[7] T. Yang and K. Yao, "Numerical error control in sliding window systems under finite precision arithmetics," submitted for publication.
[8] T. C. Yang, "Finite precision error control and array implementation of signal processing algorithms," Ph.D. dissertation, Univ. Calif. Los Angeles, 1997.
[9] T. J. Shepherd and J. G. McWhirter, "Modified givens rotations for inverse updating in QR decomposition," in *Proc. SPIE—Adaptive Signal Process.*, 1993, vol. 2027, pp. 376–387.
[10] T. C. Yang and K. Yao, "Dual-state systolic architecture for recursive least-squares updating and downdating," in *Proc. 7th IEEE Workshop VLSI Signal Process.*, Oct. 1994, pp. 316–325.
[11] H. Park and L. Eldén, "Downdating the rank-revealing URV decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 16, no. 1, pp. 138–155, Jan. 1995.

A Parallel Implementation of the 2-D Discrete Wavelet Transform without Interprocessor Communications

Francescomaria Marino, Vincenzo Piuri, and Earl E. Swartzlander, Jr.

***Abstract*—The discrete wavelet transform is currently attracting much interest among researchers and practitioners as a powerful tool for a wide variety of digital signal and imaging processing applications. This correspondence presents an efficient approach to compute the two-dimensional (2-D) discrete wavelet transform in standard form on parallel general-purpose computers. This approach does not require transposition of intermediate results and avoids interprocessor communication. Since it is based on matrix-vector multiplication, our technique does not introduce any restriction on the size of the input data or on the transform parameters. Complete use of the available processor parallelism, modularity, and scalability are achieved. Theoretical and experimental evaluations and comparisons are given with respect to traditional parallelization.**

***Index Terms*—Discrete wavelet transform, interprocessor communications, matrix-vector multiplication, parallel processing.**

I. Introduction

The discrete wavelet transform (DWT) [1]–[6] has been developed recently as a feature extraction tool for signal and image processing and has been shown to be efficient in comparison to traditional signal processing techniques in several industrial and commercial applications.

The massive computation required by the DWT can be met only with suitable computing resources. If the application is well defined and real-time operation is important, dedicated VLSI ASIC solutions should be considered (see, e.g., [7]–[18]). In particular, two efficient SIMD architectures are proposed in [10] to implement the 1-D and the 2-D DWT's, respectively. A pipeline-based realization of the 2-D DWT is described in [18]. Other efficient architectures are presented in [4], [15], and [17]. Whenever either the application or the desired DWT processing is subject to change, VLSI implementation is not suitable since it cannot easily accommodate varying specifications. In addition, if only a few processors have to be constructed, the cost of a dedicated VLSI realization is often too high. In these cases, configurable VLSI systems (e.g., FPGA- and FPL-based structures) can be adopted.

If high-speed operation is not mandatory, a feasible solution is the use of traditional general-purpose parallel computers. If high-speed operation is required, it is necessary to consider a highly parallel system with many processing elements to achieve the required speed without losing the flexibility of a general-purpose implementation. During the application, design, and experimentation phases, flexibility and modifiability of the implementation are attractive since they allow for tuning the solution, possibly before starting an ASIC development for volume production. In some applications (e.g., experimental computer graphics, medical imaging, multimedia production), even if no ASIC device is envisioned, the freedom of modifying the DWT processing parameters is important to achieve high-quality results. The use of general-purpose systems must be pursued also when the application is not required to have a flexible and adaptable implementation, but the production volume is so low that development and fabrication of a dedicated system is too expensive. In the literature, only few researchers have dealt with the DWT implementation on general-purpose machines. Parallelization of the 2-D DWT is proposed in [19] by using the *snake sweeping algorithm* [20]. Both DWT data dependence and localization analysis have been studied in [4] to design a distributed parallel memory/control architecture.

Maximum computational efficiency is an important issue in the use of multiprocessors for DWT processing. Unfortunately, the research mentioned above does not address how to avoid the interprocessor communications required to transpose the intermediate results in the 2-D DWT. Solutions to this problem have been presented in literature only for classical transforms, e.g., the DFT. An algorithm is presented in [21] to compute a N^k-point k-D DFT (where N is a prime number) by evaluating $(N^k - 1)/(N - 1)$ independent one-dimensional (1-D) DFT's. In [22], the case of the 2-D DFT for $N = p^2$ (with p a prime number) and $N = 2^n$ is considered. In [23], the parallel implementation of the algorithm described in [24] is discussed for the AT&T BT100 binary-tree computer. This algorithm computes the $N \times N$ 2-D DFT by means of $L = O(N)$ independent N-point 1-D DFT's and the discrete radon transform applied to the input matrix created by using *linear congruences*-based criteria [24]; to solve these congruences, simple formulas are provided only for some specific values of N [24]. In [25], the *reduced transform algorithm* is used to balance communication and computation in a parallel machine, but the sizes of the input array must be prime numbers. These algorithms impose restrictions on the size of the input array, thus making them nongeneral.

In [26], the matrix-vector multiplication approach has been shown highly suitable and effective for DFT when the input data are sequentially available since the matrix-vector multiplication does not need the whole input data set to begin its operation. This approach also provides also high modularity and scalability to satisfy a wide range of applications. We have explored the use of the matrix-vector multiplication approach to design a novel algorithm without limitations on the size of the input array.

This correspondence presents an approach to compute the 2-D DWT by using matrix-vector multiplications. Even though the matrix-vector multiplication approach is a well-known technique, our idea is to use this technique to avoid data transposition. Since we are

Manuscript received September 29, 1997; revised April 29, 1999. The associate editor coordinating the review of this paper and approving it for publication was Dr. Elias S. Manolakos.

F. Marino is with the Dipartimento di Ingegneria Elettrotecnica ed Elettronica, Politecnico di Bari, Bari, Italy.

V. Piuri is with the Department of Electronics and Information, Politecnico di Milano, Milano, Italy.

E. E. Swartzlander, Jr. is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA.

Publisher Item Identifier S 1053-587X(99)08298-7.

interested in maximizing the parallelism of the computation itself, we have selected the intrinsically parallel filter bank algorithm [1] to perform the DWT in standard form. Mallat's algorithm [3] is usually preferred in hardware implementations since the same circuit can be used repeatedly with the same coefficients to generate every transformation output, even though it has a higher latency. However, these two algorithms are equivalent from the point of view of the DWT results sinceMallat's algorithm can be viewed as the unrolled version of the filter bank. A suitable choice of the filter bank coefficients achieves a result quality similar to that provided by Mallat's algorithm. Coefficients of one of these two approaches can be easily transformed one into those of the other. In our software implementation, the use either of the same or different coefficients to generate each DWT output has no effect on the computational complexity. Therefore, we take advantage of the intrinsic parallelism of the filter bank to reduce the latency. Consequently, the application domain of the proposed approach covers the areas typically tackled by using Mallat's algorithm.

This corespondence is structured as follows. The parallelization approach based on matrix-vector multiplication is defined in Section II. Section III provides a theoretical analysis of the computational complexity and the performance evaluation for an experimental implementation on the AT&T DSP3 parallel computer.

II. The Matrix Approach to the Parallel Implementation of the 2-D DWT

The wavelet transform [1]–[6] is a mathematical technique that decomposes a signal by using dilated/contracted and translated versions of a single finite-duration basis function. In the literature, different 1-D DWT's have been proposed according to the nature of the signal, the time, and the scaling parameters. Any 1-D DWT can be viewed as a parallel filter bank consisting of M filters $G_m (m = 1, \cdots, M)$, where

- m filter level;
- M transform order;
- H suitable lowpass filter.

All filters operate in parallel on the same input sequence $\boldsymbol{x}$ of sampled input data $x(s)$. Each filter G_m is characterized by L_m coefficients and generates the sequence $\boldsymbol{y}_m$ having $N/2^m$ components $y_m(s)$

$$y_m(s) = \sum_{l=0} g_m(l) \cdot x(2^m(s-l)). \tag{1}$$

The lowpass filter $H = \lfloor h(0), h(1), \cdots, h(l), \cdots, h(2^M - 1) \rfloor$ generates the residual filtered sequence $\boldsymbol{y}_H$ having $N/2^M$ components $y_H(s)$

$$y_H(s) = \sum_{k=0}^{2^M-1} h(k) \cdot x(2^M s - k). \tag{2}$$

The 1-D DWT of the sequence x is obtained by collecting the $M+1$ sequences $\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_M$, and $\boldsymbol{y}_H$. It is worth noting that the $N/2^M$ integer is an intrinsic characteristic of the DWT M-level decomposition.

Since the 2-D DWT is separable [1]–[3], [15], it can be performed by two cascaded 1-D DWT's. One filter bank performs the first 1-D DWT on the rows of the 2-D square input matrix $\boldsymbol{X}$ (row filtering). The row filtered outputs are collected into an intermediate matrix $\boldsymbol{Z}$. Then, the second filter bank performs 1-D DWT's on the columns of $\boldsymbol{Z}$ (column filtering). The final result is a matrix $\boldsymbol{Y}$. Computation of the second DWT requires transposition of the intermediate matrix $\boldsymbol{Z}$.

The *first filter bank* is defined as the row 1-D DWT on the $N \times N$ input matrix $\boldsymbol{X} = \{x(r,s); 0 \le r, s < N\}$ generating the $N \times N$ matrix $\boldsymbol{Z} = \{z(p,q); 0 \le p, q < N\}$

$$z(p,q) = \sum_{i=0}^{L-1} g_1(l) \cdot x(p, q-;) \quad \text{for } 0 \le q < \frac{N}{2} \tag{3a}$$

$$z(p,q) = \sum_{l=0}^{L-1} g_m(l) \cdot x\left(p, 2^m\left(q - -N \sum_{i=1}^{m-1} 2^{-i}\right)\right) \quad \text{for } \begin{cases} 1 < m < M \\ N \sum_{i=1}^{m-1} 2^{-i} \le q < N \sum_{i=1}^{m} 2^{-i} \end{cases} \tag{3b}$$

$$z(p,q) = \sum_{k=0}^{2^M-1} h(k) \cdot x\left(p, 2^M\left(q - N \sum_{i=1}^{M} 2^{-i}\right) - k\right) \quad \text{for } N \sum_{i=1}^{M} 2^{-i} \le q < N. \tag{3c}$$

The *second filter bank* is the column 1-D DWT on the intermediate matrix $\boldsymbol{Z}$ defined in a way similar to (3). The cascade of these filter banks defines the $N \times N$ matrix $\boldsymbol{Y} = \{y(u,v); 0 \le u, v < N\}$ as the 2-D DWT of matrix $\boldsymbol{X}$. Extension to the multidimensional case consists of a cascade of filter banks (one for each dimension of the DWT) separated by data transposition.

A parallel 2-D $N \times N$ DWT implementation on P processors[1] can be obtained [27], [28] by partitioning the input and the intermediate matrices. This approach (*Algorithm A*) is described in Fig. 1. In step 1, the rows of the input matrix $\boldsymbol{X}$ are downloaded from the host computer to the processors N/P rows per processor. In step 2, each processor performs the 1-D DWT on the rows. In step 3, the partial results are uploaded by rows to the host to create the intermediate matrix $\boldsymbol{Z}$. Transposition is implicitly performed by reading matrix $\boldsymbol{Z}$ by columns in step 4. Steps 5 and 6 are the columnwise 1-D DWT and the result uploading, respectively. The result is the transposed matrix $\boldsymbol{Y}$. The column-wise computing phase (i.e., steps 4–6) can start only after results of the row-wise DWT's have been uploaded to the host; thus, PE_1 is idle during the whole time interval [C-B']. Moreover, Algorithm A requires a significant amount of interprocessor communications to transpose the intermediate matrix, inducing a long latency.

To avoid these drawbacks, we propose a matrix-based approach to parallelize the 2-D DWT. The basic idea is to describe each filter as a matrix and the filtering as matrix-vector multiplication. This is a well-known approach to implement convolution; the innovative idea is to use it in order to avoid data transposition and interprocessor communications, maintaining a high level of parallelism.

Let us consider the M matrices

$$\boldsymbol{W}_m = \left\{ w_m(i,j); 0 \le i < N, 0 \le j < \frac{N}{2^m}, 1 \le m \le M \right\}$$

and the matrix

$$\boldsymbol{R}_m = \left\{ r_M(i,j); 0 \le i < N, 0 \le j < \frac{N}{2^M} \right\}$$

where

$$w_m(i,j) \equiv \begin{cases} g_m(l), & \text{if } i = 2^m(j-l) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$r_M(i,j) \equiv \begin{cases} h(2^M(j+1) - i), & \text{if } 2^M j \le i < 2^M(j+1) \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

[1]For simplicity, we assume that N/P is an integer. When this does not hold, algorithms discussed in this correspondence will still work but will be unbalanced: Some processors will operate on $\lceil N/P \rceil$ data sets, whereas others wil operate on $\lfloor N/P \rfloor$ sets without being able to fully use their computational power.

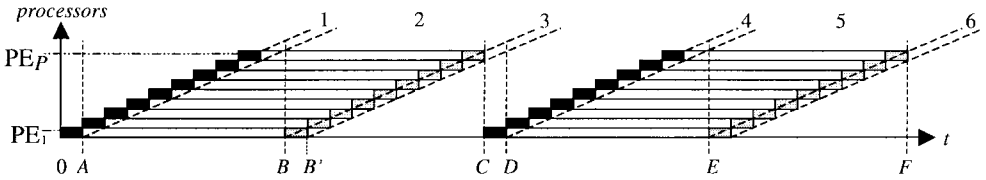

Fig. 1. Time diagram for Algorithm A. [A-0]: the first N/P rows of the input matrix are downloaded to PE$_1$. [B-A]: PE$_1$ computes the 1-D DWT of N/P rows. [C-B]: uploading of N row-wise 1-D DWT's performed by P processors (PE's). The column-wise DWT's cannot be computed till all the transformed rows have been uploaded to the host for transposition. [D-C]: the first N/P columns of the intermediate results are downloaded to PE$_1$. [E-D]: PE$_1$ computes the 1-D DWT of N/P columns. [F-E]: uploading of the column-wise 1-D DWT's, i.e., the final 2-D DWT result.

Consider the $N \times N$ sparse matrix $\boldsymbol{W}$

$$\boldsymbol{W} \equiv [[\boldsymbol{W}_1][\boldsymbol{W}_2][\boldsymbol{W}_3] \cdots [\boldsymbol{W}_M][\boldsymbol{R}_M]] \tag{6}$$

and the N-point column vector $\boldsymbol{x}_r$ given by the rth row of the input matrix $\boldsymbol{X}$. The 1-D DWT of the N-point vector $\boldsymbol{x}_r^T$ (transpose of the input vector $\boldsymbol{x}_r$) can be obtained according to (1) and (2) by

$$\boldsymbol{z}_r^T = \boldsymbol{x}_r^T \times \boldsymbol{W}. \tag{7}$$

The $N \times N$ intermediate matrix $\boldsymbol{Z}$ can thus be obtained by juxtaposition of the row vectors $\boldsymbol{z}_r^T$ resulting from (7) for each input row $\boldsymbol{x}_r$, according to (3):

$$\boldsymbol{Z} = \boldsymbol{X} \times \boldsymbol{W}. \tag{8}$$

The 2-D DWT of the matrix $\boldsymbol{X}$ can be written by using the following matrix expression:

$$\boldsymbol{Y} = \boldsymbol{Z}^T \times \boldsymbol{W} \equiv (\boldsymbol{X} \times \boldsymbol{W})^T \times \boldsymbol{W} \equiv \boldsymbol{W}^t \times \boldsymbol{X}^T \times \boldsymbol{W}. \tag{9}$$

By transposing both of the members of (9)

$$\boldsymbol{Y}^T = \boldsymbol{W}^T \times \boldsymbol{X} \times \boldsymbol{W}. \tag{10}$$

The N columns $\boldsymbol{y}_q^T (0 \le q < N)$ of the matrix $\boldsymbol{Y}^T$ (i.e., the rows of $\boldsymbol{Y}$) are given by

$$\boldsymbol{y}_q^T = \boldsymbol{W}^t \times (\boldsymbol{X} \times \boldsymbol{w}_q) \tag{11}$$

where $\boldsymbol{w}_q$ is the qth column of $\boldsymbol{W}$, and parentheses suggest the most efficient computing sequence.

Equation (11) has been obtained from (10) by partitioning $\boldsymbol{Y}^T$ by columns. This suggests an efficient parallel algorithm for computing the 2-D DWT (*Algorithm B*), which is described in Fig. 2. Downloading of the whole input matrix $\boldsymbol{X}$ is performed in steps 1 and 2. In the steps 2 and 3, each processor computes (11) for N/P different vectors $\boldsymbol{w}_q$. Uploading of N/P columns $\boldsymbol{y}_q^T$ to the host computer is performed in step 4. Downloading and computing can be overlapped (see step 2) since each processor can start its computation as soon as its data are received from the host. No time skewing is thus necessary. Uploading can be executed simultaneously by each processor [step 4 in Fig. 2(b)] if connections among processors and host allow contemporaneous data transfer. Otherwise, step 4 has to be skewed, as shown in Fig. 2(a). Unlike Algorithm A, which requires transposition of the intermediate results, Algorithm B completely avoids any interprocessor communications.[2]

[2] Equation (11) could suggest that Algorithm B needs O(N^2)-sized memory in each processor to store $\boldsymbol{W}$ and $\boldsymbol{X}$. However, (4) and (5) clearly show that only LM values (i.e., L points for each of the M filters) are necessary. Besides, in a processor, each row either of $\boldsymbol{X}$ or $\boldsymbol{Z}^T$ can be written over the previous one since any point of these vectors is processed only once by the matrix-vector multiplication; therefore, the required memory is only $O(N)$, as for Algorithm A.

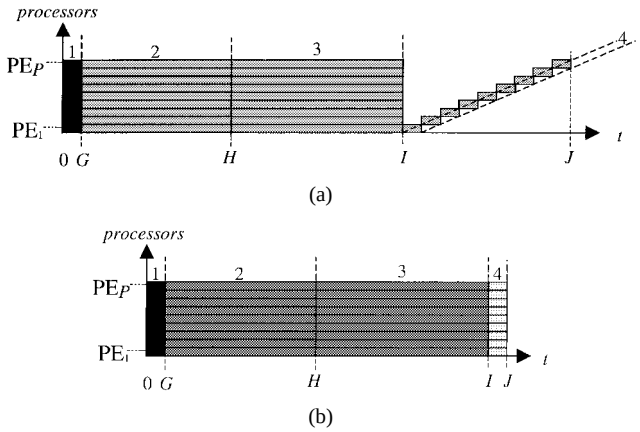

Fig. 2. Time diagram for Algorithm B, when uploading (a) cannot and (b) can be performed simultaneously by all processors (PE's). The interval [I-G] is used to perform the computation. If processing is balanced, this interval has the same duration of step 2 plus step 5 in Fig. 1. In the interval [H-G], downloading and computing can be overlapped. In Algorithm A, this overlapping is not feasible.

III. Performance Evaluation and Experimental Results

The performance of Algorithm B with respect to Algorithm A is evaluated both from a theoretical point of view and with an experimental implementation on the AT&T DSP3 parallel processor [29].

A. Theoretical Speedup

The speed-up S of Algorithm B with respect to algorithm A is defined as

$$S \equiv \frac{\text{Time to compute a 2-D DWT by using Algorithm A}}{\text{Time to compute a 2-D DWT by using Algorithm B}}. \tag{12}$$

The time intervals in the diagrams shown in Figs. 1 and 2 are as follows.

- [A-0] and [D-C] are the latencies required to download N/P input rows and N/P transformed columns, respectively

$$[A-0] = \frac{qN^2}{\beta(P)P} \quad [\text{s}] \tag{13a}$$

$$[D-C] = \frac{q'N^2}{\beta(P)P} \quad [\text{s}] \tag{13b}$$

  where $\beta(P)$ is the average bandwidth (in bits per second) of the interconnections among P processors, and the host q and q' are the precision (in bits) of input data and transformed coefficients, respectively.
- [B-A] and [D-E] are the latencies required to compute the 1-D DWT of N/P rows and N/P columns, respectively, in a single processor

$$[B-A] = [D-E] = \tau(N)\frac{N}{P} \quad [\text{s}] \tag{14}$$

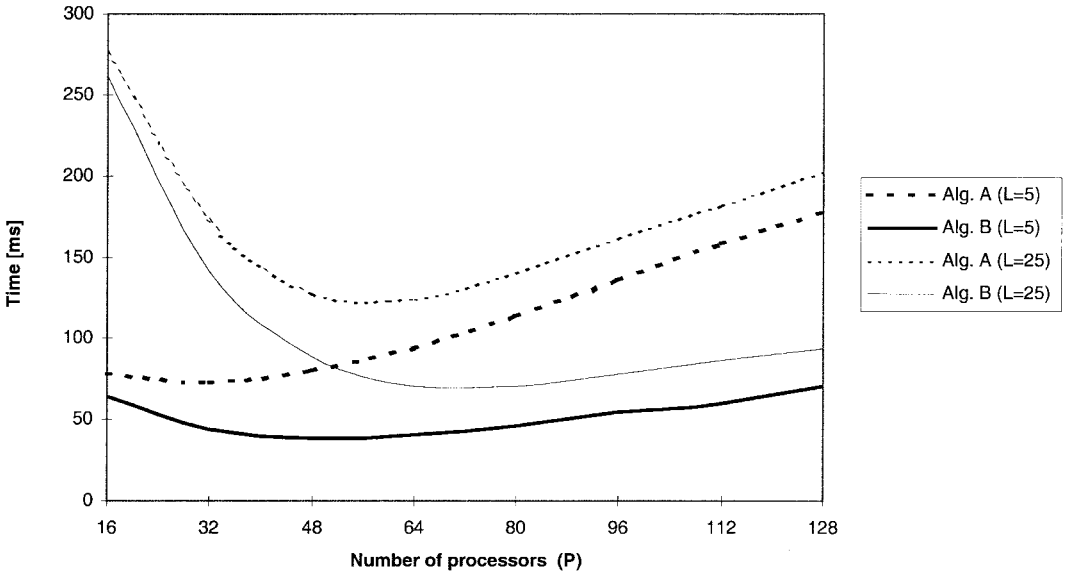

Fig. 3. Execution time required by Algorithms A and B to perform the 2-D DWT of a 128 × 128 matrix by using P processors. Time to 2-D DWT a 128 × 128 image by Algorithm A and Algorithm B.

where $\tau(N)$ is the time needed by a processor to generate the N-point 1-D DWT.

- [C-B] and [F-E] are the latencies required to upload N transformed rows and N transformed columns, respectively

$$[C-B]=[F-E]=\frac{q'N^2}{\beta(P)P}\quad[\text{s}]. \tag{15}$$

- [H-0] is the latency required to download the whole input matrix into the multiprocessor system

$$[H-0]=\frac{qN^2}{\beta(P)P}\quad[\text{s}]. \tag{16}$$

- [G-0] is the portion of [H-0] that is not overlapped with the actual computation

$$[G-0]=\eta[H-0]\quad[\text{s}]\qquad(0<\eta\leq 1). \tag{17}$$

- [J-I] is the latency required by each processor to upload the N/P vectors $\boldsymbol{y}_q^T$ either in a skewed or in a parallel way

$$[J-I]=\frac{q'N^2}{\beta(P)}\chi\quad[\text{s}] \tag{18}$$

where χ is either 1 or $1/P$ in case of skewed or parallel upload, respectively.

- [I-G] is the latency needed by a single processor to compute (11) for each of the N/P different vectors $\boldsymbol{y}_q$ assigned to the processor itself

$$[I-G]=\delta(N)\frac{N}{P}\quad[\text{s}] \tag{19}$$

where $\delta(N)$ is the time needed by a processor to compute one N-point vector $\boldsymbol{y}_q$.

The speed-up S is thus given by

$$S=\frac{[F-0]}{[J-0]}=\frac{[A-0]+[B-A]+[C-B]+[D-C]+[E-D]+[F-E]}{[G-0]+[I-G]+[J-i]}. \tag{20}$$

Since matrix $\boldsymbol{W}$ is sparse, each processor needs to store and use only the elements that are *a priori* known to be nonzero. This can be implemented without additional control circuitry since each processor is a general-purpose processor executing a software program; such a program can easily be written to skip the unnecessary multiplications. Consequently, the computation for Algorithm B is perfectly balanced among all processors, we can derive from (11) that $\delta(N)=2\tau(N)$.

The speedup becomes

$$S=\frac{(q+q')\dfrac{N^2}{\beta(P)P}+2\left(\tau(N)\dfrac{N}{P}+q'\dfrac{N^2}{\beta(P)}\right)}{q\dfrac{N^2}{\beta(P)}\eta+2\tau(N)\dfrac{N}{P}+q'\dfrac{N^2}{\beta(P)}\chi}\quad[\text{s}]. \tag{21}$$

Since, generally, $q'\geq q$, the speedup is always greater than 1 for any χ and η.

In evaluating the speedup [in particular, both $\tau(N)$ and $\delta(N)$], we did not make use of any optimization (e.g., coding strategy, compiler optimizations, fetching strategies) since they are machine or environment specific. The detailed analysis of possible influences both of optimizations and architectural factors goes beyond the scope of this correspondence, which aims to describe a machine-independent algorithm. It is worth noting that these factors have similar influence both in Algorithms A and B since the mathematical operations performed by them are similar.

B. Experimental Results

Algorithms A and B have been implemented and tested on the AT&T DSP-3 multiprocessor system, even though the generality of the proposed algorithm allows for implementation over any multiprocessor system. This architecture [29] has from 16–128 processors, each of them containing a DSP-32C unit (50 MHz, 25 MFLOPS, with a multiplier and an adder working in parallel) and 512K words (32-bit) of local memory.

Our test aims to demonstrate the relative performance improvement that can be obtained by Algorithm B with respect to Algorithm A. In our experiments, processors have been configured in a pipeline to test the worst connection scheme. Data downloading from the host to the last processor of the pipeline and, similarly, results uploading onto the host involve, therefore, all processors in the pipeline. With respect to the implementation presented here, our algorithm can achieve higher absolute performances if it is implemented on more powerful

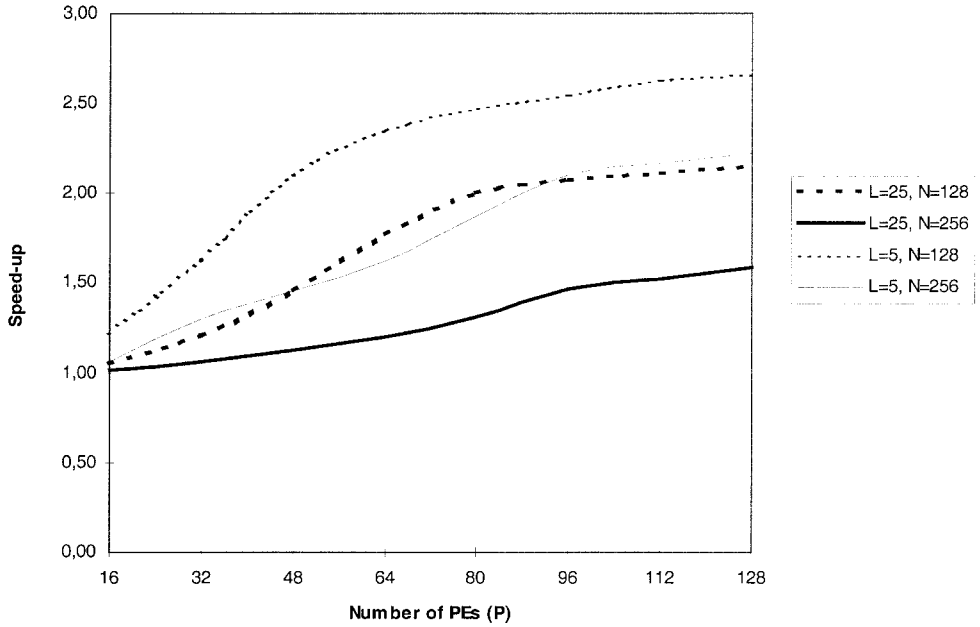

Fig. 4. Speedup of Algorithm B with respect to Algorithm A to perform the 2-D DWT of a $N \times N$ matrix by using P processors. Comparison between the times to 2-D DWT 128 $\times$ 128 and 256 $\times$ 256 images by Algorithm B and Algorithm A.

processors or on machines having more efficient connection schemes among the host and the processors. To clearly establish the advantages of avoiding transposition in Algorithm B, we have not exploited its ability to begin processing data as soon as they become available (i.e., $\eta = 1$ in our implementation); this feature could provide an even higher speedup.

Fig. 3 summarizes the results of our experiments for N =128; similar results hold for higher values of N. These experiments were performed with $M = 4$; $L = 5$, 25, with P varying from 16–128. Since the machine available in our laboratory has only 16 processors, data for $P = 16$ were obtained by measuring the performance of C++ programs running on the DSP-3, whereas data for $P > 16$ were derived by simulating the larger machines on our system.[3] The dip in Fig. 3 is due to the behavior of the average communication bandwidth, $\beta(P)$. In our pipelined processor connection scheme, the average communication bandwidth decreases with the number P of processors. For a critical value of P (depending on the number of operations performed by each processor), this negative effect overcomes the speedup achieved by the algorithm in each processor. For Algorithm A, this critical value is approximately 32 when $L = 5$ but becomes approximately 50 when $L = 25$ due to the much higher number of operations per processor. The critical values (i.e., the abscissas of the dips) are higher for the same value of L in Algorithm B since the absence of transposition reduces the impact of communications.

Fig. 4 gives the speedup by using varying numbers of processors for (L, N) equal to (5, 128), (5, 256), (25, 128), and (25, 256), respectively. The speedup is greater than 1 for all cases. Algorithm A performs transposition by uploading the intermediate results by rows and then downloading them by columns. The speedup of Algorithm B versus Algorithm A is thus mainly related to the number of input/output operations; since both algorithms need to download $\boldsymbol{X}$ and upload $\boldsymbol{Y}$, Algorithm A requires twice the total number of input/output operations of Algorithm B. Consequently, when the number P of processors is high, the speedup tends to a constant since the time intervals [B-A], [E-D], and [I-G] can be neglected (i.e., the algorithms become I/O bound).

[3]The 16 physical processors execute the computation of the first 16 units of a system having $P > 16$. The remaining $P - 16$ units are virtual; their presence is simulated only to evaluate the system operation time. Data are also downloaded for virtual processors but not used. The virtual units are assumed to operate in parallel with the physical ones. Results of the P processors are uploaded through the processors' pipeline. Propagation time of the $P - 16$ blocks that should be produced by the virtual processors is evaluated without any concern to the actual propagated values. This is achieved by propagating the last generated real block for $P - 16$ additional times.

IV. Conclusions

Implementation of 2-D discrete wavelet transforms on parallel general-purpose computers have been discussed. A traditional parallelization technique has been considered that uses input data partitioning (Algorithm A). A second algorithm (Algorithm B) based on matrix-vector multiplications is introduced. The main features of Algorithm B are the *absence* of interprocessor communications, *elimination* of transposition of intermediate results, the possibility of overlapping input and computing phases, full use of the available processor parallelism, modularity, and scalability. The approaches have been tested and compared on an AT&T DSP-3 parallel computer; experiments have shown a speedup of algorithm B from 1.01–2.63 with respect to the traditional parallelization approach.

Acknowledgment

The authors are grateful to the anonymous referees for providing comments and suggestions that greatly helped in improving this correspondence.

References

[1] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[2] I. Daubechies, *Ten Lectures on Wavelets*. Philadelphia, PA: SIAM, 1992.

[3] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 2, pp. 674–693, 1989.

[4] J. Fridman and E. S. Manolakos, "Discrete wavelet transform: Data dependence analysis and synthesis of distributed memory and control array architectures," *IEEE Trans. Signal Processing*, vol. 45, pp. 1291–1308, 1997.
[5] G. Beylkin, R. Coifman, and V. Rokhlin, "Fast wavelet transforms and numerical algorithms I," *Commun. Pure Appl. Math.*, vol. 44, pp. 141–153, 1991.
[6] J. Fridman and E. Manolakos, "On the scalability of the 2-D discrete wavelet transform algorithms," in *Multidimensional Systems and Signal Processing*. Boston, MA: Kluwer, 1997, vol. 8, pp. 185–217.
[7] G. Knowles, "VLSI architecture for the discrete wavelet transform," *Electron. Lett.*, vol. 26, pp. 1184–1185, 1990.
[8] A. S. Lewis and G. Knowles, "VLSI architecture for 2D Daubechies wavelet transform without multipliers," *Electron. Lett.*, vol. 27, pp. 171–173, 1991.
[9] K. K. Parthi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 191–202, 1993.
[10] C. Chakrabarti and M. Vishwanath, "Efficient realizations of discrete and continuous wavelet transforms: From single chip implementations to mappings on SIMD array computers," *IEEE Trans. Signal Processing*, vol. 43, pp. 759–771, 1995.
[11] J. Bae and V. K. Prasanna, "Synthesis of VLSI architectures for two-dimensional discrete wavelet transforms," in *Proc. IEEE Int. Conf. Application-Specific Array Processors*, 1995, pp. 174–181.
[12] H. Y. H. Chuang and L. Chen, "VLSI architecture for fast 2D discrete orthonormal wavelet transform," *IEEE J. VLSI Syst.*, vol. 26, pp. 225–236, 1995.
[13] J. Vega-Pineda, S. Cabrera, and Y. C. Chang, "VLSI implementation of a wavelet image compression technique using replicant coding/decoding cells," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1995, pp. 1173–1176.
[14] C. C. Hsu, J. Ding, and M. E. Zaghloul, "An image discrete wavelet transform and the hardware implementation," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1995, pp. 1315–1319.
[15] M. Vishwanath, R. M. Owens, and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 305–316, 1995.
[16] B. M. R. Lang and A. Spray, "Input buffering requirements of a systolic array for the inverse discrete wavelet transform," in *Proc. IEEE Conf. Application-Specific Array Processors*, July 1995, pp. 165–173.
[17] C. Chakrabarti, M. Vishwanath, and R. M. Owens, "Architectures for wavelet transforms: A survey," *J. VLSI Signal Process.*, vol. 14, pp. 171–192, 1996.
[18] H. Sava, M. Fleury, A. C. Downtown, and A. F Clark, "Parallel pipeline implementation of wavelet transform," *Proc. Inst. Elect. Eng., Vision Image Process.*, vol. 144, no. 6, pp. 355–359, Dec. 1997.
[19] J. Lu, "Parallelizing mallat algorithm for 2-D wavelet transform," *Inform. Process. Lett.*, vol. 45, pp. 255–259, 1993.
[20] M. Maresca and H. Li, "Morphological operations on mesh connected architectures: A generalized convolution algorithm," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision Pattern Recogn.*, 1986, pp. 299–304.
[21] L. Auslander, E. Feig, and S. Winograd, "New algorithms for the multidimensional discrete fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 388–403, 1983.
[22] R. Blahut, *Fast Algorithms for Digital Signal Processing*. Reading, MA: Addison-Wesley, 1985.
[23] I. Gertner and M. Rofheart, "A parallel algorithm for 2-D DFT computation with no interprocessor communication," *IEEE Trans. Paral. Distrib. Syst.*, vol. 1, pp. 377–382, 1990.
[24] I. Gertner, "A new efficient algorithm to compute the two-dimensional discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1036–1050, 1988.
[25] 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.
[26] V. Milutinovic, A. B. Fortes, and L. H. Jamieson, "A multiprocessor architecture for real-time computation of a class of DFT algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 1301–1309, 1986.
[27] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[28] H. C. Andrews, *Computer Techniques in Image Processing*. New York: Academic, 1970.
[29] AT&T, *DSP3 General Information Manual*.