ does not use the DFT$(n/2)$ kernel, $W_{n/2}^{kr}$, but rather a scaled DFT$(n/2)$ kernel, $\alpha_{n,r}W_{n/2}^{kr}$, where $\alpha_{n,r}$ is defined through (12). This second type of problem is a particular instance of a problem of the type

$$Y_k = \sum_{r=0}^{p-1} y_r \beta_{p,r} W_p^{kr} \qquad (k = 0, \ldots, p-1), \tag{15}$$

where $\beta_{p,r}$ are real numbers that satisfy $0 < \beta_{p,r} \le 1$ and $\beta_{p,p/2} = \beta_{p,0} = 1$. We compute the DFT$(n/2)$ by recursively applying the decomposition (14), and we compute (15) via

$$\begin{aligned} Y_{2k} &= \sum_{r=0}^{p/2-1} \left(\frac{\beta_{p,r}}{\beta_{p/2,r}} y_r + \frac{\beta_{p,r+p/2}}{\beta_{p/2,r}} y_{r+p/2}\right) \beta_{p/2,r} W_{p/2}^{kr}, \\ Y_{2k+1} &= \sum_{r=0}^{p/2-1} \left[\left(\frac{\beta_{p,r}}{\beta_{p/2,r}} y_r - \frac{\beta_{p,r+p/2}}{\beta_{p/2,r}} y_{r+p/2}\right) \widetilde{W_p^r}\right] \beta'_{p/2,r} W_{p/2}^{kr}, \\ &\qquad\qquad k = 0, \ldots, p/2-1, \end{aligned} \tag{16}$$

where $\beta_{p/2,r} = \max(\beta_{p,r}, \beta_{p,r+p/2})$ and $\beta'_{p/2,r} = \beta_{p/2,r}\alpha_{p,r}$. Note that either

$$\frac{\beta_{p,r}}{\beta_{p/2,r}} = 1 \quad \text{or} \quad \frac{\beta_{p,r+p/2}}{\beta_{p/2,r}} = 1.$$

We can therefore perform one computation of the form given by (15) by performing $p$ *real/complex multiply-adds* (that is, computations of the form $\pm a \pm bc$, where $a, c \in \mathscr{C}$ and $b \in \mathfrak{R}$), $p/2 - 2$ multiplications by scaled twiddle factors, $\widetilde{W_p^r}$, and two half-size problems of the form given by (15). A real/complex multiply-add takes only as many $m/a - ops$ (i.e., two) as does a complex addition. Step by step, the new radix-2 DIF algorithm is obtained. Because

$$\beta_{p,p/2} = \beta_{p,0} = 1, \tag{17}$$

the last stage of the new algorithm involves only two-point DFTs. Therefore, for the modified radix-2 algorithm we have replaced all of the multiplications by twiddle factors with multiplications by scaled twiddle factors and some complex additions by real/complex multiply-adds. Thus the total number of $m/a - ops$ used is equal the number of real additions used in the original radix-2 DIF algorithm.

This same procedure can be used to derive the modified radix-4 and split-radix algorithms. But rather than derive these algorithms directly, we will show how the modified algorithms can be obtained from (7). We will then at once obtain modified algorithms for any size radix. Also, the approach will be applicable to any fast transform algorithm based on a factorization similar to (7) (e.g., our work on discrete cosine transforms [8]).

For a matrix $\mathbf{S} \in \mathscr{C}^{n\times n}$ and an integer $j \in \{0, \ldots, n-1\}$, define

$$q(\mathbf{S}, j) = \max_{0\le k\le n-1}\{\max(|\operatorname{Re}(\mathbf{S}_{j,k})|, |\operatorname{Im}(\mathbf{S}_{j,k})|)\}. \tag{18}$$

Define the real diagonal matrix $\mathbf{Q}(\mathbf{S})$ by

$$\mathbf{Q}(\mathbf{S}) = (q(\mathbf{S}, j)\delta_{j,k})_{j,k=0}^{n-1}. \tag{19}$$

If $\mathbf{S}$ has no zero rows, then $\mathbf{Q}(\mathbf{S})$ is invertible, and we can define

$$\mathbf{R}(\mathbf{S}) = (\mathbf{Q}(\mathbf{S}))^{-1}\mathbf{S}. \tag{20}$$

The idea behind the new algorithms is that instead of multiplying by a sparse matrix $\mathbf{S}$, we multiply by the sparse matrix $\mathbf{R}(\mathbf{S})$. The multiplication by $\mathbf{Q}(\mathbf{S})$ is then absorbed into future computations. If the original algorithm is written as the matrix factorization $\mathbf{F} = \mathbf{M}^{(p)}\mathbf{M}^{(p-1)}\cdots\mathbf{M}^{(0)}$, then the new algorithm can be represented as the matrix factorization

$$\mathbf{F} = \mathbf{Q}(\mathbf{L}^{(p)})\mathscr{M}^{(p)}\mathscr{M}^{(p-1)}\cdots\mathscr{M}^{(0)}, \tag{21}$$

where

$$\mathbf{L}^{(0)} = \mathbf{M}^{(0)}, \qquad \mathbf{L}^{(j)} = \mathbf{M}^{(j)}\mathbf{Q}(\mathbf{L}^{(j-1)}), \quad 1 \le j \le p, \tag{22}$$

and

$$\mathscr{M}^{(j)} = \mathbf{R}(\mathbf{L}^{(j)}), \qquad 0 \le j \le p. \tag{23}$$

Because $\mathbf{Q}(\mathbf{L}^{(j-1)})$ is a real diagonal matrix, $\mathbf{L}^{(j)}$ and $\mathbf{M}^{(j)}$ have the same "structure" in the following sense: an element of $\mathbf{L}^{(j)}$ is real (or imaginary, or

zero) if and only if the corresponding element of $\mathbf{M}^{(j)}$ is real (or imaginary, or zero). Because $\mathscr{M}^{(j)} = \mathbf{R}(\mathbf{L}^{(j)})$, $\mathscr{M}^{(j)}$ also has the same structure as $\mathbf{M}^{(j)}$. The number of additions required to compute the product of a matrix and a vector (in the obvious manner) depends only on the structure of the matrix. If $\mathbf{S}$ is a real matrix or a diagonal matrix, then computing the product of $\mathbf{R}(\mathbf{S})$ and a vector requires only as many $m/a-ops$ as the number of additions required to compute the product of $\mathbf{S}$ and a vector. If we rewrite (7) in the form of (21), an algorithm based on the new factorization will require as many $m/a-ops$ as the number of additions required by the original algorithm plus up to $2n$ $m/a-ops$ for multiplication by the real diagonal matrix $\mathbf{Q}(\mathbf{L}^{(p)})$.

To be specific, let

$$\mathbf{E}^{m,l} = \mathbf{R}(\mathbf{D}^{m,l}), \qquad l = 1, 2, \ldots, m-1, \tag{24}$$

$$\mathbf{G}^{m,1} = \mathbf{Q}(\mathbf{A}^{m,1}\mathbf{Q}(\mathbf{D}^{m,1})), \tag{25}$$

$$\mathbf{G}^{m,l} = \mathbf{Q}(\mathbf{A}^{m,l}\mathbf{Q}(\mathbf{D}^{m,l})\mathbf{G}^{m,l-1}), \qquad l = 2, 3, \ldots, m-1, \tag{26}$$

$$\mathbf{B}^{m,1} = \mathbf{R}(\mathbf{A}^{m,1}\mathbf{Q}(\mathbf{D}^{m,1})), \tag{27}$$

and

$$\mathbf{B}^{m,l} = \mathbf{R}(\mathbf{A}^{m,l}\mathbf{Q}(\mathbf{D}^{m,l})\mathbf{G}^{m,l-1}), \qquad l = 2, 3, \ldots, m-1. \tag{28}$$

Then

$$\mathbf{F}^{(n)} = \mathbf{P}^{(n)}\mathbf{G}^{m,m-1}\mathbf{B}^{m,m-1}\mathbf{E}^{m,m-1}\mathbf{B}^{m,m-2}\mathbf{E}^{m,m-2}\cdots\mathbf{B}^{m,1}\mathbf{E}^{m,1}\mathbf{A}^{m,0}. \tag{29}$$

The matrix $\mathbf{G}^{m,m-1}$ in (29) plays the role of $\mathbf{Q}(\mathbf{L}^{(p)})$ in (21). We will soon see that $\mathbf{G}^{m,m-1} = \mathbf{I}^{(n)}$, whence we can conclude that the new algorithms defined by (29) use only as many $m/a-ops$ as the number of additions used by the original Cooley-Tukey or split-radix algorithm. First, we demonstrate some properties about the matrices $\mathbf{G}^{m,l}$.

**Lemma 1.** *The diagonal elements of (the diagonal matrices)* $\mathbf{G}^{m,l}$ *are in the interval* $(0, 1]$.

*Proof.* The diagonal elements of $\mathbf{Q}(\mathbf{D}^{m,l})$ are in $(0, 1]$. Thus (8), (19), (25), (26), and induction can be used to establish the lemma. □

**Lemma 2.** *For* $l = 0, 1, \ldots, m-1$ *and* $h = 0, 1, \ldots, 2^{l+1}-1$, *the* $(h2^{m-l-1})th$ *diagonal element of* $\mathbf{G}^{m,l}$ *is equal to* 1.

*Proof.* Proof is by induction on $l$. Equations (8), (11), (19), and (25) can be used to show that the theorem holds for $l = 1$. From Lemma 1, we see that the diagonal elements of (the diagonal matrices) $\mathbf{Q}(\mathbf{D}^{m,l})\mathbf{G}^{m,l}$ are bounded in absolute value by one. Assume the lemma holds for $l-1$. Then (8), (11), (19), and (26) can be used to show that the lemma holds for $l$. □

(Equation (17) is a special case of Lemma 2.) Setting $l = m-1$ in Lemma 2 shows that $\mathbf{G}^{m,m-1} = \mathbf{I}^{(n)}$. We can now rewrite (29) as

$$\mathbf{F}^{(n)} = \mathbf{P}^{(n)}\mathbf{B}^{m,m-1}\mathbf{E}^{m,m-1}\mathbf{B}^{m,m-2}\mathbf{E}^{m,m-2}\cdots\mathbf{B}^{m,1}\mathbf{E}^{m,1}\mathbf{A}^{m,0}. \tag{30}$$

A flow graph interpretation of the split-radix version of this algorithm is shown in Figure 2 (see next page).

A comparison of (7) and (30) shows that we have delivered on the promises made at the beginning of this section. The diagonal elements of $\mathbf{D}^{m,l}$ are the

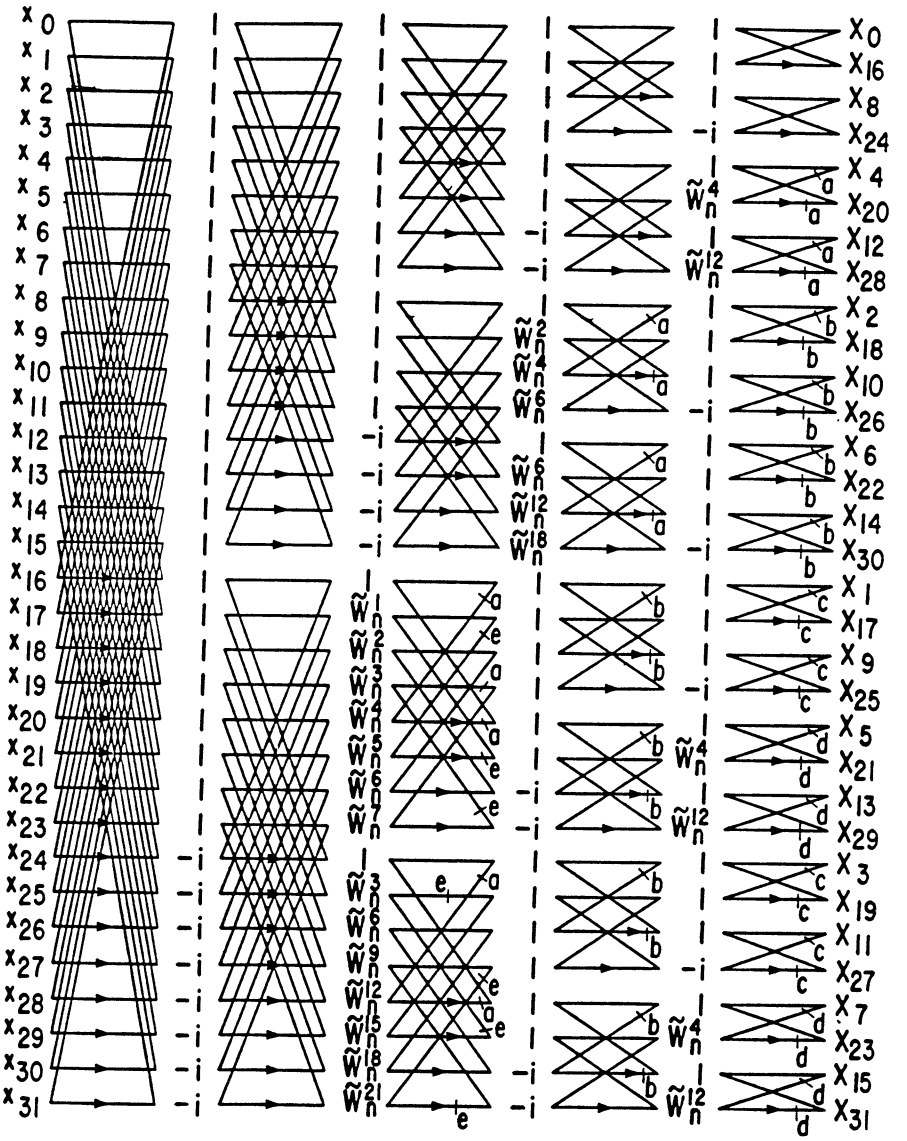

FIGURE 2. Flow graph for the "new" DIF split-radix FFT for DFT(32). The constants $a, b, c, d$, and $e$ are defined by $a = \cos(\pi/4)$, $b = \cos(\pi/8)$, $c = \cos(\pi/16)$, $d = \cos(\pi/16)\cos(\pi/4)$, and $e = \cos(3\pi/16)/\cos(\pi/16)$.

twiddle factors, but the diagonal elements of $\mathbf{E}^{m,l}$ are the scaled twiddle factors (as defined by (13)), so the ordinary real multiplications needed for the twiddle factor multiplications have been eliminated. $\mathbf{B}^{m,l}$ has the same structure as $\mathbf{A}^{m,l}$. All nonzero elements of $\mathbf{A}^{m,l}$ are equal to $\pm 1$, and at least one element of each row of $\mathbf{B}^{m,l}$ is equal to $\pm 1$. Therefore, computing the product of $\mathbf{A}^{m,l}$ or $\mathbf{B}^{m,l}$ and a complex vector requires $2n\ \ m/a - ops$, which is the number of additions used to compute the product of $\mathbf{A}^{m,l}$ and a complex vector.

Equations (4) and (5) yield the following upper bound for the number of $m/a - ops$ required to compute DFT($n$):

**Theorem 1.** *For $n = 2^m$, let $\pi_{\text{new}}(n)$ be the number of $m/a - ops$ that the split-radix version of the algorithm described above uses to compute DFT($n$). Then*

$$\pi_{\text{new}}(n) = \tfrac{8}{3}nm - \tfrac{16}{9}n + 2 - \tfrac{2}{9}(-1)^m. \tag{31}$$

A comparison of (6) and (31) shows that as $n$ tends to infinity the new algorithm uses only 80% of the number of operations used by a regular split-radix algorithm. For finite $n$ the reduction in complexity is always smaller, but it is still significant for reasonably large $n$. For example, when $n = 1024$ the new algorithm uses 25488 $m/a - ops$, which is 83.6% of the 30496 $m/a - ops$ used by a regular split-radix FFT.

## 4. Numerical properties and variations

It is possible to derive algorithms that differ from, but have the same computational complexity as, those presented in the last section. Equation (7) shows the factorization of a symmetric matrix, $\mathbf{F}^{(n)}$, into a sequence of symmetric matrices. By transposing the right-hand side of (7), we obtain the DIT factorization of $\mathbf{F}^{(n)}$. By applying the techniques that took (7) to (30) to the DIT factorization of $\mathbf{F}^{(n)}$, the modified DIT algorithm is obtained. The number of $m/a - ops$ used by the modified DIT algorithm is equal to the number of $m/a - ops$ used by the modified DIF algorithm.

It is instructive to derive the modified radix-$q$ DIT algorithm directly. From (1),

$$\begin{pmatrix} X_k \\ X_{k+n/q} \\ \vdots \\ X_{k+n(q-1)/q} \end{pmatrix} = \mathbf{F}^{(q)}\mathbf{T}^{q,n,k} \begin{pmatrix} \sum_{r=0}^{n/q-1} W_n^{qrk} x_{qr} \\ \sum_{r=0}^{n/q-1} W_n^{qrk} x_{qr+1} \\ \vdots \\ \sum_{r=0}^{n/q-1} W_n^{qrk} x_{qr+q-1} \end{pmatrix}, \qquad 0 \le k \le n/q - 1,$$

where

$$\mathbf{T}^{n,q,k} = \operatorname{diag}(1, W_n^k, \ldots, W_n^{(q-1)k}).$$

Thus, to compute a DFT($n$), we first compute $q$ DFT($n/q$)s and then perform $n/q$ multiplications of the form

$$\mathbf{F}^{(q)}\mathbf{T}^{n,q,k}\mathbf{x}, \tag{32}$$

where $\mathbf{x} \in \mathscr{C}^q$. The computation described by (32) is referred to as a *decimation-in-time butterfly.*

If $q = 2^s$, then we can write

$$\mathbf{F}^{(q)}\mathbf{T}^{n,q,k} = \mathbf{A}^{s,0}\mathbf{D}^{s,1}\mathbf{A}^{s,1}\cdots\mathbf{D}^{s,s-1}\mathbf{A}^{s,s-1}\mathbf{P}^{(q)}\mathbf{T}^{n,q,k}. \tag{33}$$

We can use the techniques of the last section to convert the right-hand side of (33) to a factorization of the form (21). We can use (11) and the fact that $\mathbf{T}_{0,0}^{n,q,k} = 1$ to establish the fact that for this new factorization the matrix that plays the role of $\mathbf{Q}(\mathbf{L}^{(p)})$ in (21) is $\mathbf{I}^{(q)}$. Therefore, we can compute a radix-$q$

DIT butterfly with as many $m/a-ops$ as the number of additions used by a regular DIT butterfly.

The DIT versions of our new algorithms have a simpler structure than the DIF versions. For the DIT algorithms, we first compute a set of smaller DFTs and then perform $O(n)$ $m/a-ops$ to recover the full-size DFT. By contrast, for the DIF versions, after we perform $O(n)$ $m/a-ops$, the smaller subproblems are not all DFTs (although they have the same complexity as DFTs). This simplified structure of the modified DIT algorithms makes it possible to write a looped computer program to compute the DFT$(2^m)$ for any $m$. Implementing the DIF algorithms would require the use of linear code. We give a detailed description of the structure of the DIT versions of our algorithms and discuss issues that affect their efficient implementation in [7].

Other algorithms that use the same number of $m/a-ops$ as those presented in the last section come from noting that the use of the "max" function in (18) is not necessary. Indeed, if instead of the function $q(\mathbf{S}, j)$ we use a function $q'(\mathbf{S}, j)$ that satisfies

$$|q'(\mathbf{S}, j)| = \mathrm{Re}(\mathbf{S}_{j,k}) \text{ or } \mathrm{Im}(\mathbf{S}_{j,k}) \quad \text{for some } k \in \{0, \ldots, n-1\},$$
$$q'(\mathbf{S}, j) \neq 0,$$

and Lemma 2 is satisfied, the algorithms that result will have the same arithmetic complexity as the algorithm described in the last section.

For reasons of numerical accuracy, it is desirable that the dynamic range of the multiplicative constants used in an algorithm be small. Because we use the "max" function in (18), we can prove the following theorem about the dynamic range of the constants used in the modified FFTs.

**Theorem 2.** *All of the real multiplicative constants used in* (30) *have absolute value at most* 1. *For* $n \geq 32$ *the smallest multiplicative constant is larger than the smallest multiplicative constant used in regular Cooley-Tukey or split-radix algorithms.*

*Moreover, for the split-radix version of* (30), *the smallest multiplicative constant used is never smaller than the smallest multiplicative constant used in the original algorithm, and, for* $n \geq 16$, *it is larger than the smallest multiplicative constant used in the original algorithm.*

*Proof.* The claims in the theorem made for the split-radix algorithm for $n < 32$ can be checked directly.

For any matrix $\mathbf{S}$ with nonzero rows, the real and imaginary parts of the elements of the matrix $\mathbf{R}(\mathbf{S})$ have absolute value at most equal to one. Because each of the factors in (30) can be written as $\mathbf{R}(\mathbf{S})$ for some $\mathbf{S}$, it follows that all of the multiplicative constants used in the new algorithms have absolute value at most one.

The smallest multiplicative constant used by a Cooley-Tukey or split-radix algorithm for DFT$(n)$ is $\sin(2\pi/n)$. The smallest nonzero real or imaginary parts of the elements of $\mathbf{E}^{m,l}$ are at least as large as $\sin(2\pi/n)/\cos(2\pi/n) > \sin(2\pi/n)$.

Let $g^{m,l}$ be equal to the smallest diagonal elements of (the real, positive, diagonal matrices) $\mathbf{G}^{m,l}$, and let $b^{m,l}$ be the absolute value of the smallest nonzero element of $\mathbf{B}^{m,l}$. All that remains to be shown is that $b^{m,l} > \sin(2\pi/n)$ for $n \geq 32$ and $l = 0, 1, \ldots, m-1$. The diagonal elements of the diagonal

matrices $\mathbf{Q}(\mathbf{D}^{m,l})$ are at least as large as $1/\sqrt{2}$. We can therefore use (25) to see that

$$g^{m,1} \geq 1/\sqrt{2}. \tag{34}$$

Likewise, we can use (26) to see that

$$g^{m,l} \geq g^{m,l-1}/\sqrt{2}, \qquad 2 \leq l \leq m-1. \tag{35}$$

From (34) and (35),

$$g^{m,l} \geq (1/\sqrt{2})^l, \qquad 1 \leq l \leq m-1. \tag{36}$$

From (27),

$$b^{m,1} \geq 1/\sqrt{2}. \tag{37}$$

Likewise, we can use (28) and (36) to see that

$$b^{m,l} \geq g^{m,l-1}/\sqrt{2} \geq (1/\sqrt{2})^l, \qquad 2 \leq l \leq m-1. \tag{38}$$

Recalling that $n = 2^m$, we can use (37) and (38) to see that for $n \geq 32$

$$\sin(2\pi/n) < (1/\sqrt{2})^{m-1} \leq b^{m,l}, \qquad 1 \leq l \leq m-1. \quad \square$$

## 5. Other DFTs

A well-known property of the DFT is that when the input sequence is real, the output sequence is conjugate symmetric. A real input DFT can be computed with fewer arithmetic computations than a complex input DFT. In the case of the split-radix DFT, savings in arithmetic computations can be obtained by simply eliminating unnecessary computations from the complex input DFT [4]. In particular, computations needed for redundant outputs are only performed once, real numbers are added together with one, rather than two, real additions, and a real number is added to an imaginary number with no arithmetic computations. Overall, half of the real multiplications and more than half of the real additions used in the original algorithm are eliminated.

These same methods can be used to reduce the number of $m/a-ops$ used in the split-radix version of the algorithm discussed in the last sections. The resulting algorithm will use $\frac{4}{3}nm - \frac{17}{9}n + 3 - \frac{1}{9}(-1)^m \quad m/a-ops$.

The $p$-dimensional DFT of an array of data of size $n_1 \times n_2 \times n_3 \times \cdots \times n_p$ is defined by

$$X_{k_1,k_2,\ldots k_p} = \sum_{r_1=0}^{n_1-1} \sum_{r_2=0}^{n_2-1} \cdots \sum_{r_p=0}^{n_p-1} W_{n_1}^{k_1 r_1} W_{n_2}^{k_2 r_2} \cdots W_{n_p}^{k_p r_p} x_{r_1,r_2,\ldots,r_p},$$

$$k_l \in \{0, 1, \ldots, n_l - 1\}.$$

Multidimensional DFTs can be computed in a row-column fashion. The algorithms derived in §3 can be used to obtain efficient row-column FFTs.

More efficient multidimensional DFT algorithms do not use a row-column approach. Such algorithms include vector-radix [13] and polynomial-transform [11, 5, 1] FFTs. With either vector-radix or polynomial-transform FFTs, the techniques of this paper can be used to obtain new algorithms that use as many $m/a-ops$ as the original algorithm had real additions. We will concentrate on

polynomial-transform based FFTs, which are more efficient than vector-radix FFTs.

Assume that, for each $l \in \{1, 2, \ldots, p\}$, $n_l$ is a power of 2, and let $n = \max(n_1, n_2, \ldots, n_p)$. A polynomial-transform FFT maps the computation of an $n_1 \times n_2 \times n_3 \times \cdots \times n_p$ DFT into a set of additions, $3n/2$ multiplication of a complex vector of size $n/2$ by the matrix $\mathbf{G}^{(n/2)}$, defined by

$$\mathbf{G}^{(n/2)} = (W_n^{j(2k+1)})_{j,k=0}^{n/2-1},$$

and one $p$-dimensional DFT of size $n_1/2 \times n_2/2 \times n_3/2 \times \cdots \times n_p/2$. By deriving an efficient algorithm to compute $\mathbf{G}^{(n/2)}\mathbf{x}$, where $\mathbf{x} \in \mathscr{C}^{n/2}$, we obtain an efficient polynomial-transform based algorithm to compute the multidimensional DFT.

Let

$$\mathbf{y} = \begin{pmatrix} y_0 \\ \vdots \\ y_{n/2-1} \end{pmatrix} = \mathbf{G}^{(n/2)}\mathbf{x} \quad \text{and} \quad \mathbf{z} = \begin{pmatrix} z_0 \\ \vdots \\ z_{n-1} \end{pmatrix} = \mathbf{F}^{(n)} \begin{pmatrix} \mathbf{x} \\ \mathbf{0}_{n/2} \end{pmatrix},$$

where $\mathbf{0}_{n/2}$ is the $(n/2)$-vector of zeros. Then $y_k = z_{2k+1}$. To compute $\mathbf{y}$, we use the split-radix version of the algorithm described in the last section to compute $\mathbf{z}$, but skip all computations needed to compute the even outputs of $\mathbf{z}$ and use no arithmetic computations for adding a number to zero. The algorithm that results allows us to compute $\mathbf{G}^{(n/2)}\mathbf{x}$ with $\frac{4}{3}nm - \frac{14}{9}n - \frac{4}{9}(-1)^m$ $m/a-ops$.

As an example, consider the computation of the DFT of an $n \times n$ array, where $n = 2^m$. Using polynomial transforms, we can compute this DFT with $\frac{3}{2}n^2m + \frac{5}{2}n^2$ additions, $\frac{3}{2}n$ multiplications of vectors by $G^{(n/2)}$, and one DFT of size $n/2 \times n/2$. If the algorithm that we have just described to multiply a vector by $G^{(n/2)}$ is used, then the number of $m/a - ops$ used by the new algorithm to compute a DFT of size $n \times n$, $\sigma_{\text{new}}(n, n)$, satisfies

$$\sigma_{\text{new}}(n, n) = \sigma_{\text{new}}(n/2, n/2) + \tfrac{7}{2}n^2m + \tfrac{1}{6}n^2 + \tfrac{2}{3}n(-1)^m.$$

With the initial condition $\sigma_{\text{new}}(2, 2) = 16$, we obtain an upper bound for the number of $m/a - ops$ required to compute the DFT of a square array.

**Theorem 5.1.** *If* $n = 2^m$, *then* $\sigma_{\text{new}}(n, n) = \frac{14}{3}n^2m - \frac{4}{3}n^2 - \frac{4}{9}n(-1)^m + \frac{16}{9}$.

## BIBLIOGRAPHY


1. L. Auslander, E. Feig, and S. Winograd, *The multiplicative complexity of the discrete Fourier transform*, Adv. in Appl. Math. **5** (1984), 87–109.
2. R. E. Blahut, *Fast algorithms for digital signal processing*, Addison-Wesley, Reading, MA, 1986.
3. J. W. Cooley and J. W. Tukey, *An algorithm for the machine computation of complex Fourier series*, Math. Comp. **19** (1965), 297–301.
4. P. Duhamel, *Implementation of "split-radix" FFT algorithms for complex, real, and real-symmetric data*, IEEE Trans. Acoust. Speech Signal Process. **34** (1986), 285–295.
5. E. Feig, *New algorithms for the 2-dimensional discrete Fourier transform*, Technical Report RC8897, IBM Research, Yorktown Heights, NY, June 1981.
6. M. T. Heideman and C. S. Burus, *A bibliography of fast transform and convolution algorithms*. II, Technical Report 8402, Rice Univ., Houston, TX, February 1984.

7. E. Linzer and E. Feig, *Implementation of efficient FFT algorithms on fused multiply/add architectures*, Technical Report RC17330, IBM Research, Yorktown Heights, NY, 1990; IEEE Trans. Signal Process (to appear).
8. ____, *New scaled DCT algorithms for fused multiply/add architectures*, IEEE Internat. Conf. Accoust. Speech, Signal Process. (Toronto, Canada), IEEE, Piscataway, NJ, 1991, pp. 2201–2204.
9. A. Mechentel, Private communication, 1990.
10. R. K. Montoye, E. Hokenek, and S. L. Runyon, *Design of the IBM RISC System/6000 floating point execution unit*, IBM J. Res. Develop. **34** (1990), 71–77.
11. H. J. Nussbaumer, *Fast computation of discrete Fourier transforms using polynomial transforms*, IEEE Trans. Acoust. Speech Signal Process. **27** (1979), 169–181.
12. C. M. Rader and N. M. Brenner, *A new principle for fast Fourier transformation*, IEEE Trans. Acoust. Speech Signal Process. **24** (1976), 264–265.
13. G. E. Rivard, *Direct fast Fourier transform of bivariate functions*, IEEE Trans. Acoust. Speech Signal Process. **25** (1977), 250–252.
14. H. V. Sorensen, M. T. Heideman, and C. S. Burus, *On computing the split-radix FFT*, IEEE Trans. Acoust. Speech Signal Process. **34** (1986), 152–156.
15. R. Tolimieri, M. An, and C. Lu, *Algorithms for the discrete Fourier transform and convolution*, Springer-Verlag, New York, 1989.
16. M. Vetterli and H. J. Nussbaumer, *Simple FFT and DCT algorithms with reduced number of operations*, Signal Process. **6** (1984), 267–278.
17. S. Winograd, *On computing the discrete Fourier transform*, Proc. Nat. Acad. Sci. U.S.A. **73** (1976), 1005–1006.
18. ____, *On computing the DFT*, Math. Comp. **32** (1978), 175–199.
19. ____, *On the multiplicative complexity of the discrete Fourier transform*, Adv. Math. **32** (1979), 83–117.

IBM THOMAS J. WATSON RESEARCH CENTER, P. O. BOX 218, YORKTOWN HEIGHTS, NEW YORK 10598

*E-mail address*: elliotl@watson.ibm.com

*E-mail address*: feig@watson.ibm.com