

# Parallel Implementation of Multidimensional Transforms without Interprocessor Communication

Francescomaria Marino and Earl E. Swartzlander, Jr., *Fellow, IEEE*

**Abstract**—This paper presents a modular algorithm which is suitable for computing a large class of multidimensional transforms in a general purpose parallel environment without interprocessor communication. Since it is based on matrix-vector multiplication, it does not impose restrictions on the size of the input data as many existing algorithms do. The method is fully general since it does not depend on the specific nature of the transform kernel and, therefore, it may be used for a wide variety of transforms. Moreover, since some one-dimensional Fast Fourier Transform algorithms map the input sequence onto two or more dimensions, the new method also may be employed to efficiently compute the 1D FFT in parallel. In addition, the proposed algorithm is exploited to derive a fully systolic VLSI architecture performing multidimensional transforms, which does not need the transposer required by classical architectures.

**Index Terms**—Parallel processing, multidimensional transforms, 2D DFT, 1D FFT, systolic VLSI architectures.

## 1 INTRODUCTION AND MOTIVATION

THE usefulness of the 2D Discrete Fourier Transform (DFT) is well-known in a large number of application areas. The multidimensional (3D or 4D) DFT also has been proposed as an effective tool, e.g., in computer vision and pattern recognition, to facilitate object recognition [1] and time dependent analysis [2], [3], in video telephony to compute motion from a sequence of images (multiframe detection) [4], [5], and in some nuclear magnetic resonance imaging algorithms [6]. The large amount of computation required by these algorithms makes a parallel implementation desirable.

Although other transforms (Hartley Transform [7], Discrete Cosine Transform [8], Discrete Sine Transform [9], Hadamard Transform [10], etc.) have been developed in order to avoid the operations on complex numbers that are required by the Fourier transform, the bulk of the literature about parallelism in transforms is devoted to the DFT.

In [11], the one- and two-dimensional Fast Fourier Transform (FFT) implementations are discussed over a  $k$ -dimensional array of  $S^k$  Processing Elements (PEs): In that model, an interconnection system allows each PE to communicate with its neighbors in any direction. The analysis shows that a minimum number of I/O operations occurs in the hypercube interconnection scheme.

Thus, though architectures with shared hierarchical memory to perform data transposition [12] or schemes requiring high connectivity [13] have been shown, it is

important to develop algorithms that reduce the need to communicate among the PEs. If communications between the PEs can be eliminated, two advantages are gained: No time is wasted on interprocessor I/O operations and no synchronization is required among the PEs. In fact, some research has focused on reducing or avoiding the interprocessor communications that are usually required to transpose the data between the 1D DFTs (see Section 2). These studies essentially exploit the possibility to compute the multidimensional DFT by a certain number of independent 1D DFTs or perform this computation by matrix-vector multiplications.

The first approach is to compute the 1D DFTs by using *fast* algorithms (many 1D FFT algorithms have been presented, e.g., in [14]). Most of the known algorithms impose some restrictions on  $N$  (the size, along each axis, of the input array), reducing the generality of this approach. For example, [15] presents an algorithm that computes an  $N^k$  point  $k$ -D DFT (where  $N$  is a prime number) by evaluating  $(N^k - 1)/(N - 1)$  independent one-dimensional DFTs.

Other results are shown in [16]: The most interesting are limited to the 2D DFT, and are related to the value of  $N = p^2$  (with  $p$  a prime number) and  $N = 2^n$ . In the first case,  $p^2 + p$  independent  $N$ -point 1D DFTs and  $p + 1$  independent  $p$ -point 1D DFTs, in the second case  $(3/2)2^n$  independent  $N$ -point 1D DFTs and one  $2^{n-1} \times 2^{n-1}$  two-dimensional DFT are required.

In [17], the authors present a parallel implementation of the algorithm described in [18] on an AT&T BT100 binary tree computer. This algorithm computes an  $N \times N$  two-dimensional DFT by  $L = O(N)$  independent  $N$ -point 1D DFTs whose input vectors are given by Discrete Radon Transforms over  $L$  data sets. These data sets are extracted from the input matrix according to criteria based on *linear congruences* [18].

- F. Marino is with the Dipartimento di Ingegneria Elettrotecnica ed Elettronica, Politecnico di Bari, 70125 Bari, Italy.  
E-mail: marino@poliba.it.
- E.E. Swartzlander Jr. is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712.  
E-mail: e.swartzlander@compmail.com.

Manuscript received 18 Aug. 1997; revised 9 Aug. 1999.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105516.

Though the algorithm does not impose any restriction on  $N$ , its first step is to *find the linear congruences which span a quadratic  $N \times N$  grid and possess a common solution  $(0, 0)$* . Unfortunately, this can be performed by means of simple formulas only for some specific values of  $N$  [18]. Moreover, it provides a solution only for the 2D case and it relies on the following relationship among the transform coefficients:

$$a_N(k+l) = a_N(k) \cdot a_N(l). \quad (1)$$

This relation is valid for the Fourier coefficients  $a_N(k) = \omega_N^k \equiv e^{\frac{-j2\pi}{N}k}$ , but not for other useful Transforms (e.g., DCT, DST, Hartley Transform, etc.).

Property (1) is also exploited in [19], where the multi-dimensional case is treated too. The Reduced Transform Algorithm is used to balance the communication and the computation in a parallel machine: Performance for an iPSC/860 is provided. Unfortunately, this algorithm requires that the sizes of the input array are equal to prime numbers.

Such restrictions are not required in approaches based on matrix-vector multiplication. This approach allows a straightforward VLSI implementation and has the same computational complexity as a 1D DFT if the problem size does not allow a *fast* implementation.

The matrix-vector approach was first explored by Winograd [20]. In [21], a detailed analysis is performed showing that matrix-vector multiplication approach is attractive when the input data are sequentially available. This is because the matrix-vector multiplication does not need the whole input data set to start its computation. This work is more suitable for fairly small transforms since the amount of numeric computation grows more rapidly than fast algorithms.

The paper is structured as follows: The 2D DFT and its standard parallel algorithm are briefly summarized in Section 2; the parallel approach based on the matrix-vector multiplication is defined in Section 3; Section 4 provides a performance evaluation; Section 5 describes a fully systolic VLSI architecture which exploits the proposed algorithm in order to avoid the transposer, which is needed by classical architectures, and Section 6 is devoted to conclusions.

## 2 CLASSICAL PARALLEL IMPLEMENTATION OF THE 2D DFT

Let  $\mathbf{X} = \{x(r, s); 0 \leq r, s < N\}$  be an  $N \times N$  matrix. Its 2D DFT is the  $N \times N$  matrix  $\mathbf{Y} = \{y(p, q); 0 \leq p, q < N\}$ , defined as [22]:

$$y(p, q) = \sum_{r=0}^{N-1} \sum_{s=0}^{N-1} x(r, s) \cdot \omega_N^{pr} \cdot \omega_N^{qs}, \quad (2)$$

where  $\omega_N$  is:

$$\omega_N \equiv e^{\frac{-j2\pi}{N}}. \quad (3)$$

The classical parallel approach to implementing the distributed computation of the  $N \times N$  2D DFT may be summarized as follows:

1. Download the rows of  $\mathbf{X}$  to the PEs (the maximum parallelism is reached by  $N$  PEs, each one devoted to a single row),
2. Perform 1D Fourier transforms in parallel for each row,
3. Exchange the results among the PEs in order to transpose the matrix,
4. Perform 1D Fourier transforms in parallel for the columns, and
5. Upload the final results.

(This algorithm may be easily extended to any dimension by iterating Steps 3, 4, and 5 for each dimension.).

Because of Step 3, if there is one PE for each row,  $N - 1$  data need to be sent and received by each PE, which limits the *speed* of this approach. Thus, for  $k = 2$ , the classical algorithm requires the input of data (i.e., one row for each PE),  $N$  independent 1D DFTs (i.e., one by each PE), data transposition,  $N$  independent 1D DFTs, and the output of the final result. In the next section, an approach based on matrix-vector multiplication which needs no interprocessor communications is described.

## 3 A MATRIX-VECTOR MULTIPLICATION APPROACH TO THE PARALLEL MULTIDIMENSIONAL DFT COMPUTATION

The following definitions are provided for the 2D case. The extension to the multidimensional case is postponed to the end of this section. The use of the method for rectangular matrices is straightforward.

Let  $\mathbf{W}_N = \{w_N(i, j); 0 \leq i, j < N\}$  be a symmetric  $N \times N$  matrix having:

$$w_N(i, j) \equiv \omega_N^{ij}. \quad (4)$$

By applying (4) to (2), we obtain:

$$y(p, q) = \sum_{s=0}^{N-1} \left( \sum_{r=0}^{N-1} w_N(p, r) \cdot x(r, s) \right) w_N(s, q) \quad (5.a)$$

or, alternatively:

$$y(p, q) = \sum_{r=0}^{N-1} w_N(p, r) \left( \sum_{s=0}^{N-1} x(r, s) \cdot w_N(s, q) \right). \quad (5.b)$$

Equation (5) may be expressed in matrix form:

$$\begin{bmatrix} y(p, 0) \\ \vdots \\ y(p, N-1) \end{bmatrix}^T = \begin{bmatrix} w_N(p, 0) \\ \vdots \\ w_N(p, N-1) \end{bmatrix}^T \times \begin{bmatrix} x(0, 0) & \cdots & x(0, N-1) \\ \vdots & \cdots & \vdots \\ x(N-1, 0) & \cdots & x(N-1, N-1) \end{bmatrix} \\ \times \begin{bmatrix} w_N(0, 0) & \cdots & w_N(0, N-1) \\ \vdots & \cdots & \vdots \\ w_N(N-1, 0) & \cdots & w_N(N-1, N-1) \end{bmatrix} \\ \text{\{for any } p = 0, 1, \dots, N-1\}} \quad (6.a)$$

$$\begin{aligned}
& \begin{bmatrix} y(0, q) \\ \dots \\ y(N-1, q) \end{bmatrix} = \\
& \begin{bmatrix} w_N(0, 0) & \dots & w_N(0, N-1) \\ \dots & \dots & \dots \\ w_N(N-1, 0) & \dots & w_N(N-1, N-1) \end{bmatrix} \times \\
& \left( \begin{bmatrix} (0, 0) & \dots & x(0, N-1) \\ \dots & \dots & \dots \\ x(N-1, 0) & \dots & x(N-1, N-1) \end{bmatrix} \times \begin{bmatrix} w_N(0, q) \\ \dots \\ w_N(N-1, q) \end{bmatrix} \right) \\
& \text{\{for any } q = 0, 1, \dots, N-1\}} \\
& \tag{6.b}
\end{aligned}$$

where:

- $\times$  denotes a matrix-vector multiplication and
- $[\cdot]^T$  denotes matrix transposition.

Now, if:

- $\mathbf{y}_p^R$  and  $\mathbf{w}_{N_p}^R$  are the  $p$ th row of  $\mathbf{Y}$  and  $\mathbf{W}_N$ , respectively, and
- $\mathbf{y}_q^C$  and  $\mathbf{w}_{N_q}^C$  are the  $q$ th column of  $\mathbf{Y}$  and  $\mathbf{W}_N$ , respectively

the following compact forms may be provided:

$$\mathbf{y}_p^R = \left( \mathbf{w}_{N_p}^R \times \mathbf{X} \right) \times \mathbf{W}_N \tag{6'.a}$$

$$\mathbf{y}_q^C = \mathbf{W}_N \times \left( \mathbf{X} \times \mathbf{w}_{N_q}^C \right). \tag{6'.b}$$

This scenario suggests a parallel approach to computing the 2D DFT: In fact, (6.a) may be computed by  $N$  independent computing phases (i.e., one for each PE), one for each value of  $p$  (alternatively, if storage by columns is preferred, (6.b) may be computed  $N$  times, one for each value of  $q$ ). In both cases, each independent computing phase needs  $O(N^2)$  operations.

The external matrix-vector multiplication in (6'.a) and (6'.b) is the mono-dimensional transform of the result from the inner multiplication, therefore, for particular values of  $N$ , it may be computed by a *fast* algorithm.

Even if a *fast* implementation is not available due to the nature of the coefficients for the particular transform, some optimizations may be considered. For example, in the case of DFT, because of the circular symmetry

$$w_N(p, r) \equiv \omega_N^{pr} = \omega_N^{(pr)^N}, \tag{7}$$

where  $(i)_N$  evaluates  $i$  modulo  $N$ . Matrix  $\mathbf{W}_N$  may be written as (a corresponding form exists for the columns because of the symmetry):

$$\mathbf{W}_N \equiv \begin{bmatrix} \left[ \mathbf{w}_{N_0}^R \right] \\ \left[ \mathbf{w}_{N_1}^R \right] \\ \dots \\ \left[ \mathbf{w}_{N_j}^R \right] \\ \dots \\ \left[ \mathbf{w}_{N_{\lceil (N-1)/2 \rceil}}^R \right] \\ \dots \\ \left[ \mathbf{w}_{N_{N-j}}^R \right] \\ \dots \\ \left[ \mathbf{w}_{N_{N-1}}^R \right] \end{bmatrix} \equiv \begin{bmatrix} [1 \quad 1 \quad \dots \quad 1] \\ \left[ \mathbf{w}_{N_1}^R \right] \\ \dots \\ \left[ \mathbf{w}_{N_j}^R \right] \\ \dots \\ \left[ \mathbf{w}_{N_{\lceil (N-1)/2 \rceil}}^R \right] \\ \dots \\ \left[ \mathbf{w}_{N_j}^R \right]^* \\ \dots \\ \left[ \mathbf{w}_{N_1}^R \right]^* \end{bmatrix}, \tag{8}$$

where:

- $[\cdot]$  denotes the smallest integer greater than or equal to the argument, and
- $[\cdot]^*$  denotes the complex conjugate.

Consequently, only  $4 \lceil \frac{N-1}{2} \rceil N$  real multiplications (instead of  $4N^2$ ) have to be effectively computed between the  $N$  complex points of the vector obtained by the first matrix-vector multiplication ( $\mathbf{X} \times \mathbf{w}_q^C$ ) and the  $N$  complex points of the first  $\lceil \frac{N-1}{2} \rceil$  rows, starting from the second one (this rough evaluation also includes the trivial multiplications by 0 and  $\pm 1$  that occur, e.g., for the first point of any row of  $\mathbf{W}$ ).

### 3.1 Multidimensional Case

Let  $\mathbf{X} = \{x(n_1, n_2, \dots, n_k); 0 \leq n_i < N\}$  be a  $k$ -dimensional array and  $\mathbf{Y} = \{y(m_1, m_2, \dots, m_k); 0 \leq m_i < N\}$  be its  $k$ -dimensional DFT defined as [22]:

$$\begin{aligned}
& y(m_1, m_2, \dots, m_k) = \\
& \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \dots \sum_{n_k=0}^{N-1} x(n_1, n_2, \dots, n_k) \cdot \omega_N^{m_1 n_1} \cdot \omega_N^{m_2 n_2} \dots \omega_N^{m_k n_k}. \tag{9}
\end{aligned}$$

Now, if  $m_k$  is the preferred direction along which to sort the output data, (9) may be rewritten as:

$$\begin{aligned}
& y(m_1, m_2, \dots, m_k) = \sum_{n_1=0}^{N-1} w_N(m_1 n_1) \\
& \left( \sum_{n_2=0}^{N-1} w_N(m_2 n_2) \left( \dots \left( \sum_{n_k=0}^{N-1} x_{n_1, n_2, \dots, n_k} \cdot w_N(m_k n_k) \right) \right) \right). \tag{10}
\end{aligned}$$

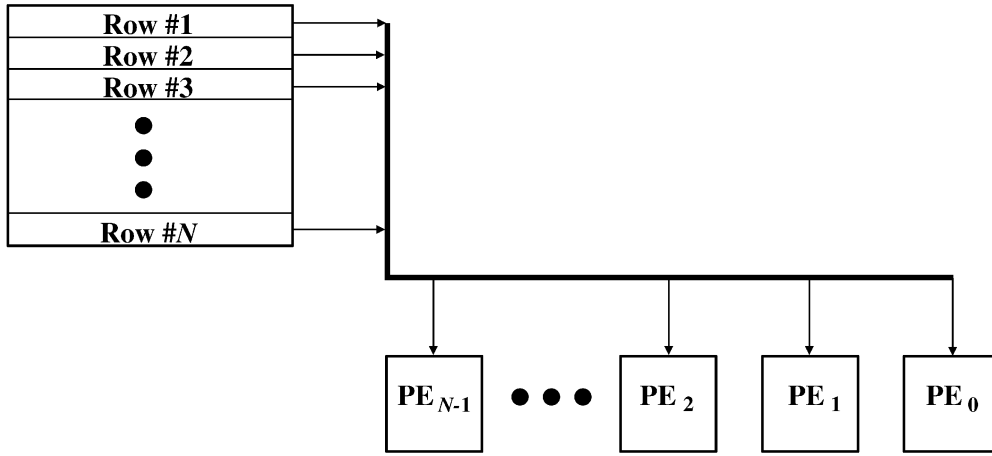
This equation may be evaluated by  $N$  independent and parallel computations, each one associated with a fixed value (i.e.,  $\overline{m_k}$ ) of the index  $m_k$ .

Each of these computations has an  $O(N^k)$  computational complexity and outputs  $N^{k-1}$  output values  $\{y(m_1, m_2, \dots, \overline{m_k}); 0 \leq m_1, m_2, \dots, m_{k-1} < N\}$ .

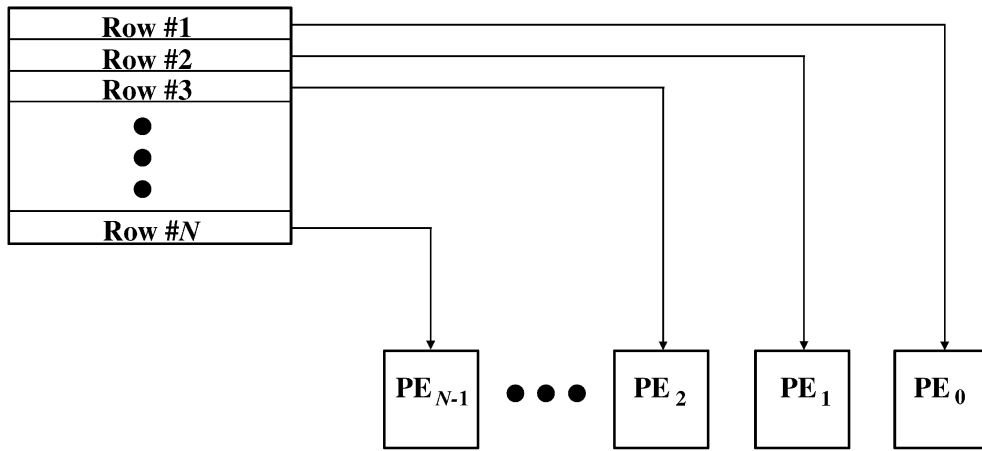
An important consideration has to do with reference to the memory requirement. From (9), it may seem that a memory size of  $O(N^k)$  is necessary for storing  $\mathbf{W}$  and  $\mathbf{X}$  in any PE, but this is not strictly required. In fact, by means of (7), only one row of  $\mathbf{W}$  is necessary:

$$w_N(p, r) \equiv \omega_N^{pr} = \omega_N^{(pr)^N} \equiv w_N(1, (pr)_N). \tag{11}$$

Moreover, any row of  $\mathbf{X}$  may be overwritten over the previous one because any point of  $\mathbf{X}$  is processed only once by the inner matrix-vector multiplication of (6).



(a)



(b)

Fig. 1. Input phase in the (a) new and (b) classic algorithm. In the classic algorithm,  $PE_i$  has to wait for the availability of Row  $(i + 1)$  before starting its task.

The possibility of processing the data by every PE in accordance with (6), where rows are multiplied by vectors  $w_{N_q}^C$ , avoids delaying the computing with respect the input phase. In other words, the processing may start as soon as the first point is received by the system: This is not possible by the classical algorithm, where the PE related to the transformation of the last row has to wait for it, as shown in Fig. 1.

In conclusion, the new algorithm requires inputting the data to transform into each PE,  $N$  independent matrix-vector multiplications (one per PE),  $N$  independent 1D transforms, and the output of the final result. The input phase and the first computing phase may be interleaved, i.e., any PE may start its processing as it receives its data.

### 3.2 1D FFT Algorithms and Other Multidimensional Transforms

FFT algorithms perform the 1D DFT efficiently. Many algorithms have been introduced [14], [23], [24], [25], in general, they exploit properties of  $\omega_N^m$  for certain values of  $N$ . Since many of them map the 1D input sequence onto two

or more dimensions [22], the proposed algorithm may be easily extended to these cases. In addition, since the proposed algorithm is completely independent of the specific nature of the Fourier coefficients, it may be used to perform a large class of  $k$ -D Transforms that may be expressed as:

$$y(m_1, m_2, \dots, m_k) = \prod_{i=1}^k c_i \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \dots \sum_{n_k=0}^{N-1} x(n_1, n_2, \dots, n_k) \cdot a_N(m_1, n_1) \cdot a_N(m_1, n_1) \cdot a_N(m_1, n_1) \quad (12)$$

where  $\prod_{i=1}^k c_i$  and  $a_N(m_i, n_i)$  are, respectively, the normalizing and the transforming coefficients. For instance, such class includes the  $k$ -D Discrete Hartley Transform [7], the  $k$ -D Discrete Cosine Transform [8], the  $k$ -D Discrete Sine Transform [26] and may be implemented in parallel with no interprocessor communications by a simple change in the matrix  $W_N$  of our algorithm.

The Discrete Wavelet Transform (DWT) [27], [28], [29] is a mathematical technique that decomposes a signal in the

TABLE 1  
Operations Required by the Classic Algorithm and the New Algorithm  
to Transform a Bidimensional  $N \times N$  Matrix by Means of  $N$  PEs

STEP	Classical Algorithm	New Algorithm
0	Receive input data	Receive input data
1	$N$ independent 1-D DFT's (one row transform per PE)	$N$ independent Matrix-vector multiplications (one per PE)
2	Corner Turning	<i>No data transposition is required</i>
3	$N$ independent 1-D DFT's (one column transform per PE)	$N$ independent 1-D DFT's (one per PE)
4	Send output data	Send output data

TABLE 2  
Time Required by the Classic Algorithm (CA) and the New Algorithm (NA)  
to Transform a Bidimensional  $N \times N$  Matrix by Means of  $N$  PEs

$N = \#PEs$	CA (fast)	NA (fast)	Speed-up (fast)	CA	NA	Speed-up
2	26 [ms]	9 [ms]	2.89	26 [ms]	9 [ms]	2.89
4	51 [ms]	17 [ms]	3.00	49 [ms]	17 [ms]	2.88
8	92 [ms]	37 [ms]	2.49	100 [ms]	40 [ms]	2.50
16	204 [ms]	88 [ms]	2.32	243 [ms]	107 [ms]	2.27
32	456 [ms]	232 [ms]	1.97	620 [ms]	300 [ms]	2.07
64	1137 [ms]	682 [ms]	1.67	1770 [ms]	950 [ms]	1.86
128	3188 [ms]	2268 [ms]	1.41	5730 [ms]	3340 [ms]	1.72

time domain by using dilated/contracted and translated versions of a single basis function, named the prototype wavelet. Recent research suggests that the DWT is preferable to other transforms, especially for image compression [30], [31], [32], [33], [34], [35]. A nontrivial extension of the proposed algorithm performing the multidimensional DWT is presented in [36].

## 4 PERFORMANCE EVALUATION

In order to evaluate the performance of the algorithm described in Section 3, we have compared it to the classic algorithm described in Section 2.

### 4.1 Preliminary Considerations

Table 1 shows the operations required by both approaches to transform a two-dimensional  $N \times N$  matrix, by means of  $N$  PEs.

For both strategies, Steps 3 and 4 are identical.

For suitable values of  $N$ , Step 1 in the classic algorithm (1D  $N$ -points DFT), may be computed by an FFT in  $O(N \log(N))$  operations, otherwise it has the same computational complexity as the  $N \times N$  matrix times  $N$ -points vector multiplication required in the new algorithm ( $O(N^2)$ ).

The new algorithm is more attractive because it does not require the data transposition (corner turning, Step 2) needed by the classic algorithm, when the number of PEs

used is less than or equal to  $N$ . In fact, the time required by this step is strongly dependent on both the architecture and the strategy chosen to perform it, but it mitigates the advantage of a *fast* computing of Step 1 in the classic algorithm.

Another advantage of the new algorithm results from the nature of Step 1. Processing can begin within each PE as soon as data becomes available. This is denoted by the dotted line between Steps 0 and 1: These steps, in fact, may conveniently be interleaved. On the contrary, the  $i$ th PE in the classic algorithm needs the  $i$ th row to transform it and start its computation (i.e., the last PE has to wait the last row before starting its task).

Comparison between the new algorithm and other approaches aiming to reduce the communications between PEs may be also performed. As we have previously indicated, the new algorithm imposes no restrictions on the value of  $N$ . Moreover, the new algorithm is general so that it may be used also for other transforms having real coefficients (i.e., in case the input data is real, not complex) because it is not based on the separability property of the Fourier Transform from (1).

### 4.2 Experimental Results

The algorithm introduced in Section 3 has been implemented and tested on an AT&T DSP-3 parallel processor.

Classic Algorithm and New Algorithm Performance

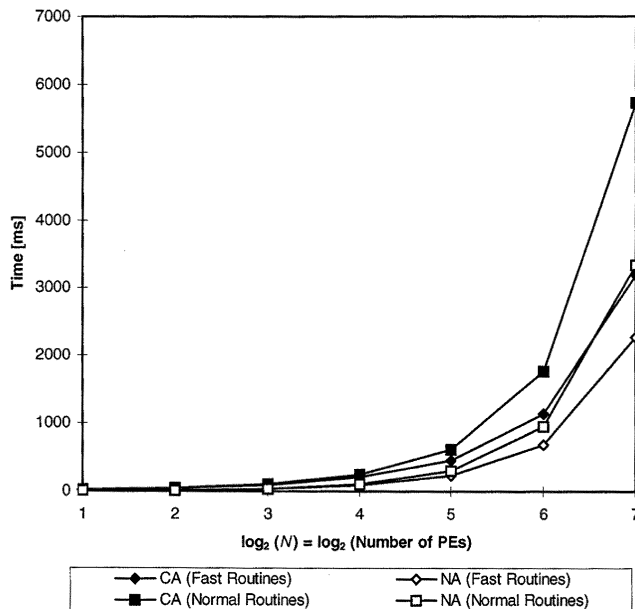


Fig. 2. Time required by the classic algorithm (CA) and the new algorithm (NA) to transform a two-dimensional  $N \times N$  matrix by means of  $N$  PEs. Both the fast and the normal computing of the Fourier Transforms are shown.

The DSP-3 parallel processor is a machine with 16 to 128 PEs. Each PE has 512K by 32-bit memory and a DSP32C processor operating at a frequency of 50 MHz, performing 25 MFLOPS, and having a multiplier and an adder that operate in parallel [37], [38].

Table 2 summarizes the experiments performed to compare the new algorithm with the classic algorithm. The data for  $N = 2, 4, 8, 16$  were obtained directly from C++ programs running on the DSP-3. The data for  $N > 16$  was derived by simulating larger machines with our hardware that has 16 PEs.

The first column of the table gives the size of the input data (for this 2D case,  $N$  is the number of rows and the number of columns) that coincides with the size of the PE array. The Fourier transforms required in Step 1 of the classic algorithm and in Step 3 of both the algorithms have been computed both by means of *fast* (second and third columns) and normal (fifth and sixth columns) routines. The fourth and the seventh columns indicate the speed-up achieved by the new algorithm over the classic algorithm. Figs. 2 and 3 graphically show the data of Table 1. The figures show that, for all cases, the new algorithm performs better than the classic one. This is more evident when the fast computing of the Fourier Transform cannot be performed. The speed-up decreases as  $N$  increases because the time consumed by Steps 0 and 4 assumes a growing weight as the number of processors grows.

In these experiments, moreover, the computing is started in both the algorithms at the same time, without exploiting the feature of processing the data as soon as available, as is possible in the new algorithm.

Speed-up of New Algorithm

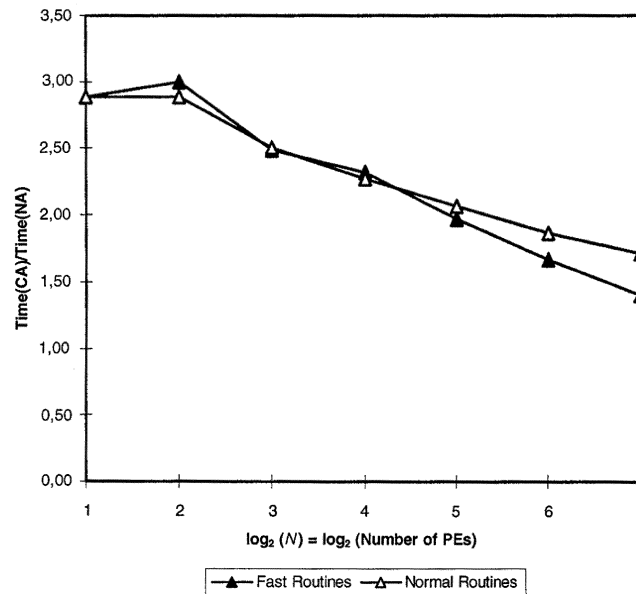


Fig. 3. Ratio of the time required by the classic algorithm (CA) and the new algorithm (NA) to transform a two-dimensional  $N \times N$  matrix by means of  $N$  PEs. The time include the I/O between the host and the parallel machine (Steps 0 and 4). Both the *fast* and the normal computing of the Fourier Transforms are shown.

## 5 VLSI IMPLEMENTATION

Advances in VLSI technology allow implementation of parallel processing on a single chip. Therefore, the approach proposed in Section 3 (new algorithm) can be exploited to design a fully systolic VLSI architecture avoiding the transposer that is required by classical architectures [39], [40]. The transposer requires a large area for global interconnection and time for loading and unloading.

Most of the architectures available in the literature restrict the value of  $N$  to a prime number or to a power of 2 [41], [42]: These limitations are not required by the new algorithm. To implement the new algorithm for the two-dimensional case, a twice iterated matrix-vector multiplication is required: In the literature, many efficient architectures are available to perform this task.

### 5.1 A Fully Systolic Architecture

In order to better exploit the VLSI technology, we have focused on systolic array implementations. These structures have these essential features:

- *synchrony*: data rhythmically computed and flowing;
- *modularity* and *regularity*: modular PEs, simply and regularly connected;
- *pipelinability*: the speed-up may be linearly increased by pipelining;
- *boundary input and output*: the I/O operations with the external world are performed only by the boundary PEs.

TABLE 3  
Processing Steps Performed by the Fully Systolic Two-Dimensional Array Computing the New Algorithm

Phase	Clock	Step	$\hat{P}_0$	$\hat{P}_1$	$\hat{P}_2$	$\hat{P}_3$
<b>1</b>	1	1.1	$R_0 \leftarrow x(0,0)w_p(0)$			
	2	1.2	$R_0 \leftarrow R_0 + x(0,1)w_p(1)$	$R_1 \leftarrow x(1,0)w_p(0)$		
	3	1.3	$R_0 \leftarrow R_0 + x(0,2)w_p(2)$	$R_1 \leftarrow R_1 + x(1,1)w_p(1)$	$R_2 \leftarrow x(2,0)w_p(0)$	
	4 (=N)	1.4	$R_0 \leftarrow R_0 + x(0,3)w_p(3)$	$R_1 \leftarrow R_1 + x(1,2)w_p(2)$	$R_2 \leftarrow R_2 + x(2,1)w_p(1)$	$R_3 \leftarrow x(3,0)w_p(0)$
<b>2</b>	5	2.1	$Out_0 \leftarrow R_0w(0,0)$	$R_1 \leftarrow R_1 + x(1,3)w_p(3)$	$R_2 \leftarrow R_2 + x(2,2)w_p(2)$	$R_3 \leftarrow R_3 + x(3,1)w_p(1)$
	6	2.2	$Out_0 \leftarrow R_0w(1,0)$	$Out_1 \leftarrow In_1 + R_1w(0,1)$	$R_2 \leftarrow R_2 + x(2,3)w_p(3)$	$R_3 \leftarrow R_3 + x(3,2)w_p(2)$
	7	2.3	$Out_0 \leftarrow R_0w(2,0)$	$Out_1 \leftarrow In_1 + R_1w(1,1)$	$Out_2 \leftarrow In_2 + R_2w(0,2)$	$R_3 \leftarrow R_3 + x(3,3)w_p(3)$
	8 (=2N)	2.4	$Out_0 \leftarrow R_0w(3,0)$	$Out_1 \leftarrow In_1 + R_1w(2,1)$	$Out_2 \leftarrow In_2 + R_2w(1,2)$	$Out_3 \leftarrow In_3 + R_3w(0,3)$
	9		$Out_1 \leftarrow In_1 + R_1w(3,1)$	$Out_2 \leftarrow In_2 + R_2w(2,2)$	$Out_3 \leftarrow In_3 + R_3w(1,3)$	
	10			$Out_2 \leftarrow In_2 + R_2w(3,2)$	$Out_3 \leftarrow In_3 + R_3w(2,3)$	
	11				$Out_3 \leftarrow In_3 + R_3w(3,3)$	

There are two well-known semisystolic arrays that perform matrix-vector multiplication [43]. Neither of them is fully systolic: The first one generates a single point of the result in each processor  $\hat{P}_i$  (i.e., the output is not available from the last PE), while the second array needs the input vector to be preloaded with one point for each processor  $\hat{P}_i$ . These features may be exploited by merging the functionality of  $\hat{P}_i$  and of  $\hat{P}_i$  in one processor  $\hat{P}_i$  in order to obtain a modular fully systolic array computing a two-dimensional transform by the new algorithm.

Let  $\hat{P}_i$  be a processor performing the task of  $\hat{P}_i$  during a first processing phase and the task of  $\hat{P}_i$  during a second phase. Data computed by the first phase are locally stored and used as input data for the second phase. Both computing phases need  $N$  steps.

Table 3 describes how the various computing steps are performed by an array constituted by four processors  $\hat{P}_i$ . A two-dimensional transform may be obtained by an  $N \times N$  array, as shown in Fig. 4. A block-diagram of  $\hat{P}_{i,j}$  is shown in Fig. 5.

This architecture:

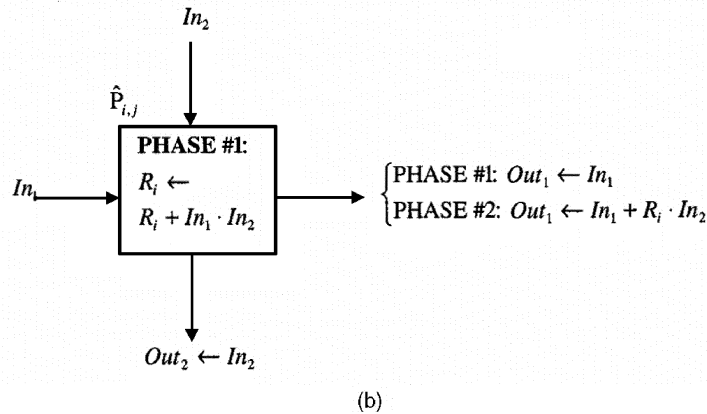
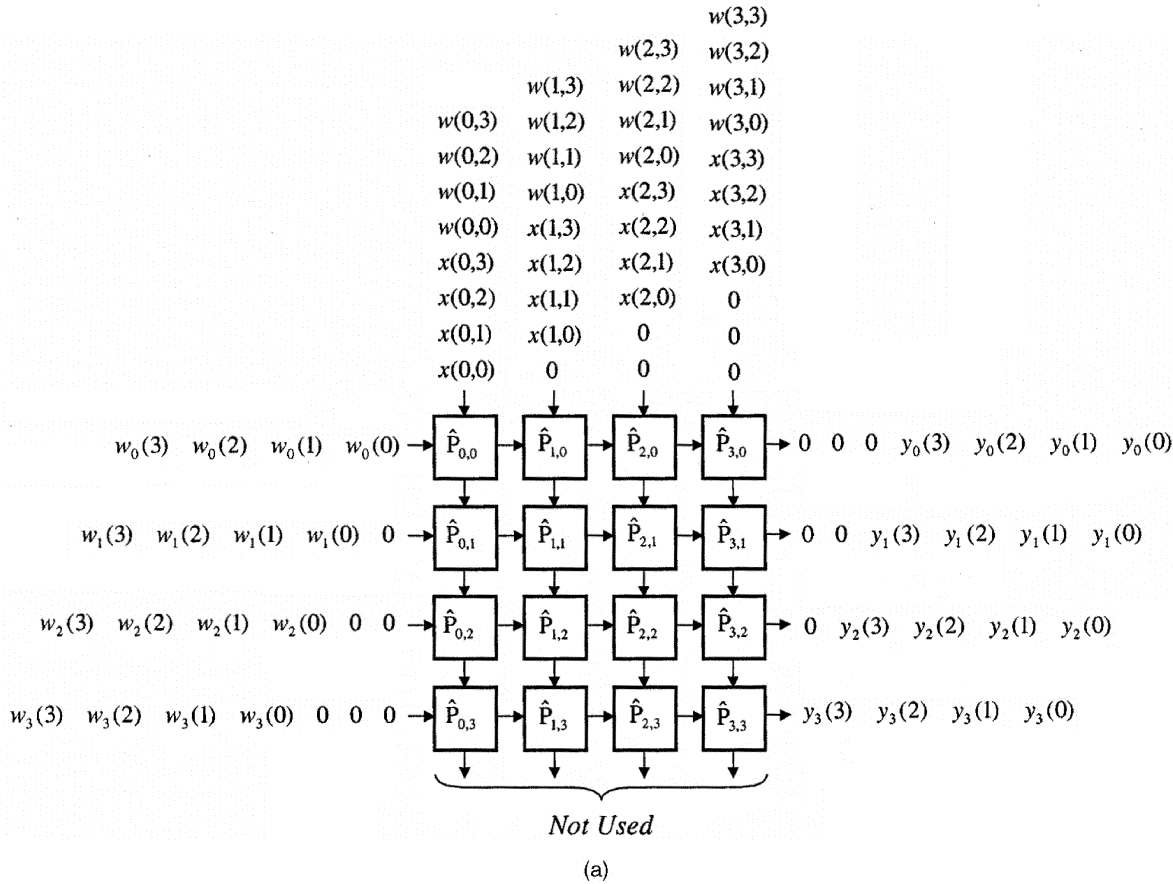


Fig. 4. (a) A VLSI fully systolic two-dimensional array computing the new algorithm. Each row of the array works as a PE of the distributed implementation. The vertical connections allow modularity, but are not used in the distributed implementation because of the broadcasting shown in Fig. 1. (b) Logic diagram of the  $\hat{P}_{i,j}$ .

- has a minimum number of boundary cells;
- uses an input data stream and generates an output data stream;
- does not require any data preloading.

These features ensure maximum throughput, minimum latency and a minimum number of I/O pins [44].

Recent advances of the 3D VLSI technology make feasible shorter and more systematic wire routing, as well as higher circuit density [45]. Also, it is important that there are no high fan-out “global” signals other than the clock. Therefore, the new algorithm for the 3D case can be directly realized by a 3D VLSI implementation [46], [47].

### 5.2 Complexity Estimation

We estimate the complexity of the 2D array shown in Fig. 4 by using the *Area · Time<sup>2</sup>* ( $A \cdot T^2$ ) complexity estimation method [45], [48].

The area complexity of each of the  $N^2$  multiply-add cells is  $O(\log(N))$ ; the area occupied by wires is  $O(N^2)$  since a vertical chord or horizontal chord crosses  $N$  wires: Therefore, the entire area complexity is  $O(N^2 \log(N))$ .

The array computes the complete transform (i.e.,  $N^2$  values) in  $2N$  clock cycles, each one of  $O(\log(N))$  units of time; therefore, the entire time complexity is  $O(N \log(N))$ .

Thus,  $A \cdot T^2 = O(N^4 \log^3(N))$ .

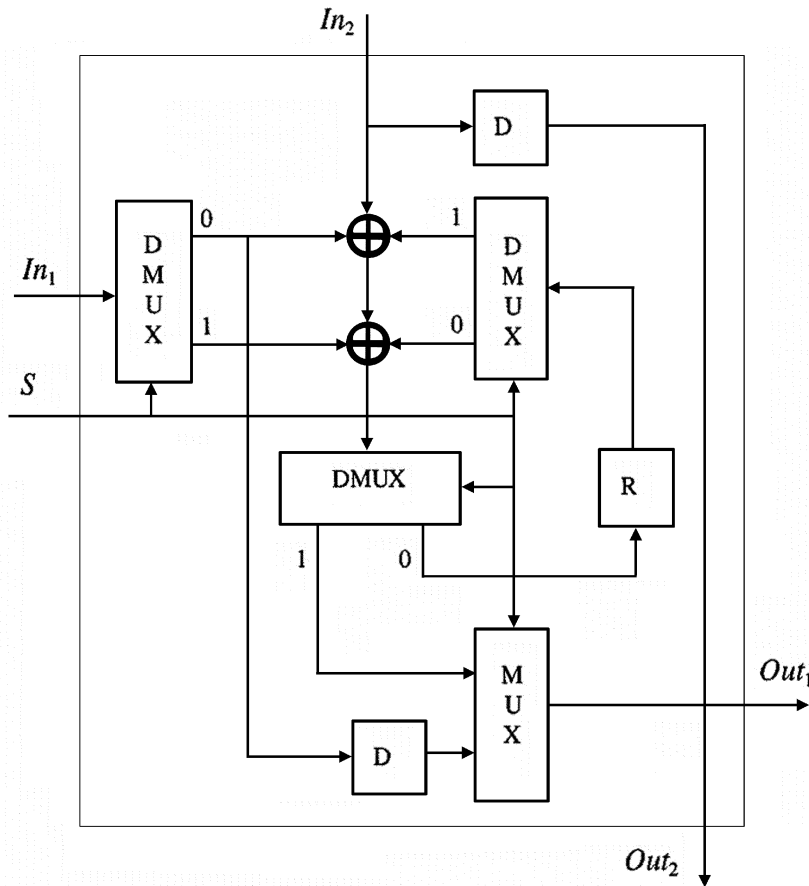


Fig. 5. Block diagram of the processor  $P_{i,j}$ . The control signal  $S$  is 0 during Phase 1 and 1 during Phase 2.

For the 2D FFT, the  $A \cdot T^2$  complexity of the lower bound is  $O(N^4 \log^2(N))$  [49], [50]: Therefore, the architecture implementing new algorithm has an  $A \cdot T^2$  complexity that is only  $O(\log(N))$  higher than the theoretical optimum one.

## 6 Conclusions

A parallel algorithm based on matrix-vector multiplication has been introduced, which is suitable for computing a large class of  $k$ -D transforms on general purpose parallel machines. Our attention is focused on the Fourier Transform, but these results are applicable to other transforms, since the algorithm has no dependency on the nature of the Fourier coefficients.

The main feature of the new algorithm is the absence of interprocessor communications: Some tests on the AT&T DSP-3 parallel machine have shown speed-up of 1.4 to 3.0 with respect to the classical 2D FFT approach.

The proposed parallel approach can be efficiently adapted to a set of 1D FFT algorithms by mapping the 1D input sequence onto a multidimensional array.

For its structure, the new algorithm does not impose any restriction on the size of the array, i.e., it may be performed with any value of  $N$ , and this is convenient: In fact, zero padding approaches that are used to increase the input size to a suitable one can enlarge the data size tremendously if performed along multiple dimensions.

The new algorithm does not need the acquisition of the whole set of input data (required by the classic approach) for starting the computation.

A modular and fully systolic VLSI implementation has been derived also. Since it exploits the proposed algorithm in order to avoid the transposer, it has an  $Area \cdot Time^2$  complexity that is within a factor of  $\log(N)$  of the lower bound for a 2D FFT. This modest increase is largely compensated by the generality of our architecture, which is also able to compute other transforms. Moreover, by needing a minimum number of boundary cells, using data streams as I/O and not requiring preloading of data into cells, it ensures a maximum throughput rate and a minimum number of I/O pins and latency.

## REFERENCES

- [1] L. Jacobson and H. Wechsler, "A Theory for Invariant Object Recognition in Front of Parallel Plane," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 6, pp. 253-331, 1984.
- [2] H. Gafni and Y.Y. Zeevi, "A Model for Separation of Spatial and Temporal Information in the Visual System," *Biological Cybernetics*, vol. 28, pp. 73-82, 1977.
- [3] H. Gafni and Y.Y. Zeevi, "A Model for Processing of Movement in the Visual System," *Biological Cybernetics*, vol. 32, pp. 165-173, 1979.
- [4] A. Kojima, N. Sakurai, and J. Kishigami, "Motion Detection Using 3D-FFT Spectrum," *ICASSP Proc.*, pp. 213-216, 1993.
- [5] B. Porat and B. Friedlander, "A Frequency Domain Algorithm for Multiframe Detection and Estimation of Dim Targets," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, pp. 398-401, 1990.

- [6] W.S. Hinshaw et al., "An Introduction to NMR Imaging: From the Block Equation to the Imaging Equation," *Proc. IEEE*, vol. 71, pp. 338-350, 1983.
- [7] R.N. Bracewell, "Discrete Hartley Transform," *J. Optical Soc. Am.*, vol. 73, pp. 1,832-1,835, 1983.
- [8] R. Storn, "A Novel Radix-2 Pipeline Architecture for the Computation of the DFT," *ISCAS '88 Proc.*, pp. 2,685-2,688, 1988.
- [9] A.K. Jain, "A Fast Karhunen-Loeve Transform for a Class of Stochastic Process," *IEEE Trans. Comm.*, vol. 24, pp. 1,023-1,029, 1984.
- [10] H.C. Andrews, *Computer Techniques in Image Processing*. New York: Academic Press, 1970.
- [11] C.R. Jesshope, "The Implementation of Fast Radix 2 Transforms on Array Processors," *IEEE Trans. Computers*, vol. 29, pp. 20-27, 1980.
- [12] A. Norton and A. Silberger, "Parallelization and Performance Analysis of the Cooley-Tukey FFT Algorithm on Shared-Memory," *IEEE Trans. Computers*, vol. 36, pp. 581-591, 1987.
- [13] L.H. Jamieson, P.T. Mueller, and H.J. Siegel, "FFT Algorithms for SIMD Processing," *J. Parallel and Distributed Computing*, vol. 3, 1986.
- [14] E.O. Brigham, *The Fast Fourier Transform*. Englewood Cliffs, N.J.: Prentice Hall, 1974.
- [15] L. Auslander, E. Feig, and S. Winograd, "New Algorithms for the Multidimensional Discrete Fourier Transform," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 31, pp. 388-403, 1983.
- [16] R. Blahut, *Fast Algorithms for Digital Signal Processing*. Reading Mass.: Addison-Wesley, 1985.
- [17] I. Gertner and M. Rofheart, "A Parallel Algorithm for 2D DFT Computation with No Interprocessor Communication," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, pp. 377-382, 1990.
- [18] I. Gertner, "A New Efficient Algorithm to Compute the Two-Dimensional Discrete Fourier Transform," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 36, pp. 1,036-1,050, 1988.
- [19] G.I. Kechriotis, M. An, M. Bletsas, R. Tolimieri, and E.S. Manolakos, "A New Approach for Computing Multidimensional DFT'S on Parallel Machines and its Implementation on the iPSC/860 Hypercube," *IEEE Trans. Signal Processing*, vol. 43, pp. 272-285, 1995.
- [20] S. Winograd, "On Computing the Discrete Fourier Transform," *Math. Comput.*, vol. 32, pp. 175-199, 1978.
- [21] V. Milutinovic, A.B. Fortes, and L.H. Jamieson, "A Multiprocessor Architecture for Real-Time Computation of a Class of DFT Algorithms," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 34, pp. 1,301-1,309, 1986.
- [22] A.V. Oppenheim and R.W. Schaffer, *Discrete-Time Signal Processing*. Englewood Cliffs, N.J.: Prentice Hall, 1989.
- [23] J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comput.*, vol. 19, pp. 297-307, 1965.
- [24] M.C. Pease, "An Adaptation of the Fast Fourier Transform for Parallel Processing," *J. ACM*, vol. 15, pp. 252-264, 1968.
- [25] H.S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Computers*, vol. 20, pp. 153-161, 1971.
- [26] Z. Wang, "Fast Algorithm for the Discrete  $W$  Transform and for the Discrete Fourier Transform," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 32, pp. 803-816, 1984.
- [27] S.G. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, pp. 674-693, 1989.
- [28] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding*. Englewood Cliffs, N.J.: Prentice Hall, 1995.
- [29] I. Daubechies, *Ten Lectures on Wavelets*. Philadelphia: SIAM, 1992.
- [30] F. Marino, T. Acharya, and L.J. Karam, "A DWT-Based Perceptually Lossless Compression Scheme and VLSI Architecture for R-G-B Digital Images," *J. Integrated Computer-Aided Engineering*, special issue on Industrial Applications of the Wavelet Transforms, to be published, 1999.
- [31] J.D. Villasenor, B. Belzer, and J. Liao, "Wavelet Filter Evaluation for Image Compression," *IEEE Trans. Image Processing*, vol. 4, pp. 1,053-1,060, Aug. 1995.
- [32] A.S. Lewis and G. Knowles, "Image Compression Using the 2D Wavelet Transform," *IEEE Trans. Image Processing*, vol. 1, pp. 244-250, Apr. 1992.
- [33] A. Averbuch, D. Lazar, and M. Israeli, "Image Compression Using Wavelet Transform and Multiresolution Decomposition," *IEEE Trans. Image Processing*, vol. 5, pp. 4-15, Jan. 1996.
- [34] J.M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Trans. Signal Processing*, vol. 41, pp. 3,445-3,462, Dec. 1993.
- [35] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image Coding Using Wavelet Transform," *IEEE Trans. Image Processing*, vol. 1, pp. 205-220, 1992.
- [36] F. Marino, V. Piuri, and E.E. Swartzlander Jr., "A Parallel Implementation of the 2D Discrete Wavelet Transform without Interprocessor Communications," submitted to *IEEE Trans. Signal Processing*.
- [37] AT&T, *DSP-3 General Information Manual*.
- [38] AT&T, *WE DSP32C Digital Signal Processor: Information Manual*.
- [39] I. Gertner and M. Shamash, "VLSI Architectures for Multidimensional Fourier Transform Processing," *IEEE Trans. Computers*, vol. 36, pp. 1,265-1,274, 1987.
- [40] M. Sheu, J. Lee, J. Wang, A. Suen, and L. Liu, "A High Throughput Rate Architecture for 8\*8 2-d DCT," *ISCAS '93 Proc.*, pp. 1,587-1,590, 1993.
- [41] J. Guo, C. Liu, and C. Jen, "The Efficient Memory-Based VLSI Array Designs for DFT and DCT," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, pp. 723-733, 1992.
- [42] N.I. Cho and S.U. Lee, "DCT Algorithms for VLSI Parallel Implementation," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 38, pp. 121-127, 1990.
- [43] S.Y. Kung, *VLSI Array Processors*. Englewood Cliff, N.J.: Prentice Hall, 1988.
- [44] N.R. Murthy and M.N.S. Swamy, "On the Real-Time Computation of DFT and DCT through Systolic Architectures," *IEEE Trans. Signal Processing*, vol. 42, pp. 988-991, 1994.
- [45] T.D. Roziner and M.G. Karpovsky, "Multidimensional Fourier Transforms by Systolic Architectures," *J. VLSI Signal Processing*, vol. 4, pp. 343-354, 1992.
- [46] H. Lim, "Multidimensional Systolic Arrays for Computing Discrete Fourier Transforms and Discrete Cosine Transforms," *Application Specific Processors*, E.E. Swartzlander Jr., ed., pp. 161-195, Norwell, Mass.: Kluwer Academic, 1997.
- [47] H. Lim and E.E. Swartzlander Jr., "Multidimensional Systolic Arrays for Multidimensional DFTs," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing*, pp. 3,277-3,280, 1996.
- [48] C.D. Thompson, "Fourier Transforms in VLSI," *IEEE Trans. Computers*, vol. 32, pp. 1,047-1,057, 1983.
- [49] C.N. Zang and D.Y.Y. Yun, "Multidimensional Systolic Network for DFT," *Proc. 11th Int'l Symp. Computer Architecture*, pp. 215-222, 1984.
- [50] D. Moldovan, "Towards a Computerized Optimal Design of VLSI Systolic Arrays," *Design Methodologies*, pp. 215-234. Amsterdam: North Holland, 1986.



**Francescomaria Marino** received his Laurea degree cum laude in electronic engineering and his PhD degree in electronic engineering, respectively, in 1991 and 1996 from the Polytechnic of Bari (Italy). He received an award from Firestone S.p.A. as the Best Graduate of the Year at the Universities of Puglia and an award from Telecom for his thesis work.

Since 1999, he has been an assistant professor in the Department of Electrical and Electronic Engineering of the Polytechnic of Bari. Previously, he worked in the Telecommunications Research Center of Arizona State University (1998), in the Electrical and Computer Engineering Department of the University of Texas at Austin (1997), and in the IESI-CNR, the Institute of Signal and Image Processing of the Italian National Council of Researches (1996). Dr. Marino coauthored two patents by Intel and one patent by CNR. His main interests are parallel algorithms and architectures for digital image and signal processing.



**Earl E. Swartzlander Jr.** (S'64, M'72, SM'79 F'88) received the BS degree in electrical engineering from Purdue University in 1967, the MS degree in electrical engineering from the University of Colorado in 1969, and the PhD degree from the University of Southern California in 1972.

In 1990, he became a professor of electrical and computer engineering at the University of Texas at Austin, where he holds the Schlumberger Centennial Chair in Engineering. His research interests are in application specific processing, the interaction between computer architecture and VLSI technology, and the history of calculators. He has made significant research contributions in computer arithmetic, VLSI development, and digital signal processor implementation.

Prior to joining the University of Texas at Austin he was with TRW Defense and Space Systems in Redondo Beach, California, from 1975 to 1990, where he held positions ranging from staff engineer to laboratory manager and, most recently, director of independent research and development for the TRW Defense Systems Group. Currently, he is the hardware area editor for *ACM Computing Reviews*, a subject area editor of the *Journal of Systems Architecture*, and editor of the Calculators column of the *IEEE Annals of the History of Computing*. He is a fellow of the IEEE and an inaugural member of the IEEE Computer Society Golden Core. He has been honored as an Outstanding Electrical Engineer and a Distinguished Engineering Alumnus of Purdue University, and as a Distinguished Engineering Alumnus of the University of Colorado.