A NEW FFT ALGORITHM AND ITS IMPLEMENTATION ON THE DSP96002

Domingo Rodriguez

Electrical and Computer Engineering Department
University of Puerto Rico, Mayagüez Campus
Mayagüez, PR 00709-5000

ABSTRACT

This work presents a new algorithm for computing the discrete Fourier transform (DFT). This fast Fourier transform (FFT) algorithm is obtained through decomposition of the Fourier matrix representing the DFT operator into a product of sparse matrices not all square matrices. The algorithm is based on additive properties of the input and output indexing sets of the Fourier transformation. Mathematical formulations of the algorithm are presented using tensor product algebra. Properties of this algebra are used to assist in the adaptation of the algorithm to the DSP96002 microprocessor architecture. This results in efficient implementations which take into account the inherent software and hardware features of this microprocessor.

1. Introduction

Computing the DFT y of an n-point complex sequence x is equivalent to the computation of the matrix product

$$y = F_n \cdot x \tag{1}$$

where F_n is the Fourier matrix of order n defined by

$$F_n = [w_n^{kl}]_{\substack{0\leq k<n \\ 0\leq l<n}}, \qquad w_n = e^{-2\pi j/n}, \quad j = \sqrt{-1} \tag{2}$$

Straightforward computation (direct method) of *Eq.* (1) requires n^2 multiplications and $n(n-1)$ additions. When n is a composite number of the form $n = r^2 \cdot m$, r and m any positive integers greater than one, the Fourier matrix F_n may be factored into a product of sparse matrices, not all square, which reduces the number of arithmetic computations (additions and multiplications) during a DFT calculation. We set out to describe this factorization technique by using properties of tensor product algebra. We first present some definitions. Then, we proceed to describe the algorithm for the specific case of $n = 2^2 \cdot m$. This is done for the purpose of clarity; but, then the algorithm is generalized.

Define the tensor product $A \otimes B$ of the matrices A and B as $A \otimes B = [a_{(i,j)} \cdot B]$; thus,

$$A \otimes B = \begin{bmatrix} a_{(0,0)}B & \cdots & a_{(0,s-1)}B \\ a_{(1,0)}B & \cdots & a_{(1,s-1)}B \\ \vdots & \ddots & \vdots \\ a_{(r-1,0)}B & \cdots & a_{(r-1,s-1)}B \end{bmatrix} \tag{3}$$

where $a_{(i,j)}$ is the entry in the i-th row and j-th column of the matrix

$$A = [a_{(i,j)}]_{\substack{0\leq i<r \\ 0\leq j<s}} \tag{4}$$

$$A \otimes B = C = [a_{(p,q)}] \otimes [b_{(r,s)}] = [c_{(t,u)}] \tag{5}$$

where

$$c_{(t,u)} = c_{(p,r;q,s)} = a_{(p,q)} \cdot b_{(r,s)} \tag{6}$$

The double indexation of the entries of the C matrix is enacted lexicographically.

Define the $m \times 1$ vector U_m as

$$U_m = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{m-1} \end{bmatrix}, \qquad u_0 = u_1 = u_2 = \ldots = u_{m-1} = 1 \tag{7}$$

If A_n is a square matrix of order n, then

$$U_m^T \otimes A_n = \Big[\underbrace{A_n \;\; A_n \;\; A_n \;\; \ldots \;\; A_n}_{m}\Big] \tag{8}$$

If A_n is symmetric, then

$$(U_m^T \otimes A_n)^T = U_m \otimes A_n^T = U_m \otimes A_n = \begin{bmatrix} A_n \\ A_n \\ A_n \\ \vdots \\ A_n \end{bmatrix} \tag{9}$$

Define I_n as the identity matrix of order n. Thus, we have

$$(I_s \otimes F_r) = diag[F_r \; F_r \; \ldots \; F_r] \tag{10}$$

A stride by s permutation matrix $P_{n,s}$ of order $n = r \cdot s$ is defined by

$$P_{n,s} \cdot d = [d_0, d_s, d_{2s}, \ldots, d_1, d_{s+1}, \ldots, d_{(r-1)s+(s-1)}]^T \quad (11)$$

$$d = [d_0, d_1, d_2, \ldots, d_{s-1}, d_s, d_{s+1}, \ldots, d_{(r-1)s+(s-1)}]^T \quad (12)$$

When $n = 2s$, the permutation matrix $P_{n,s}$ is called the perfect-shuffle matrix. If $s = 2$, $P_{n,2}$ is called the even-odd permutation matrix.

Using the matrix equality

$$P_{n,s}^T = P_{n,s}^{-1} = P_{n,r}, \qquad n = r \cdot s, \quad (13)$$

the following result is established:

$$(P_{n,s} \cdot d)^T = d^T \cdot P_{n,s}^{-1} = d^T \cdot P_{n,r} = [d_0, d_s, \ldots, d_{n-1}] \quad (14)$$

When the Fourier matrix F_n, of order $n = r \cdot s$, is multiplied on the right by the permutation matrix $P_{n,r}$, the product is defined as the matrix $C_{n,s}$:

$$C_{n,s} = F_n P_{n,r} = F_n P_{n,s}^{-1} \quad (15)$$

The diagonal matrix $D_{n,r}$, of order r is defined by

$$D_{n,r} = \text{diag}\left[1, w_n, w_n^2, w_n^3, \ldots, w_n^{r-1}\right], \qquad w_n = e^{-2\pi j/n} \quad (16)$$

The twiddle factor (phase factor) matrix $T_{n,s(r)}$, of order n, is defined as the direct sum of s diagonal matrices $D_{n,n/s}$ of order $r = \frac{n}{s}$:

$$T_{n,s(r)} = \sum_{j=0}^{s-1} \oplus D_{n,n/s}^j \quad (17)$$

If $n = r.s$, then

$$T_{n,s(n)} = T_{n,s} = \sum_{j=0}^{s-1} \oplus D_{n,n/s}^j = \sum_{j=0}^{s-1} \oplus D_{n,r}^j \quad (18)$$

2. Description of the $n = 2^2 \cdot m$ Case

<u>*Observation :*</u> If $n = 4m$, then

$$F_n = (U_{2m}^T \otimes F_2 \otimes I_{2m})(T_{mn,2m(n)})(I_{2m} \otimes U_m \otimes F_2)P_{n,2m} \quad (19)$$

<u>*Proof :*</u>

Direct matrix multiplication shows that the matrix $C_{n,2m}$ can be factorized into a product of an $n \times mn$ matrix G times an $mn \times n$ matrix H:

$$C_{n,2m} = G_{n \times mn} \cdot H_{mn \times n} \quad (20)$$

Here, we set the matrix H to be

$$H_{mn \times n} = (I_{2m} \otimes U_m \otimes F_2) \quad (21)$$

and

$$G_{n \times mn} = \begin{bmatrix} I_{2m} & D_{n,2m} & \cdots & D_{n,2m}^{2m-1} \\ I_{2m} & w^{2m} D_{n,2m} & \cdots & w^{2m} D_{n,2m}^{2m-1} \end{bmatrix} \quad (22)$$

We then proceed to extract a special twiddle factor $T_{mn,2m(n)}$ from $G_{n \times mn}$:

$$G_{n \times mn} = \begin{bmatrix} I_{2m} & I_{2m} & I_{2m} & \cdots & I_{2m} \\ I_{2m} & w^{2m} I_{2m} & I_{2m} & \cdots & w^{2m} I_{2m} \end{bmatrix} \cdot \begin{bmatrix} I_{2m} & & & & \\ & D_{n,2m} & & & \\ & & D_{n,2m}^2 & & \\ & & & \ddots & \\ & & & & D_{n,2m}^{2m-1} \end{bmatrix} \quad (23)$$

Rewriting the above matrices in tensor products form, we obtain:

$$G_{n \times mn} = (U_m^T \otimes F_2 \otimes I_{2m})(T_{mn,2m(n)}) \quad (24)$$

Since,

$$F_n = C_{n,2m} P_{n,2m} = G_{n \times mn} H_{mn \times n} P_{n,2m}, \quad (25)$$

we get the desired result

$$F_n = (U_m^T \otimes F_2 \otimes I_{2m})(T_{mn,2m(n)})(I_{2m} \otimes U_m \otimes F_2)P_{n,2m} \quad (26)$$

3. Variants of the Algorithm

Variants of the $2^2 \cdot m$ algorithm may be formulated by using some properties of tensor products algebra. For instance, we can expressed $Eq.$(19) in a form which clearly identifies the number additions required for performing the overall computation. We present a couple of examples. The first example is a decimation in time (DIT) tensor product formulation. The second example is a decimation in frequency (DIF) formulation. We proceed to describe the basic operations performed on the matrix $H_{mn \times n}$ and the matrix product $G_{n \times mn} T_{mn,2m(n)}^{-1}$ to arrive at the desired result.

$$\begin{aligned} H_{mn \times n} &= (I_{2m} \otimes U_m \otimes F_2) \\ &= I_{2m} \otimes (U_m \cdot U_1 \otimes I_2 \cdot F_2) \\ &= I_{2m} \otimes (U_m \otimes I_2) \cdot (U_1 \otimes F_2) \\ &= I_{2m} \cdot I_{2m} \otimes (U_m \otimes I_2) \cdot F_2 \\ &= (I_{2m} \otimes U_m \otimes I_2)(I_{2m} \otimes F_2) \end{aligned} \quad (27)$$

$$\begin{aligned} G_{n \times mn} T_{mn,2m(n)}^{-1} &= (U_m^T \otimes F_2 \otimes I_{2m}) \\ &= (U_1^T \cdot U_m^T \otimes F_2 \cdot I_2) \otimes I_{2m} \\ &= (U_1^T \otimes F_2)(U_m^T \otimes I_2) \otimes I_{2m} \\ &= F_2 \cdot (U_m^T \otimes I_2) \otimes I_{2m} \cdot I_{2m} \\ &= (F_2 \otimes I_{2m})(U_m^T \otimes I_2 \otimes I_{2m}) \end{aligned} \quad (28)$$

Thus,

$$F_n = F_{4m} = (F_2 \otimes I_{2m})((U_m^T \otimes I_2) \otimes I_{2m}) \cdot$$

$$T_{mn,2m(n)}(I_{2m} \otimes (U_m \otimes I_2)(I_{2m} \otimes F_2)P_{4m,2m} \quad (29)$$

This expression of the F_{4m} matrix identifies the number of required additions to be $2(2 \cdot 2m) = 8m$. From the twiddle factor matrix we can also identify the number of required multiplications to be $4m(m-1)$.

By taking the matrix transpose of F_{4m}, we obtain decimation in frequency algorithms: From $Eq.$(19),

$$F_{4m} = P_{4m,2}(I_{2m} \otimes U_m^T \otimes F_2) T_{mn,2m(n)} (U_m \otimes F_2 \otimes I_{2m}) \quad (30)$$

From (30),

$$F_{4m} = P_{4m,2}(I_{2m} \otimes F_2)(I_{2m} \otimes (U_m^T \otimes I_2)) \cdot$$
$$T_{mn,2m(n)}((U_m \otimes I_2) \otimes I_{2m})(F_2 \otimes I_{2m}) \quad (31)$$

4. General $r^2 \cdot m$ Algorithm

Here we present a generalization of the $2^2 \cdot m$ algorithm for the case when $n = r^2 \cdot m$ and r is any integer. This will result in a two-factor formulation ($n = r \cdot s$) of the algorithm. We can use this formulation in a recursive manner to obtain a general formulation of the algorithm for the case when n is a composite of the form $n = a_{k-1}^{2^{k-1}} \cdot a_{k-2}^{2^{k-2}} \cdots a_2^{2^2} \cdot a_1^2 \cdot a_0$. For the two-factor formulation we use a "divide and conquer" approach where the input and output indexing sets are represented as two dimensional arrays. We use this indexation scheme in *Eq.*(1) and properties of tensor product algebra to obtain the desired representation.

We start by considering the length of the input sequence x to be a composite integer of the form $n = r \cdot s$. Using the identities $\ell = \ell_1 + s\ell_0$ and $k = k_1 + sk_0$, the index product $k\ell$ given in *Eq.*(1) is expanded into

$$k\ell = (k_1 + sk_0)(\ell_1 + s\ell_0) = k_1\ell_1 + sk_1\ell_0 + sk_0\ell_1 + s^2k_0\ell_0 \quad (32)$$

where $0 \leq k_1, \ell_1 < s$ and $0 \leq k_0, \ell_0 < r$. Substituting this product expansion into *Eq.*(1) results in the following expression

$$y(k_1, k_0) = \sum_{\ell_1=0}^{s-1} \sum_{\ell_0=0}^{r-1} w_n^{(k_1\ell_1 + sk_1\ell_0 + sk_0\ell_1 + s^2k_0\ell_0)} x(\ell_1, \ell_0), \quad (33)$$

which we can rewrite as

$$y(k_1, k_0) = \sum_{\ell_1=0}^{s-1} w_r^{k_0\ell_1} \left\{ w_{rs}^{k_1\ell_1} \left[\sum_{\ell_0=0}^{r-1} w_r^{k_1\ell_0} x_1(\ell_0, \ell_1) \right] \right\} \quad (34)$$

Throughout, we think of the sequences x and y as one-dimensional vectors with a two-dimensional indexing scheme. This allows us to relate the sequences $x_1(\ell_1, \ell_0)$ and $x(\ell_0, \ell_1)$ through the identity

$$P_{n,s} x(\ell_1, \ell_0) = x_1(\ell_0, \ell_1), \quad 0 \leq \ell_0 < r,\ 0 \leq \ell_1 < s \quad (35)$$

We let $s = r \cdot m$. After some manipulation, we obtain the following expression for the inner bracket of *Eq.*(34)

$$y_1(k_1, \ell_1) = (U_m \otimes F_r) x_1(\ell_0, \ell_1) = (U_m \otimes I_r) F_r x_1(\ell_0, \ell_1) \quad (36)$$
$$y_1(k_1, \ell_1) = (I_s \otimes U_m \otimes I_r)(I_s \otimes F_r) x_1(\ell_0, \ell_1) \quad (37)$$

The twiddle or phase-factor $w_n^{k_1\ell_1}$ can be expressed in matrix form in the following manner

$$T_{s^2,s(n)} = \sum_{j=0}^{s-1} \oplus \left[D_{n,m}^r \otimes D_{n,r} \right]^j \quad (38)$$

Thus, we can write

$$y_2(k_1, \ell_1) = T_{s^2,s(n)} y_1(k_1, \ell_1) \quad (39)$$

Finally, the expression

$$y(k_1, k_0) = \sum_{\ell_1=0}^{s-1} w_r^{k_0\ell_1} y_2(k_1, \ell_1) \quad (40)$$

is formulated in tensor product notation in the following way

$$y(k_1, k_0) = (F_r \otimes I_s)((U_m^T \otimes I_r) \otimes I_s) y_2(k_1, \ell_1) \quad (41)$$

Combining, we obtain the general expression of the $r^2 \cdot m$ algorithm

$$y(k_1, k_0) = (F_r \otimes I_s)((U_m^T \otimes I_r) \otimes I_s) \cdot$$
$$T_{s^2,s(n)}((I_s \otimes U_m) \otimes I_r)(I_s \otimes F_r) P_{n,s} x(k_1, k_0) \quad (42)$$

5. DSP Microprocessor Implementation

Analysis of the implementation process guided us to devote a close attention to the manner in which twiddle or phase-factor operations are performed. For the special case of the $2^2 \cdot m$ algorithm, the major bulk of the multiplication operations reside in the computation of the twiddle factors. This can be evidenced in *Eq.*(19) and *Eq.*(29) above. For this reason special macros are developed for the generation of lookup tables. Properties of tensor or Kronecker product are used to go from *Eq.*(19) to *Eq.*(29). The latter equation reveals the hidden butterfly operations. These operations are performed using butterfly kernels built using the FADDSUB instruction as the basic building block.

We looked very closely at implementation procedures for the expressions $((U_m^T \otimes I_2) \otimes I_{2m})$ and $(I_{2m} \otimes (U_m \otimes I_2))$. They, in essence, control the data flow of the algorithm. The stride permutation matrix $P_{4m,2m}$ is simply the perfect-shuffle permutation. This operation was implemented using special properties of the Address Generation Unit (AGU).

We then proceeded to identify efficient implementation procedures for expressions of the form $(I_s \otimes F_r)$ and $(F_s \otimes I_r)$ which we call Fourier factors. These factors play a very important role during the implementation of linear, shift-invariant, finite impulse response (LSI-FIR) filters as can be seen in the following mathematical description of these filters using tensor product algebra.

Suppose that the sequence x is the input to the LSI-FIR system T_h. The output sequence y is given by

$$y = T_h(x) = h \bigcirc_n x \quad (43)$$

where $\bigcirc_n$ denotes cyclic convolution.

Applying (DFT) operator (represented by the Fourier matrix F_n) to the above expression results in

$$F_n(y) = F_n(T_h(x)) = F_n(h) \odot F_n(x) \quad (44)$$

This is the same as

$$(F_n \circ T_h)(x) = (F_n(h) \odot F_n)(x) \tag{45}$$

here, $\circ$ denotes composition of operators, and $\odot$ represents a point-wise multiplication of two sequences. Since the choice of x was arbitrary, we obtain the following important result

$$F_n \odot T_h = F_n(h) \odot F_n \tag{46}$$

Emphasizing the diagonalization of T_h by the action of the DFT operator gives us the following expression

$$F_n T_h F_n^{-1} = F_n(h) \odot I_n \tag{47}$$

The expression $F_n(h) \odot I_n$ is denoted by $D_{F(h)} = D_{\widehat{h}}$, where a matrix representation of $D_{\widehat{h}}$ is given by

$$D_{\widehat{h}} = \begin{bmatrix} \widehat{h}_0 & & & & \\ & \widehat{h}_1 & & & \\ & & \widehat{h}_2 & & \\ & & & \ddots & \\ & & & & \widehat{h}_{n-1} \end{bmatrix}, \widehat{h}_j = (F_n(h))(j) \tag{48}$$

Here, $j \in Z/n = \{0, 1, 2, \ldots, n-1\}$ Thus, we can write

$$T_h = F_n^{-1} D_{\widehat{h}} F_n \tag{49}$$

This last expression serves as the basis for the formulations of LSI-FIR systems using FFT algorithms. This is accomplished by computing, both, the discrete Fourier transform and its inverse using fast algorithms. For example Suppose that h is the impulse response of the LSI-FIR system T_h. If n is a composite of the form $n = r \cdot s$, $(s = r \cdot m)$ we can factor the matrix $H = F_n^{-1} D_{\widehat{h}} F_n$ representing the system T_h in a form which uses the $r^2 \cdot m$ decimation in frequency (DIF) algorithm for the computation of the Fourier transform; and the $r^2 \cdot m$ decimation in time (DIT) algorithm for the computation of its inverse. This will allow as to precompute $\widetilde{D}_{\widehat{h}} \equiv P_{n,s} D_{\widehat{h}} P_{n,s}^{-1}$

$$\widetilde{D}_{\widehat{h}} = diag\left[P_{n,s} \cdot \widehat{h}\right] = \begin{bmatrix} \widehat{h}_0 & & & & \\ & \widehat{h}_s & & & \\ & & \widehat{h}_{2s} & & \\ & & & \ddots & \\ & & & & \widehat{h}_{n-1} \end{bmatrix} \tag{50}$$

We can partition the matrix H so that we obtain a block circulant matrix with circulant blocks. This partition is always circulant. The block size can be chosen to be either r or s. Thus writing a a partition of H into submatrices H_j, $0 \le j < r$ of block size s results in the following

$$H = \sum_{j \in Z/r} S_r^j \otimes H_j \tag{51}$$

The submatrices, as we have stated above, are circulant; hence, they may represent lower order LSI-FIR systems.

Diagonalizing the shift operator S_r and the matrices H_j simultaneously produces the following known result

$$H = \sum_{j \in Z/r} [F_r^{-1} D_{\widehat{S_r \delta}} F_r] \otimes [F_s^{-1} D_{\widehat{h}}(j) F_s] \tag{52}$$

where $D_{\widehat{S_r \delta}}$ represents the diagonalized shift operator S_r, δ is the unit sample sequence of length r, and $D_{\widehat{h}}(j)$ is a diagonal matrix of order s corresponding to the diagonalization of the circulants H_j. Using properties of tensor product algebra, we obtain

$$H = \sum_{j \in Z/r} (F_r^{-1} \otimes F_s^{-1})[D_{\widehat{S_r \delta}} \otimes D_{\widehat{h}}(j)](F_r \otimes F_s) \tag{53}$$

Also

$$H = (F_r \otimes F_s)^{-1}[\sum_{j \in Z/r} (D_{\widehat{S_r \delta}} \otimes D_{\widehat{h}}(j))](F_r \otimes F_s) \tag{54}$$

Using, again, tensor product properties, we write

$$H = \mathcal{B}_{[r,s]}^{-1}\left[\sum (D_{\widehat{S_r \delta}} \otimes D_{\widehat{h}}(j)\right] \mathcal{B}_{[r,s]} \tag{55}$$

where

$$\mathcal{B}_{[r,s]} \equiv (F_r \otimes I_s)(I_r \otimes F_s) = (I_r \otimes F_s)(F_r \otimes I_s) \tag{56}$$

6. Conclusion

An algorithm has been presented for the computation of the discrete Fourier transform. Tensor product algebra was used as a tool to aid in the implementation of this algorithm on the DSP96002 microprocessor. We used properties of this algebra to match this algorithm to the architecture of this microprocessor, exploiting its inherent software and hardware features. There exists a clear demarcation between the various processing stages of this algorithm, i.e., pre-addition, data routing, multiplication, etc. This feature makes it suitable for expressing the algorithm in signal flow graph (SFG) form. This, in turn, helps in identifying feasible implementations for real time processing.

Acknowledgement:

We acknowledge the assistance provided by the undergraduate students Carlos Cabrera and Jaime Plá during the algorithm implementation process.

REFERENCES:

[1] D. Rodríguez, "Tensor Products Formulations of Additive Fast Fourier Transform Algorithms and Their Implementations," Ph.D. Thesis, City University of New York, Feb. 1988.

[2] J. Johnson, R.W. Johnson, D. Rodríguez, R. Tolimieri, "A Methodology for Designing, Modifiying, and Implementing Fourier Transform Algorithms on Various Architectures," accepted for publication in the Journal of Circuits, Systems, and Signal Processing, Birkhäuser.