Multidimensional Systems and Signal Processing, 5, 61–74 (1994)

2-D FFT Algorithm by Matrix Factorization in a 2-D Space

M. WANG* AND E.B. LEE†
Control Science and Dynamical Systems Center, University of Minnesota, Minneapolis, MN 55455

Received November 4, 1991; Revised February 16, 1993

Abstract. A new 2-D FFT algorithm is described. This algorithm applies a 2-D matrix factorization technique in a 2-D space and offers a way to do 2-D FFT in both dimensions simultaneously. The computation is greatly reduced compared to traditional algorithms. This will improve the realization of a 2-D FFT on any kind of computer. However its good parallelism will especially benefit an implementation on a computer with hypercube architecture. A good arrangement of parallel processors will save a great deal of running time. Furthermore this algorithm can be extended to M-D cases for $M > 2$.

Key Words: Fast fourier transform, parallel computation, multidimensional

1. Introduction

The discrete-time Fourier transform (DFT) plays an important role in almost every field of science and engineering. There is large demand for efficient algorithms, which leads to a study of the fast Fourier transform (FFT) realization of the DFT. Early in 1968, Theilheimer formulated the FFT transform by matrix factorization [1]. The technique is to factorize a DFT matrix as the product of a series of binary operation matrices, which produces a FFT with 2 as radix. Every binary operation matrix can be represented as a matrix with all elements except two being zero in each row. This can be represented by a butterfly signal flow graph. Late Kahaner completed a matrix description of the block diagonal factorization [2] for DFT with all different radicies. When take 2 as a radix, this actually is Sande-Tukey canonical form [3].

Historically the earliest article about the multidimensional DFT is probably by Andrews and Pratt [4]. 1D butterfly SFG is applied by Jagadeesan [5] for both of 2-D and 3-D cases. In the recent two decades many different algorithms for multidimensional DFT have been suggested to reduce the computation and time cost. For example the polynomial transform algorithm [6]–[7], the vector-radix algorithm [8] and the prime factor algorithm [9]. All these algorithms can be looked as some particular factorizations of a DFT matrix. However, all the authors for multidimensional DFT limited their attention in one dimensional space. The reason probably is because of the restriction of the architecture of computers of the past decades.

*Supported by NSF Grant CCR-8813493.
†Supported by Grants DMS-8607687, DMS-8722402, and DMS9002019.

Now a new generation of computers is knocking at our door. The charm of the new generation of computers is the parallelism which makes it possible to handle a very large amount of information simultaneously.

How to handle the data and the information in a more parallel way? This is the motivation to develop the new 2-D FFT algorithm suggested in this article. This new algorithm breaks the one dimensional space restriction and considers the task of a 2-D DFT in a 2-D space. In a 2-D space, a 2-D FFT SFG appears as a square-butterfly. This implies a possible way to perform the 2-D FFT in both dimensions simultaneously. The benefit is a big reduction of computation and time consumption.

In the light of this technique, a M-D FFT algorithm can also be implemented in a M-D space to gain better efficiency for $M > 2$. In fact the higher that M is the more efficient the reduction of computation is.

2. Description of the algorithm

Consider a matrix description of the DFT. Suppose that **W** is a DFT operation matrix, with $N = b^k$. It can be factored [2] as

$$\mathbf{W} = B \begin{bmatrix} \Delta^{(k-1)} & 0 & \cdots & 0 \\ 0 & \Delta^{(k-1)} & \cdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \Delta^{(k-1)} \end{bmatrix} \cdots \begin{bmatrix} \Delta^{(2)} & 0 & \cdots & 0 \\ 0 & \Delta^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Delta^{(2)} \end{bmatrix} \begin{bmatrix} \Delta^{(1)} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \Delta^{(1)} \end{bmatrix} [\Delta^{(0)}], \tag{1}$$

where

$$\Delta^{(j)} = \begin{bmatrix} I & I & \cdots & I \\ D^{(j)} & \Theta D^{(j)} & \cdots & \Theta^{b-1} D^{(j)} \\ \vdots & \vdots & \cdots & \vdots \\ [D^{(j)}]^{b-1} & [\Theta D^{(j)}]^{b-1} & \cdots & [\Theta^{b-1} D^{(j)}]^{b-1} \end{bmatrix}. \tag{2}$$

The size of $\Delta^{(j)}$ is $N/b^j \times N/b^j$, b is the base. For example, when $b = 2$, the size of $\Delta^{(k-1)}$ is $N/2^{k-1} \times N/2^{k-1} = 2 \times 2$ and the size of $\Delta^{(0)}$ is $N \times N$.

$$D^{(j)} = \text{diag.} \left[1 \quad a^{b^j} \quad a^{2b^j} \quad \cdots \quad a^{(b^{k-j-1}-1)b^j}\right], \quad j = 0, 1, \ldots, k-1. \tag{3}$$

The size of $D^{(j)}$ is $N/b^{j+1} \times N/b^{j+1}$.

$$\Theta = \exp\left(\frac{i2\pi}{b}\right), \quad a = \exp\left(\frac{i2\pi}{N}\right). \tag{4}$$

Here b is the base with $N = b^k$, and B is a permutation matrix which converts the data from normal order to bit-reversed order.

Take $N = 4$ as an example. When $b = 2$ and $k = 2$, then $\Theta = \exp(i2\pi/2) = -1$, $a = \exp(i2\pi/4) = e^{i\pi/2} = i$, and

$$D^{(0)} = \text{diag}\,[1 \quad a] = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad \text{and} \quad \Delta^{(0)} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & i & 0 & -i \end{bmatrix};$$

$$D^{(1)} = [1], \quad \text{and} \quad \Delta^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{so } \mathbf{W}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & i & 0 & -i \end{bmatrix}.$$

This representation for $\mathbf{W}_4$ can be realized by a butterfly interconnection which is one of the four canonical forms of the FFT realization of the DFT, (due to Sande [10]). Moreover $\mathbf{W}$ is a symmetric matrix, $\mathbf{W} = \mathbf{W}^T$.

From above example it can be seen that when the base $b = 2$, the expression of the matrix factorization has a particularly simple form—bidiagonal form, i.e., each of $\Delta^{(j)}$ is bi-diagonal shown as below:

$$\Delta^{(j)} = \begin{bmatrix} I & I \\ D^{(j)} & \Theta D^{(j)} \end{bmatrix}. \tag{5}$$

It is this bidiagonal form that offers a great benefit to implement a 2D DFT with more parallelism, less computation and lower time cost.

The factorization in a DFT is not unique. Different factorizations will produce distinct matrix descriptions. Here the focus will be put on the Sande-Tukey first canonical form. All the results in this article can be applied to the Sande-Tukey second canonical form as well.

Assume a 2-D sequence with dimensions $N_1 = N_2 = N = 2^k$ for simplicity, and let $\mathcal{X}$ denote the data sequence.

$$\mathcal{X} = \begin{bmatrix} x(0, 0) & x(0, 1) & \cdots & x(0, N-1) \\ x(1, 0) & x(1, 1) & \cdots & x(1, N-1) \\ \vdots & \vdots & \cdots & \vdots \\ x(N-1, 0) & x(N-1, 1) & \cdots & x(N-1, N-1) \end{bmatrix}.$$

Now let $\mathbf{X}$ denote its DFT

$$\mathbf{X} = \begin{bmatrix} X(0,0) & X(0,1) & \cdots & X(0,N-1) \\ X(1,0) & X(1,1) & \cdots & X(1,N-1) \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ X(N-1,0) & X(N-1,1) & \cdots & X(N-1,N-1) \end{bmatrix}.$$

where

$$X(k_1, k_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) a^{n_1 k_1} a^{n_2 k_2}.$$

$\mathbf{X}$ can be obtained by transforming $\mathcal{X}$, both its columns and its rows which can be described as $\mathbf{X} = \mathbf{W}\,\mathcal{X}\,\mathbf{W}$. Then by symmetry of $\mathbf{W}$, $\mathbf{X} = \mathbf{W}\,\mathcal{X}\,\mathbf{W}^T$. Using Kahaner's matrix description

$$\mathbf{X} = B \begin{bmatrix} \Delta^{(k-1)} & 0 & . & 0 \\ 0 & \Delta^{(k-1)} & . & 0 \\ 0 & 0 & . & 0 \\ . & . & . & . \\ . & . & . & . \\ 0 & 0 & . & \Delta^{(k-1)} \end{bmatrix} \cdots \begin{bmatrix} \Delta^{(1)} & 0 \\ 0 & \Delta^{(1)} \end{bmatrix} [\Delta^{(0)}] \mathcal{X} [\Delta^{(0)}]^T \begin{bmatrix} \Delta^{(1)} & 0 \\ 0 & \Delta^{(1)} \end{bmatrix}^T$$

$$\cdots \begin{bmatrix} \Delta^{(k-1)} & 0 & . & 0 \\ 0 & \Delta^{(k-1)} & . & 0 \\ 0 & 0 & . & 0 \\ . & . & . & . \\ . & . & . & . \\ 0 & 0 & . & \Delta^{(k-1)} \end{bmatrix}^T B^T.$$

this new algorithm executes the FFT in the following order:

Stage 1: Compute

$$\mathbf{Z_1} = [\Delta^{(0)})\ \mathcal{X}\ [\Delta^{(0)}]^T;$$

Stage 2: Compute

$$\mathbf{Z_2} = \begin{bmatrix} \Delta^{(1)} & 0 \\ 0 & \Delta^{(1)} \end{bmatrix} Z_1 \begin{bmatrix} \Delta^{(1)} & 0 \\ 0 & \Delta^{(1)} \end{bmatrix}^T;$$

Stage k: Compute

$$\mathbf{Z_k} = \begin{bmatrix} \Delta^{(k-1)} & 0 & . & 0 \\ 0 & \Delta^{(k-1)} & . & 0 \\ . & . & . & . \\ . & . & . & . \\ 0 & 0 & . & \Delta^{(k-1)} \end{bmatrix} \mathbf{Z_{k-1}} \begin{bmatrix} \Delta^{(k-1)} & 0 & . & 0 \\ 0 & \Delta^{(k-1)} & . & 0 \\ . & . & . & . \\ . & . & . & . \\ 0 & 0 & . & \Delta^{(k-1)} \end{bmatrix}^{\tau}.$$

Then $\mathbf{X} = B\mathbf{Z_k}B^{\tau}$.

We now discuss the algorithm in more detail.

Stage 1: Compute Z_1. Let

$$\mathcal{X} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}. \tag{6}$$

There are two digits for the location of each block. When the first digit is 0, the block is on the left part; when it is 1, the block is on the right half part. Similarly, when the second digit is 0, the block is on the upper half part, and 1, for the lower. Then

$$\mathbf{Z}_1 = \begin{bmatrix} I & I \\ D^{(0)} & -D^{(0)} \end{bmatrix} \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix} \begin{bmatrix} I & I \\ D^{(0)} & -D^{(0)} \end{bmatrix}^{\tau}$$

$$= \begin{bmatrix} (Y_{00}+Y_{01})+(Y_{10}+Y_{11}) & (Y_{00}-Y_{01})+(Y_{10}-Y_{11})]D^{(0)} \\ D^{(0)}[(Y_{00}+Y_{01})-(Y_{10}+Y_{11})] & D^{(0)}[(Y_{00}-Y_{01})-(Y_{10}-Y_{11})]D^{(0)} \end{bmatrix}. \tag{7}$$

Since $D^{(0)}$ is diagonal,

$$Y..D^{(0)} = [Y..(i, j)a^{j-1}], \tag{8}$$

$$D^{(0)}Y.. = [Y..(i, j)a^{i-1}], \tag{9}$$

$$D^{(0)}Y..D^{(0)} = [Y..(i, j)a^{i+j-2}], \tag{10}$$

where $Y..(i, j)$ is the element of $Y..$ on the ith row and the jth column, a^{j-1} is the element of $D^{(0)}$ on the jth row and the jth column and so on.

The diagonal form of $D^{(0)}$ implies that each element of $D^{(0)}Y..D^{(0)}$ can be computed as $Y..(i, j)$ by a^{i+j-2}. It means that in (10) the multiplication only need to be done once instead of twice. The multiplier a^{i+j-2} can be precomputed. This is the main point in reducing computation: merge two multiplications as one.

The expression of (7) shows that stage 1 can be decomposed into three steps. At step 1 compute

$$Y_{00}^{(1)} = Y_{00} + Y_{01}, \quad Y_{01}^{(1)} = Y_{00} - Y_{01}, \quad Y_{10}^{(1)} = Y_{10} + Y_{11}, \quad Y_{11}^{(1)} = Y_{10} - Y_{11}. \tag{11}$$

At step 2 compute

$$Y_{00}^{(2)} = Y_{00}^{(1)} + Y_{10}^{(1)}, \quad Y_{01}^{(2)} = Y_{01}^{(1)} + Y_{11}^{(1)},$$
$$Y_{10}^{(2)} = Y_{00}^{(1)} - Y_{10}^{(1)} \quad \text{and} \quad Y_{11}^{(2)} = Y_{01}^{(1)} - Y_{11}^{(1)}. \tag{12}$$

Then compute the multiplications

$$Y_{01}^{(3)} = Y_{01}^{(2)}D^{(1)}, \quad Y_{10}^{(3)} = D^{(1)}Y_{10}^{(2)}, \quad Y_{11}^{(3)} = D^{(1)}Y_{11}^{(2)}D^{(1)}, \tag{13}$$

at step 3, thus obtaining

$$\mathbf{Z_1} = \begin{bmatrix} Y_{00}^{(3)} & Y_{01}^{(3)} \\ Y_{10}^{(3)} & Y_{11}^{(3)} \end{bmatrix}.$$

The computation in steps 1 or 2 concludes with N^2 additions and the computation in step 3 concludes $\frac{3}{4}N^2$ multiplications. The total computation at stage 1 is N^2 (2 additions + $\frac{3}{4}$ multiplications).

Stage 2: Compute Z_2. Note that

$$\mathbf{Z_2} = \begin{bmatrix} \Delta^{(1)} & 0 \\ 0 & \Delta^{(1)} \end{bmatrix} \mathbf{Z_1} \begin{bmatrix} \Delta^{(1)} & 0 \\ 0 & \Delta^{(1)} \end{bmatrix}^{\tau}$$

where $\mathbf{Z_1}$ has been determined at stage 1.

Let

$$\mathbf{Z_1} = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix},$$

Then

$$\mathbf{Z_2} = \begin{bmatrix} \Delta^{(1)}Z_{11}\Delta^{(1)\tau} & \Delta^{(1)}Z_{12}\Delta^{(1)\tau} \\ \Delta^{(1)}Z_{21}\Delta^{(1)\tau} & \Delta^{(1)}Z_{22}\Delta^{(1)\tau} \end{bmatrix}. \tag{14}$$

It is interesting that $\mathbf{Z_1}$ has been quartered and the operation upon each of these four blocks, say upon Z_{11}, is independent from those upon Z_{12}, Z_{21} and Z_{22}. This independency is a good property which avoids sending data between different blocks. In this way a 2-D FFT has been split into four small size 2-D FFTs.

Now set

$$Z_{i,j} = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

and compute

$$\Delta^{(1)}Z_{i,j}\Delta^{(1)^T} = \begin{bmatrix} I & I \\ D^{(1)} & -D^{(1)} \end{bmatrix} \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix} \begin{bmatrix} I & I \\ D^{(1)} & -D^{(1)} \end{bmatrix}^T.$$

The results can also be obtained by repeating the procedure in stage 1 for all $i, j = 1, 2$. The difference of $Y_{\cdot\cdot}$'s here from those in stage 1 is that the size of $Y_{\cdot\cdot}$ here is $N/4$ by $N/4$ instead of $N/2$ by $N/2$, because each of these $Z_{i,j}$'s is only a quarter of $\mathbf{Z_1}$.

Consider now a more general situation. Let

$$\bar{i}_m = i|_{\text{mod}N/2^{m-1}}, \qquad \bar{j}_m = j|_{\text{mod}N/2^{m-1}}, \tag{15}$$

$$i_m = \begin{cases} 0 & \text{if } \bar{i}_m \le N/2^m, \\ (i|_{\text{mod}N/2^{m-1}}) - 1, & \text{otherwise.} \end{cases} \tag{16}$$

$$j_m = \begin{cases} 0 & \text{if } \bar{j}_m \le N/2^m, \\ (j|_{\text{mod}N/2^{m-1}}) - 1 & \text{otherwise.} \end{cases} \tag{17}$$

Using above notations, the multiplier can be expressed as

$$M_{ij}^{(m)} = a^{2^{m-1}(i_m + j_m)}. \tag{18}$$

At stage $m + 1$

$$\mathbf{Z_{m+1}} = \begin{bmatrix} \Delta^{(m)}Z_{11}^{(m)}\Delta^{(m)^T} & \Delta^{(m)}Z_{12}^{(m)}\Delta^{(m)^T} & \cdot & \Delta^{(m)}Z_{1r}^{(m)}\Delta^{(m)^T} \\ \Delta^{(m)}Z_{21}^{(m)}\Delta^{(m)^T} & \Delta^{(m)}Z_{22}^{(m)}\Delta^{(m)^T} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \Delta^{(m)}Z_{r1}^{(m)}\Delta^{(m)^T} & \cdot & \cdot & \Delta^{(m)}Z_{rr}^{(m)}\Delta^{(m)^T} \end{bmatrix}. \tag{19}$$

Here $\mathbf{Z_{m+1}^{(m)}}$ has $r \times r$ blocks, $r = 2^m$. In each of these blocks

$$Z_{i,j}^{(m)} = \begin{bmatrix} Y_{00}^{(m)} & Y_{01}^{(m)} \\ Y_{10}^{(m)} & Y_{11}^{(m)} \end{bmatrix}_{i,j}, \qquad i, j = 1, 2, \ldots, r.$$

The size of the blocks at stage m is $N/2^{k-m+1}$, $k = \log_2 N$. After $\mathbf{Z_k}$ is determined, $\mathbf{X}$ can be obtained by permuting rows and columns: $\mathbf{X} = B\mathbf{Z_k}B^T$.

3. SFG implementation of 2-D DFT and computation

Using the Sande-Tukey canonical form an 1-D DFT can be implemented by a 1D butterfly. Now by the new algorithm a 2-D DFT can also be implemented by a 2-D square butterfly.

Jagadeesan arranged the 2-D data in one line, i.e., expanded the 2-D data by Kronecker matrix product [11] then applied 1D butterfly. In this way a 2D FFT contains 2 $\log_2 N = 2k$ stages. Each stage contains one step for additions and one step for multiplications. The new algorithm in this paper arranges the 2-D data in a plane and forms a square butterfly for the 2D SFG. When $N_1 = N_2 = N = 2^k$, there are k stages. Each stage has three steps. Step 1 can be computed on a butterfly in column form. Step 2 will be accomplished on a butterfly in row form. Then step 3 performs the multiplications. Take $k = 3$ and $b = 2$ as an example. Figure 1 shows the butterfly in stage 1, and Figure 2 shows the multiplier matrix with respect to the data array. Figures 3 and 4 are for stage 2. Each block in stage 2 forms an independent square butterfly. No data communication is needed between these four blocks. In stage 3 there are also two butterfly sets, one for columns and the other for rows. $\mathbf{Z}_2$ will be split into 16 blocks, each has size 2×2. The SFGs for other stages are similar.

What is the benefit of this algorithm in computation?

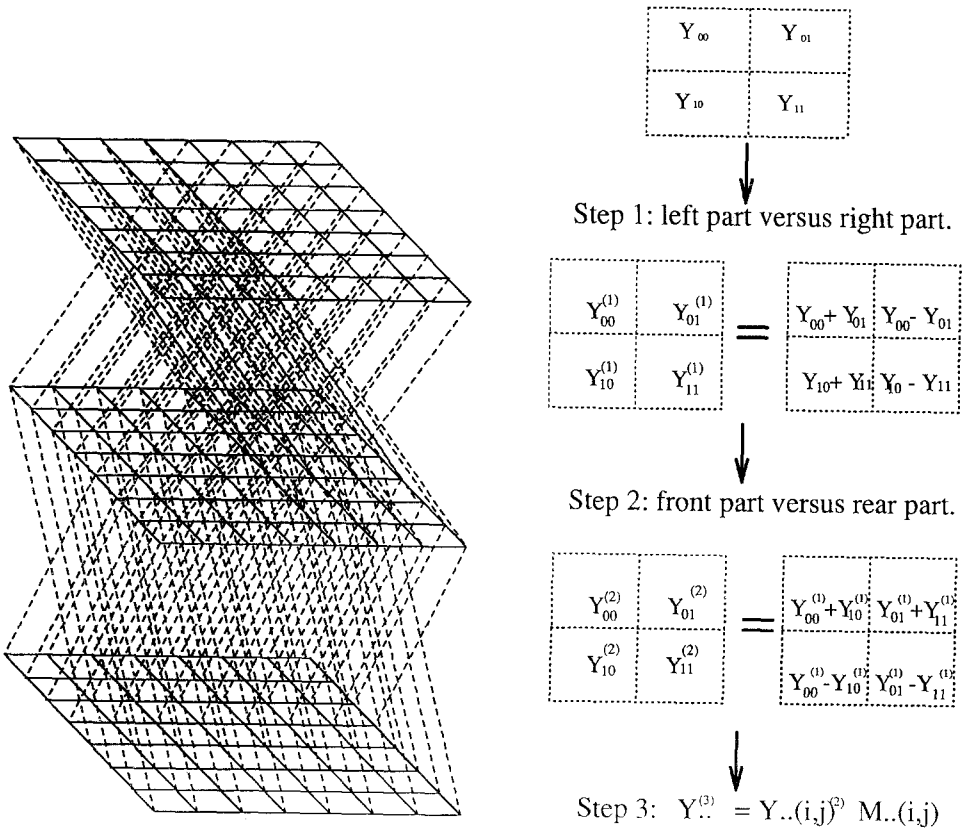


Figure 1. 2D SFG, stage 1.

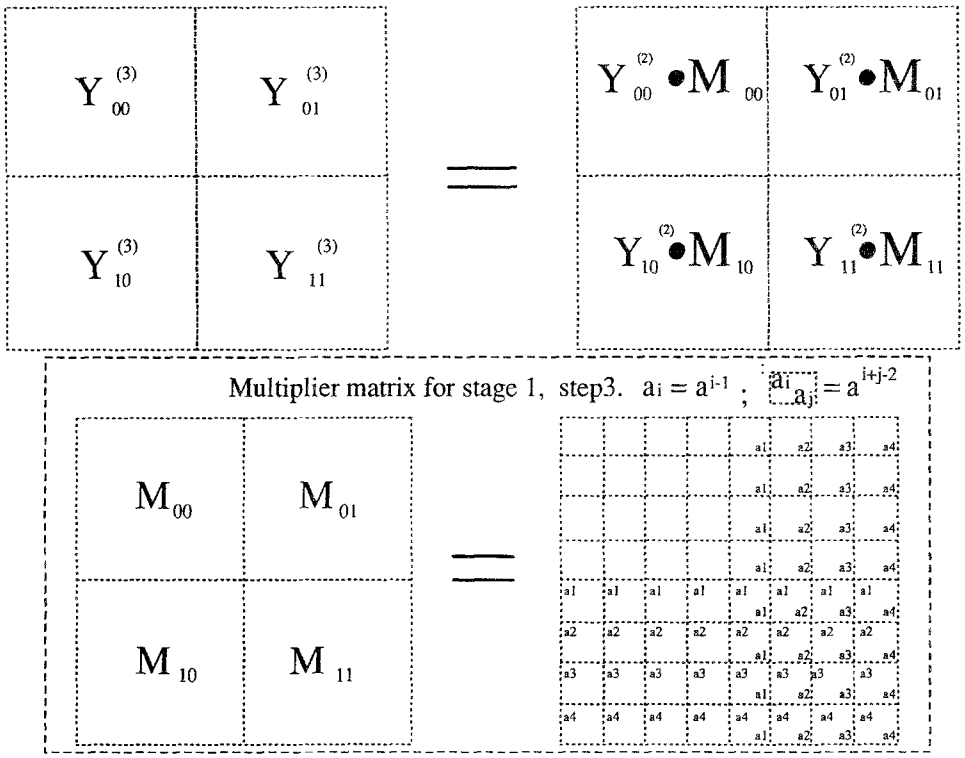
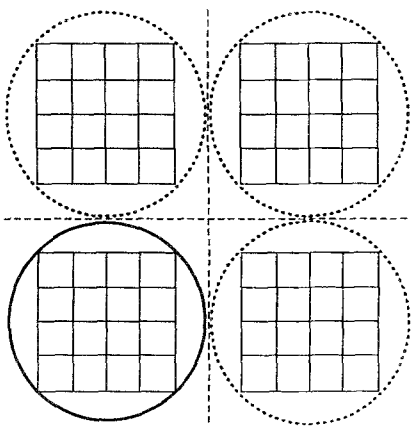
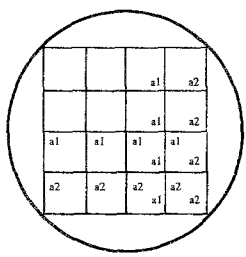


Figure 2. Stage 1, step 3: Y..(i, j)$^{(3)}$ = Y..(i, j)$^{(2)}$ M..(i, j)





Multiplier matrix for each of the right blocks

At the stage 2, the data matrix is split into four blocks. Each one is a smaller square-butterfly. All the operations and multipliers are the same for each of these square-butterflies at the stage 2.

Figure 3.

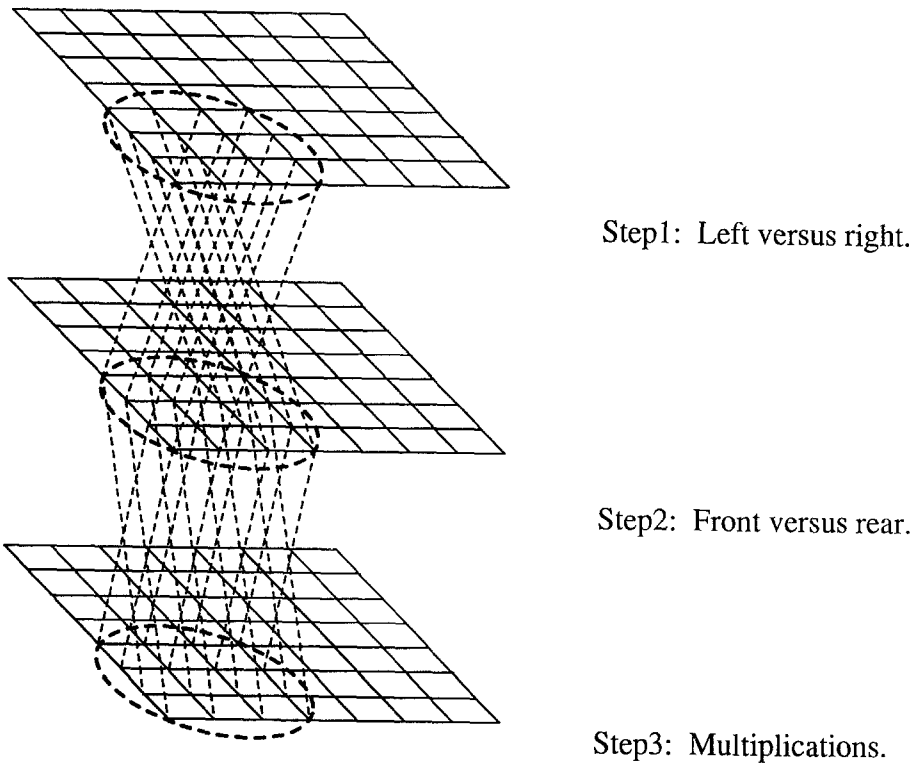


Figure 4. 2D SFG stage 2.

For a 2-D sequence, with $N_1 = N_2 = N = 2^k$, using this algorithm there are k stages and each stage contains three steps. At the first two steps there is only addition computation and the total additions required are $2N^2$. At the third step there are $\frac{3}{4}N^2$ multiplications required. So the total computation for all these stages is $\log_2 N(2N^2$ additions + $\frac{3}{4}N^2$ multiplications).

If using 1-D butterfly for the 2-D FFT there are $2k$ stages and each stage has to execute both additions and multiplications. The total computation will be $2\log_2 N(N^2$ additions + $\frac{1}{2}N^2$ multiplications).

On a parallel computer, the computation time for the new algorithm is almost one half of that for the 1-D butterfly realization.

There is something more interesting than the reduction of computation—2-D data array. The data are stored as a matrix. The form is more compact than as a long vector, so the distance is shorter between processors when the computation is distributed in parallel processors. This implies time saving in communication when the algorithm is implemented on a parallel computer. The computation complexity is improved and the efficiency can also be improved due to the computation reduction, since the square data array improves the environment of the implementation on a parallel computer. The relative problems about scalability and load balancing for a M-D FFT on a hypercube architecture computer will be discussed in subsequent articles.

4. *M*-D case

In the *M*-D case, if $N_1 = N_2 = \cdots = N_M = N$,

$$\mathbf{X} = \mathbf{W}^{[M]} \cdots \mathbf{W}^{[3]}\mathbf{W}^{[2]}\mathbf{W}^{[1]}\mathcal{X}$$

where $\mathbf{W}^{[i]}$ denotes the operation on the ith dimension. It is clear that the order of operations will not affect the result except the arrangement of the elements of $\mathbf{X}$, because

$$\mathbf{X}(k_1, k_2, k_3) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \cdots \sum_{n_M=0}^{N-1} \mathcal{X}(n_1, n_2, \ldots, n_M) a^{k_1 n_1} a^{k_2 n_2} \cdots a^{k_M n_M}$$

$$= \sum_{n_2=0}^{N-1} \cdots \sum_{n_M=0}^{N-1} \sum_{n_1=0}^{N-1} \mathcal{X}(n_1, n_2, \ldots n_M) a^{k_2 n_2} a^{k_3 n_3} \cdots a^{k_M n_M} a^{k_1 n_1}.$$

Define $\mathbf{W}^{[M]} = BH_{k-1}^{[M]} \cdots H_2^{[M]} H_1^{[M]} H_0^{[M]}$, where

$$H_m^{[M]} = \begin{bmatrix} \Delta^{(m)} & 0 & \cdot\cdot & 0 \\ 0 & \Delta^{(m)} & \cdot\cdot & 0 \\ \cdot & \cdot & \cdot\cdot & \cdot \\ \cdot & \cdot & \cdot\cdot & \Delta^{(m)} \end{bmatrix}$$

operating on the Mth dimension.

Let $B^{[M]}$ denote the permutation to the Mth dimension, then

$$\begin{aligned} \mathbf{X} &= B^{[M]} \cdots B^{[3]} B^{[2]} B^{[1]} H_{k-1}^{[M]} \cdots H_2^{[M]} H_1^{[M]} H_0^{[M]} \cdots H_{k-1}^{[3]} \cdots H_2^{[3]} H_1^{[3]} H_0^{[3]} \\ &\quad \times H_{k-1}^{[2]} \cdots H_2^{[2]} H_1^{[2]} H_0^{[2]} H_{k-1}^{[1]} \cdots H_2^{[1]} H_1^{[1]} H_0^{[1]} \mathcal{X} \\ &= B^{[M]} \cdots B^{[3]} B^{[2]} B^{[1]} H_{k-1}^{[M]} \cdots H_{k-1}^{[3]} H_{k-1}^{[2]} H_{k-1}^{[1]} \cdots H_2^{[M]} \cdots H_2^{[3]} H_2^{[2]} H_2^{[1]} \\ &\quad \times H_1^{[M]} \cdots H_1^{[3]} H_1^{[2]} H_1^{[1]} H_0^{[M]} \cdots H_0^{[3]} H_0^{[2]} H_0^{[1]} \mathcal{X} \end{aligned}$$

In a similar way there are k stages:

Stage 1: Compute $\mathbf{Z_1} = H_0^{[M]} \cdots H_0^{[3]} H_0^{[2]} H_0^{[1]} \mathcal{X}$,

Stage 2: Compute $\mathbf{Z_2} = H_1^{[M]} \cdots H_1^{[3]} H_1^{[2]} H_1^{[1]} \mathbf{Z_1}$,

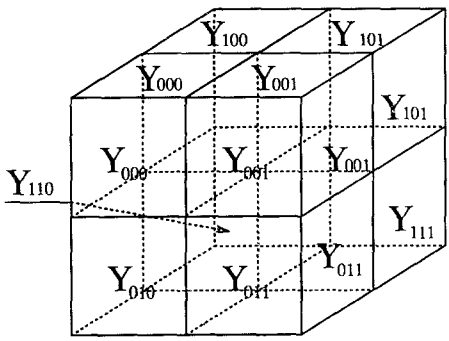
$\vdots$

Stage k: Compute $\mathbf{Z_k} = H_{k-1}^{[M]} \cdots H_{k-1}^{[3]} H_{k-1}^{[2]} H_{k-1}^{[1]} \mathbf{Z_{k-1}}$.

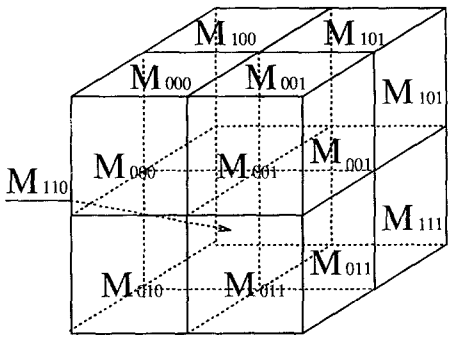
$$\mathbf{X} = B^{[M]} \cdots B^{[3]} B^{[2]} B^{[1]} \mathbf{Z_k}.$$

Take $M = 3$ as an example. The SFG is a cube-butterfly. Figure 5 shows the data and multiplier arrays. When $M = 3$ there are three steps in each stage. Stage 1 will split data into $2^3 = 8$ independent blocks with equal size. Stage 2 will split data into $(2^3)^2 = 64$ ...; Stage k will split data into $(2^3)^{2^{k-1}}$ blocks with size 2×2. Each stage can be decomposed into $M + 1 = 4$ steps. Step 1 to 3 compute the butterfly of additions with respect to every of the three dimensions. Step 4 computes the multiplications which is just simple dot-product with respect to every element of a cube data array. Figure 6 shows the four steps in the first stage of a cube-butterfly computation. The computation in each stage is N^3 (additions $+ \frac{7}{8}$ multiplications). When $M = 3$ there are still $\log_2 N = k$ stages. The total computation is $k\ N^3$(3 additions $+ \frac{7}{8}$ multiplications). If using 1D butterfly there will be $3kN^3$ multiplications. If using Sande-Tukey's canonical form there still are $3k/2 \times N^3$ multiplications. The time saving is more obvious. By the new algorithm the time cost for each processor is $\log_2 N$ (additions $+ \frac{7}{8}$ multiplications), only one third as much as by a 1-D butterfly.

From above observation it follows easily that more saving for both computation and time cost can be obtained when $M \geq 3$.



3D data array.



Multiplier array for stage 1, step 4.

Figure 5.

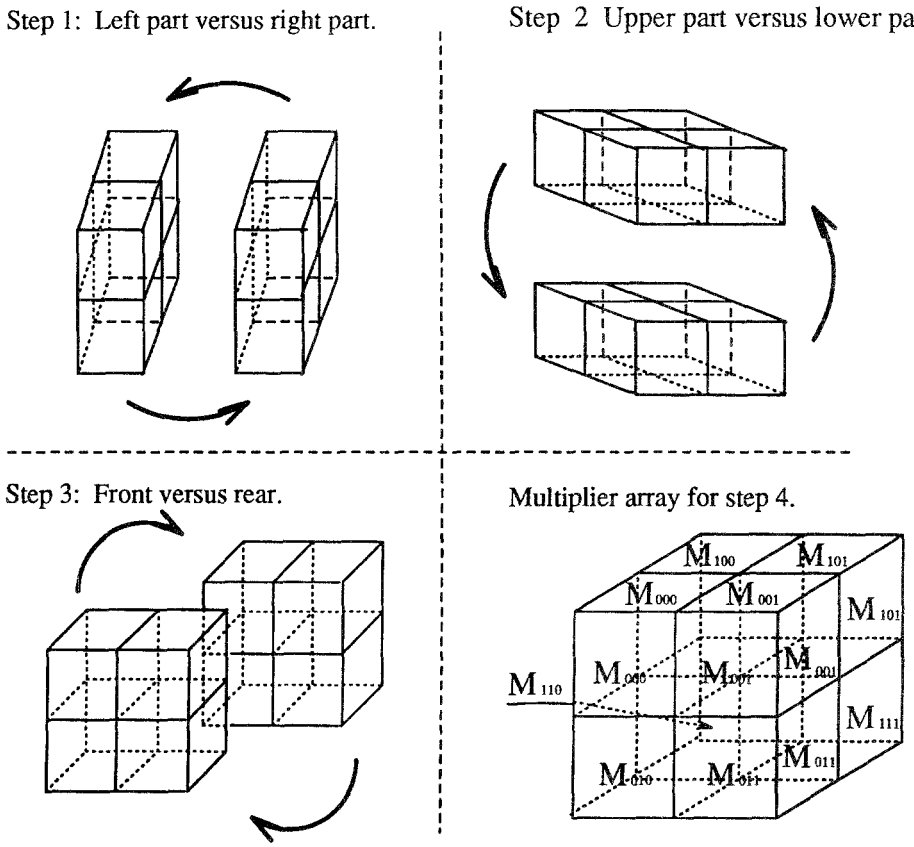


Figure 6. 3D SFG, stage 1.

References

1. F. Theilheimer, "A Matrix Version of the Fast Fourier Transform," *IEEE Trans.*, vol. AU-17, no. 2, 1969, pp. 158–161.
2. D.K. Kahaner, "Matrix Description of the Fast Fourier Transform," *IEEE*, vol. AU-18, no. 4, December 1970, pp. 442–450.
3. E.O. Brigham, "The Fast Fourier Transform," Englewood Cliffs, NJ: Prentice-Hall, 1974.
4. H.C. Andrews, and W.K. Pratt, "Digital Computer Simulation of Coherent Optical Processing Operations," *Computer Group News*, November 1968, pp. 12–19.
5. M. Jagadeesan, "*N*-Dimensional Fast Fourier Transform," *13th Midwest Symp. Circuit Theory*, Minneapolis, MN, May, 1970, pp. 3.2.1–3.2.8.
6. H.J. Nussbaumer, "New Algorithms for Convolution and DFT Based on Polynomial Transforms," *1978 Proc. IEEE Int. Conf. ASSP*, 1978, pp. 638–641.
7. H.J. Nussbaumer, and P. Quandalle, "Fast Computation of Discrete Fourier Transforms Using Polynomial Transforms," *IEEE*, vol. ASSP-27, no. 2, 1979, pp. 169–181.
8. D.B. Harris, J.H. McClellan, D.S.K. Chan, and H.W. Schuessler, "Vector Radix Fast Fourier Transform," *1977 IEEE Int. Conf. ASSP Record*, pp. 548–551.

9. A. Guessoum, and R.M. Merserean, "Fast Algorithms for the Multidimensional Discrete Fourier Transform," *IEEE*, vol. ASSP-34, no. 4, August 1986, pp. 937-943.
10. W.M. Gentleman, and G. Sande, "The Fast Fourier Transform for Fun and Profit," *AFIPS Proc., 1966 Fall Joint Computer Conf.* vol. 29, Washington, D.C., pp. 563-678.
11. A. Granham, "Kronecker Products and Matrix Calculus with Applications," New York: Ellis Horwood 1981.