On the computation of discrete Fourier transform using Fermat number transform


Indexing terms: Signal processing, Discrete Fourier transforms, Fermat number transforms

Abstract: In the paper the results of a study using Fermat number transforms (FNTs) to compute discrete Fourier transforms (DFTs) are presented. Eight basic FNT modules are suggested and used as the basic sequence-lengths to compute long DFTs. The number of multiplications per point is for most cases not more than one, whereas the number of shift-adds is approximately equal to the number of additions in the Winograd-Fourier-transform algorithm and the polynomial transform. Thus the present technique is very effective in computing discrete Fourier transforms.

1 Introduction

The introduction of the fast Fourier transform (FFT) algorithm by Cooley and Tukey [1, 2] in 1965 allowed a tremendous saving in calculating effectively the discrete Fourier transform (DFT). This algorithm reduces the number of real multiplications for a direct DFT computation from $4N^2$ to $2N \log_2 N$, while the number of real additions is reduced from $2N^2$ to $3N \log_2 N$ for $N$-point DFTs. The major disadvantages of the FFT algorithm are that (i) it still requires quite a large number of multiplications and (ii) the number of real multiplications per point is almost the same for both real and complex input data. This last point makes the FFT very inefficient for the purpose of calculating the DFT of real input data. In 1975, Winograd [3] showed that the minimum number of multiplications required to compute the circular convolution of two length-$N$ sequences is $2N - K$, where $K$ is the number of divisors of $N$ including 1 and $N$. Agarwal and Cooley [4], Winograd [5] and Kolba and Parks [6] made use of Rader’s theorem [7] on DFT with prime transform length to construct their algorithms for the computation of DFT. Compared with conventional FFT methods, the Winograd-Fourier-transform algorithms require only one-half to one-third of the number of multiplications needed for conventional FFT, with a slightly larger number of additions per point. In 1977, Nussbaumer [8] first suggested the use of polynomial transform over the field of polynomials for computing two-dimensional convolutions. This approach led to the development of fast algorithms [9–11] for computing DFTs using polynomial transforms. The method requires two to four real multiplications per point for complex input data. For the computation of real input data, both the Winograd-Fourier-transform algorithm and polynomial transform require in fact only-half of the operations needed for the case of complex data. Reed and Truong [12, 13] proposed a technique for the computation of discrete Fourier transforms, based on Winograd’s method in combination with Mersenne prime number theoretic transforms. This hybrid algorithm requires fewer multiplications than either the standard FFT or Winograd’s more conventional algorithm.

However, a very large number of additions are required and the number of multiplications per point is still relatively large (about 1.33 to 3.49 multiplications per point). Recently Siu and Constantinides [14] have shown that the number of multiplications for DFT computations can be reduced further by using number theoretic transforms (NTT) [15–18]. The total number of real multiplications in this case in fact reduces to $(P - 1)$ for computing a $P$-point DFT, where $P$ is a prime number (or $1 - 1/P$ multiplications per point). Of the various versions of the NTT proposed, the Fermat number transform (FNT), which has been thoroughly investigated by Agarwal and Burrus [17], and the Mersenne number transform (MNT), which was introduced by Rader [16], appear to offer the greater promise vis-a-vis speed and complexity. However, if we take the number of shift-adds into account for the evaluation of the entire computational effort, it turns out that, because there is no fast algorithm (or fast Fourier transform type algorithm) for the computation of Mersenne number transforms, a large number of shift-adds have to be used for the computation of discrete Fourier transforms using MNT. Some fast Fourier transform types of algorithms can possibly be used in some pseudo-Fermat number transforms [19] and pseudo-Mersenne transforms [20]. The transform length of the Fermat number transform is highly composite (it is in fact a power of two), and hence fast Fourier transform types of algorithms can be used for the computation. In this respect, it would appear that the FNT is the most prominent candidate for the computation of DFTs. However, for actual implementation, all number theoretic transforms are limited by sequence-length constraints [21]. This limits the application of FNT to a small number of relatively short DFTs only, or indeed the application of number theoretic transform for computing the discrete Fourier transform is limited as a whole.

In this paper, we present the results of our study using Fermat number transforms to compute DFTs and propose various techniques for computing long discrete Fourier transforms.

2 Basic Fermat-number-transform modules

The Fermat number transform (FNT) and its inverse are defined as follows:

$$X_k = \sum_{n=0}^{N-1} x_n a^{nk} / M$$

(1)
and

\[ x_n = \left( \frac{1}{N} \sum_{k=0}^{N-1} X_k \omega^{-nk} \right)_{M'} \]

for \( n, k = 0, 1, \ldots, N - 1 \) (2)

where \( M' = 2^t + 1 \), a Fermat number, for \( t = 1, 2, \ldots \), \( \alpha = \text{root of unity of order } N \) and is chosen to be 2 or a power of 2 for most cases, and \( \omega^m \neq 1, 1 \leq m < N \).

The expression \( \langle C \rangle_{M'} \) means the residue of the number \( C \) modulo \( M' \). \( N \) is also a power of 2 and its maximum value is \( 2^{t+1} \) for \( \alpha = 2 \). Similar to the discrete Fourier transforms, the FFT can be used to compute cyclic convolutions. Let us consider the cyclic convolution of the two sequences, \( \{x_n: n = 0, 1, \ldots, N - 1\} \) and \( \{h_n: n = 0, 1, \ldots, N - 1\} \).

\[ X_m = \sum_{n=0}^{N-1} x_n h_{m-n} \text{ for } m = 0, 1, \ldots, N - 1 \] (3)

For the computation of eqn. 3 using FFT \( X_m \) can be found from the inverse Fermat number transform (IFNT)

\[ X_m = \left( \frac{1}{N} \sum_{k=0}^{N-1} Y_k \omega^{mk} \right)_{M'} \]

for \( m = 0, 1, \ldots, N - 1 \)

The coefficients \( Y_k \) are the products of the term-by-term multiplications of the corresponding \( X_k \) and \( H_k \), and \( H_k \) is the FFT of the sequence \( \{h_n: n = 0, 1, \ldots, N - 1\} \).

Consider now the discrete Fourier transform of a real sequence \( \{x(n): n = 0, 1, \ldots, N - 1\} \)

\[ Y(k) = \sum_{n=0}^{N-1} x(n) W_{0k}^n \]

where \( k = 0, 1, \ldots, N - 1 \) and

\[ W_0 = \exp \left( -j \frac{2\pi}{N} \right) \]

Let us denote \( \langle g^s \rangle_p \) as the residue of the number \( g^s \) modulo \( P \), \( g \) is a primitive root that generates all nonzero elements inside the field modulo \( P \). If \( N \) is a prime number, \( P \) for example, eqn. 4 can be reordered [14, 7] in the following form:

\[ Y(0) = \sum_{n=0}^{p-1} x(n) \]

and

\[ Y(\langle g^s \rangle_p) = X(0) + X(\langle g^s \rangle_p) \]

where

\[ X(\langle g^s \rangle_p) = \sum_{n=1}^{p-1} x(\langle g^{-s} \rangle_p) W_0 \langle g^{-n} \rangle_p \]

for \( k = 1, 2, \ldots, P - 1 \) (7)

Eqn. 7 represents a backward cyclic convolution of length \( (P - 1) \) and can be expressed as the convolution

\[ \{x_0, x_1, \ldots, x_{P-2}\} \otimes \{W_0, W_1, \ldots, W_{(P-1)/2-1}\}, \]

\[ W_0, W_1, \ldots, W_{(P-1)/2-1} \] (8)

where

\[ x_n = x(\langle g^{-s} \rangle_p) \]

\[ W_n = W_n^{(s+1)p} \text{ for } n = 0, 1, \ldots, P - 2 \]

\[ W_n^{(s+1)p} \]

The symbols \( \ast \) and \( \otimes \) mean complex conjugate and cyclic convolution, respectively. This cyclic convolution sum can be computed by Fermat number transform. Siu and Constantinides [14] have already pointed out that the real and imaginary parts of the number theoretic transformed results of \( \{W_0, W_1, \ldots, W_{(P-1)/2-1}\}, W_1, \ldots, W_{(P-1)/2-1} \) are alternately zero, thereby reducing the total number of multiplications to \((P - 1)\) for \( P \)-point DFTs.

It is also interesting to point out that since the sequence \( \{x(n): n = 0, 1, \ldots, N - 1\} \) is real, \( X(\langle g^s \rangle_p): k = 1, 2, \ldots, P - 1 \) can be put in the form of

\[ X(\langle g^1 \rangle_p), X(\langle g^2 \rangle_p), \ldots, X(\langle g^{(P-1)/2} \rangle_p), X(\langle g^1 \rangle^*), X(\langle g^2 \rangle^*), \ldots, X(\langle g^{(P-1)/2} \rangle^*) \]

Hence this expression shows that, for actual implementation, only \( X(\langle g^s \rangle_p), X(\langle g^{(P-1)/2} \rangle_p) \) need to be computed, the other half of the \( X(\langle g^s \rangle_p) \) can be found by taking the conjugates of these values.

Notice that a Fermat number is defined as \( F_t = 2^{2^t} + 1 \); hence \( F_t - 1 \) is also a power of 2 and so if \( F_t \) is chosen as the DFT length, the corresponding \((F_t - 1)\)-point cyclic convolution can be computed by an NTT (in our case FNT) with an FFT-type algorithm.

Fermat numbers up to \( F_4 \), namely \( \{3, 5, 17, 257, 65537\} \), are all primes, and these are possible sequence lengths (i.e. \( P = F_t \) for \( t = 0, 1, 2, 3, 4 \)) for the computation of discrete Fourier transform using NTT.

For example, let us take \( F_3(= 17) \) to illustrate the idea. Consider the DFT of the sequence \( \{x(0), x(1), \ldots, x(16)\} \), i.e. \( N = P = 17 \). In this case 3 is a primitive root which can be used to generate all elements inside the field modulo 17. Hence eqn. 8 becomes

\[ \{x(6), x(2), x(12), x(4), x(8), x(14), x(16), x(11), x(15), x(5), x(13), x(10), x(9), x(3), x(1)\} \]

\[ \otimes \{W_1, W_3, W_9, W_0, W_10, W_3, W_5, W_1, W_6^*, W_0^*, W_3^*, W_9^*, W_0^*, W_10^*, W_5^*, W_1^*\} \]

This convolution sum can be computed by using a Fermat number transform. If we choose \( M \), the base for modulo arithmetic in FNT, to be \( F_4(2^{16} + 1) \), the generator \( \alpha \) for the transform is \( 2^2 \) and of course \( N = 16 \). On the other hand if we choose \( M = F_3(2^{32} + 1) \) then \( \alpha = 2^4 \) and \( N \) of course remains as 16. In all cases, the transformed results of \( \{W_0, W_3, W_9, W_0^*, W_3^*, W_9^*, W_1^*\} \) have their real and imaginary parts alternately zero. Hence real multiplications are required for the computation of a 17-point real-input-data DFT.

For actual implementation, not all Fermat primes that can be used as sequence lengths are attractive for the computation of DFT. For example, \( F_4(= 65537) \) is long enough (or perhaps too long) for nearly all kinds of applications, but there is no suitable Fermat number transform (with sequence length \( 65537 - 1 = 65536 \)) available for such a long sequence length. In the light of this situation, it would seem that the most attractive transform lengths are \( F_3(= 17) \) and \( F_4(= 257) \). Table 1 lists the number of real operations for four possible prime (Fermat primes) lengths and four other lengths (to be discussed below). The number of multiplications is found by

\[ M = P - 1 \]

and the number of shift-adds (with slight modification for length 257, see below) is found by [14]

\[ A = (P - 1)[2 \log_2(P - 1) + 1] \]

It can be seen from Table 1 that for length 257 DFT we have only a single choice of base (\( F_4 \)) for modulo arithmetic, and in this case the generator is \( 2^2 \). Agarwal and
The conjugate structure of the array dimensional mapping of the array using a map \( P - 1 \)

Consider the discrete Fourier transform length be a prime \( \langle K^2 n_2 + C^2 n_1 \rangle_{P-1} \) where follows:

- Dimensional cyclic convolution of eqn. 7 or 8 into a two-dimensional array, let us choose the map \( n = \langle 4n_2 + 3n_1 \rangle_{12} \). Hence

\[
\begin{bmatrix}
W_0^5 & W_0^4 & W_0^2 & W_0^1 & W_0^6
\end{bmatrix}
\]

Hence each row (dimension \( n_1 \)) of this resultant two-dimensional array processes the same conjugate structure as the original one-dimensional array.

**Proof**

The map \( n = \langle K_2 n_2 + CN_2 n_1 \rangle_{p-1} \) gives \( N_1 \)-point cyclic arrays. The term \( K_2 n_2 \), for \( n_2 = 0, 1, 2, \ldots, N_2 - 1 \) gives a base or an offset value for finding sets of data for \( N_1 \)-point cyclic arrays. Thus this term does not affect the property of the \( N_1 \)-point cyclic arrays. Because the \( n_1 \) map is cyclic, \( CN_2 n_1 \) can be written as \( \langle CN_2 n_1 \rangle_N \), and actually the set of integers \( \{CN_2 n_1\}_{N_1} \) is merely a permutation of the set \{0, N_2, 2N_2, \ldots, (N_1 - 1)N_2\}_{mod \ p - 1}. However

\[
\left\{\langle CN_2 n_1\rangle_{N_1} : n_1 = 0, 1, \ldots, N_1 - 1\right\}
\]

\[
= \left\{0, CN_2, \ldots, C\left(\left\lfloor N_1 - \frac{1}{2}\right\rfloor - 1\right)N_2, C\left(\left\lfloor N_1 - \frac{1}{2}\right\rfloor + \frac{1}{2}\right)N_2, \ldots, C(N_1 - 1)N_2\right\}_{mod \ N_2}
\]

From eqns. 7 and 8, it is clear that

\[
W_{\langle n \rangle_{p-1}} = W_{\langle (n+1/2) \rangle_{p-1}}
\]

Hence it can be seen from eqns. 15 and 16 that, after substituting eqn. 14 into the expression \( W_0^5 \): \( n = 0, 1, \ldots, P - 2 \), we get arrays

\[
\begin{bmatrix}
W_{n_2, 0} & W_{n_2, 1} & \cdots & W_{n_2, (N_2/2)-1} & W_{n_2, (N_2/2)} & W_{n_2, (N_2/2)+1} & \cdots & W_{n_2, (N_2-1)}
\end{bmatrix}
\]

or

\[
W_{\langle n_2 \rangle_{(N_2/2)-1}} = W_{\langle n_2 + (N_2/2) \rangle_{p-1}}
\]

This completes the proof of the theorem.

**Corollary**

It is impossible to map \( W_0^5 : n = 0, 1, \ldots, P - 2 \) into a two-dimensional cyclic array (cyclic in both dimensions) with the conjugate structure preserved in both dimensions.

**Proof**

The proof of this corollary can be seen from the fact that for the mapping from a one-dimensional cyclic array into a two-dimensional cyclic array (in both dimensions) \( N_1 \) and \( N_2 \) should be relatively prime [23]. Hence either \( N_1 \) or \( N_2 \)
can be even. From the above theorem, it is evident that for a cyclic mapping of \( \{ W_n^p : n = 0, 1, \ldots, P - 2 \} \) in both dimensions, only one dimension preserves conjugate structure. This proves the corollary.

A direct consequence of the conjugate property of the two-dimensional mapping is that the results of both real and imaginary parts of the Fermat number transformed sequence \( \{ W_{n_2, n_1} : n_1 = 0, 1, \ldots, N_1 - 1 \} \) would be absolutely zero. Hence the total number of real multiplications for each \( N_1 \)-point DFT reduces from \( 2N_1 \) to \( N_1 \). Similarly, the conjugate structure of \( X(\langle g^n \rangle_R) \) in eqn. 6 is also preserved for this mapping. As in the previous example, if \( n = \langle 4n_2 + 3n_1 \rangle_2 \), then \( k \) has to map to \( \langle 4n_2 + 9n_1 \rangle_{12} \) [23]. Hence we obtain

\[
\begin{align*}
\{X(2), X(4), X(8), X(10), X(6), X(12), X^*(2), X^*(4), X^*(8), X^*(3), X^*(6), X^*(12)\} \\
\leftrightarrow \{X(2) X(10) X^*(2) X^*(10) \} \\
\leftrightarrow \{X(6) X(4) X^*(6) X^*(4) \} \\
\leftrightarrow \{X(5) X(12) X^*(5) X^*(12) \}
\end{align*}
\]

It is clear that only \( X(2), X(10), X(6), X(4), X(5) \) and \( X(12) \) need to be computed, the other \( X(n) \) values are merely complex conjugates of these values.

Since \( N_1 \) is even, it is most worthwhile to choose \( N_1 \) as a power of two, so that the FFT-type algorithm can be used. Depending on whether the second dimension is cyclic or a linear convolution [24], pseudo-Fermat transforms, pseudo-Mersenne transforms, Mersenne number transforms, Lagrange interpolation formulas or any short convolution algorithm can be used to carry out the computation.

In practical applications, the longest transform length (with 2 or \( \sqrt{2} \) as the generator) available for a Fermat number transform is 256, but, in this case, the modulo base is \( F_6 = 2^{64} + 1 \). This wordlength is too long for some computers (especially minicomputers and microprocessors) and in some cases in hardware implementations. Hence we begin our further discussion with a basic sequence length of 64. With this basic sequence length, we can use any of the Fermat numbers in \( \{ F_2, F_3, F_6 \} \) as the modulo base. This gives a flexible choice of wordlengths and dynamic ranges for various applications. Substituting eqns. 9 and 10 into eqn. 7 we obtain

\[
X_k = \sum_{n=0}^{P-2} X_n W_{k-n} \quad \text{for } k = 0, 1, \ldots, P - 2 \tag{18}
\]

where

\[
X_n = X(\langle g^n \rangle_R) \tag{19}
\]

Recall \( N_1, N_2 = P - 1 \). Let \( N_1 = 64 \) and the map of \( n = \langle K_2 n_2 + N_2 j_2 \rangle_{P-1} \), hence eqn. 18 becomes

\[
X_{k_1, k_2} = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} X_{n_2, n_1} W_{k_2-n_2, k_1-n_1} \tag{20}
\]

The map is cyclic in \( n_1 \), and so we can take the number theoretic transform of eqn. 20 with respect to \( n_1 \) [24]. Eqn. 20 becomes

\[
X_{k_2, k_1} = \sum_{n_2=0}^{N_2-1} X_{n_2, k_1} W_{k_2-n_2, k_1} \tag{21}
\]

where

\[
X_{n_2, k_1} = \left\langle \sum_{n_1=0}^{N_1-1} X_{n_2, n_1} g^{n_1 k_1} \right\rangle_{M'} \tag{22}
\]

\[
W_{k_2-n_2, k_1} = \left\langle \sum_{n_1=0}^{N_1-1} W_{k_2-n_2, n_1} g^{n_1 k_1} \right\rangle_{M'} \tag{23}
\]

\[ k_2 = 0, 1, \ldots, N_2 - 1, \]

and

\[ M' = \text{base for modulo arithmetic} \]

Eqn. 21 is a linear convolution sum of the sequences \( \{X_{n_2, k_1} : n_2 = 0, 1, \ldots, N_2 - 1 \} \) and \( \{W_{k_2-n_2, k_1} : n_2 = 0, 1, \ldots, N_2 - 1 \} \) if \( N_1 \) and \( N_2 \) are not mutually prime. This equation can be arranged as a cyclic convolution sum if \( N_1 \) and \( N_2 \) are mutually prime. For the case of linear convolutions, \( [2 - (1/N_2)] \) [24] multiplications per point are required for the computation. These number of multiplications are still relatively large as basic transform lengths for the application of Fermat number transform to DFT. We now turn our attention to the case with cyclic mapping in \( N_2 \). A very simple mapping is \( n = \langle N_1 n_2 + N_2 n_1 \rangle_{N_2} \), where \( N_1 = 64 \). If \( n_1 \) is cyclic then a number theoretic transform can be used again to find this convolution sum. However, \( N_1 \) and \( N_2 \) are mutually prime, \( N_1 \) is even and we restrict our case to using FNT only, and so number theoretic transform is not suitable in this dimension.

Recall the fact that of the various short odd cyclic convolution algorithms introduced by Winograd [5] and Agarwal and Cooley [4], the length-3 cyclic convolution algorithm is the most efficient in terms of both the numbers of additions and multiplications per point needed. In fact, only four multiplications and 11 additions are required for the computation of a 3-point cyclic convolution sum. Let \( N_2 = 3 \) so that eqns. 20 and 21 become

\[
X_{k_2, k_1} = \sum_{n_2=0}^{3-1} \sum_{n_1=0}^{64-1} X_{n_2, n_1} W_{k_2-n_2, k_1-n_1} \tag{24}
\]

\[
X'_{k_2, k_1} = \sum_{n_2=0}^{3-1} X_{n_2, k_1} W'_{k_2-n_2, k_1} \tag{25}
\]

where

\[ k_1 = 0, 1, \ldots, 63 \quad k_2 = 0, 1, 2 \]

It is important to recall that the choice of \( N_1 \) and \( N_2 \) should always agree with our basic design, i.e. they must be chosen such that \( (P - 1) = N_1 N_2 \) or, equivalently, \( (N_1, N_2 + 1) \) must be a prime number. In our case, \( (N_1, N_2 + 1) \) is equal to 193, which is prime with 5 being the minimum primitive root.

Let us now work out the number of operations required to compute this 193-point DFT. There are 64 3-point cyclic convolutions, hence overall there are \( 64 \times 4 = 256 \) multiplications. Besides the 64 3-point cyclic convolutions, there are 64 3-point FNT transforms (forward and reverse), hence, by using eqn. 12, the total number of shift-adds (some are simply additions) becomes \( 64 \times 11 + 3 \times 64 = 3200 \). This method requires 1.33 multiplications and 16.6 shift-adds per point respectively. Similarly, length 13 (= \( 3 \times 4 + 1 \))-point DFT and length 97 (= \( 3 \times 32 + 1 \))-point DFT can be computed by using 1.23 (8) and 1.32 (14.51) multiplications (additions) per point, respectively.

On the other hand, for systems that can have relatively large word size (64-bit words for the present case), we may use a length-256 FNT with \( F_6 = 2^{64} + 1 \) as a modulo base to construct another basic transform length for DFT using FNT. In this case, the DFT transform length is
(256 \times 3 + 1) = 769, which is a prime number again, with 11 being the minimum primitive root. The total number of multiplications is 256 \times 4 = 1024 and the total number of shift-adds becomes 256 \times 11 + 3 \times 4544 = 16448. The DFT with this sequence length requires 1.33 multiplications and 24.16 shift-adds per point, respectively.

Length 3-, 5-, 17-, 257-, 13-, 97-, 193- and 769-point DFTs form eight basic Fermat number transform modules, and these modules are used to construct algorithms for computing long discrete Fourier transforms in the next Section.

3 Prime factor implementation

The basic idea in this Section arises mainly from the fact that all eight basic Fermat number transform modules obtained so far are prime in length. Thus any two or more of these modules can form basic lengths for two- or multidimensional rearrangements of discrete Fourier transforms. Furthermore, any one or more of these modules can also combine with other short (or long if there happen to be any) DFT algorithms for computing long discrete Fourier transforms.

Now let us examine the multidimensional mapping of the DFT equation. For the sake of simplicity, we consider a simple two-dimensional case. Let N of eqn. 4 be a composite number, N_3 N_4 = N. Further, let N_3 and N_4 be relatively prime so that the 'twiddle factors' [25] of the multidimensionally rearranged DFT equation can be avoided by the application of the Chinese remainder theorem (CRT) [26, 23]. In this case eqn. 4 can be written as

\[ Y_{k_1, k_2} = \sum_{n_4=0}^{N_4-1} \sum_{n_3=0}^{N_3-1} x_{n_3, n_4} W_3^{-n_3 k_3} W_4^{-n_4 k_4} \] (26)

where

\[ W_3 = \exp\left(-j \frac{2\pi}{N_3}\right) \]
\[ W_4 = \exp\left(-j \frac{2\pi}{N_4}\right) \]

and the mapping becomes

\[ n = N_3 n_4 + N_4 n_3 \]
\[ k = N_3 N_4^{-1} k_4 + N_4 N_3^{-1} k_3 \] (28)

where

\[ \langle N_3 N_3^{-1} \rangle_{N_4} = 1 \]
\[ \langle N_4 N_4^{-1} \rangle_{N_3} = 1 \]

Recall also the fact that both the prime factor algorithm (PFA) [6] and the Winograd-Fourier-transform algorithm [5] essentially make use of the same multidimensional mapping and short DFT algorithms for their computation. The number of operations per point becomes rather large for increasing DFT sizes. This is mainly due to the nature of multidimensional mapping and also due to limited number of available efficient short DFTs. There are mainly eight, namely length-2, 3, 4, 5, 7, 8, 9, 16 efficient short DFT algorithms [3, 9]. Four of these are powers of two; hence only one of the these four can appear in the same multidimensionally rearranged DFT. However, if we add our basic FNT modules to these eight short DFT algorithms we have available a much larger choice in transform lengths, and the number of multiplications per point remains a very small number even for DFT lengths longer than 10000.

Eqn. 26 can now be written in the following form suitable for a prime factor algorithm [6],

\[ Y_{k_1, k_3} = \sum_{n_4=0}^{N_4-1} \left( \sum_{n_3=0}^{N_3-1} x_{n_3, n_4} W_3^{-n_3 k_3} \right) W_4^{-n_4 k_4} \] (29)

The two-dimensional transform in eqn. 29 may be implemented by first calculating N_4 length-N_3 DFTs

\[ Y_{n_3, n_4} = \sum_{k_3=0}^{N_3-1} x_{n_3, n_4} W_3^{-n_3 k_3} \] (30)

then calculating N_3 length-N_4 DFTs

\[ Y_{k_1, k_2} = \sum_{n_4=0}^{N_4-1} Y_{n_3, n_4} W_4^{-n_4 k_4} \] (31)

The number of multiplications and additions become

\[ M = N_4 M_3 + N_3 M_4 \] (32)
\[ A = N_4 A_3 + N_3 A_4 \] (33)

where M_3(A_3) and M_4(A_4) are the number of multiplications (additions) for lengths N_3 and N_4 DFTs, respectively. Any one of our eight FNT modules can be used in N_3-point DFTs, and any one of the eight short DFTs can be used in N_4-point DFTs, provided that N_3 and N_4 are relatively prime. However, on close examination of the eight short DFT algorithms it becomes clear that the most efficient lengths are 2 and 4, in that both require trivial multiplications only. In this case eqn. 32 becomes M = N_4 M_3. This immediately doubles or quadruples the sequence lengths of the four FNT modules without increasing the number of required multiplications per point. For other transform lengths there is an unavoidable but nevertheless slight increase in the number of multiplications per point.

Eqn. 26 may also be described by the following form suitable for computation using the Winograd-Fourier-transform algorithm [5]:

\[ \tilde{Y}_{k_4} = \sum_{n_4=0}^{N_4-1} (\tilde{W} \tilde{X}_{n_4}) W_4^{-n_4 k_4} \] (34)

where

\[
\begin{bmatrix}
X_{n_3, 0} \\
X_{n_3, 1} \\
\vdots \\
X_{n_3, N_3-1}
\end{bmatrix}
= 
\begin{bmatrix}
Y_{k_4, 0} \\
Y_{k_4, 1} \\
\vdots \\
Y_{k_4, N_3-1}
\end{bmatrix}
\]

\[
\tilde{X}_{n_4} = 
\begin{bmatrix}
W_{0, 3}^0 & W_{0, 3}^0 & \cdots & W_{0, 3}^{N_3-1} \\
W_{0, 3}^1 & W_{0, 3}^1 & \cdots & W_{0, 3}^{N_3-1} \\
\vdots & \vdots & \ddots & \vdots \\
W_{0, 3}^{N_3-1} & W_{0, 3}^{N_3-1} & \cdots & W_{0, 3}^{(N_3-1)^2}
\end{bmatrix}
\]

Eqn. 34 is obtained by considering eqn. 26 as essentially a one-dimensional DFT of length N_4 with each scalar replaced by a vector of N_3 terms. The number of operations in this case becomes

\[ M = M_1 M_2 \] (35)
\[ A = N_1 A_2 + M_1 A_1 \] (36)

However, owing to the nesting nature of this algorithm, the trivial multiplications (the multiplications of \(W^0\)) in the basic FNT modules cannot be ignored in eqn. 35. From eqns. 5 and 6 it is clear that there are two trivial multiplications, namely \(\left(\sum_{n=0}^{p-1} x(n)W_0^n\right)\) and \(s(0)W_0^0\). To make a
slight simplification of the computation, let us make a rearrangement of eqn. 6

\[ \mathcal{Y}(g^k) = x(0) + \sum_{n=1}^{P-1} x(g^{-n}) W_0^{g^{kn}} \]

such that

\[ \mathcal{Y}(g^k) = \sum_{n=1}^{P-1} [x(g^{-n}) - x(0)] W_0^{g^{kn}} \]  

(37)

Hence, if eqn. 37 is used instead of eqn. 6, one \( W_0 \) multiplication is saved. The only difference between eqn. 6 and eqn. 37 is that \( x(0) \) is included before the computation of the cyclic convolution in eqn. 37 whereas \( x(0) \) is included after the computation of the cyclic convolution in eqn. 6.

As shown in Table 2, if we include the multiplication of \( \left( \sum_{n=0}^{N-1} x(n) \right) W_0 \) in eqn. 5, the number of multiplications required for the eight basic FNT modules increases slightly.

Let us illustrate this idea with \( N = 15 \). Hence \( N = 15 = 3 \times 5 \). Eqn. 27 becomes

\[ n = n_3 + 3N_3 \]

where

\[ n_3 = \{0, 1, 2, 3, 4\} \]
\[ n_4 = \{0, 1, 2\} \]

i.e.

\[ x_{0,0} = x(0) \quad x_{1,0} = x(5) \quad x_{2,0} = x(10) \]
\[ x_{0,1} = x(3) \quad x_{1,1} = x(8) \quad x_{2,1} = x(13) \]
\[ x_{0,2} = x(6) \quad x_{1,2} = x(11) \quad x_{2,2} = x(1) \]
\[ x_{0,3} = x(9) \quad x_{1,3} = x(14) \quad x_{2,3} = x(4) \]
\[ x_{0,4} = x(12) \quad x_{1,4} = x(2) \quad x_{2,4} = x(7) \]

Eqns. 26 and 34 become

\[ \mathcal{Y}_k = \sum_{n=0}^{3} \left( \sum_{n_3=0}^{3} x_{n_3} W_3^{n_3k_3} \right) W_4^{n_4k_4} \]

and

\[ \tilde{Y}_k = \sum_{n=0}^{2} \left( \tilde{W} \tilde{X}_n \right) W_4^{n_4k_4} \]  

(38)

where

<table>
<thead>
<tr>
<th>DFT length</th>
<th>Number of multiplications</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>257</td>
<td>257</td>
</tr>
<tr>
<td>13</td>
<td>17</td>
</tr>
<tr>
<td>87</td>
<td>129</td>
</tr>
<tr>
<td>193</td>
<td>257</td>
</tr>
<tr>
<td>769</td>
<td>1025</td>
</tr>
</tbody>
</table>

Using the Fermat number transform, we obtain

\[ \left[ \begin{array}{c} W_3^0 \\ W_3^1 \\ W_3^2 \\ W_3^3 \\ W_3^4 \\ W_3^5 \\ W_3^6 \\ W_3^7 \end{array} \right] = \left[ \begin{array}{cccccccc} W_0^0 & W_0^1 & W_0^2 & W_0^3 \\ W_0^3 & W_0^1 & W_0^2 & W_0^0 \\ W_0^2 & W_0^3 & W_0^1 & W_0^0 \\ W_0^1 & W_0^0 & W_0^3 & W_0^2 \end{array} \right] \]

and

\[ W_3 = \exp \left( -j \frac{2\pi}{3} \right) \quad \text{for } n_4 = 0, 1, 2 \]

Now we may use the 3-point DFT algorithm (see Reference 5, for example), to evaluate eqn. 38

\[ S_1 = \tilde{X}_1 + \tilde{X}_2 \]
\[ S_2 = \tilde{X}_1 - \tilde{X}_2 \]
\[ S_0 = \tilde{X}_0 + \tilde{S}_1 \]
\[ \tilde{M}_0 = 1 \cdot \tilde{W} \tilde{S}_0 \]
\[ \tilde{M}_1 = \left( \cos \frac{2\pi}{3} - 1 \right) \cdot \tilde{W} \tilde{S}_1 \]
\[ \tilde{M}_2 = j \sin \frac{2\pi}{3} \cdot \tilde{W} \tilde{S}_2 \]
\[ \tilde{S}_3 = \tilde{M}_0 + \tilde{M}_1 \]
\[ \tilde{S}_a = \tilde{S}_3 + \tilde{M}_2 \]
\[ \tilde{S}_5 = \tilde{S}_3 - \tilde{M}_2 \]

and

\[ \tilde{Y}_0 = \tilde{M}_0 \]
\[ \tilde{Y}_1 = \tilde{S}_a \]
\[ \tilde{Y}_2 = \tilde{S}_5 \]

All three expressions in eqn. 39 are actually length-5 DFTs multiplied by constants and are in the following form

\[ \tilde{M}_k = \text{CON} \tilde{W} \tilde{S}_k \]

where \( \text{CON} \) is a constant and \( k = 0, 1, 2 \)

Let

\[ \tilde{M}_k = \left[ \begin{array}{c} m_{k,0} \\ m_{k,1} \\ m_{k,2} \\ m_{k,3} \\ m_{k,4} \end{array} \right] \quad \text{and} \quad \tilde{S}_k = \left[ \begin{array}{c} s_{k,0} \\ s_{k,1} \\ s_{k,2} \\ s_{k,3} \\ s_{k,4} \end{array} \right] \]

Using the Fermat number transform, we obtain

\[ \left[ \begin{array}{c} m_{k,0} \\ m_{k,2} \\ m_{k,4} \\ m_{k,3} \\ m_{k,1} \end{array} \right] = \text{CON} \left[ \begin{array}{c} b_{k,0} \\ b_{k,2} \\ b_{k,4} \\ b_{k,3} \\ b_{k,1} \end{array} \right] \]  

(40)

where

\[ \left[ \begin{array}{c} b_{k,2} \\ b_{k,4} \\ b_{k,3} \\ b_{k,1} \end{array} \right] = \left[ \begin{array}{cccc} 1 & 1 & 1 & 1 \\ \alpha & \alpha^{-1} & \alpha^{-2} & \alpha^{-3} \\ \alpha^{-2} & \alpha^{-1} & \alpha & \alpha^{-2} \\ \alpha^{-3} & \alpha^{-2} & \alpha^{-1} & 1 \end{array} \right] \]
\[ \left[ \begin{array}{c} 1 & 1 & 1 & 1 \\ \alpha & \alpha & \alpha & \alpha \\ \alpha^{-1} & \alpha^{-1} & \alpha^{-1} & \alpha^{-1} \end{array} \right] \]

IEE PROCEEDINGS, Vol. 131, Part F, No. 1, FEBRUARY 1984
where $\otimes$ means term-by-term multiplication.

Notice that, for an actual implementation, all coefficients must be scaled to integers and they must be within the allowable dynamic range [17, 18]. The $CON$ in eqn. 40 and ‘1/4’ in eqn. 41 should be absorbed inside the $W_{3s}$, in which case no extra multiplication is required, and actually this is the original idea of the Winograd-Fourier-transform algorithm.

For our discussion so far, we have combined one of our basic FNT modules with one short DFT algorithm. However, in general, one may choose to combine the FNT modules to form a long transform or indeed any number of short DFT or FNT modules can be combined to form long DFT transforms using the prime factor algorithm or the Winograd-Fourier-transform algorithm. The only restriction is that the sequence lengths of these short DFT algorithms and FNT modules must be mutually prime.

4 Hybrid implementation

It is seen that both the prime factor algorithm and the Winograd-Fourier-transform algorithm can combine with eight FNT modules to compute long DFTs. PFA is especially efficient for sequence lengths of 2 and 4. WFTA is very efficient for lengths 3 and 8, hence it is worthwhile to use a combination of these methods to compute very long DFT sequences.

Table 3 lists the number of real multiplications and real additions for various DFTs computed by this hybrid technique with 5-point, 17-point, 97-point and 193-point FNT modules. In this case either $F_4$ or $F_5$ or $F_6$ can be chosen as the modulo base for the computation, and this results in a flexible choice in word lengths for the implementation. Figs. 1 and 2 give the number of multiplications and additions for WFTA [5], polynomial transform (PT) [9] and for the present DFT computation technique. As can be seen from these Figures, the number of multiplications for the WFTA and the polynomial transform is between 30 and 90% higher than the corresponding number of multiplications for the present method. On the other hand, the present method requires between 20 to 30% more shift-adds than the number of additions in WFTA and polynomial transforms for short sequence lengths. However, for sequence length 680 or above, the number of shift-adds for the present method is equal to or less than the number of additions for WFTA and PT. Hence the present method is more useful for long DFT lengths. It is worth pointing out that, for dedicated hardware or microcomputer implementation, the multiplication time is of the order of ten times or more slower than addition, notwithstanding the relative costs of such operations.

Table 4 lists the number of real operations required for various DFT using the 257-point FNT module. In this case, $F_6$ has to be used as the modulo base for the computation. It is clear from Fig. 1 that the number of multiplications per point remains constant and is equal to unity for even extremely long DFT sequences. Hence the present method gives a very efficient way to compute long DFTs.

5 Conclusion

Eight basic FNT modules for computing DFTs have been presented and used as the basic transform lengths to

**An additional 0.75 shift-add per point is required for modulo $F_6$**

<table>
<thead>
<tr>
<th>DFT length</th>
<th>Prime factors</th>
<th>Number of multiplications per point</th>
<th>Number of shift-adds per point</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>17</td>
<td>0.94</td>
<td>8.47</td>
</tr>
<tr>
<td>34</td>
<td>2 x 17</td>
<td>0.94</td>
<td>9.47</td>
</tr>
<tr>
<td>68</td>
<td>4 x 17</td>
<td>0.94</td>
<td>10.47</td>
</tr>
<tr>
<td>102</td>
<td>2 x 3 x 17</td>
<td>1.00</td>
<td>11.47</td>
</tr>
<tr>
<td>136</td>
<td>8 x 17</td>
<td>1.00</td>
<td>11.72</td>
</tr>
<tr>
<td>204</td>
<td>3 x 4 x 17</td>
<td>1.00</td>
<td>12.47</td>
</tr>
<tr>
<td>408</td>
<td>3 x 8 x 17</td>
<td>1.00</td>
<td>13.72</td>
</tr>
<tr>
<td>680</td>
<td>8 x 17</td>
<td>1.00</td>
<td>15.72</td>
</tr>
<tr>
<td>1020</td>
<td>3 x 4 x 5 x 17</td>
<td>1.00</td>
<td>16.47</td>
</tr>
<tr>
<td>2040</td>
<td>3 x 5 x 8 x 17</td>
<td>1.00</td>
<td>17.72</td>
</tr>
<tr>
<td>4080</td>
<td>3 x 5 x 16 x 17</td>
<td>1.13</td>
<td>19.10</td>
</tr>
<tr>
<td>14280</td>
<td>3 x 5 x 7 x 8 x 17</td>
<td>1.29</td>
<td>22.86</td>
</tr>
<tr>
<td>23160</td>
<td>3 x 8 x 193</td>
<td>1.33</td>
<td>25.83**</td>
</tr>
<tr>
<td>39576</td>
<td>3 x 8 x 17 x 97</td>
<td>1.33</td>
<td>28.24</td>
</tr>
<tr>
<td>78744</td>
<td>3 x 8 x 17 x 193</td>
<td>1.33</td>
<td>30.30**</td>
</tr>
</tbody>
</table>

**Fig. 1** Number of multiplications per point against sequence length for DFTs

(i) computed by FNT with modulo base $F_4$, $F_5$, or $F_6$ (O)
(ii) computed by FNT with modulo base $F_6$ only (•)
(iii) computed by polynomial transform ($\triangle$)
(iv) computed by Winograd-Fourier-transform algorithm ($\triangledown$)

**Fig. 2** Number of shift-adds (or additions) per point against sequence length for DFTs

(i) computed by FNT with modulo base $F_4$, $F_5$, or $F_6$ (O)
(ii) computed by FNT with modulo base $F_6$ only (•)
(iii) computed by polynomial transform ($\triangle$)
(iv) computed by Winograd-Fourier-transform algorithm ($\triangledown$)
compute long discrete Fourier transforms. The number of multiplications per point for most cases is not more than one and the number of shift-adds is almost equal to (for long sequence lengths, less than) the number of additions in the Winograd-Fourier-transform algorithm and the polynomial transform. Thus the present technique is very effective in computing discrete Fourier transforms.

Some possible transform lengths based on the eight basic FNT modules have been listed, but this listing is no means exhaustive in that the user may combine different short DFTs to form other transform lengths as required. Hence this method gives a large class of transform lengths for computing discrete Fourier transforms.

Fermat number transforms have already been used to compute convolutions effectively. These are mainly implemented through dedicated hardware or modules (with add-on hardware sometimes) written in assembly language for a computer/microcomputer. Hence, with just some slight modifications of these modules, one may compute a discrete Fourier transform using FNT on the same system. However, since most computer/microcomputer hardware is not designed to perform modulo Fermat number arithmetic, it is perhaps desirable to fabricate digital signal-processor chips with FNT arithmetic which can facilitate the computation of DFT and convolution using FNT. The major problem left to be solved is concerned with the design of the architecture of the digital signal processor.

6 Acknowledgment

The authors wish to thank the referees for their helpful comments and suggestions.

7 References


7 RADER, C.M.: 'Discrete Fourier transforms when the number of data samples is prime', Proc. IEEE, 1966, 56, pp. 1107–1108


22 DICKSON, L.E.: 'History of the theory of numbers', (Carnegie Institute, 1919)


Table 4: Number of operations per point for DFTs computed by FNT with modulo base $f_n$ only

<table>
<thead>
<tr>
<th>DFT length</th>
<th>Prime factors</th>
<th>Number of multiplications per point</th>
<th>Number of shift-adds per point</th>
</tr>
</thead>
<tbody>
<tr>
<td>257</td>
<td>257</td>
<td>1.00</td>
<td>17.7</td>
</tr>
<tr>
<td>514</td>
<td>$2 \times 257$</td>
<td>1.00</td>
<td>18.7</td>
</tr>
<tr>
<td>1028</td>
<td>$4 \times 257$</td>
<td>1.00</td>
<td>19.7</td>
</tr>
<tr>
<td>1542</td>
<td>$3 \times 257$</td>
<td>1.00</td>
<td>20.7</td>
</tr>
<tr>
<td>2056</td>
<td>$8 \times 257$</td>
<td>1.00</td>
<td>20.9</td>
</tr>
<tr>
<td>3084</td>
<td>$3 \times 4 \times 257$</td>
<td>1.00</td>
<td>21.7</td>
</tr>
<tr>
<td>6148</td>
<td>$3 \times 8 \times 257$</td>
<td>1.00</td>
<td>22.9</td>
</tr>
<tr>
<td>7710</td>
<td>$2 \times 5 \times 257$</td>
<td>1.00</td>
<td>24.7</td>
</tr>
<tr>
<td>10280</td>
<td>$5 \times 8 \times 257$</td>
<td>1.00</td>
<td>24.9</td>
</tr>
<tr>
<td>15420</td>
<td>$3 \times 4 \times 5 \times 257$</td>
<td>1.00</td>
<td>25.7</td>
</tr>
<tr>
<td>30840</td>
<td>$3 \times 5 \times 8 \times 257$</td>
<td>1.00</td>
<td>26.9</td>
</tr>
<tr>
<td>52428</td>
<td>$3 \times 17 \times 257$</td>
<td>1.00</td>
<td>30.2</td>
</tr>
</tbody>
</table>

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 16, 2009 at 10:01 from IEEE Xplore. Restrictions apply.