JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 3, 48–71 (1986)

FFT Algorithms for SIMD Parallel Processing Systems*

LEAH H. JAMIESON, PHILIP T. MUELLER, JR., AND HOWARD JAY SIEGEL

School of Electrical Engineering, Purdue University, West Lafayette, Indiana 47907

Received January 1, 1985

SIMD (single instruction stream – multiple data stream) algorithms for one- and two-dimensional discrete Fourier transforms are presented. Parallel structurings of algorithms for efficient computation for a variety of machine size/problem size combinations are presented and analyzed. Through these algorithms, techniques for exploiting relationships between problem size and machine size are demonstrated. The algorithms are evaluated in terms of the number of arithmetic operations and interprocessor data transfers required. The ability of various interconnection networks presented in the literature to perform the needed transfers is examined. It is shown that the efficiency of a particular data distribution/algorithm decomposition approach is a function of the machine-size/problem-size relationship. © 1986 Academic Press, Inc.

I. INTRODUCTION

Advances in VLSI technology are making large-scale parallel processing systems feasible. The MPP (Massively Parallel Processor), an existing machine, includes 2^{14} simple processors [5]. SIMD (single instruction stream – multiple data stream) machines [15] represent one type of parallel processing system. An SIMD machine typically consists of a control unit; a set of N processing elements (PEs), each a processor with its own memory; and an interconnection network. The control unit broadcasts instructions to all PEs, and each active PE executes each of these instructions on the data in its own memory. Each instruction is executed simultaneously in all active (enabled) PEs. The interconnection network allows data to be transferred among the PEs. A typical SIMD machine organization is shown in Fig. 1.

SIMD machines are especially suitable for exploiting the parallelism inherent in certain tasks performed on vectors and arrays (e.g., [33, 36]). One

*This material is based upon work supported by the National Science Foundation under Grants ECS-8120896 and ECS-7909016, and by the United States Air Force Rome Air Development Center under Contracts F30602-83-K-0119 and F30602-78-C-0025.

0743-7315/86 $3.00

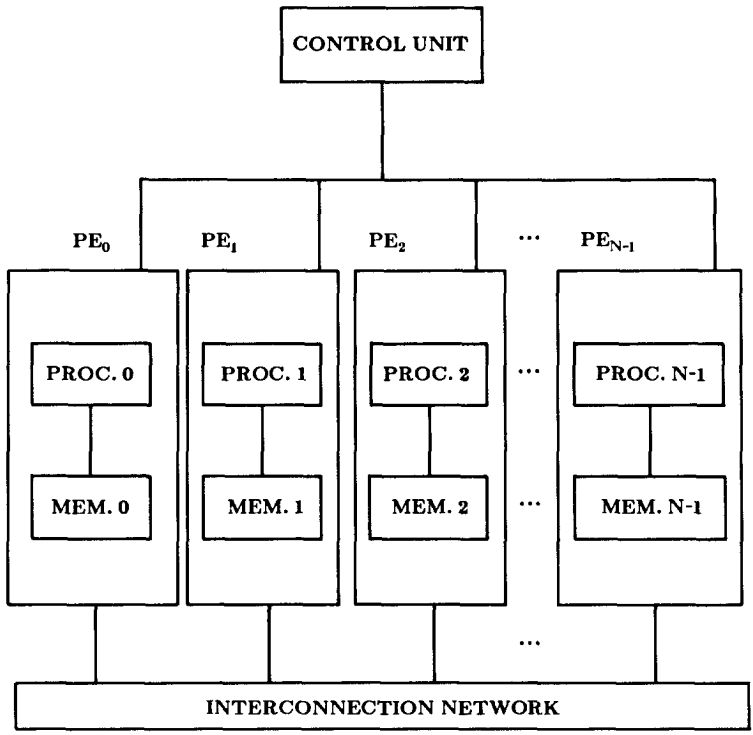

FIG. 1. SIMD machine organization.

aspect of SIMD computations which is of particular interest is the relation between the machine size (number of PEs) and problem size (number of elements in the vector or array on which operations are being performed).

In this paper, the use of SIMD machines to perform one-dimensional (1-D) and two-dimensional (2-D) discrete Fourier transforms (DFTs) [10, 21, 24] is explored. Fast Fourier transform (FFT) algorithms for a variety of problem size–machine size combinations are presented and analyzed. Algorithms developed to take advantage of a specific machine-size/problem size relationship are compared to algorithms obtained by adapting methods designed for a different machine-size/problem size relationship. The algorithms are evaluated in terms of the number of arithmetic operations and interprocessor transfers required.

Numerous papers in the literature describe special purpose machines for FFTs. In several papers, the systems described have a small number of processors with fixed interconnections [6, 11, 12, 13, 17, 18, 35]. These processors can be designed to handle algorithms based on any fixed radix. The required throughput is obtained by adding processors. The architectures presented in these papers will minimize implementation costs. Our results differ from this set of work in that we assume a general model of an SIMD ma-

chine with a flexible interconnection network as the basis for our algorithms. Thus, we are not proposing a special purpose FFT system, but rather algorithms for an SIMD machine which could also be used to execute a sequence of algorithms, in which the FFT was one step in the sequence.

In [7, 8], FFT implementations on ensembles of processors are described, where an ensemble of processors is an SIMD machine without an interconnection network. Our work differs from this in that we assume the use of an interconnection network and are concerned with analyzing the amount and type of interprocessor communication needed by the proposed FFT algorithms.

The results given in [6, 19, 22, 23, 34] are related to the work we present here. This is described in later sections.

In Section II, the SIMD machine model and interconnection capabilities required are defined. One-dimensional SIMD algorithms are summarized in Section III. In Section IV, two approaches to performing two-dimensional FFTs on SIMD machines are presented, analyzed, and compared.

II. SIMD Machine Model

The SIMD machine model assumed for the algorithms includes a set of PEs, a control unit, and an interconnection network [26]. In a machine of size N, the PEs are addressed (numbered) from 0 to $N - 1$, where $N = 2^n$. The processor in each PE contains fast access general purpose registers and an address register which contains i in PE i, $0 \leq i < N$. Inter-PE data transfers are specified by interconnection functions [25]. Formally, an *interconnection function* is a bijection on the set of PE addresses. When an interconnection function f is executed, a data item from PE i is transferred to PE $f(i)$, for all i simultaneously, $0 \leq i < N$, and PE i active. (Note that this implies that a PE can receive a data item from at most one other PE as the result of the execution of a single transfer instruction. This assumption is consistent with the network implementations for systems such as Illiac IV [9], STARAN [4], and PASM [32].) In the following sections, two classes of interconnection functions will be used.

The *Cube* interconnection network consists of $n = \log_2 N$ interconnection functions, Cube_c, $0 \leq c < n$, defined as

$$\text{Cube}_c(p_{n-1} \cdots p_{c+1} p_c p_{c-1} \cdots p_0) = p_{n-1} \cdots p_{c+1} \bar{p}_c p_{c-1} \cdots p_0,$$

where $p_{n-1} \cdots p_0$ is the binary representation of an arbitrary PE address, and $\bar{p}_c$ is the complement of p_c [25]. Thus, Cube_c specifies the connection between pairs of PEs whose addresses differ only in the cth bit position. The Cube interconnections are realizable by a single pass through a number of SIMD machine multistage networks discussed in the literature [28], in-

cluding the generalized cube [30], ADM [29], data manipulator [14], indirect binary n-cube [23], omega [20], and STARAN flip [3]. Nearest neighbor networks, such as the one used in the Illiac IV [9], cannot do a Cube_c in a single step; the time required is discussed in detail in [25, 26] and is beyond the scope of this paper.

The class of *Shift* interconnection functions (or uniform shifts) is defined as

$$\text{Shift}_i(x) = (x + i) \mod N,$$

for a given i, $0 < i < N$, and $0 \leq x < N$. From [20, 28], the omega, generalized cube, indirect binary n-cube, data manipulator, and ADM can perform each of these shifts in a single pass through the network. In general, nearest neighbor networks cannot perform Shift_i functions in a single pass. (This is discussed for i a power of two in [25, 26].)

The time required for an interprocessor data transfer is highly dependent on the way in which the network is implemented. For example, the eight-stage STARAN flip network can transfer data in 150 nanoseconds [2]. A reasonable assumption is that a data transfer in a $\log_2 N$ stage SIMD network requires no more time than a complex multiplication. (This assumption is not necessary for the results which are presented, since inter-PE transfers are treated separately in the complexity analyses.) For a discussion of multistage SIMD interconnection networks see [28].

Only those SIMD machine features necessary for the algorithms that follow have been described. The model presented is intended to provide a very general framework in which SIMD algorithms can be developed.

III. One-Dimensional FFT Algorithms

A. *Introduction*

The DFT of an M-point sequence $\{s(m)\}$, $0 \leq m < M$, is defined as

$$S(k) = \sum_{m=0}^{M-1} s(m) e^{-j(2\pi/M)mk}, \qquad 0 \leq k < M.$$

Fast Fourier transform algorithms allow computation of the DFT in $O(M \log_2 M)$ serial operations. One formulation, the radix two decimation-in-frequency (DIF) algorithm [10, 21, 24], divides $\{s(m)\}$ into sequences $\{s_1(m)\}$ equal to the first half of $\{s(m)\}$ and $\{s_2(m)\}$ equal to the second half of $\{s(m)\}$. The DFT of the M-point sequence can be computed in terms of the two $M/2$-point DFTs of the sequences $\{s_1(m) + s_2(m)\}$ and $\{[s_1(m) - s_2(m)]W_M^m\}$, where $0 \leq m < M/2$, $W_M = e^{-j(2\pi/M)}$, and $j = \sqrt{-1}$. For M a power of two, this process is repeated $\log_2 M$ times. The

solid lines in Fig. 2a show a flow graph of the computations in performing a 16-point FFT.

Bergland [6], Pease [22, 23], and Stone [34] have described methods for using $M/2$ arithmetic units to perform an M-point FFT. In the next subsection, a similar algorithm is outlined, and some properties of the algorithm are discussed. The algorithm is then used as a basis for developing and evaluating algorithms for different machine-size/problem-size relationships. It is also used in some of the 2-D FFT algorithms presented in Section IV.

B. *SIMD Algorithm for $M/2$ PEs*

For the parallel DIF algorithm, an M-point FFT is performed on an SIMD machine having $N = M/2$ PEs. PE i initially contains $s(i)$ and

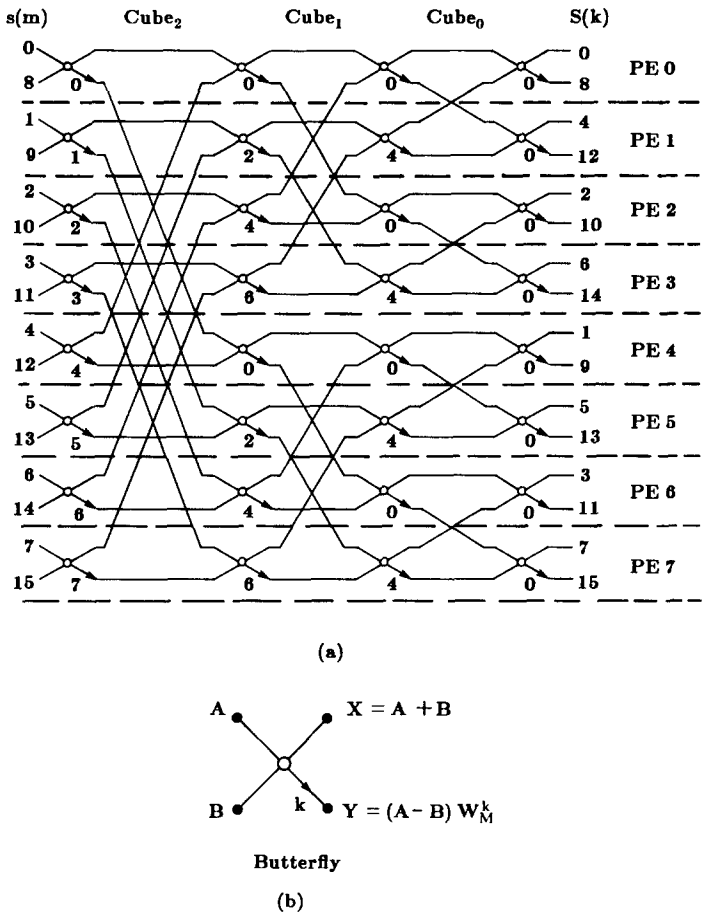

FIG. 2. (a) Computation of a 16-point FFT in 8 PEs. (b) Radix two "butterfly."

$s(i + M/2)$, $0 \leq i < N$. As in the serial method, $\log_2 M$ stages of computations are performed. At each stage, $M/2$ "butterfly" operations, shown in Fig. 2b, are executed. The items paired in a butterfly at stage σ, $0 \leq \sigma < \log_2 M$, are those whose indices differ only in the $(\log_2 M - \sigma - 1)$th bit position of their binary representation. Figure 2a illustrates the data allocation, pattern of data transfers, and computations performed in the parallel algorithm. At stage σ, the weighting factor k (see Fig. 2) is computed in PE i as $k = (i*2^{\sigma}) \bmod M/2$, $0 \leq i < N$. Because of the pairing of items which differ in a given bit position, the Cube interconnection network provides a natural means of specifying the interprocessor data transfers required for the FFT algorithm. The algorithm complexity is given in Table I. The number of butterfly steps is reduced by a factor of $M/2$, the maximum achievable with $M/2$ PEs. The number of data transfers needed is considered below.

Define the function $R(k)$ to be the maximum number of PEs from which a single PE can receive data in k or fewer transfers (where it is assumed that a PE trivially receives data from itself). The data need not remain in its original form. More formally, in a given sequence of k transfers, PE i *depends on* PE j if changing a data value in PE j alters a final value in PE i. Then $R(k)$ will be the maximum number of distinct PE j's such that PE i depends on PE j. It can be shown by induction on k that if a PE can receive at most one data item as the result of the execution of a parallel data transfer, then

TABLE I
COMPLEXITY OF ONE-DIMENSIONAL FFT ALGORITHMS

	Multiplication steps	Addition steps	Data transfer steps
Serial radix 2	$(M/2)\log_2 M$	$M \log_2 M$	—
Serial radix 4	$(3M/8)\log_2 M$	$M \log_2 M$	—
$M/2$ PEs radix 2	$\log_2 M$	$2 \log_2 M$	$\log_2 M - 1$
$M/(2^k)$ PEs[a] radix 2	$2^{k-1}\log_2 M$	$2^k \log_2 M$	$2^{k-1}(\log_2 M - k)$
$M/4$ PEs radix 4	$(3/2)\log_2 M$	$4 \log_2 M$	$(3/2)(\log_2 M - 2)$
$M/(2^k)$ PEs[b] radix 4	$(3/4)2^{k-1}\log_2 M$	$2^k \log_2 M$	$(3/4)2^{k-1}(\log_2 M - k)$
M/R PEs[c] radix R	(R/M)(no. of multiplications in serial radix R)	(R/M)(no. of additions in serial radix R)	$(R - 1)(\log_R M - 1)$

[a] $1 \leq k \leq \log_2 M$.
[b] k even, $2 \leq k \leq \log_2 M$.
[c] R a power of 2.

$R(k) \leq 2^k$. A binary tree structure can be used to attain $R(k) = 2^k$. (Gentleman has proposed a measure that is the "opposite" of $R(k)$, i.e., the data movement required to disseminate data originally available in a single PE [16].)

It can be shown that the computation of an M-point FFT on an SIMD machine with $M/2$ PEs, where the M data points are distributed evenly over the $M/2$ PEs, requires at least $\log_2(M/2)$ data transfer steps. Let l be the address of a PE which, after the FFT computation, contains an element $S(l')$ of the transform sequence, and let d be the number of transfers needed to compute $S(l')$. PE l must contain data from all $M/2$ PEs since $S(l')$ depends on every element of the input sequence $s(m)$, $0 \leq m < M$. Thus $R(d) \geq M/2$, and $d \geq \log_2(M/2)$. Therefore, the algorithm presented attains the lower bound on the number of data trasnfer steps when the M data points are distributed evenly over the $M/2$ PEs.

C. *SIMD Algorithm for Fewer than $M/2$ PEs*

As examples of possible techniques when $M/2$ PEs are not available, algorithms using $N = M/4$ PEs to compute the M-point transform are discussed. These techniques are then generalized for use with fewer PEs.

A simple solution to the problem of having fewer PEs is to replicate the steps in the radix two, $M/2$-PE algorithm. Two computations that were performed in parallel in different PEs are now performed sequentially in the same PE, as shown for $N = 16$ in Fig. 3. The choice of the data points which were allocated to the same PE was made to eliminate the need for a transfer at the first step of the algorithm. This approach can be generalized to perform an M-point FFT in $N = M/(2^k)$ PEs, $2 \leq k \leq \log_2 M$. Each PE will initially contain 2^k elements. The data transfers performed will be the Cube_c functions for $\log_2 N > c \geq 0$; each Cube_c function will be used 2^{k-1} times. The algorithm complexities are given in Table I.

An alternative approach for using $M/4$ PEs when M is a power of four performs the operations of a radix four DIF FFT. The input sequence $\{s(m)\}$ is divided into four sequences $\{s_i(m)\}$ of $M/4$ points, $0 \leq i < 4$, and $0 \leq m < M/4$, where $s_i(m) = s(m + i(M/4))$. The DFT of the M-point sequence can be computed by calculating four $M/4$-point DFTs of subsequences $\{t_i(m)\}$, $0 \leq i < 4$, where for a given m, each $t_i(m)$ value is a function of the four values $s_i(m)$, $0 \leq i < 4$. Figure 4a shows the computations performed in a radix four 16-point FFT [24]. Let $r = \log_4 M$, and let $m = d_{r-1} \cdots d_1 d_0$ be the base four representation of m, $0 \leq m < M$, $0 \leq d_i < 4$. At stage σ in the computation, where the stages are numbered from 0 to $r - 1$, the four items whose indices differ in the digit $d_{r-\sigma-1}$ are combined in the radix four butterfly operation.

The algorithm to compute the M-point DFT using $N = M/4$ PEs will consist of r stages of four-input butterflies. Using the butterfly notation of Fig. 4b, registers A, B, C, and D of PE m are initially loaded with points

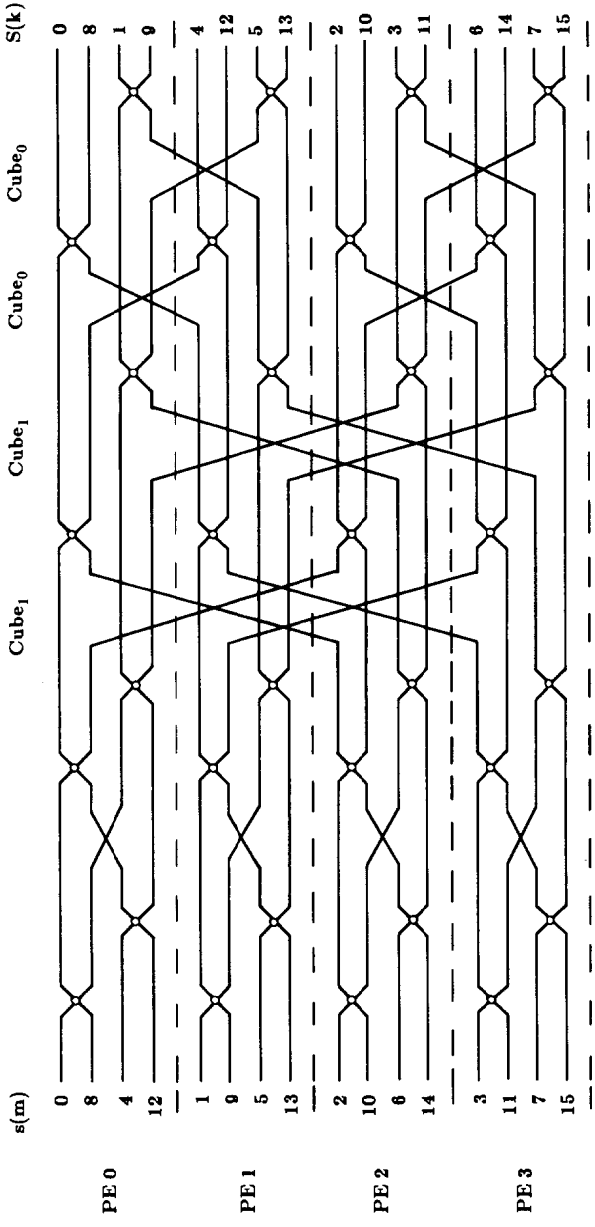

FIG. 3. Computation of a 16-point FFT in 4 PEs, using a radix two algorithm.

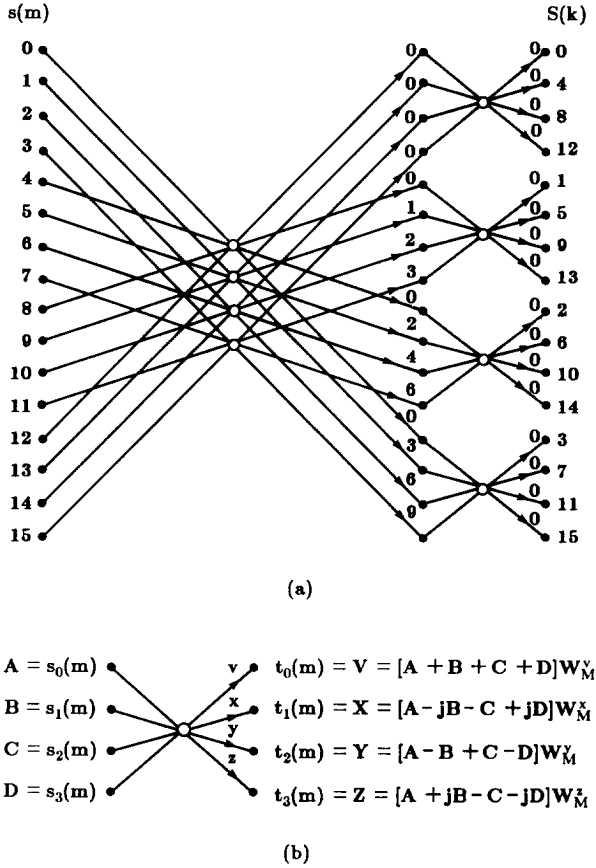

FIG. 4. (a) Operations in the computation of a radix four 16-point FFT [24]. (b) Radix four "butterfly."

$s_0(m)$, $s_1(m)$, $s_2(m)$, and $s_3(m)$, respectively, $0 \leq m < M/4$. The result of one butterfly executed in parallel in all PEs is to generate the four t_i subsequences, one in each of the sets of V, W, Y, and Z registers, with the mth element of each subsequence in PE m, $0 \leq m < M/4$. This is stage 0 of the algorithm. From this point, the same processing steps are performed for each stage of the FFT, and for each subsequence. A butterfly operation in stage $\sigma - 1$ of the FFT, $0 < \sigma < r$, generates four $\{t_i\}$ subsequences, with one $Q = M/4^{\sigma}$-point subsequence in each of the V, X, Y, and Z registers of a partition consisting of $Q = M/4^{\sigma}$ PEs. (The Q PEs in a partition will be the Q PEs consecutively numbered from k to $k + Q - 1$, where $k = pQ$, $0 \leq p < N/Q$.) In order to process each Q-point subsequence, the size-Q partition is quartered, and the data are moved as shown in Fig. 5. Within each partition at stage σ, the four subsequence elements whose indices dif-

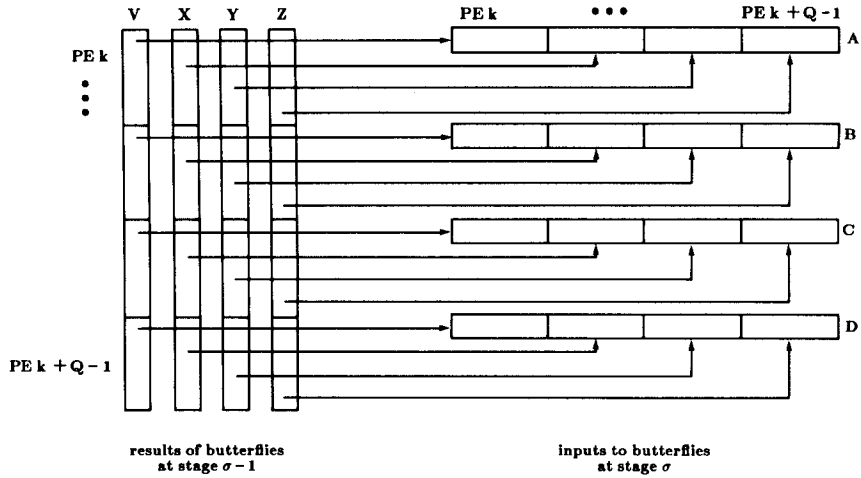

FIG. 5. Alignment of data for stage σ of $M/4$-PE radix four FFT, $0 < \sigma < r$. $Q = M/4^{\sigma}$.

fer in the δth digit, where $\delta = \log_4(Q/4) = r - \sigma - 1$, are the A, B, C, and D inputs to a butterfly. The data transfers from the size-Q partition to the four size-$Q/4$ partitions must therefore be done so that these four elements are in the same PE. Each PE retains one of its data items and receives data from three other PEs. Since within a partition, the lth element of each of the V, X, Y, and Z subsequences is in PE $k + l$, $0 \leq l < Q$, the elements needed for the stage σ butterfly to be performed in PE $k + l$ are in the PEs whose addresses, base four, differ from l in the δth digit. This is accomplished if the first $Q/4$ elements of each of the V, X, Y, and Z subsequences move to the A registers of the appropriate size-$Q/4$ partition, the second $Q/4$ elements to the B registers, and so on, with the relative order of the elements being preserved among each set of $Q/4$ elements. Table II summarizes the transfers of the data in the V, X, Y, and Z registers that must be performed.

The transfers needed at each stage can be accomplished in three parallel data transfers if the interconnection network needs only one step to perform the uniform shifts mod Q within each group of Q consecutive PEs. In [20], it is shown that an omega network of size N can perform uniform shifts mod N. In [27], it is shown that an omega can be partitioned into subnetworks (of size Q, for example) each of which has the properties of an omega of size Q. Thus, the omega network can perform each of these shifts in one step. From the relationships in [28, 31], the indirect binary n-cube, generalized cube (with "individual box control"), and ADM networks can do this also. The radix four, $M/4$-PE algorithm complexity is summarized in Table I. The algorithm achieves the maximum possible reduction in the number of arithmetic operations that an $M/4$-PE algorithm can achieve over a serial radix four FFT.

TABLE II
DATA TRANSFERS NEEDED TO ALIGN DATA FOR STAGE σ OF $M/4$-PE, RADIX FOUR FFT.[a]

Logical PE l	Register V	Register X	Register Y	Register Z	Destination
$0 \le l < Q'$	—	$+Q'$	$+2Q'$	$+3Q'$	A register
$Q \le l < 2Q'$	$+3Q'$	—	$+Q'$	$+2Q'$	B regester
$2Q \le l < 3Q'$	$+2Q'$	$+3Q'$	—	$+Q'$	C register
$3Q \le l < Q$	$+Q'$	$+2Q'$	$+3Q'$	—	D register

[a] $0 \le \sigma < r$. $Q = M/4^{\sigma}$; $Q' = Q/4$. Arithmetic is mod Q and the PEs are logically numbered from 0 to $Q - 1$. An entry "$+d$" for register R of PE l means that the contents of register R are to be moved from PE l to the destination register of PE$(l+d)$ mod Q.

In a manner similar to the way in which the radix two $M/2$-PE algorithm was generalized for use when $N = M/(2^k)$, the radix four $M/4$-PE algorithm can be adapted for use when $N = M/(4^p)$, M a power of four, $2 \le p \le \log_4 M$. The algorithm complexities are summarized in Table I, letting $k = 2p$, and $N = M/(2^k)$.

The above techniques can also be generalized in the sense that it is possible to derive a radix eight algorithm for $N = M/8$, a radix sixteen algorithm for $N = M/16$, etc. For the radix $R = 2^p$ M/R-PE algorithm, it will be required for M to be a power of R. The number of arithmetic operations will be reduced from the number for a serial radix R algorithm by a factor of $N = M/R$. As in the radix four case, the transfers required will be uniform shifts mod R^y, for $1 \le y < \log_R M$. The number of parallel transfers at each stage will be $R - 1$.

Table I summarizes the results for computing an M-point FFT by the various 1-D algorithms considered. For an M-point sequence, the $M/2$-PE algorithm is the fastest, but requires the greatest number of PEs. For a given machine size $N = M/(2^k)$, $1 < k < \log_2 M$, the radix four algorithm requires $2^{k-3} \log_2 M$ fewer multiplication steps and $2^{k-3}(\log_2 M - k)$ fewer transfer steps than the radix two algorithm for N PEs, but the radix four algorithm is restricted by the condition that M and 2^k be powers of four. (It is possible to remove this restriction for the case where M and 2^k (and therefore N) are powers of two but not of four. Use a radix two DIF algorithm for the first stage, producing two subproblems of size $M/2$. Use the radix four method with $N/2$ PEs on the two subproblems. Since N and M were powers of two but not of four, both $N/2$ and $M/2$ are powers of four.) In general, for a given machine size N, fewer transfer steps are needed for a radix R algorithm, $R = 2^p$, $R > 2$, than for a radix two algorithm using the same number of PEs. For $R > 16$, the difficulty of developing serial radix

R algorithms is generally considered to outweigh possible gains in speed. For $2 \leq R \leq 16$, however, the number of serial arithmetic operations decreases as R increases [10]. For R in this range, it will therefore be advantageous to choose R as large as possible, since both arithmetic and transfer operations will be reduced. In all cases, the number of "overhead" steps—i.e., steps incurred in performing inter-PE data alignment rather than performing actual computations of the DFT—is less than the number of computation steps. The asymptotic complexity of each N-PE parallel algorithm is therefore $(1/N)$th the asymptotic complexity of the corresponding serial algorithm.

Another alternative method which can be used when $N < M/2$ is presented in [19]. For all of these approaches, the method of choice will depend upon the actual values of N and M.

IV. Two-Dimensional FFT Algorithms

A. *Introduction*

A function which can be useful for studying the spatial spectral characteristics of an $M \times M$ digitized image is the 2-D $M \times M$ point discrete Fourier transform

$$F(v, w) = \sum_{l=0}^{M-1} \sum_{m=0}^{M-1} S(l, m) W_M^{vl} W_M^{wm}, \qquad 0 \leq v, w < M,$$

where S is the input image, and F is the DFT of S.

For an SIMD machine, the number of PEs and the way in which the data are distributed play a large part in determining how the 2-D DFT is implemented. Some implementations for various machine sizes are discussed below.

B. *Row–Column Methods*

A standard approach to computing the 2-D DFT is to perform the 1-D DFT on the rows of S, giving an intermediate matrix G, then performing the 1-D DFT on the columns of G. The resulting matrix F^T is the transpose of the 2-D DFT of S [24]. F^T can be transposed to give F; however, this may not be necessary depending on what further processing is done on F.

1. *SIMD Algorithms for $N \leq M$ PEs.* Suppose that an SIMD machine has $N = M$ PEs, each of which has one row of an $M \times M$ input image, S. An efficient method for obtaining F, the DFT of S, is to perform M 1-D FFTs in parallel on the rows of S to get G, transpose G among the PEs, then perform M 1-D FFTs in parallel on the columns of G to get F^T. This is shown in Fig. 6.

To form the transpose, G^T, of G such that each row of G^T is in a different

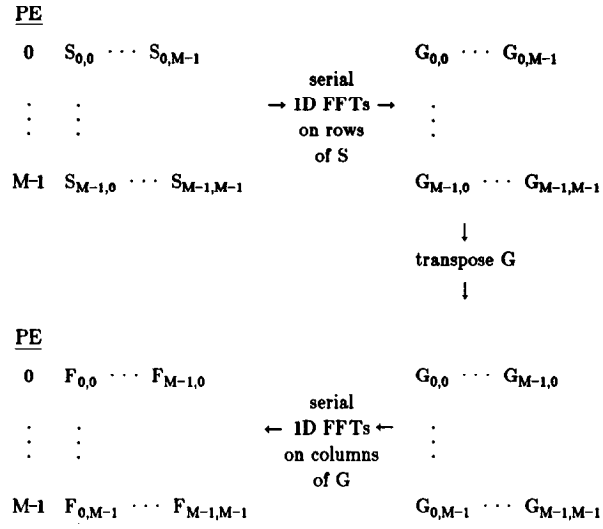

FIG. 6. Computation of two-dimensional DFT of $M \times M$ array S using M PEs.

PE, the basic operation performed is the transfer of array element $G(l, w)$ from PE l to PE w. This is done for M $G(l, w)$'s in parallel, using an interconnection function which sends data from PE l to PE $(l + i) \bmod M$ for all of the $G(l, w)$ for which $(w - l) \bmod M = i$. The parallel transfer operation is performed once for each i, $1 \leq i < M$. For each i value, the element which PE l sends is the wth element of the row of G held in PE l, i.e., $G(l, w)$, where $w = (l + i) \bmod M$. That element, received in PE w, is stored as the lth element of the row of G^T being created in PE w, i.e., $G^T(w, l)$, where $l = (w - i) \bmod M$. The transfers can be expressed in terms of the Shift interconnection functions. Performing the transpose therefore requires $M - 1$ parallel data transfers for networks which can perform each Shift function in a single pass.

The serial complexity of 2M 1-D FFTs (i.e., an M × M 2-D DFT) is $M^2 \log_2 M$ butterflies. The above parallel implementation of the 2-D DFT executes two serial FFT algorithms and has a complexity of $M \log_2 M$ butterfly steps. Thus, an ideal speedup of M is achieved for butterfly operations with a cost of $M - 1$ data transfers. A counting argument can be used to show that the transpose algorithm achieves the lower bound on the number of transfers needed to transpose G. Element $G(i, j)$ must be transferred from PE j, for all $0 \leq i, j < M$, and $i \neq j$. The total number of elements which must be transferred is therefore $M(M - 1)$. At most M elements can be moved in one data transfer step, so at least $M - 1$ data transfer steps are needed.

This approach can be generalized for $N \leq M$. For example, if $N = M/2$, a speedup of approximately N can be achieved. For this implementation

each PE is given two rows of the input matrix S. The FFTs on the rows of S are performed by two serial FFTs, executed one after the other, on the two rows in each PE. This yields G, with each PE having two rows of G. The second step is to form the transpose of G, G^T, where each PE has two rows of G^T (i.e., each PE has two columns of G). If PE i contains rows $2i$ and $2i + 1$, then, in general, $G(r, c)$ is transferred from PE $[r/2]$ to PE $[c/2]$, $0 \leq r, c < M$. The complexity associated with the transpose is $2M - 4$ parallel transfers. The -4 term appears because the diagonal and near diagonal terms are already in the correct PE. The final step is to perform a 1-D DFT on the columns of G. This is done by two serial FFTs in each PE, as above. This gives F^T, with each PE having two rows of F^T. This implementation has a complexity of four serial FFT algorithms, or $2M \log_2 M$ butterfly steps. This is the maximum possible reduction in the number of butterfly steps given $M/2$ PEs. The overhead associated with the transpose is $2M - 4$ transfers.

When this "row–column" method is implemented on N PEs, $N \leq M$, the complexity will be derived directly from the 1-D FFT algorithm used. If the complexity of the serial 1-D FFT algorithm is C, then the complexity of the 2-D FFT algorithm is $2[M/N]C$ plus the cost of performing the transpose. If $N = M/(2^p)$, the cost of the transpose is $2^p(M - 2^p)$ data transfers. The -2^p term appears because before the transpose each PE holds 2^p rows, and after the transpose each PE holds 2^p columns. Thus, only $M - 2^p$ elements of each row need to be transferred. The algorithm complexity is given in Table III.

2. *SIMD Algorithms for $M < N \leq M^2/2$ PEs.* When more than M PEs are available, the row–column approach can use parallel 1-D algorithms such as those presented in Section III to transform the rows (columns). An algorithm for $M^2/2$ PEs is described, then generalized for $N < M^2/2$ PEs.

For $N = M^2/2$, the PEs are initially logically configured as M rows of $M/2$ PEs, logically numbered (i, j), where $0 \leq i < M$ and $0 \leq j < \mathrm{M}/2$. The physical address of PE(i, j) is $i(M/2) + j$. The physical address can be represented in binary as $p_{2\mu-2}p_{2\mu-3} \cdots p_{\mu-1}p_{\mu-2} \cdots p_1p_0$, where $\mu = \log_2 M$. Bits $p_{\mu-2} \cdots p_0$ are the binary representation of j, and bits $p_{2\mu-2} \cdots p_{\mu-1}$ are the binary representation of i. The input matrix S is distributed such that PE(i, j) has $S(i, j)$ and $S(i, j + M/2)$. Thus, each row of PEs can perform the 1-D FFT on its row of S using the method described in Section III for 1-D FFTs with $N = M/2$. In this case, the Cube functions required for data transfers will exchange data based on the low order $\mu - 1$ bits of the physical address; i.e., the functions will act on j independently of i. Thus, the 1-D FFT can be performed on each row independently and on all rows simultaneously. The result, G, is distributed such that each column of PEs holds two columns of G, with each PE holding one element from each of the two columns of G.

The PEs are now logically reconfigured to form M columns of $M/2$ PEs,

TABLE III
COMPLEXITY OF TWO-DIMENSIONAL FFT ALGORITHMS

	Multiplication steps	Addition steps	Data transfer steps
Serial radix 2	$M^2 \log_2 M$	$2M^2 \log_2 M$	—
M PEs row–column	$M \log_2 M$	$2M \log_2 M$	$M - 1$
$M/(2^r)$ PEs row–column[a]	$2^r M \log_2 M$	$2^{r+1} M \log_2 M$	$2^r M - 2^{2r}$
$M^2/(2^k)$ PEs row–column[b]	$2^{k-\mu} M \log_2 M$	$2^{k-\mu+1} M \log_2 M$	$2^{k-\mu} M - 2^{2(k-\mu)}$
$M^2/2$ PEs row–column	$2 \log_2 M$	$4 \log_2 M$	$2 \log_2 M - 1$
$M^2/(2^k)$ PEs[c] row–column	$2^k \log_2 M$	$2^{k+1} \log_2 M$	$2^k(\log_2 M - k) + 2^k - 1$
$M^2/4$ PEs 2-D DIF	$4 \log_2 M$	$8 \log_2 M$	$3(\log_2 M - 1)$
$M^2/(2^k)$ PEs[d] 2-D DIF	$2^k \log_2 M$	$2^{k+1} \log_2 M$	$2^{k-2} 3(\log_2 M - k/2)$

[a] $0 \leq r < \log_2 M$.
[b] This entry is the same as the previous entry with r replaced by $k - \mu$, where $\mu = \log_2 M$. $\text{Log}_2 M \leq k < 2 \log_2 M$.
[c] $0 < k < \log_2 M$.
[d] k even, $0 < k < 2 \log_2 M$.

with each column of PEs having one column of G, two column elements in each PE. To do this, PE(i, j) is renumbered (k, l) where $k = i \bmod (M/2)$ and $l = 2j + \lfloor i/(M/2) \rfloor$. Effectively, this renumbering takes each column of the original configuration, divides it in half, and aligns the halves to form two adjacent columns as shown in Fig. 7. (The lower half of Fig. 7 shows the reconfiguration but not the renumbering.) In terms of the binary representation of the physical address, the binary representation of k is $p_{2\mu-3} \cdots p_{\mu-1}$, and the binary representation of l is $p_{\mu-2} \cdots p_0 p_{2\mu-2}$.

After the renumbering, G is distributed such that each pair of columns of PEs, PE$(k, 2\lambda)$ and PE$(k, 2\lambda + 1)$, where $0 \leq k, \lambda < M/2$, holds two columns of G. Within such a pair of columns of PEs, PE$(k, 2\lambda)$ has two points from the kth row of G, and PE$(k, 2\lambda + 1)$ has the two corresponding points from the $(k + M/2)$th row of G. Within each of the two columns of G, these are precisely the points which must be paired at the start of the 1-D M-point, $M/2$-PE algorithm. Using the $\text{Cube}_{2\mu-2}$ function, PE$(k, 2\lambda)$ and PE$(k, 2\lambda + 1)$, $0 \leq k, \lambda < M/2$, can exchange data so that each column of PEs gets a different column of G, with each PE holding the two elements of G needed to start the 1-D FFT. In terms of physical PE addresses, $p_{2\mu-2}$ corresponds to the high order bit position and $\text{Cube}_{2\mu-2} = \text{Cube}_{n-1}$. In terms

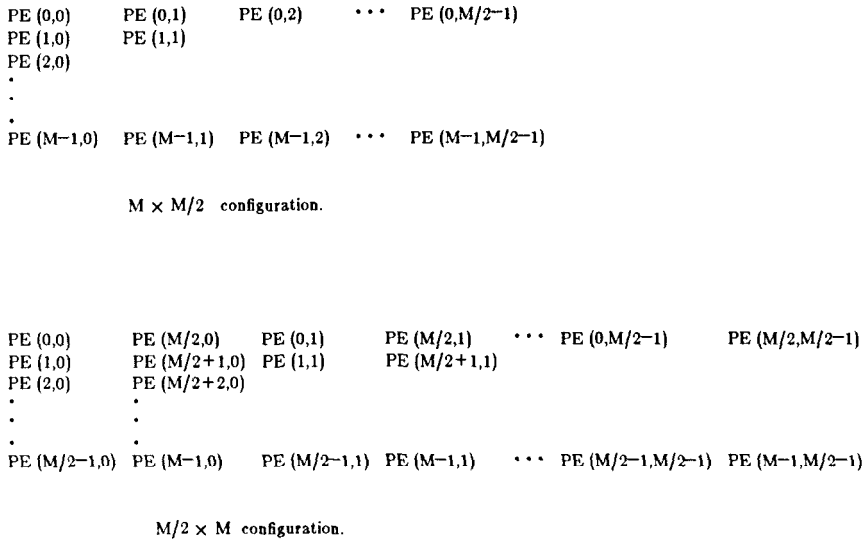

FIG. 7. Reconfiguration of PEs from $M \times M/2$ grid to $M/2 \times M$ grid, without renumbering.

of the logical numbering, $p_{2\mu-2}$ is the low order bit in the binary representation of the logical column index, so the $\text{Cube}_{2\mu-2}$ function effects the exchange of elements between the corresponding rows in columns 2λ and $2\lambda + 1$, $0 \leq \lambda < M/2$.

Each column of PEs now performs the 1-D FFT on its column of G using the method described in Section III. However, the Cube interconnection functions used are chosen differently. To perform the FFT on a column, it is necessary to perform data transfers based on the row index, which in this case is given by k, represented by bits $p_{2\mu-3} \cdots p_{\mu-1}$ of the physical PE address. Therefore, whenever Cube_i is executed in the original algorithm, $\text{Cube}_{i+\mu-1}$ is executed in this algorithm. In this way, the Cube functions allow communication within a column instead of a row. The complexity of this algorithm is derived from the 1-D case, and is summarized in Table III. The speedup for arithmetic operations is $M^2/2$, which is ideal. It is worth noting that all but one of the transfers are incurred by the 1-D FFTs. The reconfiguring of the data between the row transform step and the column transform step—in effect, the "transpose" operation—is done by the logical renumbering and a single inter-PE transfer.

This approach can be generalized for $N = M^2/2^k$, $1 < k \leq \log_2 M$. (In the case $k = \log_2 M$, this approach reduces to the method given above for $N = M$.) In the generalized algorithm the PEs are logically configured as M rows of $M/2^k$ columns. Each row of the PEs holds a row of S; each PE holds 2^k elements of S. Each row of PEs executes a 1-D FFT on its row of S using a 1-D FFT algorithm for $N = M/2^k$ PEs, as discussed in Section III.

The PEs are then logically reconfigured to form M columns of $M/2^k$ rows. This is done by dividing each column of PEs into 2^k segments of $M/2^k$ PEs, and aligning the segments to form 2^k columns. This is shown for column j in Fig. 8. After the reconfiguration, G is distributed so that each group of 2^k columns of PEs holds 2^k columns of G, with each PE holding one element from each of 2^k columns of G. The data are then aligned so that each column of PEs holds a different column of G with each PE holding 2^k elements of a single column of G. This alignment can be done using $2^k - 1$ transfer steps, as follows. If the columns within each group of 2^k columns are numbered from 0 to $2^k - 1$, then the alignment can be done by having column j send its data to column $(j + r) \bmod 2^k$ in transfer step r, where $0 \leq j < 2^k$, $0 < r < 2^k$. The physical addresses of the $M = 2^\mu$ PEs in each group of 2^k columns agree in their $\mu - k$ lower order bit positions. The transfers are among the corresponding elements of the columns of each group. Specifically, the numbering of the columns within a group and the transfers are based on the high order k physical bit positions. Thus, the logical shifts of $+r \bmod 2^k$ are physical shifts of $+r(2^{2(\mu-k)}) \bmod N$, $0 < r < 2^k$. The needed transfers are therefore uniform shifts, mod N. After the data are aligned, each column of PEs executes a 1-D FFT on its column of G using a 1-D FFT algorithm for $N = M/2^k$ PEs.

The complexity of this general method is derived directly from the 1-D FFT algorithm used. If the complexity of the general 1-D FFT algorithm is C, then the complexity of the 2-D FFT is $2C$ plus the cost of the re-

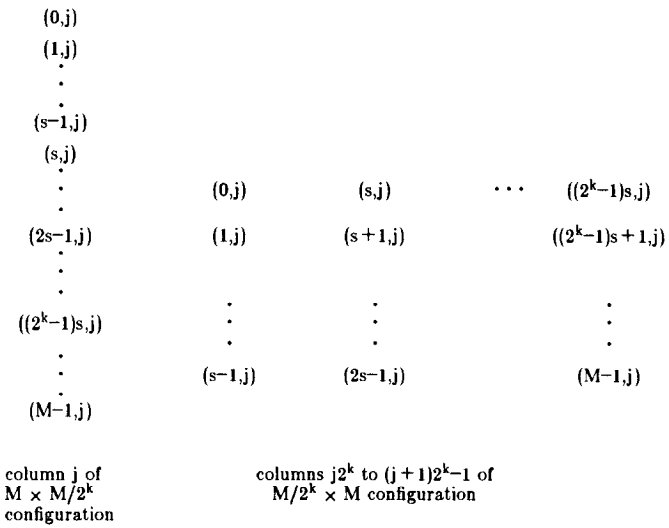

FIG. 8. Reconfiguration of column j in $M \times M/2^k$ grid to columns $j2^k$ to $(j + 1)2^k - 1$ in $M/2^k \times M$ grid, where $0 \leq j < M/2^k$, and $s = M/2^k$.

configuring. Table III summarizes the algorithm complexity, under the assumption that the radix two algorithm presented in Section III is used. The $2^k - 1$ term in the data transfer complexity is the number of transfers required to align the data after the logical renumbering of the PEs.

3. *Relationship between the Row–Column Methods.* The two row–column methods (i.e., the generalizations of the M-PE and $M^2/2$-PE methods) can be thought of as a single method defined on different ranges of N (the machine size). The $M^2/2$-PE method was generalized for machines of size $N = M^2/(2^k)$, $0 < k < \log_2 M$, whereas the M-PE method was generalized for machines of size $N = M/(2^r)$, $0 \leq r < \log_2 M$. For the latter method, the range for N can be rewritten as $N = M^2/(2^k)$, $\log_2 M \leq k < 2 \log_2 M$. Thus, the row–column method is defined for $N = M^2/(2^k)$, $0 < k < 2 \log_2 M$, where the $M^2/2$-PE approach is used if $0 < k < \log_2 M$ and the M-PE approach is used if $\log_2 M \leq k < 2 \log_2 M$. The fundamental difference between the approaches applied to the two ranges is that when the number of PEs is such that each PE holds at least a full row (column), i.e., for $\log_2 M \leq k < 2 \log_2 M$, serial 1-D algorithms are used to transform the rows (columns). When N is such that a row (column) is distributed over more than one PE, i.e., for $0 < k < \log_2 M$, parallel 1-D algorithms, such as those presented in Section III, are used to transform the rows (columns).

C. *Two-Dimensional Decimation-in-Frequency Method*

In the case where N is a perfect square, a different approach can be used. Instead of reducing the 2-D DFT to 1-D DFTs, a 2-D decimation-in-frequency (DIF) algorithm is derived. This is shown first for $N = M^2/4$, then generalized for smaller N.

The 2-D FFT is obtained by dividing the matrix into four quadrants and obtaining expressions for the $M \times M$ point DFT in terms of four $M/2 \times M/2$ point DFTs. The four DFTs are given by

$$F_i(v, w) = \sum_{l=0}^{M/2-1} \sum_{m=0}^{M/2-1} S_i(l, m)\, W_{M/2}^{lv/2}\, W_{M/2}^{mw/2}, \qquad 0 \leq v, w < M,$$

for $0 \leq i < 4$, where for F_0, v and w are even; for F_1, v is even and w is odd; for F_2, v is odd and w is even; and for F_3, v and w are odd. The S_i's are given by

$$S_0(l, m) = S(l, m) + S(l, m + M/2) + S(l + M/2, m) + S(l + M/2, m + M/2)$$

$$S_1(l, m) = [S(l, m) - S(l, m + M/2) + S(l + M/2, m) - S(l + M/2, m + M/2)]\, W_M^m$$

$$S_2(l, m) = [S(l, m) + S(l, m + M/2) - S(l + M/2, m) - S(l + M/2, m + M/2)]\, W_M^l$$

$$S_3(l, m) = [S(l, m) - S(l, m + M/2) - S(l + M/2, m) + S(l + M/2, m + M/2)]\, W_M^m\, W_M^l$$

for $0 \leq l, m < M/2$. Each F_i is the 2-D $M/4 \times M/4$ point DFT of S_i. This process can be repeatedly applied to obtain a 2-D DIF algorithm.

The SIMD machine is viewed as an $M/2 \times M/2$ matrix of PEs. Each PE is given a logical address of the form (l, m), with logical PE (l, m) corresponding to physical PE $lM/2 + m$. Initially, PE (l, m) is given data points $S(l, m)$, $S(l, m + M/2)$, $S(l + M/2, m)$, and $S(l + M/2, m + M/2)$. The algorithm consists of $\log_2 M$ steps, numbered from 1 to $\log_2 M$. The first step is to compute $S_i(l, m)$, $0 \leq i < 4$, $0 \leq l, m < M/2$, for the original $M \times M$ image. The jth step is to compute $S_i(l, m)$, $0 \leq i < 4$, $0 \leq l, m < M/2^j$, for each of the $M/2^{j-1} \times M/2^{j-1}$ matrices. Each step can be divided into two parts: the parallel data transfers to align the data and the computations required to compute the expressions S_0, S_1, S_2, and S_3.

In the first step, the data are already aligned and S_0, S_1, S_2, and S_3 are computed. At the end of the first step, PE (l, m) contains one point from each of the four matrices of which the DFTs need to be computed to give the required $M \times M$ DFT. The next step is to divide the set of PEs, PE (l, m), $0 \leq l, m < M/2$, into four quadrants, as shown in Fig. 9. Each quadrant will compute the DFT of one of the four matrices. The data transfers for this step must put the data for each DFT into a different quadrant. Consider the case where there are $N = 2^n$ PEs and $\sqrt{N}$ is an integer. If the array of PEs is partitioned into quadrants, then: (1) PEs in quadrants 0 and 1 can exchange data with the corresponding PEs in quadrants 2 and 3 using the Cube_{n-1} interconnection function; (2) PEs in quadrants 0 and 2 can exchange data with quadrants 1 and 3 using the $\text{Cube}_{n/2-1}$ function; and (3) PEs in diagonally oposite quadrants can exchange data using the $\text{Cube}_{n/2-1}$ and Cube_{n-1} functions. (The omega, indirect binary n-cube, generalized cube, and ADM networks can implement the combination of the $\text{Cube}_{n/2-1}$ and Cube_{n-1} functions in a single transfer.) After the data are aligned, each

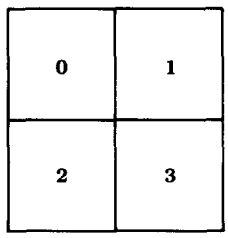

FIG. 9. Quadrant labeling.

quadrant can compute the DFT of the matrix it contains. For example, quadrant i can compute the DFT of S_i.

This process of dividing each set of PEs into four quadrants, and then assigning a different DFT to each quadrant, is repeated. Interconnection functions analogous to those described above can be used to align the data. The total number of steps (including the first step) is $\log_2 M$. After all $\log_2 M$ steps have been executed, each PE contains four values, each of which is an element in the $M \times M$ transform matrix. The algorithm complexity is given in Table III. The speedup for arithmetic operations is $M^2/4$, which is equal to the number of PEs used, which is ideal.

This technique can be generalized for machines of size $N = M^2/(4^p)$, $0 < p < \log_2 M$. For $p > 1$, computations performed in parallel in different PEs in the $M^2/4$ algorithm are now performed sequentially in the same PE. The number of steps in the algorithm is still $\log_2 M$; however, each step computes 4^{p-1} sets of S_0, S_1, S_2, and S_3. Thus, the number of computations is increased by a factor of 4^{p-1}. The number of transfers is also increased by a factor of 4^{p-1}, except that in the last $p - 1$ steps no transfers are required. This is because after each step of the algorithm the problem is reduced to finding four times as many DFTs of matrices one-quarter the size as before. If $N = M^2/(4^p)$, each PE initially has 4^p points. After $\sigma = \log_2 M + 1 - p$ steps, the size of the matrices of which the DFT is being computed is reduced from $M \times M$ to $M/2^{\sigma-1} \times M/2^{\sigma-1} = 4^p$ points. So, for the last $p - 1$ steps no transfers are required. The algorithm complexity is summarized in Table III, letting $k = 2p$, and writing the number of PEs as $N = M^2/(2^k)$.

D. *Comparison of Two-Dimensional Algorithms*

Table III summarizes the results for computing a 2-D $M \times M$ point FFT by the various algorithms presented. For a given problem size and machine size, the information in this table can be used to determine which of the algorithms presented is fastest. Table III shows that for a given problem size and machine size, the arithmetic complexity is the same for the 2-D DIF method and the row–column method (where this will refer to the generalizations of both the M-PE and $M^2/2$-PE algorithms). Therefore, the fastest algorithm is the one which uses the fewest data transfers. For an $M \times M$ matrix and $M^2/(2^k)$ PEs, where $0 < k < 2 \log_2 M$, a choice must be made between the 2-D DIF and row–column methods. For $\log_2 M \leq k < 2 \log_2 M$, corresponding to $1 < N \leq M$, the choice will be between the 2-D DIF approach and the generalization of the M-PE method. For $0 < k < \log_2 M$, corresponding to $M < N \leq M^2/2$, the choice will be between the 2-D DIF approach and the generalization of the $M^2/2$-PE method. Since the 2-D DIF method can be used only when k is even, only cases for which k is even will be considered.

For $\log_2 M \leq k < 2 \log_2 M$ (i.e., $1 < N \leq M$), the comparison of the

two methods can be obtained by solving the inequality

$$2^{k-2}3(\log_2 M - (k/2)) > 2^{k-\mu}M - 2^{2(k-\mu)},$$

where $\mu = \log_2 M$. The left-hand side represents the number of transfers in the 2-D DIF algorithm and the right-hand side is the number of transfers in the row–column method. Substituting for $\mu = \log_2 M$ and $k = 2\log_2 M - \log_2 N$ and simplifying gives

$$3\log_2 N + (8/N) > 8.$$

The inequality is clearly true for $N \geq 16$ (recalling that k must be even, i.e., N must be a perfect square). Thus, for $16 \leq N \leq M$, the row–column method will be faster than the 2-D DIF algorithm. For $N = 4$, the only remaining allowable value for N, equality holds between the two sides of the equation, meaning that the two methods will perform the same number of transfers.

For $0 < k < \log_2 M$ (i.e., $M < N \leq M^2/2$), the comparison of the two methods can be obtained by solving the inequality

$$2^k(\log_2 M - k) + 2^k - 1 > 2^{k-2}3(\log_2 M - (k/2)),$$

where the left-hand side of the inequality is the number of transfers incurred by the row–column method and the right-hand side is the number of transfers for the 2-D DIF algorithm. If an error of one transfer or less is acceptable (i.e., the method chosen for particular values of M and N will require at most one transfer more than the fastest method), then the -1 term can be dropped, giving

$$2^k(\log_2 M - k) + 2^k > 2^{k-2}3(\log_2 M - (k/2)).$$

Simplifying and substituting $2\log_2 M - \log_2 N$ for k gives

$$(2/5)\log_2 M + 8/5 > 2\log_2 M - \log_2 N,$$

which simplifies to

$$2^{8/5}N > M^{8/5}.$$

Thus, when $2^{8/5}N > M^{8/5}$ and $M < N \leq M^2/2$ (i.e., $0 < k < \log_2 M$), the 2-D DIF method is faster, and when $2^{8/5}N < M^{8/5}$ and $M < N \leq M^2/2$, the row–column method is faster. These results can be interpreted by considering two types of decisions which involve the choice of an algorithm. First, given a machine of size N, and assuming $M < N \leq M^2/2$, determine for which values of M the 2-D DIF method is faster than the row–column method. From the above inequalities, the DIF

approach will be faster for $M \times M$ images for which $M < 2N^{5/8}$. For example, for $N = 256$, the DIF approach is faster for images for which $M < 64$; for $N = 1024$, the DIF method is faster for images for which $M < 152$. Alternatively, given an $M \times M$ image, and assuming that $M < N \leq M^2/2$, determine for which machine sizes N the DIF method is faster. Again from the above analysis, it can be shown that the DIF method will be faster for $N > (M/2)^{8/5}$. For example, for a 128×128 image, a machine of more than 776 PEs will be needed for the DIF algorithm to be faster; for a 256×256 image, N must be greater than 2353 for the DIF algorithm to be faster; for a 512×512 image, the DIF algorithm will be faster when $N > 7132$.

Combining the above results, a comparison of the 2-D decimation-in-frequency and row–column methods can be made over the full range for which the algorithms have been defined, i.e., for $1 < N \leq M^2/2$. The row–column method will be faster for $1 < N < (M/2)^{8/5}$, and the 2-D DIF method will be faster for $(M/2)^{8/5} < N \leq M^2/2$.

V. Conclusions

Parallel 1-D and 2-D FFT algorithms for a variety of problem size–machine size combinations have been presented and analyzed. The design and analysis of parallel algorithms differs from that of serial algorithms in that the asymptotic time complexity of a serial algorithm will be approximately the same for any canonic model of a serial computer [1]. As has been shown by the algorithms above, the complexity of parallel algorithms depends on architecture features such as the number of PEs and the type of interprocessor connection network, and on the way in which the data are distributed among the processors. For each of the algorithms developed, a precise description of the inter-PE data transfers needed has been presented, along with an examination as to which networks proposed in the literature can perform the needed inter-PE transfers efficiently.

The analyses presented provide a means of evaluating the effect of machine size on FFT algorithms. For a given machine size and signal or image size, the analyses provide the information needed to select an algorithm. For a given signal or image size, the relative speeds attainable by machines of different sizes can be compared. For a given signal or image size and given speed requirements, a machine size can be selected. The algorithm analyses therefore provide a valuable tool for both potential SIMD machine users and designers.

Acknowledgments

The authors wish to thank P. H. Swain for his comments. Preliminary versions of portions of this material were presented by the authors at the Seventeenth Annual Allerton Conference

on Communication, Control, and Computing (Oct. 1979) and the Fifth International Conference on Pattern Recognition (Dec. 1980).

REFERENCES

[1] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms*. Addison–Wesley, Reading, Mass., 1976.

[2] Batcher, K. E. STARAN parallel processor system hardware. *AFIPS Conf. Proc. 1974 NCC*, May 1974, Vol. 43, pp. 405–410.

[3] Batcher, K. E. The flip network in STARAN. *1976 Int. Conf. Parallel Processing*, Aug. 1976. pp. 65–71.

[4] Batcher, K. E. STARAN series E. *1977 Int. Conf. Parallel Processing*, Aug. 1977, pp. 144–153.

[5] Batcher, K. E. Bit serial parallel processing systems. *IEEE Trans. Comput.* **C-31** (May 1982), 337–384.

[6] Bergland, G. D. Fast Fourier transform hardware implementations—an overview. *IEEE Trans. Audio Electroacoust.* **AU-17** (June 1969), 104–108.

[7] Bergland, G. D., and Wilson, D. E. A fast Fourier transform for a global, highly parallel processor. *IEEE Trans. Audio Electroacoust.* **AU-17** (June 1969), 125–127.

[8] Bergland, G. D. A parallel implementation of the fast Fourier transform algorithm. *IEEE Trans. Comput.* **C-21** (Apr. 1972), 366–370.

[9] Bouknight, W. J., et al. The Illiac IV system. *Proc. IEEE* **60** (Apr. 1972), 369–388.

[10] Brigham, E. O. *The Fast Fourier Transform*. Prentice–Hall, Englewood Cliffs, N.J., 1974.

[11] Collesidis, R. A., Dutton, T. A., and Fisher, J. R. An ultra-high speed FFT processor. *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Apr. 1980, pp. 784–787.

[12] Corinthios, M. J. The design of a class of fast Fourier transform computers. *IEEE Trans. Comput.* **C-20** (June 1971), 617–623.

[13] Corinthios, M. J. A fast Fourier transform for high-speed signal processing, *IEEE Trans. Comput.* **C-20** (Aug. 1971), 843–846.

[14] Feng, T. Data manipulating functions in parallel processors and their implementations, *IEEE Trans. Comput.* **C-23** (Mar. 1974), 309–318.

[15] Flynn, M. J. Very high-speed computing systems. *Proc. IEEE* **54** (Dec. 1966), 1901–1909.

[16] Gentleman, W. M. Some complexity results for matrix computations on parallel processors. *J. Assoc. Comput. Mach.* **25** (Jan. 1978), 112–115.

[17] Gold, B., and Bially, T. Parallelism in fast Fourier transform hardware. *IEEE Trans. Audio Electroacoust.* **AU-21** (Feb. 1973), 5–16.

[18] Gottlieb, P., and De Lorenzo, L. J. Parallel data streams and serial arithmetic for fast Fourier transform processors. *IEEE Trans. Acoust. Speech Signal Process.* **ASSP-22** (Apr. 1974), 111–117.

[19] Jesshope, C. R. The implementation of fast radix 2 transforms on array processors, *IEEE Trans. Comput.* **C-29** (Jan. 1980), 20–27.

[20] Lawrie, D. H. Access and alignment of data in an array processor. *IEEE Trans. Comput.* **C-24** (Dec. 1975), 1145–1155.

[21] Oppenheim, A. V., and Schafer, R. W. *Digital Signal Processing*. Prentice–Hall, Englewood Cliffs, N.J., 1975.

[22] Pease, M. C. An adaptation of the fast Fourier transform for parallel processing. *J. Assoc. Comput. Mach.* **15** (Apr. 1968), 252–264.

[23] Pease, M. C. The indirect binary n-cube microprocessor array. *IEEE Trans. Comput.* **C-26** (May 1977), 458–473.

[24] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing.* Prentice–Hall, Englewood Cliffs, N.J., 1975.

[25] Siegel, H. J. Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks. *IEEE Trans. Comput.* **C-26** (Feb. 1977), 153–161.

[26] Siegel, H. J. A model of SIMD machines and a comparison of various interconnection networks. *IEEE Trans. Comput.* **C-28** (Dec. 1979), 907–917.

[27] Siegel, H. J. The theory underlying the partitioning of permutation networks. *IEEE Trans. Comput.* **C-29** (Sept. 1980), 791–801.

[28] Siegel, H. J. *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies.* Lexington Books, Heath, Lexington, Mass. 1985.

[29] Siegel, H. J., and McMillen, R. J. Using the augmented data manipulator network in PASM. *Computer* **14** (Feb. 1981), 25–33.

[30] Siegel, H. J., and McMillen, R. J. The multistage cube: A versatile interconnection network. *Computer* **14** (Dec. 1981), 65–76.

[31] Siegel, H. J., and Smith, S. D. Study of multistage SIMD interconnection networks. *5th Ann. Symp. Computer Architecture,* Apr. 1978, pp. 223–229.

[32] Siegel, H. J., Siegel, L. J., Kemmerer, F. C., Mueller, P. T., Jr., Smalley, H. E., and Smith, S. D. PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition. *IEEE Trans. Comput.* **C-30** (Dec. 1981), 934–947.

[33] Siegel, L. J., Siegel, H. J., and Feather, A. E. Parallel processing approaches to image correlation. *IEEE Trans. Comput.* **C-31** (Mar. 1982), 208–218.

[34] Stone, H. S. Parallel processing with the perfect shuffle. *IEEE Trans. Comput.* **C-20** (Feb. 1971), 153–161.

[35] Veenkart, R. L. A serial minded FFT. *IEEE Trans. Audio Electroacoust.* **AU-20** (Aug. 1972), 180–185.

[36] Warpenburg, M. R., and Siegel, L. J. Image resampling in an SIMD environment. *IEEE Trans. Comput.* **C-31** (Oct. 1982), 934–942.