Parallel Architecture for the Discrete Wavelet Transform based on the Lifting Factorization

Wenqing Jiang and Antonio Ortega

Integrated Media Systems Center
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-2564
Email: [wqjiang,ortega]@sipi.usc.edu

ABSTRACT

One major difficulty in parallel architecture design for the Discrete Wavelet Transform (DWT) is that the DWT is not a block transform, with the exception of the trivial Haar transform. As a result, frequent communication has to be set up between processors to exchange data to ensure that correct boundary wavelet coefficients are computed. The significant communication overhead thus hampers the improvement of the efficiency of parallel systems, specially for processor networks with large communication latencies. In this paper we propose, a new technique, called *Boundary Postprocessing*, that allows the correct transform of boundary samples. The basic idea is to model the DWT as a *Finite State Machine* (FSM) based on the lifting factorization of the wavelet filterbanks. Application of this technique leads to a new parallel DWT architecture, *Split-and-Merge*, which only requires one communication setup between neighboring processors for any arbitrary level of wavelet decompositions. Example designs and performance analysis for 1D and 2D DWT show that the proposed technique helps to reduce greatly the interprocessor communication overhead. For the best available parallel lifting DWT algorithm, a speedup of abut 30% on average is observed in our experiments using the proposed technique.

Keywords: Discrete Wavelet Transform, Parallel Architecture, Finite State Machine, Boundary Postprocessing, Split-and-Merge

1. INTRODUCTION

In this paper we study the problem of parallel architecture design for the Discrete Wavelet Transform (DWT). The significance of the problem can be seen in that fast DWT computation is essential in a number of DWT-based image processing applications, such as satellite imagery compression and analysis in remote sensing,[1,2] fast image retrieval and browsing in large databases[3] and real-time pattern recognition and autonomous tracking.[4] Fast DWT algorithms are being actively developed to reduce the number of multiplication-add operations.[5,6] Another alternative consists of making use of devoted Massively Parallel Processors (MPP), such as Intel's Paragon and MasPar's MP-1/2, or cheap multiple Processing Elements (PE), such as the Network of Workstations (NOW)[7] and the Local Area Multicomputers (LAM),[8] to perform a parallel implementation of the DWT.[1,9–11]

One key issue in an efficient parallel algorithm design is to reduce the interprocessor communication overhead. This becomes even more critical for a DWT parallel implementation since, with the exception of Haar transform, the DWT is not a block transform.

Note that boundary issues are also encountered in standard filtering using the FFT and can be easily dealt with appropriate data overlapping. However, because the DWT consists of the *recursive* application of a filtering operation, the boundary problems become particularly important and deserve special attention. As an example, assume a three level decomposition is to be performed using two processors. In this case, either the two processors are given sufficient overlapped data to carry on the whole computation without communicating with each other (and this overlap can be large) or alternatively they have to communicate samples after each of the levels of the decomposition has been computed. This example shows the increased importance of boundary processing when recursive filterbanks are used. The example also points out to the two parameters we can use to measure performance, namely, the amount of data to be transmitted between processors (or to be stored in the processor if a sequential computation is used) and the number of times data has to be communicated between processors.

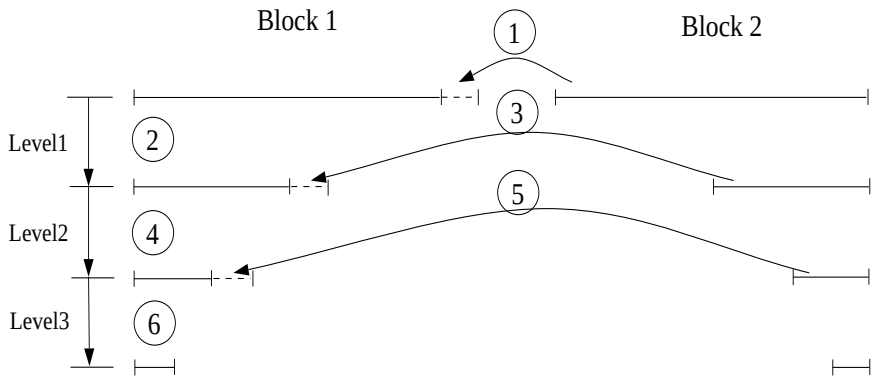

Figure 1. An example DWT dataflow chart using *Boundary Preprocessing* in a two-processor parallel system. Processors 1 and 2 are allocated with input data block 1 and 2 respectively. Solid lines: completely transformed data; Dashed lines: boundary samples from the neighboring block. Operations 1,3,5: communicate boundary data samples to neighboring blocks; Operations 2,4,6: transform for current level.

Most existing parallel architecture designs adopt a *Boundary Preprocessing* approach in which raw input data samples are exchanged between neighboring processors before the start of wavelet decomposition at each level.[9,11] This is shown in Fig.1 using a two-processor system as an example. For a three-level wavelet decomposition, three communications are necessary to transfer boundary samples from one processor to another. In general, for a J-level wavelet decomposition, there would be J data exchanges at each decomposition level across neighboring processors. Obviously this communication overhead adversely affects the speedup of parallel systems, specially those with large communication latencies, such as NOWs and LAMs.[12]

There are two approaches proposed in the literature aimed to reduce the interprocessor communication in parallel DWT algorithms. The first is the *overlap* technique which stores enough input data samples at each processor so that no communication is needed.[13] This approach, however, increases the buffer requirements for each processor exponentially with the increase of decomposition level since extra raw data samples have to be stored for the DWT computation at each level. Moreover, as the number of levels of decomposition increases, the amount of overlap also becomes larger and thus the memory requirements increase. The second approach is the *tiling* method,[10] which approximates, at each processor, unavailable boundary data samples by symmetric extensions or periodic extensions, for example. This approach, although it completely removes the interprocessor communication, results in incorrect wavelet coefficients along block boundaries. Performance degradation has been reported in low bit rate image coding due to incorrect wavelet coefficients near block boundaries.[14] Furthermore, the computation error can also negatively affect the accuracy of pattern recognition and tracking in image analysis applications.

In this paper, we propose a new technique, *Boundary Postprocessing*, for the DWT computation near block boundaries. Using this technique, the DWT can be computed correctly while the interprocessor communication overhead is significantly reduced. The basic idea is to model the DWT as a *Finite State Machine (FSM)*, which updates/transforms each raw input sample (initial state) progressively into a wavelet coefficient (final state) as long as there are enough neighboring samples present. Obviously, data samples near block boundaries can only be updated to intermediate states due to lack of enough neighboring samples. Rather than communicating raw data samples before the start of the decomposition at each level, as in *Boundary Preprocessing*, these partially updated boundary samples, called *state information*, are collected at each level and communicated after the independent transform of each block. A postprocessing operation is then initiated to complete the transform for boundary samples.

The proposed FSM model takes advantage of filterbank factorizations such as the lattice factorization by Vaidyanathan and Hoang[15,16] and the ladder structure by Marshall.[17] In particular, Daubechies and Sweldens[6] have shown recently that, the polyphase matrix representation of any FIR wavelet filters can be factored into a product of a series of "lifting" steps (elementary upper/lower triangular matrices). DWT computation based on such lifting factorizations have some attractive properties, such as fast computation, in-place calculation and easy convertibility to integer transforms. It is the in-place computation property that allows us to introduce the FSM approach. For this reason we use the lifting factorizations as the basis for our work. Note that filterbank factorizations have been motivated

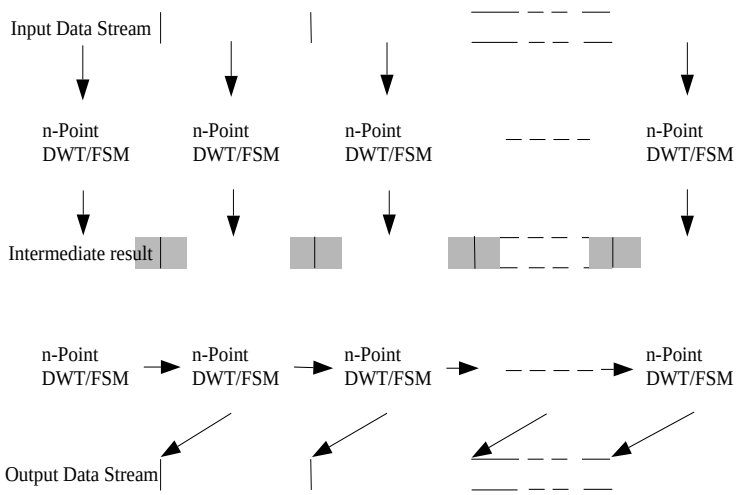

Figure 2. The proposed *Split-and-Merge* parallel DWT architecture. The shaded parts store the state information. In *Split* stage, each processor computes its allocated data independently up to the required decomposition level. In *Merge* stage, a one-way communication is initiated to communicate the state information to neighboring processors. A postprocessing operation is then started to complete the transform for boundary samples.

traditionally by the reduction of memory and number of operations, whereas here we demonstrate that they can also contribute to a reduction in the communication overhead in a parallel computation.

Application of the proposed *Boundary Postprocessing* technique results in a new parallel DWT architecture, *Split-and-Merge*, shown in Fig.2. As one can see, for 1D wavelet decompositions, only one interprocessor data exchange is needed for any J-level wavelet decompositions. Compared to existing approaches which requires J communications,[1,9–11] the interprocessor communication overhead is significantly reduced. The proposed technique not only can be applied to parallel systems built on NOWs and LAMs, but also to MPP systems. Interestingly, it will be shown that the proposed design can make use efficiently of both the *single-port* and *multi-port* communication model in a mesh connected processor network.[9] We will also show that with this algorithm it is the communication overhead can be reduced when going from a single-port to a multi-port communication model, in contrast with what was observed when standard DWT algorithms are used.[9]

The remainder of this paper is organized as follows. In the next Section, the FSM model for the DWT and the *Boundary Postprocessing* technique based on the lifting factorization are introduced. The *Split-and-Merge* parallel architecture is then introduced. Section 3 provides details on two variations on parallel architecture designs for 2D DWT, block parallel and strip parallel, which correspond to MPPs and LAMs systems, respectively. To show the effectiveness of the proposed architecture, performance analysis and experimental results are given in Section 4.

2. THE SPLIT-AND-MERGE PARALLEL ARCHITECTURE

Throughout this paper, we focus on the Mallat style[18] multilevel octave-band wavelet decomposition system with critical sampling using a two-channel wavelet filterbank. The extensions of our study to systems of *standard* DWTs,[19] multichannel wavelet filterbanks, continuous DWT (no subsampling at each stage after filtering operations),[5] and wavelet packet decomposition, if not trivial, are straightforward.

2.1. DWT as a Finite State Machine

Using the Euclidean algorithm, Daubechies and Sweldens[6] have shown that, for any FIR wavelet filters, the polyphase matrix $\mathbf{P_s}(z)$ (subscript s stands for the synthesis) has a factorization form as

$$\mathbf{P_s}(z) = \prod_{i=1}^{m} \begin{bmatrix} 1 & -s_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -t_i(z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \tag{1}$$

and the corresponding analysis polyphase matrix $\mathbf{P}_a(z)$ as

$$\mathbf{P_a}(z) = \begin{bmatrix} 1/K & 0 \\ 0 & K \end{bmatrix} \prod_{i=m}^{1} \begin{bmatrix} 1 & 0 \\ t_i(z) & 1 \end{bmatrix} \begin{bmatrix} 1 & s_i(z) \\ 0 & 1 \end{bmatrix} \tag{2}$$

where $(s_i(z), t_i(z))$ are Laurent polynomials and $m \leq \lfloor L/2 \rfloor$ (L is the filter length) is determined by the specific factorization from. The Perfect Reconstruction (PR) property can be easily verified as $\mathbf{P}_s(z)\mathbf{P}_a(z) = \mathbf{I}$ where $\mathbf{I}$ is the identity matrix. It has been shown that such a lifting-factorization based DWT algorithm is, asymptotically for long filters, twice as fast as that of the Standard algorithm (Theorem 8 in Daubechies and Sweldens[6]). For example, for the popular $(9,7)$ filters,[20] taking into consideration the filters' symmetries, the cost of Standard algorithm for one-level wavelet decomposition is 11.5 mult/add operations per output sample while the cost of lifting-based algorithm is 7.

From the lifting point of view,[21] these elementary matrices (upper triangular and lower triangular ones) in the factorization form (2) can be further classified into *prediction/lifting, updating/dual lifting* operations. However, from a computational point of view, there is no big difference among these elementary matrices, each of which essentially updates the input data samples using linear convolutions. Without loss of generality, we use $l^i(z)$ to represent either $s_i(z)$ or $t_i(z)$ and $\mathbf{e}^i(z)$ the corresponding elementary matrix. That is

$$\mathbf{e}^i(z) \equiv \begin{bmatrix} 1 & l^i(z) \\ 0 & 1 \end{bmatrix} \qquad or \qquad \mathbf{e}^i(z) \equiv \begin{bmatrix} 1 & 0 \\ l^i(z) & 1 \end{bmatrix}$$

The inverses of $\mathbf{e}^i(z)$ are the matrix inverses, denoted as $\mathbf{e}^{-i}(z)$.

Let us consider the input $\mathbf{X}(z)$ as a column vector, define the intermediate states in the process of transformation, $\{\mathbf{X}^i(z), i = 0, 1, \cdots, 2m+1\}$, where $\mathbf{X}^i(z)$ is the result of applying the operation $\mathbf{e}^{i-1}(z)$ to $\mathbf{X}^{i-1}(z)$, and where the initial input is $\mathbf{X}^0(z) = \mathbf{X}(z)$. Obviously, the forward transform starts from the raw input data samples, the initial state $\mathbf{X}^0(z) = \mathbf{X}(z)$, and, using these elementary matrices $\mathbf{e}^i(z)$, progressively updates the input into the wavelet transform coefficients, the final state $\mathbf{Y}(z) = \mathbf{X}^{2m+1}(z)$. The inverse transform reverses this process to reconstruct the input. One can see that, because of the in-place computation property, every time we generate $\mathbf{X}^i(z)$, we only need to store this set of values, i.e., we do not need to know any of the other $\mathbf{X}^j(z)$, for $j < i$, in order to compute the output. Thus, it is clear that the filtering operation can be seen as a *Finite State Machine* (FSM) as depicted in where each elementary matrix $\mathbf{e}^i(z)$ updates the FSM state $\mathbf{X}^i(z)$ to the next higher level $\mathbf{X}^{i+1}(z)$.

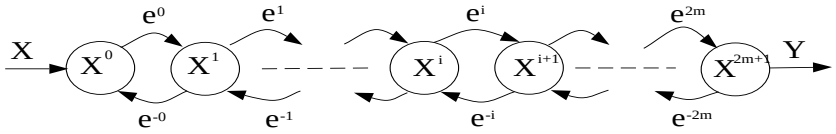

Figure 3. State transition diagram of DWT as a Finite State Machine.

2.2. Boundary Postprocessing

To appreciate the benefits of the FSM model, let us consider one lifting operation $\mathbf{e}^i(z)$ which updates odd samples using neighboring even samples.

$$\mathbf{e}^i(z) = \begin{bmatrix} 1 & 0 \\ az^{-1} + b & 1 \end{bmatrix}$$

Assume the input vector $\mathbf{X}$ is segmented at point $x(n)$ into two subvectors and transformed independently at different processors. Without loss of generality, let index n be even. In vector form, this can be written as:

$$\begin{bmatrix} \vdots \\ x^{i+1}(n-6) \\ x^{i+1}(n-5) \\ x^{i+1}(n-4) \\ x^{i+1}(n-3) \\ x^{i+1}(n-2) \\ x_s^{i+1}(n-1) \end{bmatrix} = \begin{bmatrix} \vdots \\ x^i(n-6) \\ bx^i(n-6) + x^i(n-5) + ax^i(n-4) \\ x^i(n-4) \\ bx^i(n-4) + x^i(n-3) + ax^i(n-2) \\ x^i(n-2) \\ bx^i(n-2) + x^i(n-1) \end{bmatrix}$$

$$\begin{bmatrix} x^{i+1}(n) \\ x^{i+1}(n+1) \\ x^{i+1}(n+2) \\ x^{i+1}(n+3) \\ x^{i+1}(n+4) \\ \vdots \end{bmatrix} = \begin{bmatrix} x^i(n) \\ bx^i(n) + x^i(n+1) + ax^i(n+2) \\ x^i(n+2) \\ bx^i(n+2) + x^i(n+3) + ax^i(n+4) \\ x^i(n+4) \\ \vdots \end{bmatrix}$$

As one can see, this simple filtering operation $\mathbf{e}^i(z)$ updates all the odd samples while the even samples are preserved. The first observation is that no extra memory is needed since the updated results can be stored back to their original memory locations. The second observation is that the boundary sample $x^i(n-1)$ is not fully updated since $x^i(n)$ is in the next block. As a result, $x^i(n-1)$ is updated into $x_s^{i+1}(n-1) = bx^i(n-2) + x^i(n-1)$. However, if we preserve this partially updated value $x_s^{i+1}(n-1)$, then as soon as $x^i(n)$ is communicated from the next block, $x_s^{i+1}(n-1)$ can be updated immediately as $x^{i+1}(n-1) = x_s^{i+1}(n-1) + ax^i(n)$.

This approach of preserving intermediate states (the partially updated value $x_s^{i+1}(n-1)$ in this case) and then continuing later is exactly what a FSM enables us to do. That is, the wavelet transform can be stopped at any intermediate stage and continued later as long as the state information (partially updated sample values) at the break point is preserved. It can be shown that this is also true for multilevel wavelet decompositions.

One may have noticed that, if the updating of sample $x_s^{i+1}(n-1)$ requires the original sample value $x^i(n+1)$, then the above approach will not work since $x^i(n+1)$ will have been updated after the independent processing of the two vectors by $\mathbf{e}^i(z)$. However, the polyphase factorization given in (1) guarantees that such situations never occur. That is, at each stage $e^i(z)$, the samples to be updated only need, besides itself, original values of samples which are not going to be updated at this stage. Following the lifting formulation, at each stage, either odd samples are updated using only even samples or otherwise. Odd and even samples are never updated simultaneously at the same stage except for the final scaling/normalization.

In general, each state transition by one elementary matrix $\mathbf{e}^i(z)$ will leave partially updated samples near the block boundaries. These partially updated samples will be called as the *state information* hereafter. As long as necessary state information in each block is preserved, the boundary transform can be completed after independent transformations of each block. This is done by communicating this state information across blocks and postprocessing operations are initiated to complete the transform. We thus call this boundary transform technique as *Boundary Postprocessing* in contrast to the *Boundary Preprocessing* approach which communicates raw data samples before the start of transform of each block.[22–25,9,11]

What makes this *Boundary Postprocessing* technique attractive is that it can be generalized to any arbitrary number of decomposition levels. The intuition is that nothing prevents us to generalize the FSM model to multilevel wavelet decompositions. After one level of decomposition, half of the samples (the high frequency subband) will remain unchanged while the other half (the low frequency subband) starts over another round of state transitions exactly the same as in the previous level of decomposition. This process continues until the transform reaches the deepest level of decomposition. That is, a DWT with any number of decomposition level can always be modeled as a *Finite State Machine* and the *Boundary Postprocessing* technique can always be applied. Each block is independently transformed up to the required level of decomposition. The state information is communicated after and postprocessing is initiated to complete the transform for boundary samples. In Fig.4 we show an example dataflow chart of a three-level wavelet decomposition using the *Boundary Postprocessing* technique.

2.2.1. An example using the (9,7) filters

The analysis polyphase matrix factorization of the $(9, 7)$ filters, given by Daubechies and Sweldens,[6] is

$$\begin{aligned} \mathbf{P}_a(z) &= e_4(z)e_3(z)e_2(z)e_1(z)e_0(z) \\ &= \begin{bmatrix} \zeta & 0 \\ 0 & 1/\zeta \end{bmatrix} \begin{bmatrix} 1 & l^3(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ l^2(z) & 1 \end{bmatrix} \begin{bmatrix} 1 & l^1(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ l^0(z) & 1 \end{bmatrix} \\ &= \begin{bmatrix} \zeta & 0 \\ 0 & 1/\zeta \end{bmatrix} \begin{bmatrix} 1 & \delta(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \gamma(1+z^{-1}) & 1 \end{bmatrix} \begin{bmatrix} 1 & \beta(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \alpha(1+z^{-1}) & 1 \end{bmatrix} \end{aligned}$$

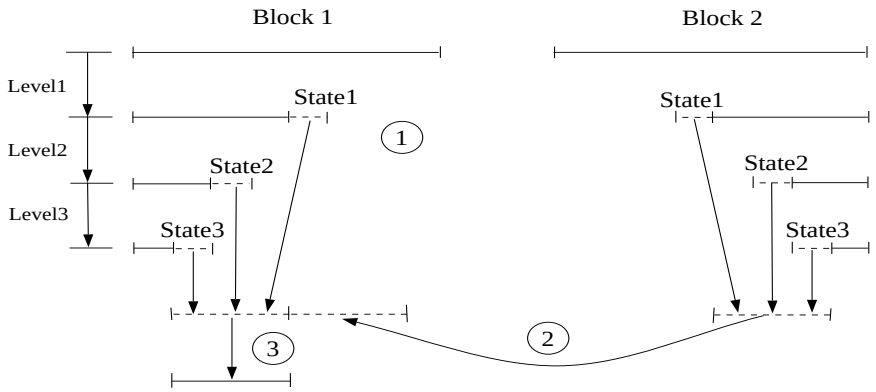

Figure 4. An example dataflow chart of a three-level wavelet decomposition using the proposed *Boundary Post-processing* technique. Solid lines: completely transformed data; Dashed lines: partially transformed data. Operation 1: each block transforms its own allocated data independently and state information is buffered; Operation 2: state information is communicated to neighboring blocks; Operation 3: complete transform for the boundary data samples.

where $\alpha = -1.586134342, \beta = -0.05298011854, \gamma = 0.8829110762, \delta = 0.4435068522$, and $\zeta = 1.149604398$. The corresponding FSM implementation is shown in Fig.5, where each box represents one input sample and the number inside the box denotes the current state of the input sample. Note that for simplicity the state numbering system in this figure is slightly different from the definitions given before. That is, if the *actual value* of an input sample is updated then it goes into next higher state. Otherwise, the state number is not changed. As one can see, for an input 9D vector $\mathbf{X} = [x(-4)\, x(-3) \cdots x(3)\, x(4)]^t$, only the center sample $x(0)$ is updated fully, i.e., transformed into a wavelet coefficient since it has all the 9 samples in its neighborhood. All others (4 samples to the left and 5 to the right) are left in the intermediate states due to lack of neighboring samples. However, once the missing samples ($x(5)$ and $x(6)$ in this case) are communicated from the neighboring processors, then the transform can be continued. A complete two-processor system using *Boundary Postprocessing* example is shown in Fig.6. In the *Split* stage, the input data is segmented into two blocks, each of which is allocated to a different processor. After independent transform in each processor, each processor has partially transform samples along the block boundaries. Next in the *Merge* stage, a one way communication is setup. The state information from processor 1 is communicated to processor 2. The state information from the two processors is combined together and transformed completely which completes the transform for boundary samples.

2.3. The Split-and-Merge Architecture

In Fig.2 the proposed parallel DWT architecture is shown. The striped data partition scheme, as described by Fridman and Manolakos,[9] is used to allocate the input data sequence uniformly onto P available processors. Each processor computes its own allocated data up to the required wavelet decomposition level J. This stage is called as *Split*. The output from this stage consists of two parts: (i)completely transformed coefficients and (ii) the state information (partially updated boundary samples). In the second stage, *Merge*, a one-way communication is initiated and the state information is transfered to the neighboring processors. The state information from the neighboring processor is then combined together with its own corresponding state information to complete the whole DWT transform. The corresponding pseudo C-code algorithm is given in Table 2.3.

3. PARALLEL ARCHITECTURES FOR 2D DWT

In Fig.7 an example 2D DWT with two level decompositions is shown. The data is row transformed first and then column transformed. Naturally, data samples along block boundaries can not be fully transformed due to lack of neighboring samples. This constitute the row and column state information at each level. Buffer size analyses can be found in our more detailed description of the system.[26]

3.1. Block Parallel

We first consider a 2D mesh-connected processor network depicted in Fig.8(a), where each processor is only connected with its immediate neighboring processors. The message passing mechanism is *virtual-cut-through* which models the

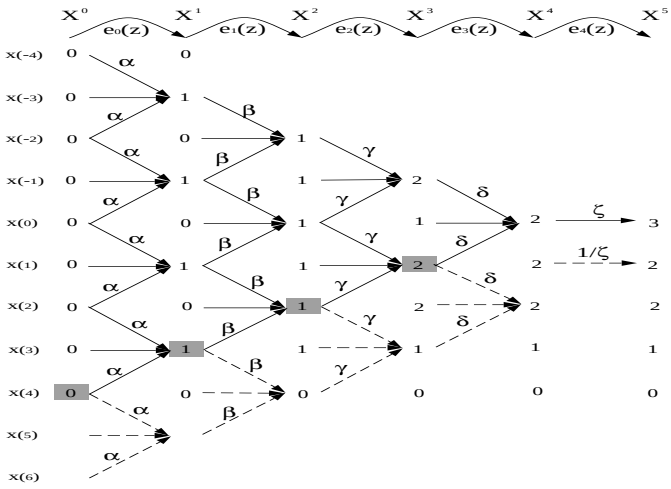

Figure 5. Illustration of DWT as a FSM using the $(9,7)$ wavelet filters. Solid lines represent operations performed for the transform of input pair $(x(0), x(1)$ while dashed lines represent operations to be performed later for input pair $(x(2), x(3)$. Along each line is the multiplication factor with default value 1. The operation at each end node is a summation. Shaded boxes represent state information on one side of the input vector $\mathbf{X}$.

Table 1. The proposed parallel DWT algorithm.

```
begin{ transform in processor p}
  for( j = 0; j < J; j++)
    {
      transform at current level j.
      store state information.
    }
  send state information to processor p + 1;
  receive state information from processor p − 1;
  for( j = 0; j < J; j++)
    {
      transform boundary data samples at current level j.
    }
end
```

communication time T_c for a size-m message over l-link as $T_c = t_s + mt_w + lt_h$ where t_s is the connection establishing time, t_h is the propagation time over a single link, and t_w is the time to transmit a size-1 message. If one message unit is an integer, than t_w is the time to transmit one integer. Other cases are defined accordingly.

Using such a model, the natural partition for 2D data is the block partition strategy shown in Fig.8(b). Processor $P_{m,n}$ is given the input samples with indices $(x, y), mN_r \le x \le (m+1)N_r, mN_c \le y \le (m+1)N_c$. Without loss of generality, assume $W = MN_r$ and $H = NN_c$ where (N_r, N_c) are the block row and column length, (M, N) are the number of processors in row and column direction, and (W, H) are the original 2D data sizes. We also limit the analysis here to cases $J \le min(\log_2 N_r, \log_2 N_c)$. The processor network model used here is the same as that by Fridman and Manolakos[9] for the purpose of comparison.

As shown before, in the first phase, *Split*, each processor is allocated with its portion of data and starts the transform all the way to the required decomposition level J. Upon completion, the data configuration at each processor is shown in Fig.9(a). The center part of each block is completely transformed while the boundaries are left with the partially transformed samples, i.e., the state information. The next stage, *Merge*, communicates the state information and completes the transform for boundary samples. If the *single-port* model is used, then three communications are necessary to complete the transform, one for row state, one for column and one for the intersection

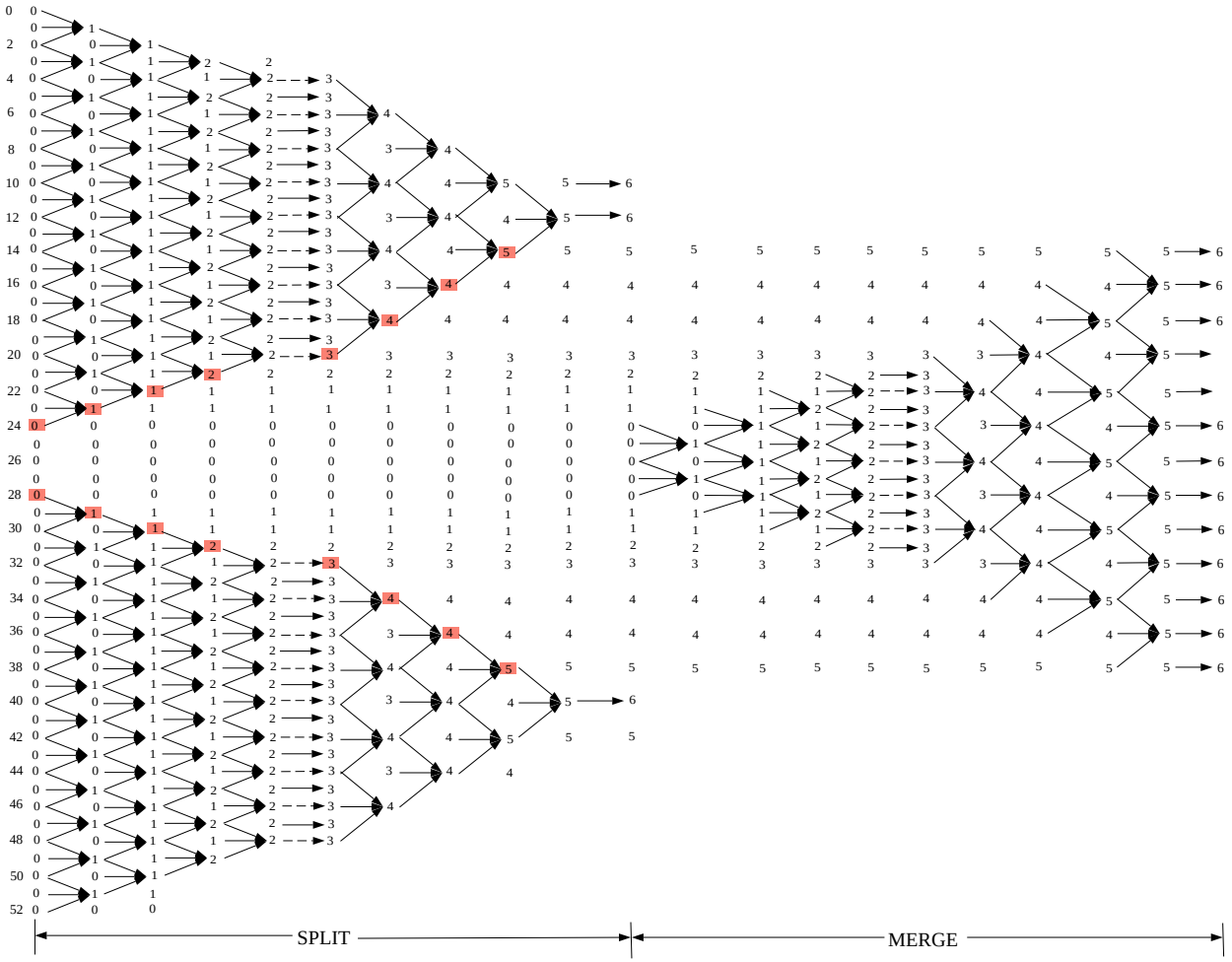

Figure 6. An example *Split-and-Merge* parallel DWT architecture using $(9, 7)$ filters for two level decompositions. Shaded boxes represent partially updated samples to be exchanged between processors in the *Merge* stage.

of row and column state. However, if the *multi-port* model is used, the row and column state information exchange can be implemented simultaneously thus reducing one communication. This *Merge* process is shown in Fig.9 from (a) to (d) for the *single-port* model. If the *multi-port* model is used, (a) and (b) can be combined to simultaneously transmit/receive the row and column state information to/from neighboring processors. This is in contrast to the observation given by Fridman and Manolakos[9] that it was not possible to effectively utilize more than a single communication port at a time in the 2D DWT. The proposed parallel algorithm can reduce the communication overhead if a multi-port model is used.

3.2. Strip Parallel

We now consider another type of processor network in which each processor can communicate to every other processor. A typical example is the LAM/NOW systems where locally connected machines are reconfigured into a parallel system. One example LAM is the bus network shown in Fig.10(a). Though virtually any arbitrary topology can be built upon such a physical processor network, for a parallel system local interprocessor communication is preferred to reduce the network traffic, and hence the communication overhead. Consequently, we propose to use the the strip partition to allocate data to different processors. This is depicted in Fig.10(a) where processor P_n is allocated with input samples of indices $(x, y), 0 \leq x \leq W-1, nN_c \leq y \leq (n+1)N_c$. The block size is now $W \text{x} N_c$.

In the first stage, *Split*, each processor is allocated with its own strip and transforms up to the required level of decomposition J. Since no segmentation is done in the row direction, state information obviously will only

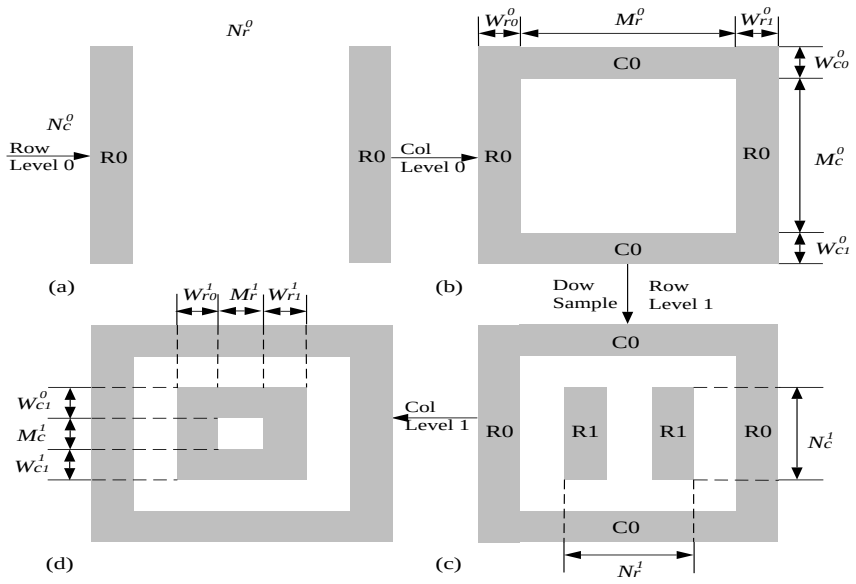

Figure 7. Illustration of 2D DWT as a FSM. Shaded areas represent the state information with $R0/C0$ the row/column state information at level 0 and so on. The input block is first row transformed (a) and column transformed (b) at level 0, then downsampled (taking the LL0 subband) and row transformed at level 1 (c), and column transformed at level 1 (d).

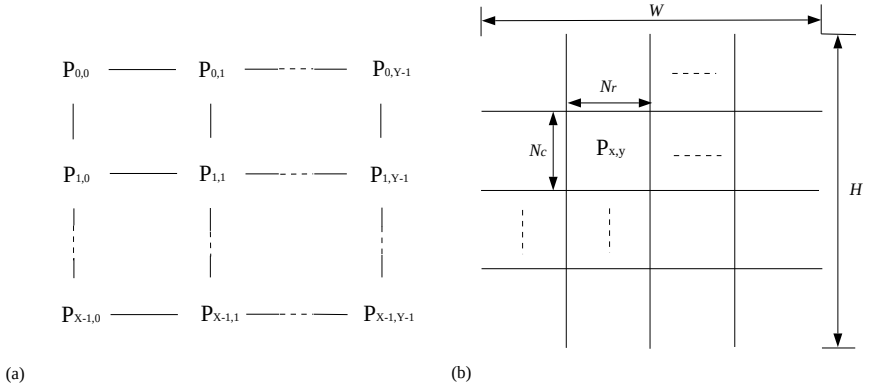

Figure 8. 2D mesh processor network (a) and corresponding data partition (b).

appear along up and down boundaries in each block. This is shown in Fig.11. Next, in the *Merge* stage, only one communication is necessary to transfer/receive the column state information from neighboring processors.

4. PERFORMANCE ANALYSIS AND EXPERIMENTAL RESULTS

4.1. Performance Analysis

The performance analysis is given for 1D DWT. It can be easily extended to the 2D DWT using the separable transform approach. Comparison is done among three different parallel algorithms:

1. *Standard*:[9,11]: each processor computes the DWT using the standard subsample-filtering algorithm.[5] The boundary data transform is completed using the *Boundary Preprocessing* technique;

2. *Lifting*: same as the *Standard* algorithm except that each processor computes the DWT using the fast lifting algorithm[21,6]; and

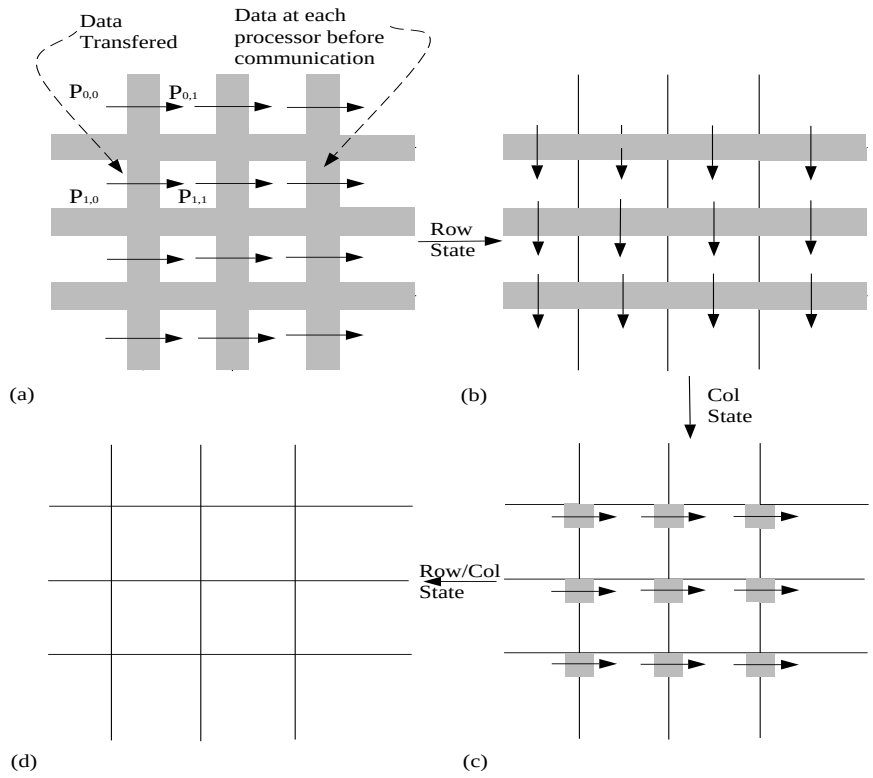

Figure 9. *Merge* operations in 2D mesh processor network. (a) transfer row state information from $P_{i,j}$ to $P_{i,j+1}$; (b) transfer column state information from $P_{i,j}$ to $P_{i+1,j}$; (c) transfer newly generated row state information from $P_{i,j}$ to $P_{i,j+1}$; (d) complete transform for boundary samples. Notice the total amount of data in each processor in the final state (d) is different from the original uniform allocation due to the *Merge* operations.

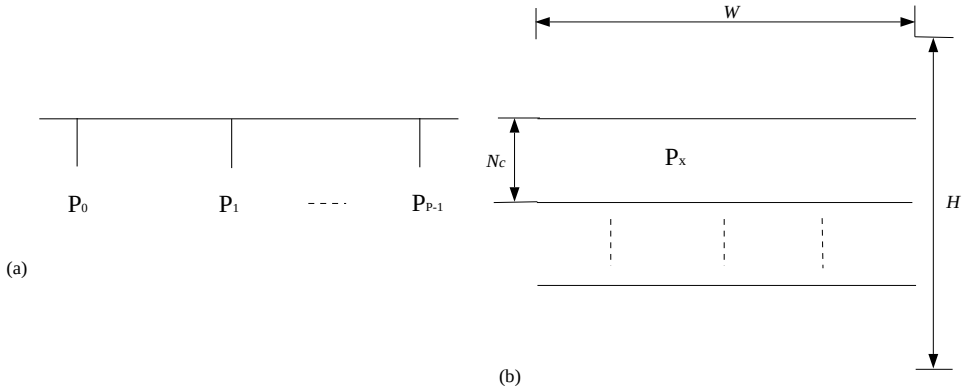

Figure 10. (a) Bus-connected processor network and (b) corresponding strip data partition.

3. *Proposed*: each processor computes the DWT using the fast lifting algorithm and the proposed *Boundary Postprocessing* technique is used for transform of boundary samples.

The performance is given for an N-point sequence 1D DWT with J-level decompositions. The data partition is the stripped partition[9] which allocates the input sequence uniformly to P processors so that each processor has N/P (assumed to be an integer) consecutive samples. The runtime of a parallel algorithm T consists of two parts: (i) time for transform computation (multiplications and additions); and (ii) time for interprocessor communication.

Without loss of generality, denote the execution time of one mult/add operation as t_{ma}[9] and the time to communicate one data sample between neighboring processors t_c. Let t_{setup} be the communication setup time and the

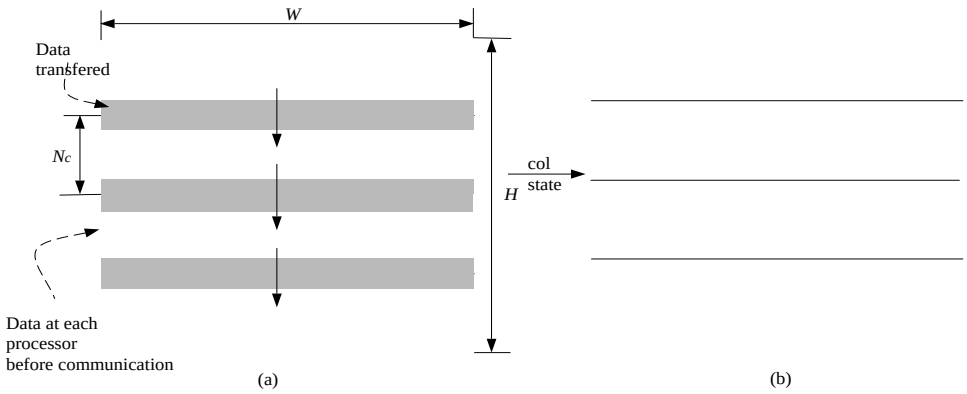

Figure 11. *Merge* operations for strip parallel implementation. (a) transfer row state information from P_i to P_{i+1}; and (b) complete transforms for boundary samples in each processor.

filter length be L. The total number of multiplications and additions of a standard sequential algorithm is

$$\begin{aligned} C_{seq} &= \underbrace{2NL(1-2^{-J})}_{Multiplications} + \underbrace{2N(L-1)(1-2^{-J})}_{Additions} \\ &= 2N(2L-1)(1-2^{-J}), \end{aligned}$$

and the total transform time $T_{seq} = C_{seq} t_{ma}$ is

$$T_{seq} = 2N(2L-1)(1-2^{-J})t_{ma}. \tag{3}$$

For the *Standard* algorithm, the transform computation time can be approximated as $C_{seq}t_{ma}/P$. Before each level of decomposition, $(L-2)$ boundary samples need to be communicated to the adjacent processor.[27] Therefore the total runtime is

$$T_S = \underbrace{2N(2L-1)(1-2^{-J})t_{ma}/P}_{Computation} + \underbrace{J(t_{setup} + (L-2)t_c)}_{Communication} \tag{4}$$

The *Lifting* algorithm has the same communication overhead as that of the *Standard* algorithm. However, it reduces the number of multiplications and additions asymptotically to half of the standard algorithm.[6] Therefore the total runtime is

$$T_L = \underbrace{N(2L-1)(1-2^{-J})t_{ma}/P}_{Computation} + \underbrace{J(t_{setup} + (L-2)t_c)}_{Communication} \tag{5}$$

In the *Proposed* algorithm, the transform computation time is the same as that of the *Lifting* parallel algorithm, however, the communication overhead is reduced. As shown before, only one communication setup is necessary to communicate the state information between adjacent processors. Furthermore, the size of the state information at each decomposition level can be shown to be upper bounded by $(L-2)$.[26] Therefore the total runtime of the proposed parallel algorithm can be estimated as

$$T_P = \underbrace{N(2L-1)(1-2^{-J})t_{ma}/P}_{Computation} + \underbrace{t_{setup} + J(L-2)t_c}_{Communication} \tag{6}$$

Comparing T_S, T_L and T_P, the conclusion is clear that the proposed parallel architectures decreases the algorithm runtime by: (i) reducing the computation time using the lifting DWT algorithm and (ii) reducing the communication overhead using the *Boundary Postprocessing* technique. One can easily show that the proposed parallel algorithm improves also the speedup and efficiency,[28] and keeps the same isoefficiency as existing algorithms.[9]

4.2. Experimental Results

In the simulation the (9,7) wavelet filters are used and the three different parallel DWT algorithms discussed in Section 2, i.e., the standard, the lifting and the proposed one, are implemented. The baseline sequential algorithm, however, is chosen to be the fast lifting DWT algorithm.[6] The strip partition strategy is used in the experiment to segment an input 512x512 image into two strips of size 256x512, each of which is loaded into one machine for transform. The parallel platform is LAM 6.1 from Ohio Supercomputer Center,[8] which runs over Ethernet connected SUN ULTRA-1 workstations in our lab (CPU clock frequency $133MHz$). Two workstations are used to simulate a parallel system with two processors. The algorithm running time is measured using the $MPI\_Wtime()$ function call from MPI libraries averaging over 50 running instances. The relative speedup is calculated against the sequential lifting algorithm as $T_{seq}/T_{para} - 1$. The results of DWT running times for different decomposition levels are given in Table.4.2.

As one can see, the simple parallel standard algorithm and the parallel lifting algorithm do not improve that much from the sequential lifting algorithm (relative speedup is only about 10% to 30%) due to communication overhead between the two workstations. However, using the *Boundary Postprocessing* technique, the proposed parallel algorithm provides speedup from 50% to 70% for all five levels of decompositions. It can be concluded that the proposed parallel algorithm can reduce the DWT computation time by significantly reducing the communication overhead.

Table 2. DWT running time of different parallel algorithms (in seconds).

Level	Sequential Lifting	Parallel Standard		Parallel Lifting		Parallel Proposed	
		time	speedup	time	speedup	time	speedup
1	0.3638	0.3115	17%	0.2745	33%	0.2045	78%
2	0.3649	0.3275	11%	0.2899	26%	0.2338	56%
3	0.3952	0.3490	13%	0.2938	34%	0.2369	67%
4	0.4028	0.3513	15%	0.3041	34%	0.2383	69%
5	0.4041	0.3675	9%	0.3165	28%	0.2417	67%

5. CONCLUSIONS

As a summary, a new *Boundary Postprocessing* technique has been proposed in this paper for parallel DWT computations. Application of this new technique results in a new parallel architecture, *Split-and-Merge*. The basic procedure is that, each processor can compute the transform for its own allocated up to the required level of decompositions. Only one interprocessor communication is necessary to exchange boundary state information to complete the whole transform. Example architecture design and performance analysis have shown that this technique helps to reduce significantly interprocessor communication overhead. Our experimental results show that the proposed parallel algorithm is particularly useful for processor networks with large communication latencies such as LAM systems.

REFERENCES

1. A. Uhl, "A parallel approach for compressing satellite data with wavelets and wavelet packets using PVM," in *Workshop Paragraph '94, RISC - Linz Report Series No. 94-17*, 1994.
2. T. Bell, "Remote sensing," *IEEE Spectrum* , pp. 25–31, 1995.
3. A. Andresen, T. Yang, O. Egecioglu, O. Ibarra, and T. Smith, "Scalability issues for high performance digital libraries on the world wide web," in *Proc. of ADL'96, Forum on Research and Technology Advances in Digital Libraries* , pp. 716–731, 1996.
4. F. Yu and D. Gregory, "Optical pattern recognition: architectures and techniques," *Proc. of the IEEE* **84**(5), pp. 733–752, 1996.
5. O. Rioul and P. Duhamel, "Fast algorithms for discrete and continuous wavelet transforms," *IEEE Trans. on Information Theory* **38**, pp. 569–586, Mar. 1992.

6. I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl.* **4**(3), pp. 247–269, 1998.
7. T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW team, "A case for NOW (Networks of Workstations)," *IEEE Micro* , Feb. 1995.
8. *http://www.osc.edu/lam.html.*
9. J. Fridman and E. S. Manolakos, "On the scalability of 2-D discrete wavelet transform alogrithms," *Multidimensional Systems and Signal Processing* (8), pp. 185–217, 1997.
10. L. Yang and M. Misra, "Coarse-grained parallel algorithms for multi-dimensional wavelet transforms," *The Journal of Supercomputing* **12**(1/2), pp. 99–118, 1998.
11. O. Nielsen and M. Hegland, "TRCS9721: A scalable parallel 2D wavelet transform algorithm," tech. rep., The Australian National University, Dec. 1997.
12. R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson, "Effects of communication latency, overhead, and bandwidth in a cluster architecture," in *ICSA 24*, pp. 716–731, June 1997.
13. F. Kossentini, "Spatially segmented wavelet transform," tech. rep., UBC, 1998. ISOIEC JTC 1SC29WG1 WG1N868.
14. "Report on core experiment codeff1 "Complexity reduction of SSWT"," tech. rep., Motorola Australia, UBC, 1998. ISOIEC JTC 1SC29WG1 WG1N881.
15. P. P. Vaidyanathan and P.-Q. Hoang, "Lattice structures for optimal design and robust implementation of two-channel perfect-reconstruction QMF banks," *IEEE Trans. Acoust., Speech, Signal Processing* **36**, pp. 81–94, Jan. 1988.
16. P. P. Vaidyanathan, "Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial," *Proceedings of The IEEE* **78**, pp. 56–93, Jan. 1990.
17. T. G. Marshall, "Zero-phase filter bank and wavelet coder matrices: Properties, triangular decompositions, and a fast algorithm," *Multidimensional Systems and Signal Processing* **8**, pp. 71–88, 1997.
18. S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. on Patt. Anal. and Mach. Intell.* **11**(7), pp. 674–693, 1989.
19. G. Beylkin, R. Coifman, and V. Rokhlin, "Fast wavelet transforms and numerical algorithms I," *CPAM* **44**, pp. 141–183, 1991.
20. M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using the wavelet transform," *IEEE Trans. Image Proc.* **1**, pp. 205–220, Dec. 1992.
21. W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," in *Wavelet Applications in Signal and Image Processing III*, A. F. Laine and M. Unser, eds., pp. 68–79, Proc. SPIE 2569, 1995.
22. M. Vishwanath, "The recursive pyramid algorithm for the discrete wavelet transform," *IEEE Trans. on Signal Processing* **42**, Mar. 1994.
23. C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementations to mappings on SIMD array computers," *IEEE Trans. on Signal Processing* **43**, pp. 759–771, Mar. 1995.
24. C. Chrysafis and A. Ortega, "Line based, reduced memory, wavelet image compression," in *Proc. of Data Compression Conf.*, 1998.
25. G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, and I. Bolsens, "Optimal memory organization for scalable texture codecs in MPEG-4," *IEEE Trans. on Circuits and Systems for Video Technology* **9**, pp. 218–243, Mar. 1999.
26. W. Jiang and A. Ortega, "Discrete wavelet transform system architecture design using filterbank factorization," tech. rep., University of Southern California, Los Angeles, CA, USA, 1999.
27. M. Vetterli and D. L. Gall, "Perfect reconstruction FIR filter banks: Some properties and factorizations," *IEEE Trans. on Acoustic, Speech and Signal Processing* **37**, p. 1057, July 1989.
28. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1994.