IEEE TRANSACTIONS ON

ACOUSTICS, SPEECH, AND SIGNAL PROCESSING

FEBRUARY 1990 VOLUME 38 NUMBER 2 (ISSN 0096-3518)

A PUBLICATION OF THE IEEE ACOUSTICS, SPEECH, AND SIGNAL PROCESSING SOCIETY

PAPERS

CORRESPONDENCE

Fast Fourier Transforms Over Finite Groups by Multiprocessor Systems

TATYANA D. ROZINER, MARK G. KARPOVSKY, SENIOR MEMBER, IEEE, AND
LAZAR A. TRACHTENBERG, SENIOR MEMBER, IEEE

Abstract—**This paper presents a method for an optimal implementation of General Discrete Fourier Transform (GDFT) algorithms over finite groups (Abelian and non-Abelian) in a multiprocessor environment. Tradeoffs between hardware complexity/speed and computation time are investigated for different multiprocessor implementations with local nonshared memories (unibus, complete communication network). Formulas are presented for the number of arithmetic operations, for the number of interprocessor data transfers, and for the number of communication links among the processors.**

I. Introduction

FAST Fourier Transforms (FFT) are well known to play an important role in many application fields, such as spectral analysis, digital filtering, image processing, video transmission, etc. The increasing requirements of speed in many real-time applications stimulated in recent years the development of a number of new very fast FT algorithms as well as numerous investigations on comparative complexity of different FFT versions [2], [6], [18], [19], [23]. A systematic approach to the problems of design of FFT and complexity evaluation was developed by Beth [2], [3] who generalized the results formulated in [1], [5]–[7], [12] having considered the classical FDFT as a special case of GDFT (General Discrete Fourier Transform) based on the theory of representations of finite groups. The powerful mathematical apparatus allowed Beth to obtain complexity evaluations for different existing FT algorithms as well as to design new versions of very fast GDFT algorithms for uniprocessor systems in case of decomposable group structure introduced on the input data set. The methods of the theory of representations of finite groups (Abelian and non-Abelian) were also used by other researchers [9]–[14], [28], [29] for different applications related to FFT.

As noted in [6], [19], and [20] and substantiated in [2], the decrease in the number of multiplications in a very fast FT algorithm involves inevitably an increase in the number of additions and/or preprocessing operations. From the viewpoint of further advance as to speed increase of FFT, and with high progress in VLSI technology, the multiprocessor highly parallel systems become an attractive alternative compared to uniprocessor architectures. Many recent publications suggested a wide variety of multiprocessor implementations for FFT, among them numerous works on systolic arrays and array processors with different interconnection networks (e.g., [17], [24]–[27], [29]–[32]).

In a multiprocessor environment, a new factor arises that may strongly influence the efficiency of FFT algorithm performance. The structure of the algorithm involves numerous interprocessor data transfers which can, in case of inappropriate or too slow processor communication network, become a reason for a degradation of performance. Even in uniprocessor systems, the interregister data transfers, loads, stores, and data copying operations may take a considerable part of the total execution time (up to 80 percent, in case of certain architectures, as shown in [21]). The evaluations performed in [22] showed that the minimum number of the interregister data transfers in case of uniprocessor system is at least of the same order as the number of arithmetic operations.

Algorithm-independent upper bounds on complexity including the evaluation of the number of interprocessor transfers were determined in [23] for the classical FFT algorithm performance in terms of multiprocessor communication network design. From the viewpoint of group theory approach, the results of [23] apply to the case of the cyclic group structure only with multiple processors performing the group algorithm.

The subject of the present paper is the generalization of the results mentioned above for the case of finite decomposable (possibly non-Abelian) group structure introduced on the input data set, for the problems of GDFT spectrum calculations by a multiprocessor SIMD system with nonshared memory. The investigation of tradeoffs between the number of operations (including data transfers) needed for GDFT over arbitrary finite groups and hardware complexity (multiple processors and different communication networks) is performed. It is shown for sample evaluations with different group structures that the use of non-Abelian groups (e.g., quaternions) may result in many cases in optimal (fastest) performance of GDFT.

Manuscript received November 18, 1987; revised May 10, 1989. This work was supported by the Office of Naval Research under Naval Research Contract N00014-87-K-0735, and in part by the National Science Foundation under Grant ECS-8512748.

T. D. Roziner and M. G. Karpovsky are with the College of Engineering, Boston University, Boston, MA 02215.

L. A. Trachtenberg is with Drexel University, Philadelphia, PA 19104.

IEEE Log Number 8932777.

0096-3518/90/0200-0226$01.00

II. GDFT as the Generalization of FFT

Discrete Fourier transforms considered in this work are the generalizations of the classical Fourier transforms (FT) in the following sense. For the classical 1-D (one-dimensional) DFT, the vector of input data $f(0), f(1), \cdots, f(N-1)$ is multiplied by the matrix whose elements are the powers of the Nth root of unity, resulting in the vector of N spectrum values:

$$\hat{f}(w) = (1/N) \sum_{m=0}^{N-1} f(m) W_N^{wm}$$

$$\text{where } W_N = \exp(-2\pi j/N); \quad j = \sqrt{-1};$$

$$w = 0, 1, \cdots, N-1.$$

In terms of group theory approach, the matrix of FT corresponds to introducing the cyclic group structure C_N on the set of input data: the set $\{0, 1, \cdots, N-1\}$ can be considered as a group; the group operation $*$ is modulo-N addition (commutative operation). The set of group elements is closed with respect to group operation. Also, for each element a of C_N, its inverse element is defined as a^{-1} such that $a * a^{-1} = 0 = N (\text{mod } N)$. Element 1 is the generator of C_N; any other element can be obtained from 1 by repeated use of group operation.

Example 1: Group C_4 of the order 4 contains four elements: 0, 1, 2, 3. Group operation $*$ is modulo-4 addition. Evidently, for each pair of the group elements, the result of their modulo-4 addition is also the group element, e.g., $1 * 2 = 3$, $2 * 3 = 2$, $3 * 3 = 1$, etc. Inverse of 0 is 0, inverse of 1 is 3, inverse of 2 is 2, inverse of 3 is 1.

For the cyclic group C_N, N distinct roots of unity $1, W_N, W_N^2, \cdots, W_N^{N-1}$ form the set of N distinct 1-D representations. A 1-D representation R is the homomorphism $R: C_N \to \mathbb{C}$ where $\mathbb{C}$ is the field of complex numbers. For any two group elements $a, b \in C_N$, $R(a * b) = R(a) \cdot R(b)$ (usual multiplication). Therefore, for a cyclic group, it is enough to define N distinct 1-D representations for group generator 1.

Example 2: For C_4, $R_0(1) = 1$ (trivial representation); $R_1(1) = W_4 = j$; $R_2(1) = W_4^2 = -1$; $R_3(1) = W_4^3 = -j$. To find $R_3(3)$, the property of the homomorphism can be used: $R_3(3) = R_3(1) \cdot R_3(1) \cdot R_3(1) = (-j)^3 = -j$, etc.

Representation / Group Element x	$R_0(x)$	$R_1(x)$	$R_2(x)$	$R_3(x)$
0	1	1	1	1
1	1	$W_4 = j$	$W_4^2 = -1$	$W_4^3 = -j$
2	1	$W_4^2 = -1$	$W_4^4 = 1$	$W_4^6 = -1$
3	1	$W_4^3 = -j$	$W_4^6 = -1$	$W_4^9 = j$

The table is the usual DFT matrix. To obtain FFT, the inverses of group elements in the leftmost column must be taken (row reordering):

Group Element x	$R_0(x)$	$R_1(x)$	$R_2(x)$	$R_3(x)$
$0 = 0^{-1}$	1	1	1	1
$3 = 1^{-1}$	1	$-j$	-1	j
$2 = 2^{-1}$	1	-1	1	-1
$1 = 3^{-1}$	1	j	-1	$-j$

which is the matrix of FFT (Fast Chrestenson Transform for 4 points; [13], [27], [28]).

The spectrum $\hat{f}(w)$ is obtained by multiplication of vector of data by the matrix given by the table above:

$$\hat{f}(w) = \frac{1}{4} \sum_{x=0}^{3} f(x) R_w(x^{-1}); \quad w = 0, 1, 2, 3. \tag{2.1}$$

The extension of this approach is to introduce more general group structure on the input data set. For a general group structure, its group operation may be noncommutative (non-Abelian groups). The theory of non-Abelian groups allows us to construct generalized transform (GDFT) based on group representations that in the general case may be multidimensional (matrices with complex-valued elements). The GDFT in this case can be also defined by a transform matrix; when the input data vector is multiplied by this matrix, the result is the vector of generalized spectrum elements. The GDFT has properties analogous to DFT (linearity, scaling property, group convolution, Parseval's theorem, etc. [2], [12], [16]). For GDFT over certain non-Abelian groups, the transform matrix has the sparse and simple form; for example, non-Abelian quaternion group Q_2 (of the order 8, that is, a group of 8 elements) implies GDFT matrix 8×8 with 16 zero elements, and the nonzero elements are ± 1, $\pm j$ only, that is, no multiplication is needed (see Section III, Fig. 3).

Multidimensional DFT that is of interest in many applications is defined for m dimensions as [1]:

$$\hat{f}(w_0, w_1, \cdots, w_{m-1}) = \frac{1}{n_0 \cdot n_1 \cdot \cdots \cdot n_{m-1}} \sum_{x_0=0}^{n_0-1} \sum_{x_1=0}^{n_1-1} \cdots \sum_{x_{m-1}=0}^{n_{m-1}-1} \cdot f(x_0, x_1, \cdots, x_{m-1}) \cdot W_{n_0}^{w_0 x_0} W_{n_1}^{w_1 x_1} \cdot \cdots \cdot W_{n_{m-1}}^{w_{m-1} x_{m-1}}, \tag{2.2}$$

where

$$W_{n_i} = \exp(-2\pi j/n_i); \quad 0 \le x_i \le n_i - 1;$$

$$0 \le w_i \le n_i - 1; \quad i = 0, 1, \cdots, m-1;$$

the dimension i has n_i points.

In terms of group theory, multidimensional DFT is equivalent to the FT over the group G with the following structure:

$$G = G_0 \times G_1 \times \cdots \times G_{m-1},$$

where "×" denotes the direct product of cyclic (commutative) groups $G_0, G_1, \cdots, G_{m-1}$. The group theory approach allows us to obtain the matrix of m-dimensional FFT as the Kronecker product ($\otimes$) of FFT matrices for the constituent groups.

Example 3: 3-D FFT (with 2 points in each dimension) can be described in terms of the group structure $G = C_2 \times C_2 \times C_2$. The FFT matrix is the direct product of three matrices $\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$:

$$\left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{array}\right] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

(Fast Walsh Transform [1], [13]).

In this work, even more general multidimensional FFT (i.e., multidimensional GDFT) is considered. The group structure $G = G_0 \times G_1 \times \cdots \times G_{m-1}$ introduced on the N-point data set ($N = n_0 \cdot n_1 \cdot \cdots \cdot n_{m-1}$ where N is the order of group G, that is, the number of elements of G) may include noncommutative groups as well as commutative (e.g., cyclic) ones. That implies that the matrix of GDFT may be expressed in terms of smaller transform matrices (for commutative groups, in terms of their Kronecker product) [8], [12]. These generalized FFT's over finite non-Abelian groups have been widely used in different applications [9], [14], [15]. The rule of multiplication of the input data vector by such a matrix for fast GDFT is equivalent to the factorization of the GDFT matrix. As a consequence, the GDFT algorithm can be performed in m steps. Step number i corresponds to the butterfly algorithm of the GDFT for constituent group G_i performed in parallel on a number of disjoint subsets of data (input data or the intermediate spectra). This creates a possibility of exploiting the parallelism of the algorithm in each step by use of a multiprocessor system. However, multiprocessor execution creates a problem of interprocessor data exchange. Both aspects of multidimensional GDFT (parallelism and additional time needed for data transfers) are considered in the following sections. Estimation will be given for the execution time and for the number of data transfers as a function of group structure and of the number of processors.

III. Evaluation of the Number of Operations Needed for FFT Over Group Structure $G = G_0 \times G_1 \times \cdots \times G_{m-1}$ Using Multiple Processors

Let G be an arbitrary finite group of order $|G| = N$; let V be the vector space of dimension d over the field of complex numbers, and let $GL(V)$ be the group of all nonsingular $d \times d$ matrices with elements in . A representation of G is a homomorphism $R: G \to GL(V)$, that is, for $x, y \in G$, $x * y \in G$, $R(x * y) = R(x) \cdot R(y)$ (where $*$ is the group operation). A representation R is irreducible if there are no nontrivial subspaces of V which are mapped to themselves by all matrices $R(x)$, $x \in G$. Every representation R_w is equivalent to some unitary representation R_u [i.e., to a representation with unitary matrix $R_u(x)$], that is, there exists a $Q \in GL(V)$ such that $R_u(x) = Q^{-1} R_w(x)\, Q$ for all $x \in G$. The set of all irreducible unitary representations for G has the following orthogonality properties [7]. Let $R_w^{(s,t)}$ denote the (s, t)th element of matrix representation $R_w(x)$, and let $R_w(x)$ be of dimension d_w; then

$$\frac{1}{|G|} \sum_{x \in G} R_w^{(s,t)}(x)\, \overline{R_q^{(p,r)}(x)} = \frac{1}{d_w} \delta_{wq} \delta_{sp} \delta_{tr} \quad (3.1)$$

$$\sum_{R_w \in R(G)} d_w \operatorname{Tr} R_w(x) = N \delta_{xe} \quad (3.2)$$

where $R(G)$ is the set of all irreducible unitary representations of G; e is the identity element of G; δ is the Kronecker symbol; $\operatorname{Tr} A$ is the trace of matrix A, and $\overline{A}$ denotes the adjoint matrix (transposed and complex conjugate) of A. The relations (3.1) and (3.2) allow us to define the direct and inverse General Discrete Fourier Transform (GDFT) over G as follows. If f is a function $f: G \to$, then

$$\hat{f}(w) = \frac{d_w}{N} \sum_{x \in G} f(x)\, R_w(x^{-1})$$

$$f(x) = \sum_{R_w \in R(G)} \operatorname{Tr}\left(\hat{f}(w)\, R_w(x)\right) \quad (3.3)$$

where x^{-1} is the inverse of x in G (cf. formula (2.1), Example 2 of Section II where $d_w = 1$).

Let G, $|G| = N$, be a direct product of groups G_j, $0 \le j \le m-1$, where $|G_j| = n_j$, $N = n_0 n_1 \cdot \cdots \cdot n_{m-1}$.

In the case of non-Abelian groups, G_j and consequently G have matrix representations [7], [12], [15].

Denote the elements of G as $X_k \in G$, $0 \le k \le N-1$, and the elements of group G_j as $x_j \in G_j$; $0 \le j \le m-1$. Let $R(G)$ be the set of all irreducible unitary representations of G with elements $R_w \in R(G)$, and let $R(G_j)$ be the set of all irreducible unitary representations of G_j with elements $R_{wj} \in R(G_j)$, $0 \le j \le m-1$ [12].

Following [12], note the following.

1) If $X \in G$, then $X = (x_0, x_1, \cdots, x_{m-1})$, $x_j \in G_j$ (X may be represented as m-tuple of elements of groups G_j, $0 \le j \le m-1$).

2) If $R_w \in R(G)$, then $R_w(X) = R_w(x_0, x_1, \cdots, x_{m-1}) = \bigotimes_{j=0}^{m-1} R_{wj}(x_j)$, where $R_{wj} \in R(G_j)$; symbol $\otimes$ denotes the Kronecker product of matrices. Thus, for every $R_w \in R(G)$, we may denote $w = (w_0, w_1, \cdots, w_{m-1})$.

The direct Fourier transform over group G may be defined as follows [12]:

$$\begin{aligned}\hat{f}(w) &= \hat{f}(w_0, w_1, \cdots, w_{m-1}) \\ &= \frac{d_w}{N} \sum_{x_0} \sum_{x_1} \cdots \sum_{x_{m-1}} f(x_0, x_1, \cdots, x_{m-1}) \\ &\quad \cdot \bigotimes_{j=0}^{m-1} R_{w_j}(x_j^{-1}) \\ &= \frac{d_w}{N} \sum_{x_{m-1}} \left(\cdots \left(\sum_{x_1} \left(\sum_{x_0} (f(x_0, x_1, \cdots, x_{m-1}) \right.\right.\right. \\ &\quad \cdot R_{w_0}(x_0^{-1})) \\ &\quad \left.\left.\left. \otimes R_{w_1}(x_1^{-1}) \right) \otimes \cdots \right) \otimes R_{w_{m-1}}(x_{m-1}^{-1}) \right).\end{aligned} \tag{3.4}$$

The calculation of spectrum $\hat{f}(w_0, w_1, \cdots, w_{m-1})$ is performed in m steps (Step j over group G_j, $0 \le j \le m-1$).

Step 0:

$$\begin{aligned}&f_0(x_0, x_1, \cdots, x_{m-1}) \\ &\quad \triangleq f(x_0, x_1, \cdots, x_{m-1}); \\ &f_1(w_0, x_1, \cdots, x_{m-1}) \\ &\quad = \sum_{x_0} f_0(x_0, x_1, \cdots, x_{m-1}) R_{w_0}(x_0^{-1}).\end{aligned}$$

Step 1:

$$\begin{aligned}&f_2(w_0, w_1, x_2, \cdots, x_{m-1}) \\ &\quad = \sum_{x_1} f_1(w_0, x_1, \cdots, x_{m-1}) \otimes R_{w_1}(x_1^{-1}).\end{aligned}$$

Step j:

$$\begin{aligned}&f_{j+1}(w_0, w_1, \cdots, w_j, x_{j+1}, \cdots, x_{m-1}) \\ &\quad = \sum_{x_j} f_j(w_0, w_1, \cdots, w_{j-1}, x_j, \cdots, x_{m-1}) \\ &\qquad \otimes R_{w_j}(x_j^{-1}).\end{aligned}$$

Step $(m-1)$:

$$\begin{aligned}&f_m(w_0, w_1, \cdots, w_{m-1}) \\ &\quad = \sum_{x_{m-1}} f_{m-1}(w_0, w_1, \cdots, w_{m-2}, x_{m-1}) \\ &\qquad \otimes R_{w_{m-1}}(x_{m-1}^{-1}).\end{aligned}$$

During Step j, the variables $w_0, w_1, \cdots, w_{j-1}, x_{j+1}, \cdots, x_{m-1}$ are fixed, and the summation is performed by variable $x_j \in G_j$.

$$\hat{f}(w_0, w_1, \cdots, w_{m-1}) \triangleq \frac{d_w}{N} f_m(w_0, w_1, \cdots, w_{m-1}).$$

Let X_i be the ith element of G: $X_i \in G$, $0 \le i \le N-1$. It will be convenient to list the elements of G in a certain order. If x_j^i designates the ith element of group G_j, $0 \le i \le n_j - 1$, $0 \le j \le m-1$, let the elements $X_0, X_1, \cdots, X_{N-1}$ of G be ordered as follows:

$$\left.\begin{aligned}X_0 &= (x_0^0, x_1^0, x_2^0, \cdots, x_{m-1}^0) \\ X_1 &= (x_0^1, x_1^0, x_2^0, \cdots, x_{m-1}^0) \\ X_2 &= (x_0^2, x_1^0, x_2^0, \cdots, x_{m-1}^0) \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ X_{N-1} &= (x_0^{n_0-1}, x_1^{n_1-1}, x_2^{n_2-1}, \cdots, x_{m-1}^{n_{m-1}-1}).\end{aligned}\right\} \tag{3.5}$$

Element $X_p = (x_0^{j_0}, x_1^{j_1}, \cdots, x_{m-1}^{j_{m-1}})$ precedes the element $X_q = (x_0^{j_0}, x_1^{j_1}, \cdots, x_{m-1}^{j_{m-1}})$ if $i_0 = j_0$, $i_1 = j_1$, $\cdots$, $i_{k-1} = j_{k-1}$, and $i_k < j_k$ for some k, $0 \le k \le m-1$.

Example 4: Let $G = C_2 \times S_3 \times C_2$; $n_0 = n_2 = 2$, $n_1 = 6$; $N = n_0 n_1 n_2 = 24$; $m = 3$ (S_3 is the permutation group of the order 6). Elements of G are listed as follows (only superscripts of x_j^i are given); group number j, $0 \le j \le 2$, is implicitly determined by the position of superscript in a 3-tuple corresponding to $X_i \in G$.

$$\left.\begin{aligned}X_0 &= (0,0,0) & X_6 &= (0,3,0) & X_{12} &= (0,0,1) & X_{18} &= (0,3,1) \\ X_1 &= (1,0,0) & X_7 &= (1,3,0) & X_{13} &= (1,0,1) & X_{19} &= (1,3,1) \\ X_2 &= (0,1,0) & X_8 &= (0,4,0) & X_{14} &= (0,1,1) & X_{20} &= (0,4,1) \\ X_3 &= (1,1,0) & X_9 &= (1,4,0) & X_{15} &= (1,1,1) & X_{21} &= (1,4,1) \\ X_4 &= (0,2,0) & X_{10} &= (0,5,0) & X_{16} &= (0,2,1) & X_{22} &= (0,5,1) \\ X_5 &= (1,2,0) & X_{11} &= (1,5,0) & X_{17} &= (1,2,1) & X_{23} &= (1,5,1)\end{aligned}\right\} \tag{3.6}$$

The generalized Fourier transform over group $G = G_0 \times G_1 \times \cdots \times G_{m-1}$ is performed in m steps as shown earlier. Step j, $0 \le j \le m-1$, involves the calculation of intermediate spectrum using the FFT algorithm specified by the nature of group G_j. To perform the spectrum calculation of step j, N/n_j subsets of elements of the intermediate spectrum are to be taken as input data for the algorithm of group G_j. The structure of each one of N/n_j subsets is as follows: $\{\tilde{X}_{k_0}, \tilde{X}_{k_1}, \cdots, \tilde{X}_{k_{n_j}}\}$ ($\tilde{X}_i$ denotes an element of intermediate spectrum array), where

$$\left.\begin{aligned}\tilde{X}_{k_0} &= (x_0, x_1, \cdots, x_{j-1}, x_j^0, x_{j+1}, \cdots, x_{m-1}) \\ \tilde{X}_{k_1} &= (x_0, x_1, \cdots, x_{j-1}, x_j^1, x_{j+1}, \cdots, x_{m-1}) \\ \tilde{X}_{k_2} &= (x_0, x_1, \cdots, x_{j-1}, x_j^2, x_{j+1}, \cdots, x_{m-1}) \\ \tilde{X}_{k_{n_j-1}} &= (x_0, x_1, \cdots, x_{j-1}, x_j^{n_j-1}, x_{j+1}, \cdots, x_{m-1}).\end{aligned}\right\} \tag{3.7}$$

that is, each set (3.7) must contain all elements of group G_j in $(j+1)$th position: elements $x_0 \in G_0$, $x_1 \in G_1$, $\cdots$, $x_{j-1} \in G_{j-1}$, $x_{j+1} \in G_{j+1}$, $\cdots$, $x_{m-1} \in G_{m-1}$ are constant for each set (3.7) but different for different sets.

From the viewpoint of practical calculation, we can consider the formally defined procedure of spectrum evaluation (3.4) as the m-step procedure that transforms the input array of N values into the output array of the same size. In the following consideration, we shall denote the elements of input, output, or any intermediate spectrum array by their numbers only, since the values of array elements are not important for our purpose (that is, to trace the data distribution and transfers in case of multiple processors). We shall assume also that for the butterfly algorithm of any constituent group, the numbers of input elements are retained for the outputs. For example, if $\bar{X}_i$, $\bar{X}_k$ are entered in Step j into the algorithm of C_2 group, the resulting outputs $\bar{X}_i + \bar{X}_k$, $\bar{X}_i - \bar{X}_k$ will be considered as updated elements number i and number k, respectively, of the intermediate spectrum array of N elements. (For C_2, $R_w(x) = (-1)^{wx}$; $w, x \in \{0, 1\}$.)

In the following evaluations of the number of operations, we do not take into account the operations needed for spectrum reshuffling and normalization.

Let p denote the number of processors operating in parallel during the calculation of the generalized spectrum. Assume that each processor contains local memory, but there is no global shared memory [23]. In each one of m steps of spectrum calculation, the N elements of the intermediate spectrum are partitioned into disjoint subsets (3.7). In step j, N/n_j subsets (each containing n_j elements) are to be used as input data for butterfly algorithm of the constituent group G_j. In order to avoid idle processors in any step, $N/(pn_j)$ must be an integer for any j, $0 \le j \le m-1$.

Denote the number of operations needed to perform the butterfly algorithm of G_j on a single input data set (3.7) (with one processor) as $L(G_j)$. (In the general case, the algorithm is characterized by the pair of numbers—the number of additions/subtractions and the number of multiplications. We shall assume that $L(G_j)$ is the equivalent number of additions.)

To perform the spectrum calculation over $G = G_0 \times G_1 \times \cdots \times G_{m-1}$ by a single processor, $L(G)$ equivalent operations are needed:

$$L(G) = \sum_{j=0}^{m-1} L(G_j) \prod_{\substack{i=0 \\ i \neq j}}^{m-1} n_i = N \sum_{j=0}^{m-1} \frac{L(G_j)}{n_j}. \quad (3.8)$$

To generalize (3.8) for the case of p parallel processors, denote $L^{(p)}(G)$ the number of operations needed for spectrum calculation over G with p processors. Evidently,

$$L^{(p)}(G) = \frac{L(G)}{p} = \frac{N}{p} \sum_{j=0}^{m-1} \frac{L(G_j)}{n_j}. \quad (3.8a)$$

However, for $p > 1$, the data transfers among the processors are needed (at least in one of the algorithm steps). Therefore, the total time $T^{(p)}(G)$ needed for spectrum calculation over G with p processors is:

$$T^{(p)}(G) = L^{(p)}(G) \cdot t_a + M^{(p)}(G)\, t_c,$$

where t_a is the time needed for one addition/subtraction, t_c is the time needed for one data transfer (communication time), and $M^{(p)}(G)$ is the total number of transfers needed for spectrum calculation over G with p processors. The value of $M^{(p)}(G)$ is not only a function of p and of the group structure for a fixed $|G| = N$, it depends also on the type of the processor communication network.

To minimize $M^{(p)}(G)$, the communication network is needed where each processor can communicate directly with any other processor (if the communication is required by the algorithm), and is able to receive or to transfer a unit of data during communication time t_c. This type of network is of high cost for a large number of processors (although not always the complete network is needed). It seems more reasonable to assume another extreme case when all processors communicate via single bus ("unibus" connection). In the last case, during the transfer time t_c, only one data unit can be transferred from one processor to another. This type of connection puts some restrictions on t_c — the unibus must be fast enough, in order not to increase significantly the total calculation time.

Assume the following.

a) For any j, $0 \le j \le m-1$, the N/pn_j value is an integer (no idle processors in any step). The job load is distributed equally among p processors.

b) The input data are entered into local memories of p processors as follows: first N/p elements of the input array are given to processor 1, next N/p elements to processor 2, etc.

c) For any group G_j (that is, in each step of the algorithm), any set of input data containing n_j elements is delivered to a single processor. It can be shown that the violation of the rule "single group set — single processor" results in a drastic increase in the number of interprocessor communications needed, and in many cases the number of the operations is also growing.

Define two parameters: $p_1 = N/p$ — the number of elements of G per processor; $n_0 \cdot n_1 \cdot \cdots \cdot n_j$ — block size for group G_j, $0 \le j \le m-1$.

The concept of block size will be helpful in evaluation of the number of interprocessor transfers. In consequence of (3.5), the accepted ordering of the elements of G is as follows. For processing over group G_0 (Step 0 of the algorithm), N/n_0 sets of elements of G are to be entered into the algorithm of G_0 as input data. Each set contains n_0 elements, and they follow in succession (again, we list only the numbers of the elements):

$$\{0, 1, 2, \cdots, n_0 - 1\} \quad \text{(Set 0)}$$

$$\{n_0, n_0 + 1, n_0 + 2, \cdots, 2n_0 - 1\} \quad \text{(Set 1)}$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$\{n_0 \cdot n_1 \cdot \cdots \cdot n_{m-1} - n_0, \cdots, n_0 \cdot n_1 \cdot \cdots \cdot n_{m-1} - 1 = N - 1\}$$

$$\text{(Set } (N/n_0) - 1\text{)}.$$

For processing over group G_1 (Step 1), N/n_1 sets of elements of the intermediate spectrum are to be entered into the butterfly algorithm of G_1; each set contains n_1 elements. The elements are chosen "skipping over $n_0 - 1$":

$$\left.\begin{aligned} \{0, n_0, 2n_0, \cdots, (n_1-1)n_0\} &\quad \text{(Set 0)} \\ \{1, n_0+1, 2n_0+1, \cdots, (n_1-1)n_0+1\} &\quad \text{(Set 1)} \\ \{2, n_0+2, 2n_0+2, \cdots, (n_1-1)n_0+2\} &\quad \text{(Set 2)} \\ \{n_0-1, 2n_0-1, 3n_0-1, \cdots, (n_1-1)n_0+n_0-1 = n_0n_1-1\} &\quad \text{(Set } (n_0-1)\text{).} \end{aligned}\right\} \tag{3.9}$$

The elements listed above form block 1 for group G_1; there are N/n_0n_1 blocks in total for G_1, each containing n_0n_1 elements. Similarly, for group G_j, $0 \le j \le m-1$, there are $N/(n_0n_1 \cdot \cdots \cdot n_j)$ blocks each containing $n_0n_1 \cdot \cdots \cdot n_j$ elements. (Evidently, for G_{m-1}, there is a single block containing all N elements of group G.) For the group G_j, elements in block lines are chosen "skipping over $n_0n_1 \cdot \cdots \cdot n_{j-1} - 1$."

Example 5: Let $G = C_2 \times S_3 \times C_2$ (where S_3 is the permutation group of the order 6); $N = |G| = 24$, $m = 3$, $n_0 = n_2 = 2$, $n_1 = 6$. Block structure for 3 steps of the spectrum algorithm is as follows:

Step 0

$\{0, 1\}$ Block 1
$\{2, 3\}$ Block 2
$\{4, 5\}$ Block 3
$\cdots$
$\{20, 21\}$ Block 11
$\{22, 23\}$ Block 12

Step 1

$\left.\begin{matrix}0, 2, 4, 6, 8, 10\\ 1, 3, 5, 7, 9, 11\end{matrix}\right\}$ Block 1

$\left.\begin{matrix}12, 14, 16, 18, 20, 22\\ 13, 15, 17, 19, 21, 23\end{matrix}\right\}$ Block 2

Step 2

$\left.\begin{matrix}0, 12\\ 1, 13\\ 2, 14\\ \cdots\\ 10, 22\\ 11, 23\end{matrix}\right\}$ Block 1

Since before processing (before Step 0) $N/p = p_1$ successive elements of G were delivered to the 1st processor, next p_1 successive elements to the 2nd processor, etc., the block sizes in successive steps of the spectrum calculation allow us to trace the distribution of the intermediate spectrum elements in block lines among different processors and to determine whether the interprocessor data transfers are needed in a certain step. The following rule is valid: if the block size $n_0n_1 \cdot \cdots \cdot n_j \le p_1 = N/p$, no data transfers are needed in Step j. If the block size $n_0n_1 \cdot \cdots \cdot n_j > p_1$, the transfers among p processors must be performed. (Note that the term "transfers needed in Step j" means transfers needed before processing of Step j.)

It can be easily seen that the transfers are never needed in Step 0 (block size n_0) if our previous assumption is valid that $N/(pn_j)$ is an integer for any j, $0 \le j \le m-1$. In particular, $N/(pn_0) \ge 1$ is an integer which means $n_0 \le p_1$ (no transfers needed). For Step 0, there are N/n_0 trivial blocks each containing only one line.

For the case when transfers are needed ($n_0n_1 \cdot \cdots \cdot n_j > p_1 = N/p$), define parameter K_j, $1 \le j \le m-1$, as $K_j \triangleq n_0n_1 \cdot \cdots \cdot n_j/p_1 = (n_0n_1 \cdot \cdots \cdot n_jp)/N$; K_j is the number of processors per block of group G_j. By the definition, K_j is an integer greater than 1. The number of lines (i.e., group G_j sets) in a block of G_j is

$$n_0n_1 \cdot \cdots \cdot n_{j-1} = \frac{N}{n_jn_{j+1} \cdot \cdots \cdot n_{m-1}} = \frac{N}{pn_j}\frac{pn_0n_1 \cdot \cdots \cdot n_j}{N} = \frac{N}{pn_j} \cdot K_j. \tag{3.10}$$

Since $n_0n_1 \cdot \cdots \cdot n_{j-1}$ is an integer as well as $N/(pn_j)$ (by the assumption made above), K_j is also an integer.

The relation (3.10) means also that it is always possible to distribute the lines of a block equally among p processors (in case transfers are needed in Step j).

The number of transfers in Step j of the algorithm depends upon the relation between n_j and K_j. Consider two cases: $n_j \ge K_j$ and $n_j < K_j$.

Case $n_j \ge K_j$: Assume that Step j is the one where transfers are needed. Since the elements in any block column follow in succession, and since the $n_0n_1 \cdot \cdots \cdot n_j$ elements of a block belong to K_j different processors, then the group order n_j must be an integer multiple of K_j: $n_j = A_jK_j$, $A_j \ge 1$ integer.

Block size for group G_j may be expressed as

$$n_0n_1 \cdot \cdots \cdot n_j = n_0n_1 \cdot \cdots \cdot n_{j-1} \cdot K_j\frac{n_j}{K_j}.$$

Since $n_0n_1 \cdot \cdots \cdot n_j$ and K_j are integers, the value of $A_j = n_j/K_j$ is also an integer.

Deliver first $n_0 n_1 \cdot \cdots \cdot n_{j-1}/K_j$ lines of a block for G_j to the first one of the participating processors, the second $n_0 n_1 \cdot \cdots \cdot n_{j-1}/K_j$ lines to the second processor, etc., then in each line of a block $A_j = n_j/K_j$ elements belong to the proper processor (that is, they are already located in its local memory), and the remaining $n_j - n_j/K_j$ elements are to be transferred from other processors. The total number of transfers in all blocks of G_j (that is, in N/n_j lines of all blocks together) is

$$\frac{N}{n_j}\left(n_j - \frac{n_j}{K_j}\right) = N\left(1 - \frac{1}{K_j}\right).$$

Case $n_j < K_j$: In this case, $n_0 n_1 \cdot \cdots \cdot n_j < K_j n_0 n_1 \cdot \cdots \cdot n_{j-1}$, or $n_0 n_1 \cdot \cdots \cdot n_{j-1} > n_0 n_1 \cdot \cdots \cdot n_j/K_j = p_1$. In other words, one block column contains more elements than the number of elements per processor. The block column size must be an integer multiple of p_1 (block column is distributed among $N_j > 1$ processors):

$$n_0 n_1 \cdot \cdots \cdot n_{j-1} = N_j p_1;$$

block size

$$n_0 n_1 \cdot \cdots \cdot n_j = \frac{n_0 n_1 \cdot \cdots \cdot n_{j-1}}{p_1} p_1 n_j = N_j \cdot p_1 n_j;$$

since p_1 and n_j are integers, N_j is also an integer. It follows by definition that $N_j > 1$:

$$N_j = \frac{n_0 n_1 \cdot \cdots \cdot n_{j-1}}{p_1} = \frac{n_0 n_1 \cdot \cdots \cdot n_j}{p_1 n_j} = \frac{K_j}{n_j};$$

$N_j = 1$ for $K_j = n_j$ which is the case we considered above.

In the last case, all elements of each block line belong to different processors. It is possible to distribute $n_0 n_1 \cdot \cdots \cdot n_{j-1}$ lines of the block among K_j processors in such a way that only one element of each line belongs to the proper processor (that is, this element is already located in memory of the processor that the line is delivered to), and the remaining $n_j - 1$ elements are to be transferred from the other processors. The total number of transfers for all blocks is

$$\frac{N}{n_j}(n_j - 1) = N\left(1 - \frac{1}{n_j}\right).$$

Both cases ($n_j \geq K_j$ and $n_j < K_j$) may be unified in one formula for the number of transfers among p processors in the beginning of Step j:

$$M^{(p)}(G_j) = N\left(1 - \frac{1}{\min(n_j, K_j)}\right)$$

(note that K_j by definition depends on the number of processors).

Formula (3.8) may be generalized now for the case of p processors. Total time $T^{(p)}(G)$ needed for spectrum calculation over $G = G_0 \times G_1 \times \cdots \times G_{m-1}$ is

$$\begin{aligned} T^{(p)}(G) &= L^{(p)}(G)\, t_a + M^{(p)}(G) \cdot t_c \\ &= \frac{N}{p}\left\{ t_a \sum_{j=0}^{m-1} \frac{L(G_j)}{n_j} + k_1 t_c \sum_{j=1}^{m-1} \right. \\ &\qquad \left. \cdot\, \phi_j(p, n_0, \cdots n_j)\left(1 - \frac{1}{\min(n_j, K_j)}\right)\right\} \end{aligned} \tag{3.11}$$

where $N = |G| = n_0 n_1 \cdot \cdots \cdot n_{m-1}$; $n_j = |G_j|$, $0 \leq j \leq m-1$; p is the number of processors; t_a is the time for one addition; t_c is the time for one data transfer; $L(G_j)$ is the number of operations (converted to the equivalent number of additions) to calculate the spectrum over G_j (single butterfly algorithm with one processor); $k_1 = 2$, for the complete communication network, and $k_1 = p$, for unibus connection of processors; $p_1 = N/p$ is the number of elements of G per processor; $\phi_j(p, n_0, \cdots, n_j) = 0$ for $n_0 n_1 \cdot \cdots \cdot n_j \leq p_1 = N/p$, and $\phi_j(p, n_0, \cdots, n_j) = 1$ otherwise; K_j is the number of processors per one block of G_j; $L^{(p)}(G)$ is the total number of operations for spectrum calculation over G (converted to equivalent number of additions), with p processors; $M^{(p)}(G)$ is the total number of interprocessor transfers of data.

The upper limit of the number of processors for this approach is

$$p \leq N/(\max_{0 \leq j \leq m-1} n_j), \qquad N = |G|, \qquad n_j = |G_j|.$$

Example 6: $G = C_2 \times C_2 \times Q_2$ by 4 processors; $N = 32$, $m = 3$, $n_0 = n_1 = 2$, $n_2 = 8$ (see Fig. 1). Number of additions/subtractions: $8 + 8 + 20 = 36$. (A processor can perform C_2-butterfly with a pair of input data at a time; for example, P_1 processes the pair (0, 1) and then the next pair (2, 3) in Step 0.) Number of data transfers: $24 = 32 \times (1 - 1/4)$ ($\phi_1 = 0$, $\phi_2 = 1$).

Special Case of $n_j = B_j p$, *Integer* $B_j \geq 1$ *for All* j, $0 \leq j \leq m-1$

In this case, formula (3.11) can be simplified. It can be shown that the data transfers are needed in the last step only:

$$\phi_0 = \phi_1 = \cdots = \phi_{m-2} = 0, \qquad \text{and } \phi_{m-1} = 1.$$

In the last step, $K_{m-1} = p$, and $n_{m-1} \geq p$, so that $\min(n_{m-1}, K_{m-1}) = p$. Formula (3.11) is simplified as follows:

$$T^{(p)}(G) = \frac{N}{p}\left(t_a \sum_{j=0}^{m-1} \frac{L(G_j)}{n_j} + k_1 t_c\left(1 - \frac{1}{p}\right)\right);$$

$$n_j = B_j p, \qquad 0 \leq j \leq m-1. \tag{3.12}$$

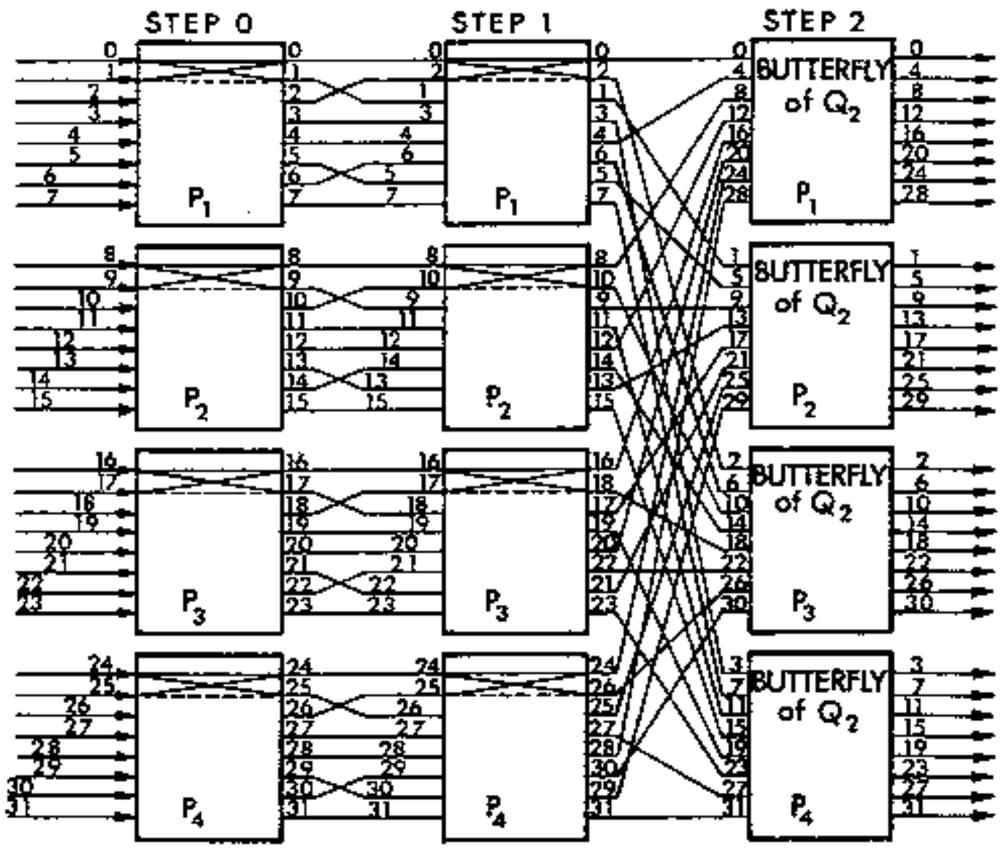

Fig. 1. The group $G = C_2 \times C_2 \times Q_2$ by 4 processors.

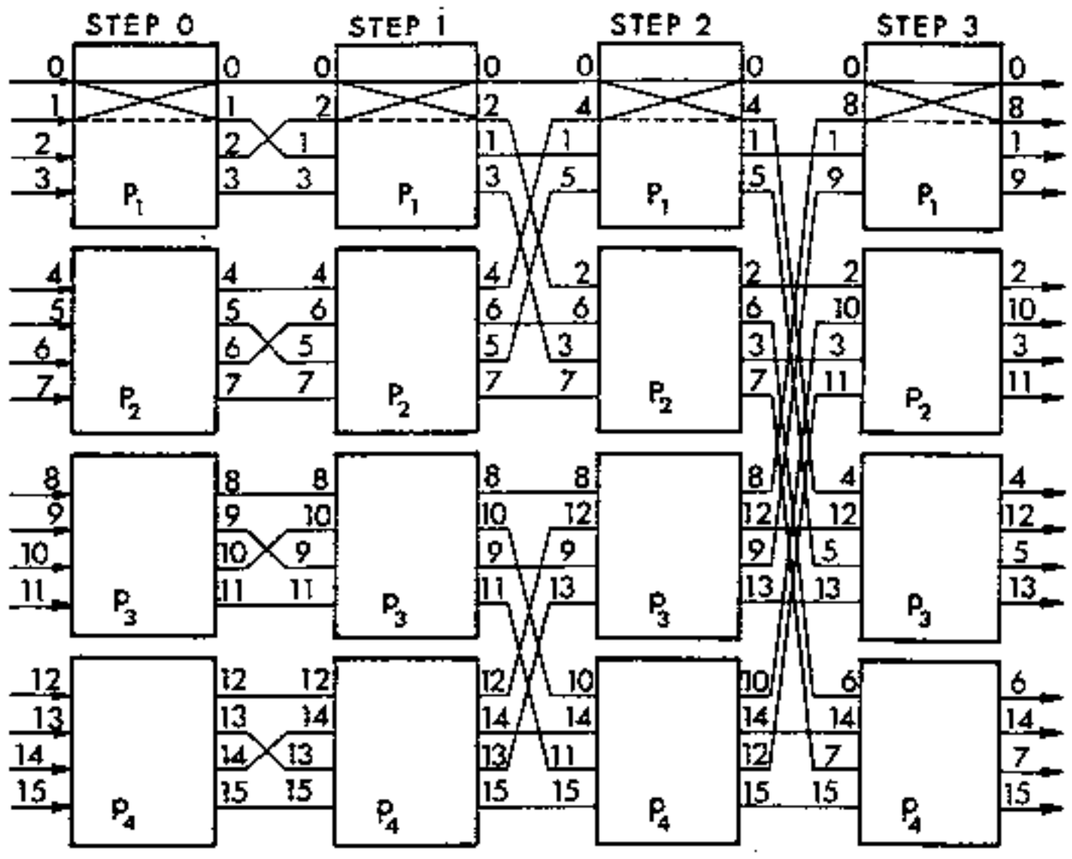

Fig. 2. The group $G = C_2 \times C_2 \times C_2 \times C_2$ by 4 processors.

Special Case of Fast Walsh Transform [13]

$$G = C_2 \times C_2 \times \cdots \times C_2 = C_2^m (m \text{ groups});$$

$$|G| = N = 2^m; \quad n_j = 2, \quad L(G_j) = 2,$$

$$0 \le j \le m - 1.$$

For the number of processors $p = 2^i$, where $i \le m - 1$, transfers are needed in steps with numbers $j \ge m - i$, since block size for Step j is

$$n_0 n_1 \cdot \cdots \cdot n_{m-1} = 2^{j+1}; \quad p_1 = \frac{N}{p} = 2^{m-i}; \quad 2^{j+1} > 2^{m-i}$$

$$\text{for } j + 1 > m - i, \quad \text{or } j \ge m - i.$$

For unibus connection ($k_1 = p = 2^i$)

$$T^{(2^i)}(C_2^m) = m \cdot 2^{m-i} \cdot t_a + i \cdot 2^{m-1} \cdot t_c. \quad (3.13)$$

Example 7: $G = C_2 \times C_2 \times C_2 \times C_2$ by 4 processors: 16 transfers (unibus), and 16 additions/subtractions (see Fig. 2). Data transfers are needed in Steps 2 and 3 (before the processing).

The number of transfers in (3.13) for Fast Walsh Transform is similar to the result of Gannon and Rosendale [23] (differing by a factor of 2 since the algorithm used in [23] includes spectrum reshuffling).

Special Case of Fast Chrestenson Transform [13]

$$G = C_q \times C_q \times \cdots \times C_q = C_q^m;$$

$$|G| = N = q^m; \quad n_j = q; \quad L(G_j) \le q(q - 1),$$

$$0 \le j \le m - 1.$$

For the number of processors $p = q^i$, where $i \le m - 1$, the data transfers are needed in steps with numbers $j \ge m - i$.

$$T^{(q^i)}(C_q^m) \le mq^{m-i}(q - 1)\, t_a + q^{m-1} \cdot i(q - 1)\, t_c. \quad (3.14)$$

(In (3.14), the relation $L(C_q) \le q(q - 1)$ was used that overestimates the number of operations needed for spectrum calculation.) In case of FFT over a group C_q, $q = 2^l$,

$$L(C_q) = 2q(\log_2 q - 1) + 2.$$

By (3.11) for $p = 2^i = 2^{li}$, we obtain

$$T^{(2^{li})}(C_{2^{li}}^m) = 2^{l(m-i-1)+1} \cdot m\big(2^l(l - 1) + 1\big)\, t_a + 2^{l(m-1)} \cdot i \cdot (2^l - 1) \cdot t_c. \quad (3.15)$$

The Number of Communication Links Among p Processors

Up to now, the term "complete communication network" has been used, without entering into details of processors connection. It was assumed that the network allows communication in parallel among any processors that need to exchange data. In fact, our model of "complete" network does not always need to have all $p(p - 1)/2$ communication links among p processors; which links are to be present in the network is determined by the structure of group $G = G_0 \times G_1 \times \cdots \times G_{m-1}$. The concept of block introduced in this section allows us to obtain the formula for the exact number of communication links for the arbitrary group structure $G = G_0 \times G_1 \times \cdots \times G_{m-1}$ and for any $p \le (N/(\max_{0 \le j \le m-1} n_j))$.

One case when the complete network is needed is rather evident: the case when $p = n_{m-1}$. For $j = m - 1$, there

is a single block of N elements, and $K_{m-1} = P$. By definition, $N_j = K_j/n_j$ (we recall that K_j is the number of different processors per block, and N_j is the number of different processors per one block column). If $p = n_{m-1}$, then $N_{m-1} = 1$, and all the processors in each block line are different. Evidently, the complete network is needed in this case. As to other cases, consider again two possibilities: $n_j \geq K_j$ and $n_j < K_j$. (Note that bidirectional communication links among all processors are assumed.)

Case $n_j \geq K_j$ ($n_0 n_1 \cdot \cdots \cdot n_j > p_1 = N/p$)

It has been shown that in this case, $n_j = A_j K_j$, where $A_j \geq 1$ is the number of block columns with the elements located in the memory of one processor (there are K_j processors in a block). The number of communication links is $\binom{K_j}{2} n_{j+1} \cdot \cdots \cdot n_{m-1} = (p/2)(K_j - 1)$ (where $n_{j+1} \cdot \cdots \cdot n_{m-1}$ is the number of blocks).

Case $n_j < K_j$

In this case, $N_j = n_0 \cdot n_1 \cdot \cdots \cdot n_{j-1}/p = K_j/n_j$; $N_j > 1$ is an integer (N_j is the number of different processors in one block column). A block column contains $n_0 n_1 \cdot \cdots \cdot n_{j-1}$ elements; the number of different processors in one block line is $n_j = K_j/N_j$, and the number of links needed for the transfers among n_j processors is $\binom{n_j}{2}$. This value is to be multiplied by p/n_j, to obtain the total number of needed communication links, since there are K_j processors per one block, K_j/n_j sets of different processors (each set consisting of n_j processors), and the number of blocks is $n_{j+1} \cdot \cdots \cdot n_{m-1}$:

$$\frac{K_j}{n_j} \cdot n_{j+1} \cdot \cdots \cdot n_{m-1} = \frac{n_0 n_1 \cdot \cdots \cdot n_j}{p_1} \cdot \frac{n_{j+1} \cdot \cdots \cdot n_{m-1}}{n_j} = \frac{N}{(N/p)\, n_j} = \frac{p}{n_j}.$$

The general formula for the number of communication links needed for the spectrum calculation over an arbitrary group $G = G_0 \times G_1 \times \cdots \times G_{m-1}$, with p processors that may communicate in parallel, is as follows:

$$CL^{(p)}(G) = \sum_{j=1}^{m-1} \phi_j(p, n_0, n_1, \cdots, n_j) \cdot \left(\psi_{1,j}(n_j, K_j) \frac{p}{2} (K_j - 1) + \psi_{2,j}(n_j, K_j) \frac{p}{2} (n_j - 1) \right), \quad (3.16)$$

where

$\phi_j(p, n_0, \cdots, n_j) = 1$, if $n_0 n_1 \cdot \cdots \cdot n_j \geq p_1 = N/p$, and 0 otherwise;

$\psi_{1,j}(n_j, K_j) = 1$, for $n_j \geq K_j$, and 0 otherwise;

$\psi_{2,j}(n_j, K_j) = 1$, for $n_j < K_j$, and 0 otherwise;

$$K_j = \frac{n_0 n_1 \cdot \cdots \cdot n_j p}{N}.$$

Special Case of Fast Walsh Transform [1], [13]

For $G = C_2 \times C_2 \times \cdots \times C_2 = C_2^m$, the number of communication links does not depend on the order of group G. For any $|G| = N = 2^m$, the number of links is the function of the number of processors only. It is the minimal network for p processors; all the other group structures involve the larger number of links for $p > 2$. Assume $p = 2^i$; as it has been shown earlier, for Fast Walsh transform, the interprocessor transfers are needed in steps with the numbers $j \geq m - i$ where $i = \log_2 p$.

$$K_j = 2^{j-(m-i)+1}; \quad \text{for } j \geq m - i, \quad K_j \geq 2, \quad \text{and } n_j = 2 \leq K_j.$$

$$CL^{(2^i)}(C_2^m) = (m - 1 - (m - i - 1)) \binom{2}{2} 2^{i-1} = \frac{1}{2} p \log_2 p \quad (3.17)$$

(bidirectional links). The network for Fast Walsh Transform is the binary cube with p vertices and $(1/2)\, p \log_2 p$ edges. Table I summarizes the numbers of communication links needed for different groups with $N = 64$ and $N = 512$. The minimal communication network is the binary cube (for C_2^6 and C_2^9 groups) for any value of p. For comparison, one of the table columns shows the number of links in the complete network for each p. It can be seen that for the smaller groups ($N = 64$), in most cases, the complete network is needed; the exceptions are C_2^6 (for all p) and C_4^3 (for $p = 8, 16$, i.e., for $p > n_{m-1}$). In the case of large groups (e.g., with $N = 512$), for $p > n_{m-1}$, some of the sample groups are next to the best (minimal) network of C_2^9 having smaller network than the complete one. However, these networks, although being noncomplete, contain about twice as many links as the binary cube network of Fast Walsh Transform. Note that among the groups next to the best, there are those containing non-Abelian quaternion groups Q_2 or a factor of Q_2 combined with C_2 groups (Q_2^3, $C_2^3 \times Q_2^2$, $C_2^6 \times Q_2$).

IV. Computer Verification of the Number of Transfers and Sample Calculations for Different Groups

To verify the number of interprocessor transfers in formula (3.11), a simulated multiprocessor spectrum calculation has been performed by a uniprocessor computer.

TABLE I
THE NUMBER OF BIDIRECTIONAL COMMUNICATION LINKS NEEDED FOR INTERPROCESSOR DATA TRANSFERS WITH p PROCESSORS, FOR DIFFERENT GROUPS

N = 512 G / p	Complete network	C_2^9	C_8^3 Q_2^3	$C_4 \times C_8 \times C_{16}$	$C_2^3 \times C_8^2$ $C_2^3 \times Q_2^2$	$C_2^6 \times C_8$ $C_2^6 \times Q_2^2$	$C_2 \times C_{16}^2$
2	1	1	1	1	1	1	1
4	6	4	6	6	6	6	6
8	28	12	28	28	28	28	28
16	120	32	64	120	64	64	120
32	496	80	160	256	160	144	496
64	2016	192	448	–	448	320	–

N = 64 G / p	Complete network	C_2^6	C_4^3	C_8^2 Q_2^2	$C_2 \times C_4 \times C_8$ $C_2 \times C_4 \times Q_2$	$C_2^3 \times C_8$ $C_2^3 \times Q_2$
2	1	1	1	1	1	1
4	6	4	6	6	6	6
8	28	12	16	28	28	28
16	120	32	48	–	–	–
32	496	80	–	–	–	–

TABLE II
THE NUMBER OF INTERPROCESSOR TRANSFERS $M^{(p)}(G)$ FOR G COMPOSED AS A DIRECT PRODUCT OF SIX C_2 GROUPS AND ONE C_8 GROUP

G / p	$C_2^6 \times C_8$	$C_2^5 \times C_8 \times C_2$	$C_2^4 \times C_8 \times C_2^2$	$C_2^3 \times C_8 \times C_2^3$	$C_2^2 \times C_8 \times C_2^4$	$C_2 \times C_8 \times C_2^5$	$C_8 \times C_2^6$
2	256	256	256	256	256	256	256
4	384	512	512	512	512	512	512
8	448	640	768	768	768	768	768
16	704	704	1024	1024	1024	1024	1024
32	960	960	1152	1152	1280	1280	1280
64	1216	1216	1216	1216	1408	1536	1536

The simulation algorithm provided an independent way to count the number of transfers; computer experiments with numerous sample groups confirmed the correctness of (3.11) and consequently of the derived formulas (3.12)–(3.15).

With the use of introduced concept of block, a simple rule for the ordering of constituent groups has been determined. To minimize the number of transfers for a fixed set of the constituent groups G_j, it is necessary to order the groups according to the rule:

$$|G_0| \le |G_1| \le \cdots \le |G_{m-1}|.$$

Small groups in the initial steps of the algorithm produce small block sizes, and the transfers are required in the minimal number of steps. Table II shows the transfer numbers for a fixed sample set of seven constituent groups (six groups C_2 and one C_8 group; $N = 512$) taken in different order. For two processors, the number of transfers is the same for all versions of G. However, for $p > 2$, the group $C_2^6 \times C_8$ always has the lowest number of transfers, while the group $C_8 \times C_2^6$ is "the worst" having the greatest number of transfers for all $p > 2$.

For sample calculations and comparison, several groups with $N = 64$ and with $N = 512$ were chosen. Table III gives the numbers of transfers for eleven sample groups of order 512, for different numbers of processors. It may seem at the first sight that low-order group implementation of G (when n_j are small) may result in the lowest number of transfers due to the lower degree of "data intermixing" in the successive steps of spectrum calculation algorithm. However, small-size group "design" of G results in the larger number of algorithm steps. As can be seen from Table III, C_2^9 group seems to be the worst choice having the largest number of data transfers. Nevertheless, C_2^9 turns out to be a good implementation in many cases due to relatively low number of the arithmetic operations.

To compare the different groups of Table III with respect to the speed of spectrum calculation, the assumption $t_m = t_a$ has been made (t_m is the time needed for one real multiplication). The ratio t_m/t_a varies within the limits from 1 (Cyber-205, Cray-1, CDC Star-100 [33]) to 20 ÷ 30 (Zilog Z-8000 [2]) for different computers and types of operands. Since in the constituent group algorithms chosen for our estimations, all multiplications are real ([1], Ch. 5, cyclic groups) or trivial (the butterfly for the quaternion group shown in Fig. 3 [15]), the choice of $t_m = t_a$ seems to be reasonable. The ratio $k = t_c/t_a$ varied from 0.02 (fast bus as the Nanobus of Encore Multimax [34]) to $k = 1 \div 1.5$. The choice of the upper limit for k

TABLE III
THE NUMBER OF INTERPROCESSOR TRANSFERS (UNIBUS) FOR DIFFERENT CONSTITUENT GROUPS WITH $N = 512$

p \ G	C_2^9	$C_2^6 \times Q_2$	C_8^3					$C_2^3 \times Q_2^2$
		$C_2^6 \times C_8$	Q_2^3	$C_2 \times C_4^4$	$C_4 \times C_8 \times C_{16}$	$C_2 \times C_{16}^2$	$C_{16} \times C_{32}$	$C_2^3 \times C_8^2$
2	256	256	256	256	256	256	256	256
4	512	384	384	384	384	384	384	384
8	768	448	448	640	448	448	448	448
16	1024	704	704	768	480	480	480	704
32	1280	960	832	1024	736	736	–	832
64	1536	1216	896	1152	–	–	–	–

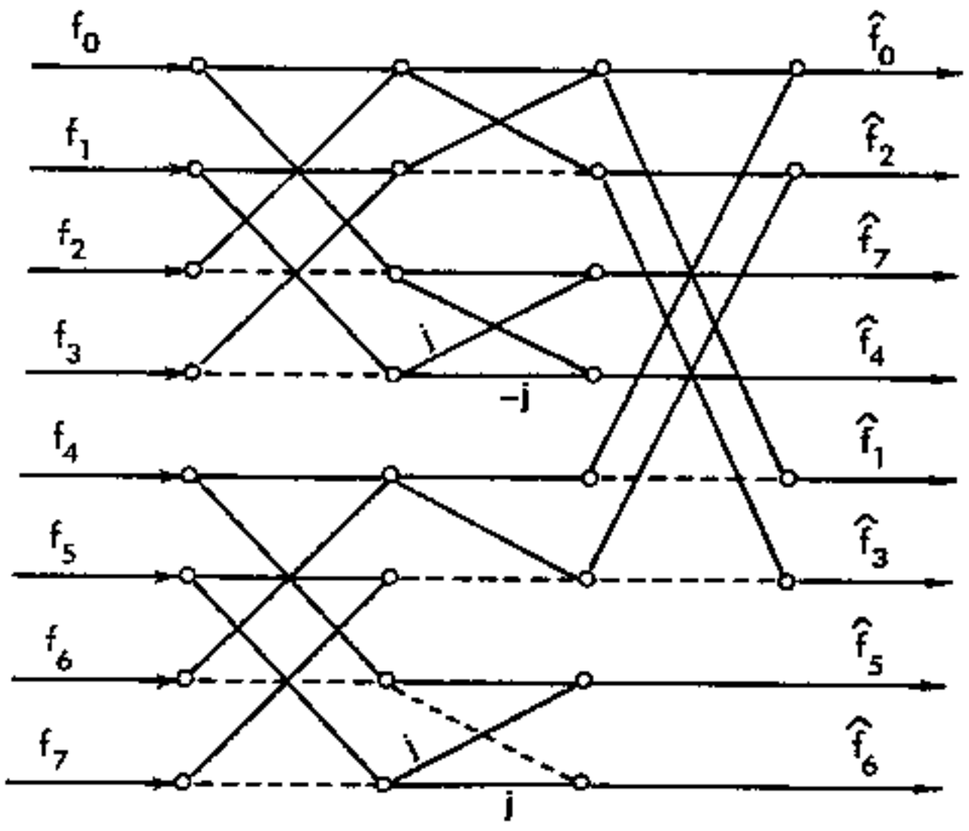

Fig. 3. Flow diagram ("butterfly") for the quaternion group Q_2.

depends on speedup values to be obtained below; it will be shown that for $k \geq 2$, the speedup that can be achieved with two processors instead of one becomes too low to compensate the expenses on the additional hardware. With the increase of p, the speedup decreases.

The use of p processors instead of one reduces the number of the arithmetic operations for spectrum calculation to $L^{(p)}(G) = L(G)/p$. We shall use the term "ideal speedup" to denote the speedup equal to p for p processors. The ideal speedup can never be achieved because of the additional time needed for data transfers. However, for a bus that is fast enough, it is possible to obtain the speedup rather close to ideal.

The total time $T^{(p)}(G)$ needed for spectrum calculation over $G = G_0 \times G_1 \times \cdots \times G_{m-1}$ was evaluated for 8 sample groups of the order 512, and for 6 sample groups of the order 64, for different number of processors p and for different bus speeds characterized by the parameter $k = t_c/t_a$. The $T^{(p)}$ value has been compared to $T^{(1)}$—time needed for uniprocessor spectrum calculation (no data transfers). Both $T^{(p)}$ and $T^{(1)}$ were calculated in terms of the equivalent number of additions (multiplications and data transfers were converted to equivalent numbers of add operations) using the relations $t_m = t_a$, $t_c = k \cdot t_a$.

We shall take the ratio $T^{(1)}/(pT^{(p)})$ as a measure of closeness of an actual speedup $T^{(1)}/T^{(p)}$ to the ideal speedup. The value of the ratio $T^{(1)}/(pT^{(p)})$ may be chosen as a criterion for the comparison of different groups for different p and k. Other criteria may be the total system cost, the number of arithmetic operations, the accuracy of calculation, etc.

Fig. 4 shows the curves of relative speedup $T^{(1)}/(pT^{(p)})$ for sample groups $C_2^6(N = 64)$ and $C_2^9(N = 512)$ for different values of k (that is, for different speeds of the bus). The smaller group has lower values of speedup due to the relatively large number of transfers compared to the number of arithmetic operations. It can be seen that if the minimum acceptable speedup is chosen as 0.9 of the ideal speedup value ($=p$ for p processors), then for bus speed parameter $k \geq 1$ it is not worthwhile to have two processors instead of one for $G = C_2^6$. Fast increase in the number of transfers results in a decrease of the speedup for $k = 1$, $p = 2$ to 0.86 of the "ideal" value only. For $k = 1$, $p = 4$ for the same group the relative speedup becomes only 0.6, that is, the absolute speedup is 2.4 only. For a fast bus ($k = 0.02$), both groups C_2^6 and C_2^9 may have up to 16 processors (relative speedup values for $p = 2, 4, 8, 16$ and $k = 0.02$, are higher than or equal to 0.9).

For another pair of sample groups $G = Q_2 \times Q_2$ ($N = 64$) and $G = Q_2 \times Q_2 \times Q_2$ ($N = 512$), the same rule can be observed: the smaller group allows us to have the smaller number of processors for the same bus speed, to provide the relative speedup not lower than a given cutoff value (0.9 for our examples). It can be concluded that smaller groups are more sensitive to the decrease of the bus speed; for the same value of k, the speedup curves of the smaller group are lower than those for the larger group. Figs. 5 and 6 illustrate the speedup change with the increase in the number of processors for different bus speeds in case of two quaternion groups. From Figs. 5 and 6, it can be seen that there is a fast speedup decrease (below the cutoff level 0.9) for 4 processors and $k \geq 0.25$ in case of $Q_2 \times Q_2$ while the large group $Q_2 \times Q_2 \times Q_2$ allows $k = 0.5$ for 4 processors for the chosen cutoff value 0.9. Fast bus with $k = 0.02$ in case of $Q_2 \times Q_2 \times Q_2$ allows us to have up to 32 processors for the same restrictions on the relative speedup; for $k = 0.02$ and $p = 32$, the speedup will be about $30 = 32 \times 0.94$.

For the chosen minimum speedup value, the bus speed of $k = 0.5$ is about the critical value for small groups

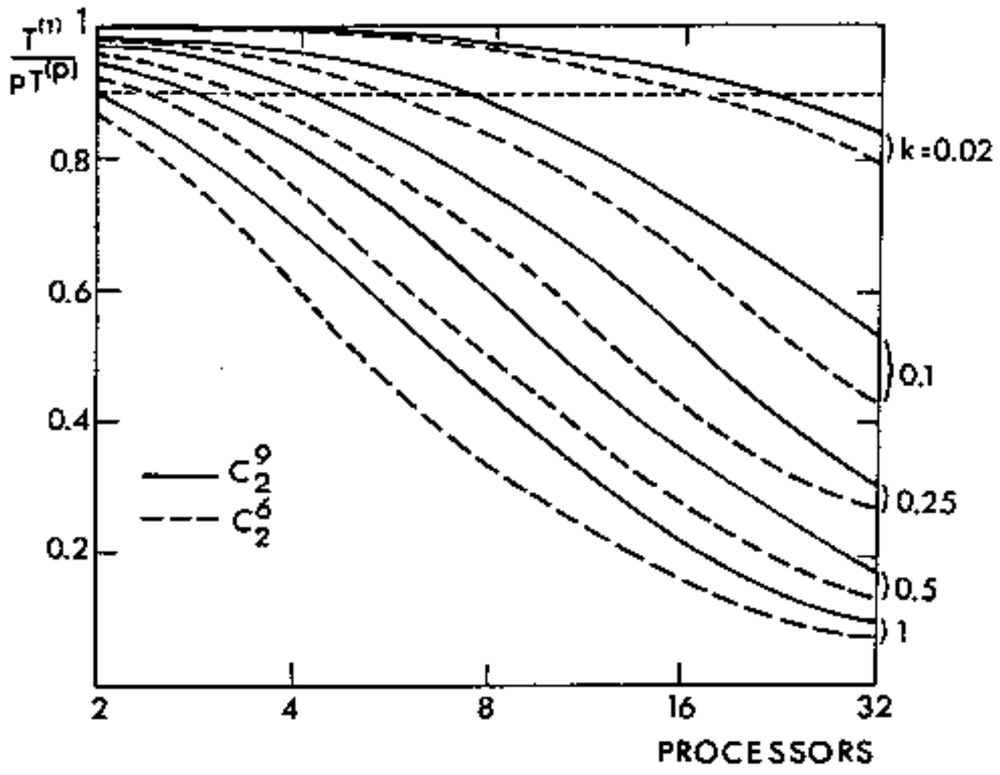

Fig. 4. The relative speedup $T^{(1)}/(pT^{(p)})$ for groups $G = C_2^9$ ($N = 512$) and for $G = C_2^6$ ($N = 64$) as a function of the number of processors and the bus speed. The value of $k = 0.02$ corresponds to the fast bus case; dotted lines show speedup for C_2^6.

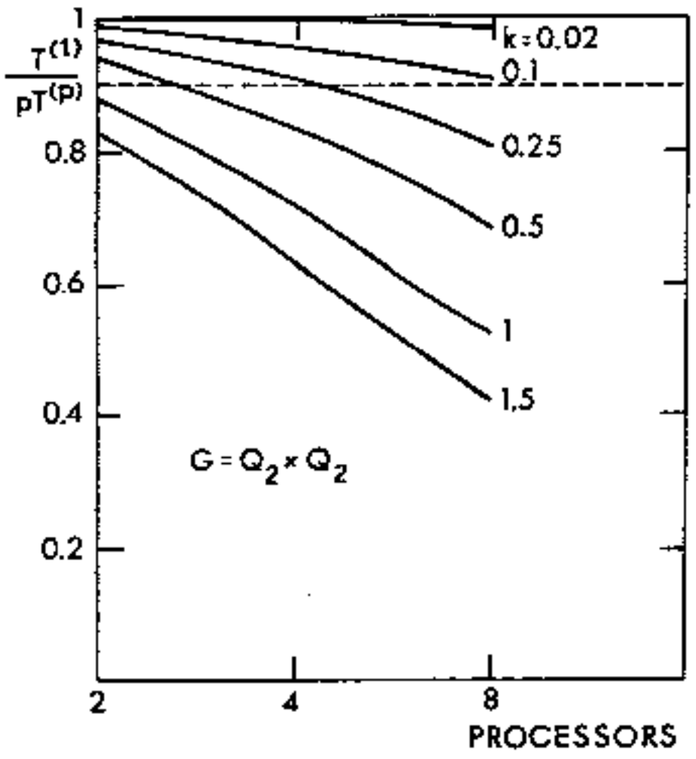

Fig. 5. The relative speedup $T^{(1)}/(pT^{(p)})$ for the small quaternion group $Q_2 \times Q_2$ ($N = 64$) as a function of the number of processors and of the bus speed.

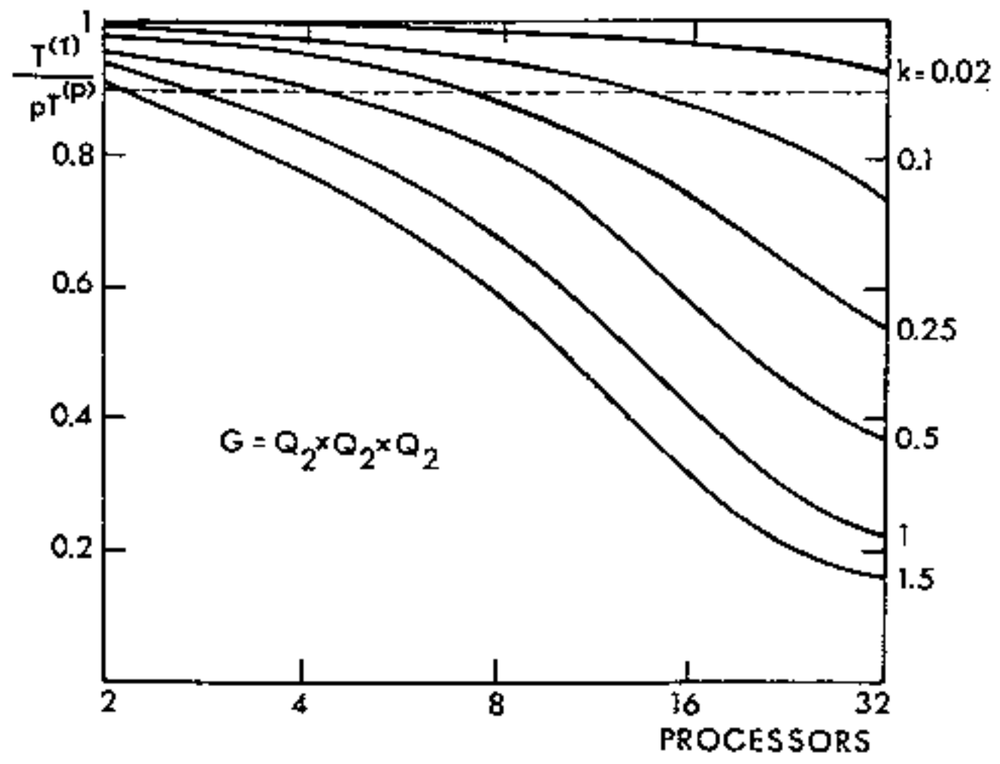

Fig. 6. The relative speedup $T^{(1)}/(pT^{(p)})$ for the large quaternion group $G = Q_2 \times Q_2 \times Q_2$ ($N = 512$) as a function of the number of processors and of the bus speed.

C_2^6 and $Q_2 \times Q_2$ (two processors give the relative speedup not less than 0.9). For $k > 0.5$, the relative speedup is less than 0.9 for $p = 2$. It means that if we choose the relative speedup ≥ 0.9 as the only criterion for the choice of the number of processors, it makes no sense to have two processors instead of one, if the communication time is greater than a half of one addition time. For large groups (C_2^9 and $Q_2 \times Q_2 \times Q_2$), the bus may be slower under the same restriction on the minimum relative speedup: for $Q_2 \times Q_2 \times Q_2$, it can be shown that the critical value of k is about 1.55, and for C_2^9, it is about 1.1 (for two processors). Large groups are "less sensitive" to the bus speed because of the relatively large number of the arithmetic operations compared to the number of data transfers; the bus speed becomes less significant.

The quaternion-based groups Q_2^2 and Q_2^3 compared to C_2^6 and C_2^9 have the higher values of relative speedup for the same number of processors and bus speed. However, both quaternion-based groups as well as the groups C_2^6, C_2^9 are characterized by rather low speedup values compared to other groups of the same order (see Tables IV and VI and Figs. 7 and 8). The relative speedup $T^{(1)}/(pT^{(p)})$ may be considered as an efficiency index of a multiprocessor system or as a measure of how much the expenses on the additional hardware (processors) are covered. With the increase in the number of processors, the relative speedup tends to decrease drastically; however, the total time of spectrum calculation also decreases. To compare different group performance (for different p and k) in terms of calculation time, one can compare the equivalent number of additions (multiplications and data transfers converted to the equivalent number of additions).

Table V summarizes the results of calculation for the equivalent numbers of additions (including converted numbers of transfers) which allows one to compare different groups G and to determine the ones with the fastest performance. The two criteria—relative speedup and the equivalent number of additions—are evidently conflicting. With the increase of the number of processors, the number of operations tends to decrease as well as the relative speedup.

The comparison of different structures of G with $N = 512$ (Table V) reveals that the fastest performance in most cases is provided by $C_2^6 \times Q_2$ group. This group is the best one for all p values in case of the fast bus ($k = 0.02$). With the decrease of bus speed ($k > 0.02$), the group is still the best for $p = 2, 4, 8, 16$, while for the larger values of p and $k > 0.02$, the $C_2^3 \times Q_2^2$ group has the smallest number of operations. The groups next to the best are C_2^9 (for all p in fast bus case and for the lower range of p with the decrease of bus speed); $C_2^3 \times Q_2^2$ (for $p = 32$ and $k = 0.1$ as well as for $k = 0.5, 1$ and $p = 8, 16$), and Q_2^3 (for $k = 0.5, 1$ and $p = 32, 64$).

The calculations of the number of equivalent additions were repeated for $N = 512$ for the slow multiplication case $t_m = 5a$, to examine the influence of the multiplication speed on the performance of groups that need nontri-

TABLE IV
RELATIVE SPEEDUP FOR DIFFERENT GROUPS OF THE ORDER $N = 512$ AS A FUNCTION OF BUS SPEED AND OF THE NUMBER OF PROCESSORS (p): $t_m = t_a$

p \ G	C_2^9	Q_2^3	C_8^3	$C_4 \times C_8 \times C_{16}$	$C_2^3 \times C_8^2$	$C_2^3 \times Q_2^2$	$C_2^6 \times Q_2$	$C_2 \times C_{16}^2$
k = 0.02 (FAST BUS)								
2	.998	.998	.999	.999	.999	.998	.998	.999
4	.992	.995	.997	.997	.996	.994	.993	.996
8	.976	.988	.992	.99	.98	.987	.983	.99
16	.93	.966	.98	.98	.97	.96	.95	.98
32	.84	.92	.94	.95	.93	.91	.88	-
64	.71	.85	.89	-	.86	.82	.74	-
k = 0.1								
2	.988	.992	.994	.994	.993	.991	.988	.994
4	.958	.977	.98	.985	.98	.97	.97	.98
8	.88	.947	.96	.96	.95	.94	.92	.96
16	.73	.85	.89	.93	.86	.83	.79	.92
32	.53	.71	.77	.81	.72	.67	.59	-
64	.32	.53	.61	-	.55	.48	.36	-
k = 0.5								
2	.95	.96	.97	.98	.964	.955	.94	.97
4	.82	.89	.92	.93	.90	.88	.85	.92
8	.60	.78	.83	.85	.79	.75	.71	.83
16	.36	.53	.61	.72	.55	.49	.44	.69
32	.18	.32	.40	.46	.34	.29	.22	-
64	.08	.18	.24	-	.19	.16	.10	-
k = 1.0 (SLOW BUS)								
2	.90	.93	.95	.95	.93	.91	.89	.94
4	.69	.81	.85	.87	.82	.78	.74	.85
8	.43	.64	.71	.74	.66	.60	.56	.71
16	.22	.36	.44	.56	.38	.32	.28	.53
32	.10	.19	.25	.30	.21	.17	.12	-
64	.04	.10	.14	-	.11	.09	.05	-

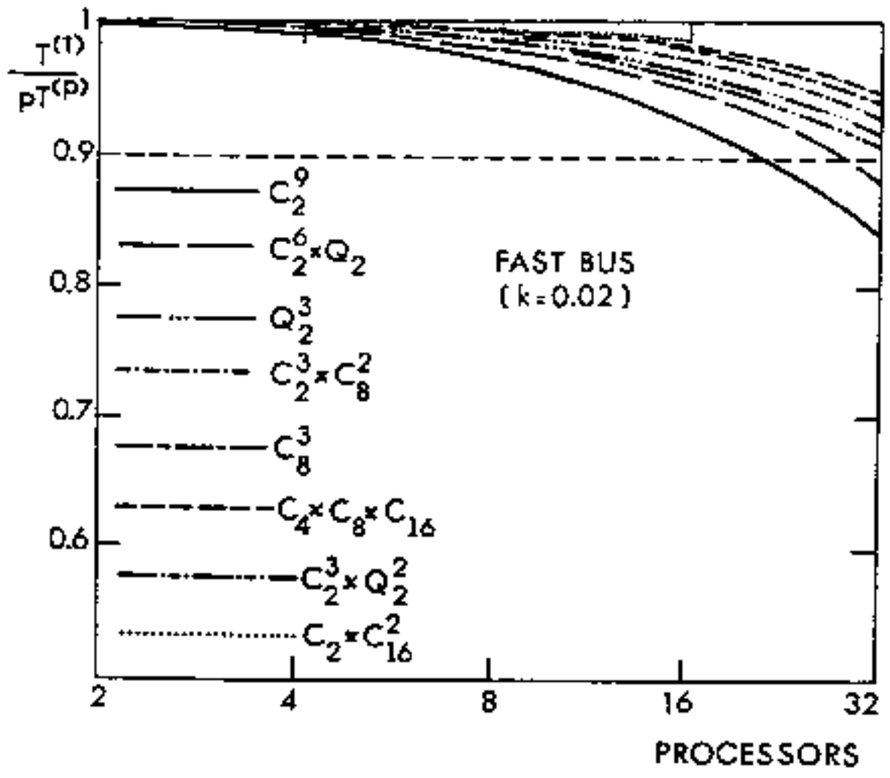

Fig. 7. The relative speedup $T^{(1)}/(pT^{(p)})$ for eight sample groups of the order 512 as a function of the number of processors (fast bus with $k = 0.02$).

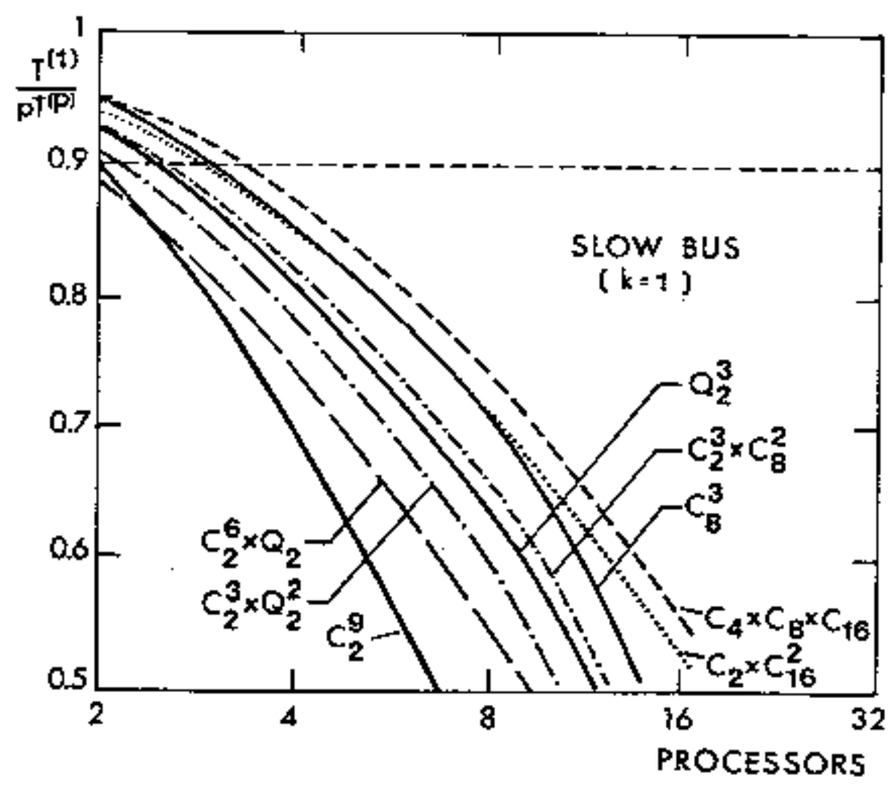

Fig. 8. The relative speedup $T^{(1)}/(pT^{(p)})$ for eight sample groups of the order 512 as a function of the number of processors (slow bus with $k = 1$).

vial multiplications. The results showed the same pattern—the fastest groups were again $C_2^6 \times Q_2$, C_2^9, $C_2^3 \times Q_2^2$, and Q_2^3.

It can be assumed that fast performance is typical for the groups C_2 and Q_2 used as constituent groups in the direct product of G since the group algorithms for C_2 and Q_2 do not need multiplications. It can be expected that the group $G = C_2 \times C_4 \times Q_2$ will also be characterized by a low number of operations since C_4 also has only trivial multiplications by ± 1, $\pm j$. The total calculation time for the group turns out to be very close to the time for the Q_2^2 group, being only about 5 percent greater.

For small sample groups ($N = 64$, Table VI), the group $C_2^3 \times Q_2$ is the fastest one (for $p = 2, 4, 8$). The groups next to the best are C_2^6 (for the fast bus, $k = 0.02$, 0.1 and/or $p = 2, 4$), and Q_2^2 (for $k = 0.5$, 1.0 and $p = 8$). The slowest groups in both cases of $N = 64$ and $N = 512$ are those that contain high-order cyclic groups C_8 and C_{16} in the direct product for G.

For $N = 64$, the C_4^3 group is about 1.6–2 times slower compared to the best group $C_2^3 \times Q_2$. It is very likely that, in general, the best performance (the fastest one) can be achieved with a combination of C_2 and Q_2 groups in the direct product of G.

V. CONCLUSIONS

Investigations of tradeoffs between the number of operations needed for GDFT over arbitrary finite groups

TABLE V
EQUIVALENT NUMBER OF ADDITIONS ($t_m = t_a$) FOR DIFFERENT GROUPS OF THE ORDER $N = 512$ AS A FUNCTION OF BUS SPEED PARAMETER ($k = t_c/t_a$) AND OF THE NUMBER OF PROCESSORS (p)

p \ G	C_2^9	Q_2^3	C_8^3	$C_4 \times C_8 \times C_{16}$	$C_2^3 \times C_8^2$	$C_2^3 \times Q_2^2$	$C_2^6 \times Q_2$	$C_2 \times C_{16}^2$
k = 0.02 (FAST BUS)								
2	2309	3205	4485	4997	3461	2693	2181	4293
4	1162	1608	2248	2504	1736	1352	1096	2152
8	591	809	1129	1257	873	681	553	1081
16	309	414	574	634	446	350	286	546
32	170	217	297	327	233	185	155	-
64	103	118	158	-	126	102	92	-
k = 0.1								
2	2330	3226	4506	5018	3482	2714	2202	4314
4	1203	1638	2278	2534	1766	1382	1126	2182
8	653	845	1165	1293	909	717	589	1117
16	390	470	630	672	502	406	342	584
32	272	283	363	386	299	251	232	-
64	226	190	230	-	198	174	190	-
k = 0.5								
2	2432	3328	4608	5120	3584	2816	2304	4416
4	1408	1792	2432	2688	1920	1536	1280	2336
8	960	1024	1344	1472	1088	896	768	1296
16	800	752	912	864	784	688	624	776
32	784	616	696	680	632	584	616	-
64	840	548	462	-	556	532	674	-
k = 1.0 (SLOW BUS)								
2	2560	3456	4736	5248	3712	2944	2432	4544
4	1664	1984	2624	2880	2112	1728	1472	2528
8	1344	1248	1568	1696	1312	1120	992	1520
16	1312	1104	1264	1104	1136	1040	976	1016
32	1424	1032	1112	1048	1048	1000	1096	-
64	1608	996	1036	-	1004	980	1284	-

TABLE VI
RELATIVE SPEEDUP AND THE EQUIVALENT NUMBER OF ADDITIONS ($t_m = t_a$) FOR DIFFERENT GROUPS OF THE ORDER $N = 64$ AS A FUNCTION OF BUS SPEED PARAMETER ($k = t_c/t_a$) AND OF THE NUMBER OF PROCESSORS (p)

p \ G	C_2^6	C_4^3	Q_2^2	$C_2 \times C_4 \times Q_2$	$C_2 \times C_4 \times C_8$	$C_2^3 \times Q_2$
RELATIVE SPEEDUP:	k = 0.02 (FAST BUS)					
2	.997	.998	.998	.998	.998	.996
4	.987	.992	.992	.992	.994	.99
8	.96	.977	.984	.984	.986	.98
	k = 0.1					
2	.983	.99	.986	.988	.99	.982
4	.94	.97	.96	.965	.97	.95
8	.83	.91	.91	.92	.93	.89
	k = 0.5					
2	.92	.95	.938	.94	.95	.92
4	.75	.87	.83	.84	.87	.79
8	.50	.66	.68	.69	.74	.61
	k = 1.0 (SLOW BUS)					
2	.86	.91	.88	.89	.91	.85
4	.60	.77	.71	.73	.77	.65
8	.33	.50	.52	.53	.59	.44
NUMBER OF ADDITIONS:	k = 0.02 (FAST BUS)					
2	193	321	241	257	321	177
4	97	161	121	129	161	89
8	50	82	61	65	81	45
	k = 0.1					
2	195	323	243	259	323	179
4	102	165	125	133	165	93
8	58	88	66	70	86	50
	k = 0.5					
2	208	336	256	272	336	192
4	128	184	144	152	184	112
8	96	120	88	92	108	72
	k = 1.0 (SLOW BUS)					
2	224	352	272	288	352	208
4	160	208	168	176	208	136
8	144	160	116	120	136	100

(Abelian and non-Abelian) and the hardware complexity/speed have been performed for a multiprocessor system with nonshared local memories of the processors. For arbitrary finite group structure $G = G_0 \times G_1 \times \cdots G_{m-1}$ introduced on the set of N input data ($N = |G|$), general formulas have been developed for the number of arithmetic operations, the number of interprocessor data transfers, and the number of communication links among p processors operating in parallel.

Sample calculations with several groups of different order and different structures revealed some typical features of the constituent groups and the influence of the group order on the performance speed. Smaller groups ($N = 64$) are more sensitive to the speed of the processor communication network. For these groups, the increase in the communication time relative to the addition time results in a drastic decrease of the speedup which can make the use of multiple processors unreasonable. The fastest groups, in most cases, were the ones combining small cyclic groups (C_2) and non-Abelian quaternions (Q_2) in the direct product for G. As shown by an earlier work [15], the quaternion groups as the components of the direct product for G in many cases show the optimal performance as to the accuracy of calculations.

ACKNOWLEDGMENT

The authors would like to thank Dr. R. C. Brower and Dr. L. Levitin for useful discussions.

REFERENCES

[1] D. F. Elliot and K. R. Rao, *Fast Transforms: Algorithms, Analyses and Applications*. New York: Academic, 1982.
[2] T. Beth, "Verfahren der schnellen Fourier-Transformationen," in *Leitfaden der angewandten Mathematik und Mechanik LAMM*, Bd. 61. Stuttgart: B. G. Teubner, 1984.
[3] T. Beth, W. Fung, and R. Muhlfeld, "Zur Algebraischen Diskreten Fourier-Transformation," *Arch. Math.*, vol. 40, pp. 238-244, 1983.
[4] T. Beth and K. S. Vijayan, "Some filing schemes based on finite groups of automorfism," *Computing*, vol. 20, pp. 95-108, 1978.

[5] G. Apple and P. Wintz, "Calculation of Fourier transforms on finite Abelian groups," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 233–236, 1970.
[6] M. D. Atkinson, "The complexity of group algebra computations," *Theor. Comput. Sci.*, vol. 1, pp. 205–209, 1977.
[7] J. Neubuser, "Computing with groups and their character tables," *Computing*, Suppl. 4, pp. 45–56, 1982.
[8] L. Dornhoff, *Group Representation Theory*. New York: Marcel Dekker, 1971.
[9] E. A. Trachtenberg, "Systems over finite groups as suboptimal Wiener filters: A comparative study," presented at the Int. Symp. Math. Theory Syst. and Networks, Beer-Sheva, Israel, 1983.
[10] ——, "Construction of group transforms subject to several performance criteria," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-33, pp. 1521–1531, 1985.
[11] M. G. Karpovsky and E. A. Trachtenberg, "Statistical performance of generalized Wiener filters," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 303–307, 1986.
[12] M. G. Karpovsky, "Fast Fourier transforms on finite non-Abelian groups," *IEEE Trans. Comput.*, vol. C-26, pp. 1020–1030, 1977.
[13] ——, *Finite Orthogonal Series in the Design of Digital Devices*. New York: Wiley, 1976.
[14] M. G. Karpovsky and E. A. Trachtenberg, "Fourier transforms over finite groups for error detection and error correction in communication channels," *Inform. Contr.*, vol. 40, pp. 335–358, 1979.
[15] E. A. Trachtenberg and M. G. Karpovsky, "Filtering in a communication channel by Fourier transforms over finite groups," in *Spectral Techniques and Fault Detection*, M. Karpovsky, Ed. New York: Academic, 1985.
[16] M. G. Karpovsky and E. A. Trachtenberg, "Some optimization problems for convolution systems over finite groups," *Inform. Contr.*, vol. 34, pp. 227–247, 1977.
[17] T. D. Roziner, M. G. Karpovsky, and E. A. Trachtenberg, "Complexity analysis of generalized Fast Fourier Transforms in a multiprocessor environment," presented at the 12th IMACS Congr. Scientific Computation, Paris, France, July 18–22, 1988.
[18] S. Winograd, "On computing the discrete Fourier transform," *Math. Comput.*, vol. 32, pp. 175–199, 1978.
[19] J. Morgenstern, "Note on a lower bound of the linear complexity of the fast Fourier transform," *J. ACM*, vol. 20, pp. 305–306, 1973.
[20] Z. Wang, "Fast algorithms for the discrete W transform and fast DFT," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 803–816, 1984.
[21] M. A. Mehalic, P. L. Rustan, and G. P. Route, "Effects of architectural implementation on DFT algorithm performance," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-33, pp. 684–693, 1985.
[22] H. Nawab and J. H. McClellan, "Bounds on the minimum number of data transfers in WFTA and FFT programs," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-27, pp. 394–398, 1979.
[23] D. B. Gannon and J. van Rosendale, "On the impact of communication complexity on the design of parallel numerical algorithms," *IEEE Trans. Comput.*, vol. C-33, pp. 1180–1194, 1984.
[24] M. J. Corinthios, "3D cellular arrays for parallel/cascade image/signal processing," in *Spectral Techniques and Fault Detection*, M. Karpovsky, Ed. New York: Academic, 1985.
[25] ——, "Cellular arrays for parallel image signal processing," in *Proc. Int. Workshop Fault Detection Spectral Techniques*, Boston, MA, Oct. 1983, pp. 1001–1025.
[26] ——, "Architecture of signal processors," *Int. J. Mini and Microcomput.*, vol. 4, pp. 63–69, 1982.
[27] C. Moraga, "Design of a multiple-valued systolic system for the computation of the Chrestenson spectrum," *IEEE Trans. Comput.*, vol. C-35, pp. 183–188, 1986.
[28] C. Moraga and K. Seseke, "The Chrestenson transform in pattern analysis," in *Spectral Techniques and Fault Detection*, M. Karpovsky, Ed. New York: Academic, 1985.
[29] S. Yalamanchili and J. K. Aggarwal, "A characterization and analysis of parallel processor interconnection networks," *IEEE Trans. Comput.*, vol. C-36, pp. 680–691, 1987.
[30] A. Norton and A. J. Silberger, "Parallelization and performance analysis of the Cooley-Tukey FFT algorithm for shared-memory architectures," *IEEE Trans. Comput.*, vol. C-36, pp. 581–591, 1987.
[31] K. O. Siomalas and B. A. Bowen, "Performance of crossbar multiprocessor systems," *IEEE Trans. Comput.*, vol. C-32, pp. 689–695, 1983.
[32] C. J. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Trans. Comput.*, vol. C-34, pp. 66–77, 1985.
[33] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.
[34] Encore Multimax Technical Summary, Encore Corp., Marlboro, MA, 1986.

Tatyana D. Roziner was born in Moscow, U.S.S.R. She received the M.S. degree in mathematics from Moscow University, Moscow, U.S.S.R. in 1959, and the Ph.D. degree in applied mathematics and physics from the Acoustics Institute of the U.S.S.R. Academy of Sciences in 1975.

In 1960–1976 she worked as a Research Scientist at the Acoustics Institute in Moscow. She joined the National Institute for Industrial Automation in 1976, and in 1977–1978 she worked at the Central Institute for Integrated Automation. Between 1979 and 1985 she worked as Senior Research Scientist for Israel Aircraft Industries. She joined the Department of Electrical, Computer, and Systems Engineering of Boston University in 1986. Her research interests include data processing in multiprocessor environment, parallel architectures and fast algorithms for parallel processing, digital design, and VLSI architectures.

Mark G. Karpovsky (M'80–SM'84) received the M.S. degree in 1963 and the Ph.D. degree in 1967 in computer engineering.

He has published about 100 papers and several books in the areas of signal processing, logic design, testing, fault-tolerant computing, error-correcting codes. He is the author of the book *Finite Orthogonal Series in the Design of Digital Devices* (New York: Academic, 1976), and the Editor of the book *Spectral Techniques and Fault Detection* (New York: Academic, 1985). Since 1983 he has been teaching at Boston University, Boston, MA, and also doing consulting work for IBM, Digital Corp., Honeywell, and AT&T. He is currently a Professor of Computer Engineering and Director of the Research Laboratory on Design and Testing of Computer and Communication Systems at Boston University.

Lazar A. Trachtenberg (M'82–SM'86) was born in Tula, U.S.S.R., on January 29, 1946. He received the degree of engineer-physicist in control systems and the D.Sc. degree in computer science and applied mathematics from the Leningrad Polytechnical Institute, Faculty of Physics and Mechanics, and from the Technion, Haifa, Israel, in 1970 and 1978, respectively.

In 1977–1978 he was a Lecturer at the Division of Computer Science, Tel-Aviv University, Israel. From 1978 to 1979 he was a Senior Lecturer, and from 1979 to 1982 an Associate Professor, at the Department of Mathematics and Applied Mathematics, University of South Africa. In 1981–1982 he was a Visiting Professor at Scuola Normale Superiore, Department of Mathematics, Pisa, Italy. In 1982–1989 he was Associate Professor at the Department of Electrical and Computer Engineering at Michigan Technological, Houghton, and Drexel University, Philadelphia, PA, respectively. He is currently an Associate Professor at the Department of Electrical and Computer Engineering at Drexel University. His current research interests include fault tolerant computing, logic and systems design, digital filtering, and digital signal and image processing.