Efficient Realizations of the Discrete and Continuous Wavelet Transforms: from Single Chip Implementations to Mappings on SIMD Array Computers

Chaitali Chakrabarti [1]

Department of Electrical Engineering
Telecommunications Research Center
Arizona State University
Tempe, AZ 85287-5706
chaitali@asu.edu

Mohan Vishwanath

Computer Science Lab
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
mohan@parc.xerox.com

Abstract

This paper presents a wide range of algorithms and architectures for computing the 1-D and 2-D Discrete Wavelet Transform (DWT), and the 1-D and 2-D Continuous Wavelet Transform (CWT). The algorithms and architectures presented here are independent of the size and nature of the wavelet function. New on-line algorithms are proposed for the DWT and the CWT which require significantly small storage. The proposed systolic array and the parallel filter architectures implement these on-line algorithms, and are optimal both with respect to area and time (under the word-serial model). Moreover, these architectures are very regular, and support single chip implementations in VLSI. The proposed SIMD architectures implement the existing pyramid and a'trous algorithms, and are optimal with respect to time.

[1] This work was supported in part by a grant from NSF, no. MIP-9309504.

1 Introduction

In the last few years there has been a great amount of interest in wavelet transforms, and the interest has been highly inter-disciplinary. The wavelet transform can be viewed as a decomposition of a signal in the time-scale plane. There are several types of wavelet transforms depending on the nature of the signal (continuous or discrete) and the nature of the time and scale parameters (continuous or discrete). In this paper we focus on the realizations of the Discrete Wavelet Transform (DWT) and the Continuous Wavelet Transform (CWT) as defined in [11]. According to this definition, the time and scale parameters as well as the input and output signals of both the transforms are discrete; the two transforms differ mainly in the manner in which the time-scale plane is tiled. The CWT and the DWT have a very wide range of applications: from signal analysis and signal coding to numerical analysis [4],[6],[8],[14]. The large application-domain of these transforms makes the study of their implementations in VLSI very important. In this paper we develop efficient algorithms and architectures for computing the 1-D and 2-D DWT, and the 1-D and 2-D CWT. The proposed implementations are *independent* of the size and nature of the wavelet. This feature makes them very attractive, since the wavelet function can be varied with the application requirements.

There exist only a handful of VLSI architectures for wavelet transforms. The first architecture for computing the DWT was designed by Knowles[5]. This architecture was not well suited for VLSI since it used large multiplexers for routing the intermediate results. Later, Lewis and Knowles[7] designed an architecture for computing the 2-D DWT with Daubechies 4-tap filter as the wavelet function. A drawback of their architecture was that it was heavily dependent on the properties of the specific wavelet, and did not work efficiently for other wavelets. Aware Inc. has developed a chip for DWT called the Wavelet Transform Processor (WTP) [1]. WTP essentially consists of a 4-tap filter and some external memory and relies on software for computing the DWT. Recently, Parhi and Nishitani have proposed folded architectures and digit-serial architectures for the 1-D DWT [10]. Their architectures require minimum number of registers, but do not scale easily with the filter size and the number of octaves computed. Earlier, we have proposed systolic architectures for computing the DWT which are optimal (asymptotically) in the word-serial model [17]. These architectures require $2N$ cycles (versus N cycles in the architectures proposed here) to compute N outputs. As far as the CWT is concerned, we are not aware of any other work on mapping the CWT into VLSI.

In this paper we propose a wide variety of algorithms and architectures for computing the 1-D and 2-D DWT, and the 1-D and 2-D CWT. A preliminary version of this paper appeared in [2]. The proposed architectures range from systolic arrays and parallel filters to SIMD arrays. We introduce new on-line algorithms for computing the DWT and the CWT. The systolic array and the parallel filter architectures implement these on-line algorithms for DWT and CWT in a word-serial manner. These architectures are very regular and support single-chip VLSI implementations. Moreover, these architectures are optimal under the word serial model, both with respect to the area required and the time taken. We also present parallel architectures (SIMD) for computing Mallat's pyramid algorithm[8] for the DWT, and the a'trous algorithm for the CWT[11]. These architectures process the signal very fast ($O(L)$), but have an area that is proportional to the size of the input. The specific contributions of this paper are listed below:

- Development of the Modified Recursive Pyramid Algorithm(MRPA) for parallel filters. This algorithm computes the N-point DWT in a 'running' fashion in N steps. The amount of storage required is $O(JLk)$, where J is the number of octaves, L is the size of the filter, and k is the precision.

- Development of the On-line CWT Algorithm (OCA) for computing the 1-D CWT in a 'running' manner on both systolic and parallel filters. This algorithm computes the N-input, NJ-output CWT in NJ steps (1-input every J cycles).

- Development of optimal (both area and time) architectures for the 1-D and 2-D DWT. These architectures are based on the MRPA, and support efficient VLSI implementations.

- Development of optimal (both area and time) architectures for the 1-D CWT. These architectures are based on the OCA, and are ideal for VLSI implementations.

- Development of SIMD architectures based on SIMD array and SIMD multigrid for both the DWT and CWT. These architectures implement the direct pyramid algorithm for the DWT, and the a'trous algorithm for the CWT, and are suitable for multiple chip implementations.

The rest of the paper is organized as follows. In Section 2, the wavelet transform is briefly described, followed by the lower bound results of [18]. Section 3 contains a brief introduction to the RPA, followed by descriptions of MRPA and OCA. Sections 4 and 5 describe the systolic array, the parallel filter, and the SIMD array architectures for 1-D and 2-D DWT, while Sections 6 and 7 describe the same set of architectures for 1-D and 2-D CWT.

2 Preliminaries

The Wavelet Transform (WT) of a signal $\hat{x}(t)$ is given by,

$$\hat{w}(u,s) = \int \hat{x}(t)\hat{h}(\frac{t-u}{s})dt$$

where $\hat{h}(\frac{t-u}{s})$ is the wavelet corresponding to the 'prototype' wavelet $\hat{h}(t)$ at scale s and time u. The Wavelet Transform of a sequence $x(i)$ (sampled version of the continuous signal $\hat{x}(t)$), discretized on a grid whose samples are arbitrarily spaced both in time b and scale a is given by [11]

$$W(b,a) = \frac{1}{\mid a \mid^{1/2}} \sum_{i=b}^{i=aL+b-1} x(i)h\left(\frac{i-b}{a}\right) \tag{1}$$

where L is the size of the support of the basic wavelet h, and h is obtained by sampling $\hat{h}(t)$. In addition, N is the number of input samples and J is the number of scales.

In this paper we focus on two special cases of the WT, the Discrete Wavelet Transform (DWT) and the Continuous Wavelet Transform (CWT).

DWT: The DWT can be viewed as the multiresolution representation of a sequence $x(n)$ with $N/2$ values at the highest resolution, $N/4$ values at the next resolution, and so on [8]. The structure of the DWT is due to the dyadic nature of its time-scale grid. Here $a = 2^k$, where k is an integer, and b is a multiple of a. If N is a power of 2, then $J \leq \log N$, and $1 \leq k \leq J$. At each scale k, the number of samples in the time dimension is $N/2^k$.

CWT: The CWT is defined as a WT with no decimation in time and scale (see equation 1). However, in this paper we consider a commonly used version of the CWT where the scale resolution is logarithmic as in the case of DWT [11]. In other words, $a = 2^k$ for $1 \leq k \leq J$, and $J \leq \log N$. At each scale k, the number of samples in the time dimension in N.

It is to be noted that the properties of the wavelet transform are heavily *dependent* on the properties of the basic wavelet. The architectures developed in this paper are *independent* of the wavelet function and are hence flexible.

2.1 Lower bounds

In this section we present our earlier results on lower bounds for computing the Wavelet Transforms [15],[18]. These bounds are for single chip implementations and are derived under the following practical spatial restrictions on the I/O protocol:

1. Unilocal: Each input/output bit is available at only one pad.

2. Place-determinate: I/O data are available at prespecified (instance-independent) places.

3. Word-local: For any cut l partitioning the chip, only a constant number of input (output) words have some bit entering (exiting) the chip on each side of l. That is except for maybe a small number of inputs (outputs) all the k bits of the inputs (outputs) enter (exit) the chip on either the left or the right side of the partition.

The results outlined below are for the 1-D case. All these bounds hold under the assumption that $J \leq \lfloor \log_2 \left(\frac{N}{L}\right) \rfloor + 1$. Bounds have also been derived for the case when J does not satisfy this condition in [15]. Let N be a power of 2. Then the area A and time T satisfy the following lower bounds. Here T is the computation delay or the period [2].

- For 1-D DWT, $AT^2 \geq (J^2 L^2 k^2)$.
- For 1-D DWT under the word-serial [3] model, $A \geq (JLk)$ and $T \geq N$.
- For 1-D CWT, $AT^2 \geq (N^2 L^2 k^2)$.
- For 1-D CWT under the word-serial model, $A \geq (NLk)$ and $T \geq NJ$.

Note that under the word-serial model, the period for an N-input N-output function is lower bounded by N. Also, under this model, the computation delay is lower bounded by the period. Lower bounds for the 2-D case can be obtained by simply replacing N by N^2 and L by L^2 in the bounds for the 1-D case (This is in contrast to the 1-D and 2-D DFT case, where a different technique had to be used to obtain the bounds for the 2-D DFT). A point to be noted is that for the 2-D case, the word-serial model does not place any restriction on the order of the inputs as long as they are input in a word-serial manner. However, in practical imaging systems, where the digital image is available in a raster scan format, an additional constraint has to be introduced in the derivation of the lower bounds. It is conjectured that for 2-D DWT, when the image is available in the raster-scan format, the area is bounded by $A \geq (NLk)$, and the time is bounded by $T \geq N^2$.

[2] The period is defined as the time between the initiation of two consecutive sets of computations.

[3] The word-serial model is one in which at any time instant, at most one input (output) word has some, but not all, of its bits already read (written).

The proposed systolic array and the parallel filter architectures for 1-D and 2-D DWT satisfy the word-serial model, and are optimal both in terms of area and time. The proposed systolic and parallel filter architectures for 1-D CWT are also optimal under the word-serial model for the case when only 1 output is computed every cycle.

3 On-line Algorithms

3.1 Recursive Pyramid Algorithm

The Recursive Pyramid Algorithm (RPA) is a running algorithm for computing 1-D DWT [16]. The RPA is not only efficient from a VLSI (single chip) point of view but it can also be implemented very efficiently on existing programmable DSP chips without any buffering. We give a brief description here for completeness. The DWT can be computed in an efficient manner on general purpose computers and SIMD machines using the pyramid algorithm (PA) developed by Mallat [8],[9] (see Figure 1). However, there are a number of applications of the DWT where a 'running' (or real time/on-line) implementation is desirable. Running implementations of the PA for an N-point DWT, requires either $O(N)$ storage or a cascade of $\log N$ computation units. Both these alternatives are expensive, especially since the input sequence in real-time applications is quasi-infinite. The Recursive Pyramid Algorithm (RPA) computes the N-point DWT in real time using a single computation unit and $L(\log N - 1)$ cells of storage, where L is the length of the wavelet filter (QMF) and generally $L \ll N$. The latency of the computation unit is assumed to be one time unit. The RPA consists of rearranging the order of the N outputs such that an output is scheduled at the 'earliest' instance that it can be scheduled. The earliest instance is based on the following precedence relation: if the earliest 'instance' of the ith octave clashes with that of the $(i+1)$th octave, then the ith octave gets scheduled first. In the resulting schedule, the first octave computations take place every other cycle, and the higher octave computations are scheduled between the first octave computations. Let $y_i(n)$ be the nth lowpass output of the ith octave. Then the output schedule generated by the RPA for $N = 16$ and $J = 4$ is as follows. $y_1(1), -, y_2(1), -, y_1(2), -, y_3(1), -, y_1(3), -, y_2(2), -, y_1(4), -, y_4(1), -, y_1(5), -, y_2(3), -, y_1(6), \ldots$. A '$-$' signifies that an output is not generated in that clock cycle.

3.2 Modified Recursive Pyramid Algorithm

The modified Recursive Pyramid Algorithm (MRPA) is a version of the RPA that is suitable for parallel filter and other implementations which have a latency that is greater than 1. The main difference with the RPA is that the earliest instance at which an octave output can be scheduled now depends not only on the scheduling times of the lower octaves, but also on the latency of the computation unit.

In the MRPA scheduling, the lower octaves are scheduled before the higher octaves, in order to avoid possible clashes. If the first output of any octave is scheduled such that there is no conflict with *any* of the lower octave outputs, then it is guaranteed that there will be no conflicts for all the outputs of that octave. We next derive a general formula for the time when the inputs for the ith octave computation are fed into the computation unit.

Let $off(k)$ be the additional delay in the scheduling of the first output of the kth octave that is caused by scheduling conflicts with octaves smaller than k, and let T be the latency of the computation unit. Then the inputs for the ith octave computation are fed at times

$$(i-1)T + \sum_{k=1}^{i} off(k) + 2^i m_i, \quad 0 \le m_i < \frac{N}{2^i}$$

Let $off(1) = 1$. The remaining $off(k)$s are calculated in the following way:
$off(k)$ is the smallest positive integer such that the time at which the first input for the kth octave is fed does not conflict with the times at which the inputs for the jth octave are fed, $1 \le j \le k-1$. In other words, for $1 \le j \le k-1$,

$$(k-1)T + \sum_{p=1}^{k} off(p) \ne (j-1)T + \sum_{p=1}^{j} off(p) + 2^j m_j, \quad 0 \le m_j < \frac{N}{2^j}$$

This implies that for $1 \le j \le k-1$,

$$off(k) \ne 2^j m_j - (k-j)T - \sum_{p=j+1}^{k-1} off(p), \quad 0 \le m_j < \frac{N}{2^j}$$

We explain this procedure with the help of the following example. Let the latency of the computation unit $T = 4$. Then $off(2)$ is the smallest positive integer such that $off(2) \ne 2m_1 - 4$. Thus $off(2) = 1$. Similarly, $off(3)$ is the smallest positive integer such that (i) $off(3) \ne 2m_1 - 9$ and (ii) $off(3) \ne 4m_2 - 4$. Thus $off(3) = 2$. In this example, the inputs to the first octave are fed at times $1 + 2m_1$, $0 \le m_1 < N/2$, the inputs to the second octave are fed at times $6 + 4m_2$, $0 \le m_2 < N/4$, the inputs to the third octave are fed at times $12 + 8m_3$, $0 \le m_3 < N/8$, and so on.

Figure 2 describes the corresponding timing diagram. The pseudo-code for the MRPA in included in Appendix A. In that implementation, the last dc term is not computed.

3.3 On-line CWT Algorithm

The On-line CWT Algorithm (OCA) is a running algorithm for computing 1-D CWT. Here the NJ outputs (N outputs per octave) are scheduled in such a way that the output of a particular octave is produced once every J cycles. Such a scheduling scheme requires that the input be fed at the rate of one every J cycles. As in the case of RPA and MRPA, the outputs are scheduled at the earliest possible instance. If the latency of the computation unit is 1, then there are no clashes, and the outputs of the various octaves are scheduled one after the other. For instance if $J = 4$, then the outputs are scheduled as follows: $y_1(1), y_2(1), y_3(1), y_4(1), y_1(2), y_2(2), y_3(2), y_4(2), y_1(3), \ldots$ and so on. If the latency of the computation unit is greater than 1, then the outputs have to be scheduled in a way such that in case of a clash, the lower output octaves have precedence over the higher octave outputs. Once the first output of any octave is scheduled such that there is no conflict with any of the lower octaves, the remaining outputs of that octave are scheduled every J cycles then on. In an example with $J = 4$ and latency $T = 4$, the outputs of the various octaves are scheduled as follows. $y_1(1), -, -, -, y_1(2), y_2(1), -, -, y_1(3), y_2(2), y_3(1), -, y_1(4), y_2(3), y_3(2), y_4(1), y_1(5), y_2(4),$ $y_3(3), y_4(2), y_1(6), \ldots$.

Like the RPA, the OCA is not only efficient from a VLSI (single chip) point of view but it can also be implemented very efficiently on existing programmable DSP chips without any buffering. Pseudo-code for the OCA, when the latency of the computation unit is 1, is included in Appendix B. In that implementation, the last dc term is not computed.

4 1-D Discrete Wavelet Transform

In this section we describe parallel filter, systolic array, SIMD array and SIMD multigrid architectures for 1-D DWT. The parallel filter and the systolic array architectures are optimal with respect to area and time under the word serial model. We consider both these architectures, since it allows the designer to trade-off (within a constant factor) between period, chip area and clocking frequency.

4.1 Parallel Filter Architecture

The parallel filter architecture for 1-D DWT implements the MRPA algorithm (see Section 3.2). The first octave outputs are computed every other cycle, and all higher octave outputs are computed between the two first octave output computations. The main components of this architecture are two parallel filters to compute the low pass and the high pass outputs, and a storage unit of size LJ to store the inputs that are required for the computation of the J octave outputs. Figure 3 gives the block diagram of an architecture for computing J octaves.

Parallel filters: Each parallel filter consists of L fixed multipliers and a tree of $(L-1)$ adders to add the products. The latency of the parallel filter is $T_m + T_a \log L$, where T_m is the time taken to do a multiplication, and T_a is the time taken to do an addition. The latency plays an important role in scheduling the octave outputs (see Section 3.2).

Storage unit: The storage unit consists of J serial-in parallel-out shift registers, each of length L. The ith shift register stores the inputs required for the ith octave computation $1 \leq i \leq J$. An output is written into the ith shift register every 2^{i-1} cycles, and all L elements of the shift register are fed to the parallel filters every 2^i cycles. Since the number of outputs of the ith octave is half the number of outputs of the $(i-1)$th octave, the ith shift register is clocked at half the rate of the $(i-1)$th shift register. The time when the first output is written into or read out of a shift register is determined by the MRPA scheduling. For instance, in the example when $L = 4$, the first octave outputs are written into the first shift register at time instants $5, 7, 9, 11, \ldots$, and fed to the parallel filter at time instants $6, 10, 14, 18, \ldots$. Control signals $\{c_i\}$ and $\{a_i\}$ are generated at appropriate times to AND the signals into or out of the shift register.

The hardware components of this architecture include $2L$ multipliers, $2(L-1)$ adders, JL storage units and a control unit to generate the appropriate control signals. The computation time as well as the period is $\approx N$. Note that the parallel filter architecture can be clocked at high sample rates by pipelining the multipliers and the adders. In such a case, the latency increases, resulting in a different MRPA scheduling. The size of the storage unit does not increase since the value of $off(k)$ is bounded by 2^{k-1}.

4.2 Systolic Architecture

Systolic architectures for computing the 1-D DWT which implement the RPA have been presented in [17]. We give a brief description of the architecture in [17] for the sake of comparisons. The architecture consists of a linear systolic array to compute both the low pass and the high pass outputs, and a storage unit to store the inputs for higher octave computations. The x-inputs that are required for the first octave computation are fed in alternate cycles to one end of the array, while the inputs that are required for the higher octave computations are fed in parallel from the storage unit. This architecture requires an area $O(LJk)$, and computes the DWT with a delay (and period) of $2N$ cycles, the first output appearing only 1 cycle after the first input. The architecture satisfies the word serial model and is optimal both with respect to area and time.

We next propose a modification of the systolic architecture of [17] which computes the DWT with a delay of only N cycles. The modified architecture consists of two linear arrays, one to compute the low pass outputs and the other to compute the high pass outputs. The x-inputs are first fed into the storage unit, and then loaded in parallel to the two systolic arrays. The order in which the outputs are computed here is the same as in the previous architecture. The area is also $O(LJk)$.

4.3 SIMD Linear Array

We propose an implementation of 1-D DWT on a reconfigurable SIMD architecture [3]. The architecture consists of a linear array of N processors where each processor contains a reconfigurable switch. If the switch is set to 1, the data passes through it without *any* delay. We refer to the processors which do multiply-add or which pass data to a neighboring processor with a delay of one cycle as 'active' processors. The reconfigurable switches can be used to reconfigure the N processor array to a smaller array of active processors. Since the configuration patterns are known apriori, they can be stored in each processor. A similar implementation has been proposed in [12].

In the proposed scheme, for the mth octave computation, $1 \leq m \leq J$, the N processor array is reconfigured to an array of size $\frac{N}{2^{m-1}}$ with processors $P(2^{m-1}j)$ being active, $0 \leq j < \lfloor \frac{N}{2^{m-1}} \rfloor$. Figure 4 describes the interconnection for the computation of 3 octaves on a 8 processor array. In this scheme, all the outputs of any particular octave are computed at the same time. The high pass and the low pass filter coefficients are broadcast to each processor. Each processor computes the products of the data elements and the filter coefficients, and updates the partial results of the low pass and the high pass outputs. The data element is then sent to its right neighbor.

The computation time for computing J octaves in this architecture is LJ and the period is also LJ. The area of each processor is $O(k)$ and the overall area complexity is $O(Nk)$.

4.4 SIMD multigrid

The 1-dimensional DWT of N points can be efficiently mapped into a multigrid architecture of size N. The multigrid architecture of size N consists of $(\log N + 1)$ levels, with $\frac{N}{2^i}$ processors in level i, $0 \leq i \leq \log N$. Figure 5 describes the multigrid architecture for $N = 8$. The mapping is as follows: the ith level of the multigrid computes the $(i+1)$th octave outputs (high pass and low pass) and sends the low pass outputs to level $(i+1)$, $0 \leq i < \log N$.

The procedure to compute the ith octave outputs is the same as that of the SIMD linear array. The high pass and the low pass filter coefficients are broadcast to each processor, and each processor updates the partial results of the low pass and the high pass outputs by the products of the data element and the corresponding filter coefficients. The computation time for computing J octaves in this architecture is $O(LJ)$, and the area is $O(Nk \log N)$. Since each level of the multigrid operates on a different set of inputs, the period of this design is only L, compared to LJ for the linear array architecture.

The period of the multigrid architecture can be reduced to 1 cycle by increasing the number of processors in each level by a factor of L. In the modified multigrid, level i consists of L sublevels with $\frac{N}{2^i}$ processors in each sublevel, $0 \leq i < \log N$. The interconnection among the processors between 2 levels as well as between sublevels is described in Figure 6. In each level, there are two types of processors, active processors described by label A, and inactive processors. Each active processor has a filter coefficient assigned to it. It computes the products of its input data with the assigned filter coefficients and updates the low-pass and the high-pass partial sums. It sends the partial sums to its active south neighbor and the input data to its inactive south-east neighbor. Each inactive processor passes the input data to its active south-east neighbor. This architecture has an area of $A = O(NLk)$. The computation time is LJ, and the period is 1.

4.5 Comparisons

Table 1 compares the number of multipliers, the asymptotic area and period for the various architectures for 1-D DWT. Note that both the systolic array and the parallel filter match the area and time lower bounds for 1-D DWT (in the word-serial model), and are thus optimal with respect to both area and time.

5 2-D Discrete Wavelet Transform

5.1 Parallel Filter Architecture

The parallel architecture for 2-D non-separable DWT implements the MRPA algorithm in 2-D. As in the 1-D case, the higher octave computations are interspersed between the first octave computations. Here the windows of the first octave computations are centered in even-numbered rows, while the windows for the higher octave computations are centered in odd-numbered rows. For instance, the first octave computation windows are centered in rows $0, 2, 4, 6, 8, 10, \ldots$, the second octave computation windows are centered in rows $1, 5, 8, 13, \ldots$, the third octave computation windows are centered in rows $1, 9, 17, 25, \ldots$, and so on. Along odd-numbered rows, the higher octave computations are interspersed in the same way as in 1-D DWT. However, since the filter is of size $L \times L$, the delay in the parallel filter is different, and consequently, the time when the first output of any octave is computed is different.

The proposed architecture consists of two programmable parallel filters (one filter per two bands) and the storage unit to store the lowest band outputs required for higher octave computation. Figure 7 describes the block diagram of this architecture.

Parallel Filters: Each parallel filter consists of L^2 programmable multipliers and a tree of $(L^2 - 1)$ adders to add the products. Each filter computes the outputs of two bands. The outputs of the lowest band are stored in the storage unit and are used for higher octave computation. The latency of the parallel filter is $T_m + 2T_a \log L$, where T_m and T_a are the multiplication and addition times respectively.

Storage Unit: The storage unit consists of J serial-in parallel-out shift register units, the ith unit storing the inputs required for ith octave computation, $1 \leq i \leq J$. The ith unit consists of a 2-D array of storage cells of size $L \times \frac{N}{2^{i-1}}$. New data are shifted into the last row of each unit, and when the last row is filled, the data shift up by one row in the array. Each shift register unit has two clocks associated with it: a row clock and a column clock. The ith unit row clock has a period, $t_r(i)$ that is 2^{i-1} times the sample period t_s, $t_r(i) = 2^{i-1}t_s$, and has a duty cycle that is 2^{1-i} times that of the sample clock. The ith unit column clock has a period $t_c(i)$, that is N times $t_r(i)$, $t_c(i) = Nt_r(i)$, and a duty cycle of 2^{1-i}. Data is written into a shift register unit when both the row clock and the column clock are high. L^2 data are read out of the unit at time instants when every second row clock and every second column clock are high. The time when the first output of any octave is written into or read out of the shift register unit is determined by the MRPA algorithm in a way similar to that of 1-D DWT. Control signals are generated at appropriate times to AND the data into and out of the shift register unit.

The hardware components of this architecture include $2L^2$ multipliers, $2(L^2-1)$ adders, NL storage cells, and a control unit. The computation time as well as the period is $\approx N^2$.

5.2 Systolic-Parallel Architecture

A systolic-parallel architecture for computing the 2-D separable DWT has been presented in [17]. We give a brief description here for the sake of comparisons. This architecture computes the 2-D pyramid algorithm using the RPA. It consists of a systolic filter, a parallel filter, and a bank of registers. It requires area proportional to NLk and computes the 2-D DWT with a delay (and period) of N^2+N cycles, the first output appearing 1 cycle after the first input.

5.3 SIMD 2-D array

The proposed implementation consists of a 2-D array of $(N \times N)$ SIMD processors which are reconfigured to form smaller arrays of active processors for higher octave computations. Specifically, the array is reconfigured to form an array of size $(\frac{N}{2^{m-1}} \times \frac{N}{2^{m-1}})$ for mth octave computation, $1 \leq m \leq J$. Figure 8 describes the interconnection during the computation of the first three octaves on an (8×8) array.

In this scheme, all the outputs of a particular octave are computed at the same time. One possible way of computing the outputs of a particular octave is to have the pixels move and the partial results stay. Then every alternate active processor needs to participate in updating the partial results. The other possible way is to have the pixels stay and the partial results move. In this case computation should be initiated in every other active processor. The area of this architecture is $O(N^2k)$. The computation time as well as the period is L^2J.

5.4 Comparisons

Table 2 compares the number of multipliers, the asymptotic area and period for the various architectures for 2-D DWT. Note that both the systolic-parallel architecture and the parallel filter match the area and time lower bounds for 2-D DWT (in the word-serial model with raster-scan input), and are thus optimal with respect to both area and time. Note that the systolic-parallel architecture handles the separable case, while the parallel filter architecture handles the non-separable case.

6 1-D Continuous Wavelet Transform

In this section we describe architectures for computing 1-D CWT which are based on the reorganized "a trous" computational structure of [11]. Figure 9 describes such an implementation. In this implementation there are 2^{j-1} cells at each octave, with each cell working at the rate of $1/2^{j-1}$th that of the cell of the first octave. This is because the inputs to the jth octave cell are subsampled by 2^{j-1} compared to the original input.

6.1 Parallel Filter and Systolic Array Architectures

In this subsection we map the algorithm of [11] into systolic and parallel filter architectures. We first describe architectures where all the elementary cells at the jth octave are combined to form a single computation unit. In such a realization, J outputs are computed simultaneously. Next, we describe word-serial architectures which have only 1 computation unit, and compute only 1 output at a time.

6.1.1 J outputs are computed every cycle:

In this case, both the systolic array and the parallel filter architectures consist of J computation units; each unit consists of one low pass and one high pass filter. The output of the low pass filter of the jth computation unit is sent to the $(j+1)$th computation unit.

Parallel Filter: In the parallel filter architecture, the inputs to the jth computation unit have to be subsampled by a factor of 2^{j-1}. This is achieved by storing the data in a delay line of size $2^{j-1}L$, and tapping the delay line every 2^{j-1} delay units. Figure 10 describes the architecture when the filter size is $L = 4$.

The hardware components of the jth computation unit include $2L$ multipliers, $2(L-1)$ adders and $2^{j-1}(L-1)$ storage cells. Thus the total number of multipliers, adders and storage units in this design are $2JL$, $2J(L-1)$ and $(2^J-1)(L-1)$ respectively. The period is $\approx N$.

Systolic array: In this architecture, each computation unit consists of two linear systolic arrays (one for the high pass computation and the other for the low pass computation). In the systolic arrays, the inputs are broadcast to all the processors, the weights stay in the processors, and the partial results get updated as they move from one processor to another in the array. In the jth unit, subsampling by a factor of 2^{j-1} is obtained by storing the partial results in registers of size 2^{j-1} in between consecutive processors. Figure 11 describes the systolic array architecture when

$L = 4$.

The total number of multipliers, adders and storage cells in this design are $2JL$, $2J(L-1)$ and $2(2^J-1)(L-1)$ respectively. Thus the systolic array architecture has twice as many storage cells as the parallel filter architecture. The period is N.

6.1.2 1 output is computed every cycle:

In this case, both the parallel filter and the systolic array architectures consist of only one computation unit. Thus the outputs of J octaves are now computed by the same unit. The outputs are scheduled by the on-line CWT algorithm OCA (see Section 3.3).

The block diagram of this architecture is described in Figure 12a. Both the parallel filter and the systolic array architectures consist of a computation unit to compute the low pass and the high pass filter outputs, and a storage unit to store the inputs required for the computation of the octave outputs. The storage unit consists of J subunits of storage cells; the first subunit stores L x-inputs, the second subunit stores $2L$ y_1-outputs, the third subunit stores $4L$ y_2-outputs and so on. The number of cells in the jth subunit is $2^{j-1}L$, since the inputs for the computation of the jth octave are subsampled by a factor of 2^{j-1}. All the cells of the storage unit are clocked uniformly (in contrast to the storage unit for 1-D DWT). A data is read into the storage unit every cycle. In fact, each of the J subunits reads in a data every J cycles. L data are read out from the storage unit every cycle. The way the data are read out is different for the parallel filter and the systolic array architectures. Figure 12b describes the storage unit for the case when $L = 4$. Control signals $\{a_k\}$ and $\{c_{k,l}\}$ are generated at appropriate times by a finite state machine to read in(out) data to(from) the storage unit.

Parallel filter: In this architecture, all L elements of the same subunit are fed to the low pass and the high pass filters at the same time. Thus for the computation of an jth octave output, control signals $c_{j,4}$, $c_{j,3}$, $c_{j,2}$ and $c_{j,1}$ are set to 1, and L elements of the jth subunit are fed to the filters. Note that these L elements are 2^{j-1} apart in the jth subunit.

Systolic array: In this architecture, the weights stay in the processors, the inputs are fed in parallel, and the partial results get updated as they move from one processor to another in the array. This computation scheme demands that the inputs for the computation of a particular output are fed to the array in a skewed fashion. Also, L different outputs are computed at the same time in the array, and the L elements that are loaded in parallel from the storage unit come from J different subunits (if $L \geq J$) or L different subunits (if $L < J$). In an example where $L = J = 4$, if at time $t = T$, processors $P(4)$, $P(3)$, $P(2)$, and $P(1)$ are scheduled to compute the partial results of $y_1(3)$, $y_4(2)$, $y_3(2)$ and $y_2(2)$, then the control signals $c_{1,4}$, $c_{4,3}$, $c_{3,2}$, and $c_{2,1}$ are set to 1, and $L = 4$ data are read in from the storage unit.

Area and Time Complexities: The hardware components for both the architectures include $2L$ multipliers, $2(L-1)$ adders and $(2^J - 1)L$ storage cells. The period is $\approx NJ$ for parallel filter and NJ for the systolic array architecture. Thus both the architectures are optimal with respect to both area and time in the word serial model.

6.2 SIMD linear array

The proposed SIMD architecture consists of a linear array of N processors which can be reconfigured to form arrays of sizes $N/2$, $N/4$, and so on. The reconfiguration pattern is different from the one described in Section 3.3 for 1-dimensional DWT. For the 1st octave computation, the linear array interconnection is used, and the computation time is L. For the 2nd octave computation, the N processor array is reconfigured to form two $N/2$ processor arrays, once with processors $P(2j)$, and once with processors $P(2j+1)$, $0 \leq j < \lfloor N/2 \rfloor$. The computation time is now $2L$. In general, for the $(k+1)$th octave computation, the N processor array is reconfigured to form a set of 2^k arrays, each of size $N/2^k$. The mth processor array in this set is formed with processors $P(2^k j + m - 1)$, where $0 \leq k < \log N$, $0 \leq j < \lfloor N/2^k \rfloor$, and $0 \leq m < 2^k$. Figure 13 describes the reconfiguration pattern when $N = 8$. The total computation time of this design is $L(1+2+\ldots+2^{J-1}) = L(2^J - 1)$. The period of this architecture is also $L(2^J - 1)$. The area complexity is $A = O(Nk)$.

6.3 Comparisons

Table 3 compares the number of multipliers, the asymptotic area and period for the various architectures for 1-D CWT. Note that both the systolic array and the parallel filter for the case when 1 output is computed per cycle, match the area and time lower bounds for 1-D CWT (in the word-serial model), and are thus optimal with respect to both area and time.

7 2-D Continuous Wavelet Transform

In this section we describe systolic array and parallel filter architectures for 2-D CWT for the case when J outputs are computed every cycle. Both the architectures are extensions of the ones that we proposed for 1-D CWT. We assume that the input is fed in the line scan mode, and that the output is also produced in the line scan mode. The proposed architecture for computing J octaves consists of J computation units; each unit consists of one low pass and one high pass filter. The output of the low pass filter of the jth unit is sent to the $(j+1)$th unit, $1 \leq j < J$.

7.1 Parallel Filter Architecture

In the parallel filter architecture, each parallel filter consists of L^2 multipliers grouped in L subunits with L multipliers and $L-1$ adders per subunit. The outputs of each of the subunits is added by a tree of adders. The inputs to the jth filter unit have to be subsampled along the rows as well as along the columns by a factor of 2^{j-1}. Subsampling along the rows is achieved by storing the data in a delay line of size $2^{j-1}L$, and tapping the delay line every 2^{j-1} delay units. Since the inputs are fed in the line scan mode, subsampling along the columns is obtained by storing the data in a delay line of size $2^{j-1}LN$, and tapping the delay line every $2^{j-1}N$ delay units. Figure 14 describes the architecture when the filter size is (4×4). As in the architecture for 1-dimensional CWT, each filter unit is active all the time. The hardware components of the jth computation unit of the parallel filter architecture include $2L^2$ multipliers, $2L^2-1$ adders, and $2^{j-1}(N+L)(L-1)$ storage units. Thus the total number of multipliers, adders and storage units are $2JL^2$, $2J(L^2-1)$ and $(2^J-1)(N+L)(L-1)$ respectively. The area is $O(N^2Lk)$ and the period is $\approx N^2$.

7.2 Systolic array

In the systolic array architecture, each filter consists of L^2 processors, grouped in L subunits with a linear array of L processors in each subunit. The outputs of the subunits are added by a row of adders (see Figure 15). Subsampling along the rows is obtained by storing the partial results in 2^{j-1} storage cells in between consecutive processors in each subunit. Subsampling along the columns is obtained by storing the partial results in $2^{j-1}N$ storage cells in between two subunits. Figure 15 describes the systolic array architecture when the filter size is (4×4). The total number of multipliers, adders and storage units of the systolic array architecture are $2JL^2$, $2J(L^2-1)$ and $2(2^J-1)(N+L)(L-1)$ respectively. The area is $O(N^2Lk)$ and the period is N^2.

7.3 2-D SIMD array

The proposed SIMD architecture consists of a 2-D array of $(N \times N)$ processors which can be reconfigured to form arrays of sizes $(N/2 \times N/2)$, $(N/4 \times N/4)$, and so on. The reconfiguration pattern is a 2-D extension of the one used for SIMD linear array (see Section 6.2). For the $(k+1)$th octave computation, the $(N \times N)$ processor array is reconfigured to form a set of 4^k arrays, each of size $N/2^{k-1} \times N/2^{k-1}$. The total computation time as well as the period of the 2-D SIMD array architecture is $L^2(1+4+\ldots+4^{J-1}) = L^2(4^J-1)/3$. The area is $A = O(N^2k)$.

8 Comparisons

Table 4 compares the number of multipliers, and the asymptotic area and time complexities for the various 2-D CWT architectures.

9 Conclusion

In this paper, we have presented a wide range of algorithms and architectures for computing the 1-D and 2-D DWT and the 1-D and 2-D CWT. We have presented two on-line algorithms, the MRPA and the OCA, for implementing the 1-D DWT and the 1-D CWT respectively. The systolic and parallel filter architectures implement these on-line algorithms, and are optimal both with respect to area and time. The 1-D DWT architectures are highly amenable to single chip design, since they require area that is independent of the length of the 1-D input sequence. The systolic and the parallel filter architectures for the 2-D DWT can also fit on one chip (even though the storage required depends on the square-root of the input size). We have also presented SIMD array architectures for computing the DWT and CWT. These architectures have an area that is proportional to size of the input sequence, but they are very fast ($T = O(LJ)$ or $O(L)$).

The architectures proposed in this paper are independent of the size and nature of the wavelet. Further reductions in the hardware complexity can be achieved by exploiting the property of the specific wavelet. For instance, for orthonormal wavelets, the number of multipliers in these implementations can be reduced by a factor of 2 by using polyphase decomposition techniques of [13].

References

[1] Aware, Inc., Cambridge, MA. *Aware Wavelet Transform Processor (WTP) Preliminary*, 1991.

[2] C. Chakrabarti, M. Vishwanath, and R.M. Owens. Architectures for wavelet transforms. *Proc of the VLSI Signal Processing Workshop*, pages 507–515, 1993.

[3] H. Li, and Q. F. Stout. Reconfigurable SIMD massively parallel computers. *Proceedings of the IEEE*, 79(4):429-443, 1991.

[4] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Trans. Info. Theory*, 36(5):961–1005, Sept 1990.

[5] G. Knowles. VLSI architecture for the discrete wavelet transform. *Elec. Letters*, 26(15):1184–1185, Jul 1990.

[6] R. Kronland-Martinet, J. Morlet, and A. Grossmann. Analysis of sound patterns through wavelet transforms. *Intl. J. of Patt. Recog. and Artif. Intell*, 1(2):273–302, 1987.

[7] A.S. Lewis and G. Knowles. VLSI architecture for 2-d daubechies wavelet transform without multipliers. *Elec. Letters*, 27(2):171–173, Jan 1991.

[8] S. Mallat. Multifrequency channel decompositions of images and wavelet models. *IEEE Trans. Acoustics Speech and Sig. Proc*, 37(12):2091–2110, Dec 1989.

[9] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Analysis and Mach. Intell*, 11(7):674–693, July 1989.

[10] K. K. Parhi and T. Nishitani. VLSI architectures for discrete wavelet transforms. *IEEE Trans on VLSI Systems*, 1:191–202, 1993.

[11] O. Rioul and P. Duhamel. Fast algorithms for discrete and continuous wavelet transforms. *IEEE Trans. on Information Theory*, 38(2):569–586, Mar 1992.

[12] G. Sullivan. *Vector and parallel implementations of the wavelet transform*, MS thesis, University of Illinois at Urbana-Champaign, 1991.

[13] P.P. Vaidyanathan. Quadrature Mirror Filter Banks, M-Band Extensions and Perfect-Reconstruction Techniques. *IEEE ASSP Magazine*, 4–20, July 1987.

[14] M. Vetterli. Wavelets and filter banks for discrete time signal processing. In R. Coifman et al., editors, *Wavelets and their Applications*. Jones and Barlett, 1991.

[15] M. Vishwanath. *Time-Frequency Distributions: Complexity, Algorithms and Architectures*. PhD thesis, Pennsylvania State University, May 1993.

[16] M. Vishwanath. The recursive pyramid algorithm for the discrete wavelet transform, *IEEE Trans. on Signal Proc.*, 42(3):673-676, Mar 1994.

[17] M. Vishwanath, R.M. Owens, and M.J. Irwin. VLSI architectures for the discrete wavelet transform. Accepted for publication in the IEEE Trans. Circuits and Systems.

[18] M. Vishwanath, R.M. Owens, and M.J. Irwin. The computational complexity of time-frequency distributions. *Proc. of the the Sixth SP Workshop on Statistical Signal and Array Processing*, Oct 1992.

Appendix A

The pseudo-code for the Modified Recursive Pyramid Algorithm (MRPA) for the 1-D DWT is as follows.

```
begin{MRPA}
  input: X(0,i)=x(i),  i:[1,N]                      /* N is a power of 2. */

   /* Calculate the values of off(k).                                      */

  off(1)=1
  for (i=2 to J)
     off(i) is the smallest positive integer such that the following
     conditions are true
     for (j=1 to i-1)
```

$$off(i) \neq 2^j m_j - (i-j)T - \sum_{p=j+1}^{i-1} off(p)$$

```
  /* Calculate the offset a(j) in the recursion to determine the correct
     octave in rdwt(i,j)  when i is even                                   */

     for (i=2 to J)
```

$$Z = (i-1)T + \sum_{k=1}^{i} off(k)$$

```
        for (j=2 to i-1)
            Z=(Z-a(j))/2
        a(i)=Z-2

  for(i=1 to N-1)                              /* Once for each output  */
     rdwt(i,1)
end{MRPA}
```

```
rdwt(i,j)
begin{rdwt}
  if(i is odd)
    k=(i+1)/2

   /* Compute output number k of octave j.  This is computed using the  */
   /* last L outputs of octave (j-1).                                    */

    sumL=0, sumH=0

    for(m=0 to L-1)
       sumL=sumL+X(j-1,i-m)*w[m]
       sumH=sumH+X(j-1,i-m)*h[m]
    X(j,k)=sumL                      /* Low pass output */
    y(j,k)=sumH                     /* High pass output */

  else
    rdwt((i-a(j))/2,j+1)       /*  Recursion to determine correct octave */
end{rdwt}
```

Appendix B

The pseudo-code for the On-line CWT Algorithm (OCA) for the 1-D CWT, when the latency of the computation unit is one, is as follows.

```
begin{OCA}
  input: X(0,i)=x(i),  i:[1,N]  /* N is a power of 2.                  */
                                /* For ease of notation the "0"th octave  */
                                /* lowpass output, X(0,i), is set to x(i) */


  for(i=1 to N-1)               /* For each input                       */
     for(k=1 to J)              /* If we have J octaves, then for each   */
        cwt(k,i)                /* input, there are J outputs.           */
end{OCA}


cwt(l,m)
begin{cwt}
   /* Compute output number m of octave l.  This is computed using the   */
   /* L outputs of octave (l-1), where the L outputs are spaced 2**(l-1) */
   /* from each other.  Note that the m{th} o/p of octave l is y(l,m)    */
   /* and 2**(l) is 2 to the power l.  The o/ps of the low-pass filter   */
   /* (intermediate variables) are the X's.  The filters have L coeffs.  */
    sumL=0
    sumH=0
    for(p=0 to L-1)
      sumL=sumL+ X(l-1,m-(p*2**(l-1))) * lowp[p]   /* lowp[], highp[]    */
      sumH=sumH+ X(l-1,m-(p*2**(l-1))) * highp[p]  /* are the filters    */

    X(l,m)=sumL;                                   /* Low pass output */
    y(l,m)=sumH;                                   /* High pass output */
end{cwt}
```

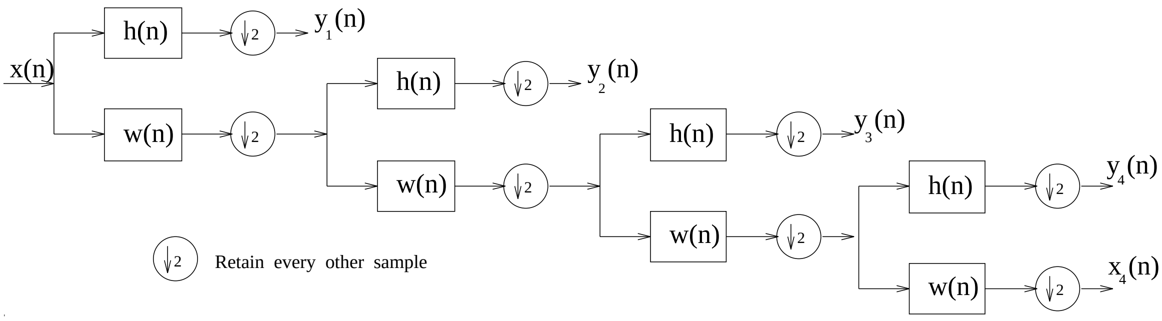

Figure 1: The DWT Pyramid. $N = 16$ and $J = 4$

Time instants	1	3	5		7	9		11		13		15		17		19		21		23		25
x inputs are fed at	x	x	x		x	x		x		x		x		x		x		x		x		x
y_1 is computed at			x		x	x		x		x		x		x		x		x		x		x
y_1 inputs are fed at				x			x				x				x				x			
y_2 is computed at							x				x				x				x			
y_2 inputs are fed at									x								x					
y_3 is computed at													x								x	

Figure 2: Timing diagram illustrating the time instants at which the 1st, 2nd, and 3rd octaves are scheduled using MRPA. The latency of the computation unit is $T = 4$.

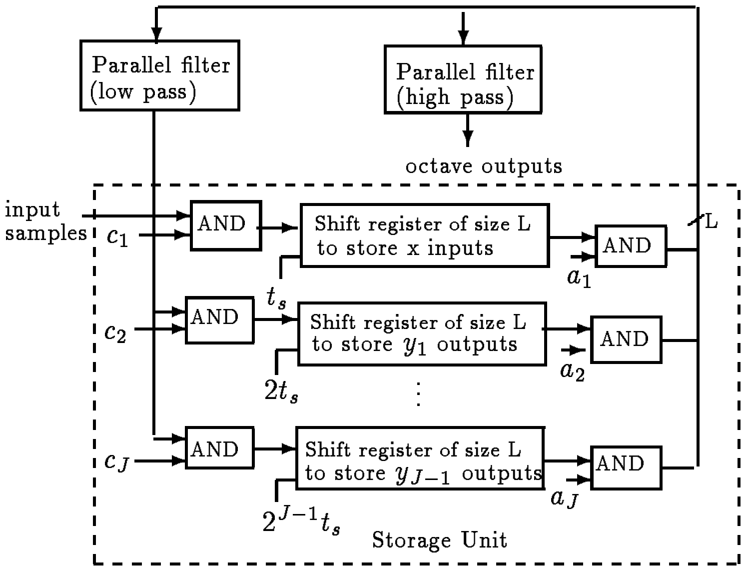

Figure 3: Block diagram of the parallel filter architecture for 1-D DWT

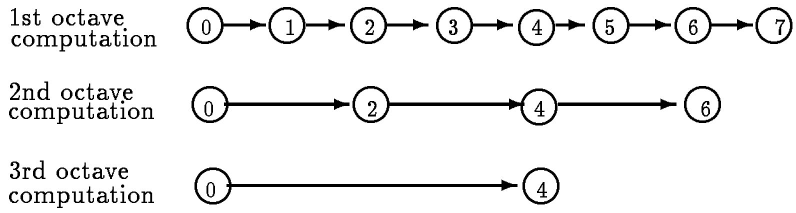

Figure 4: Interconnection among processors in a SIMD linear array for 1-D DWT computation, $N = 8$.

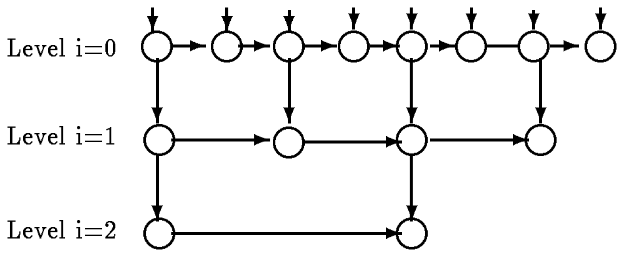

Figure 5: Multigrid architecture for $N = 8$

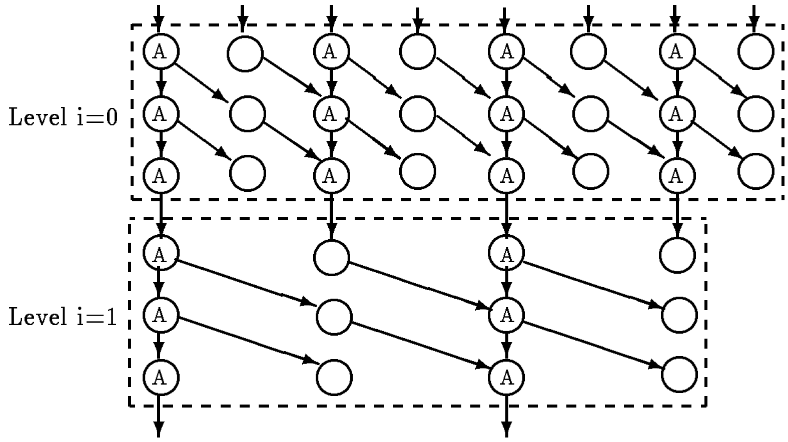

Figure 6: Modified multigrid architecture for computing 1-D DWT. $N = 8$ and $L = 3$.

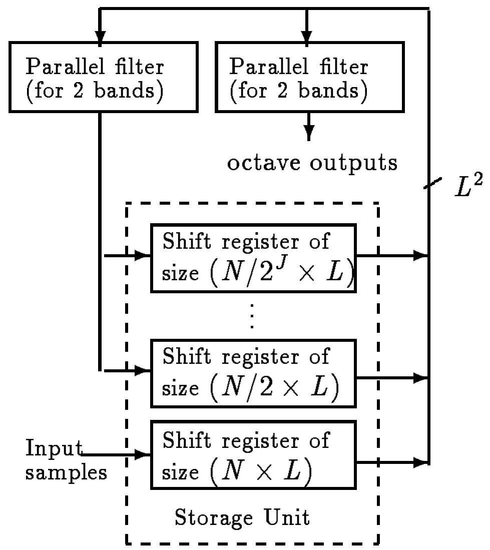

Figure 7: Block diagram of parallel filter architecture for 2-D DWT

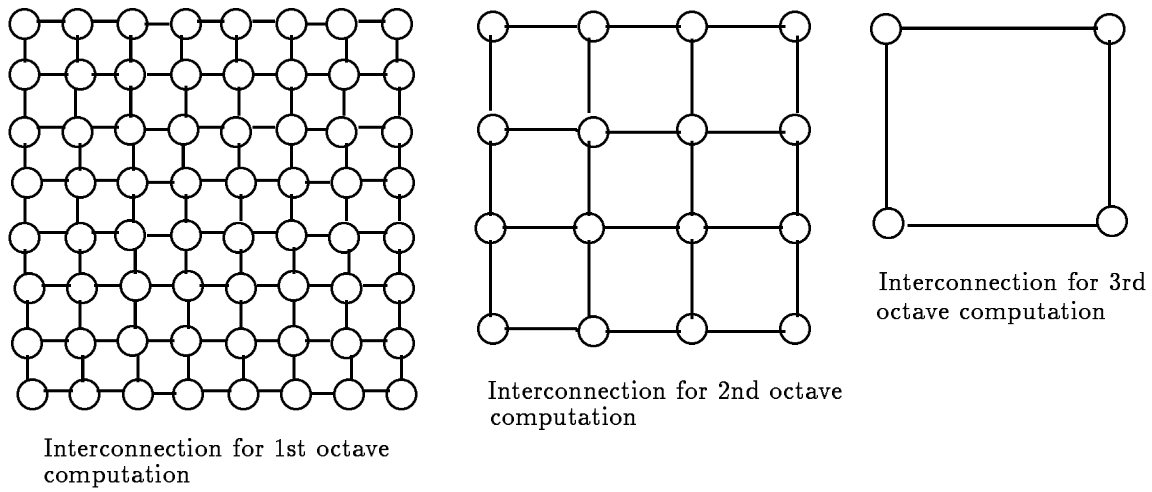

Figure 8: Interconnection among processors in a 2-D SIMD array for 2-D DWT computation

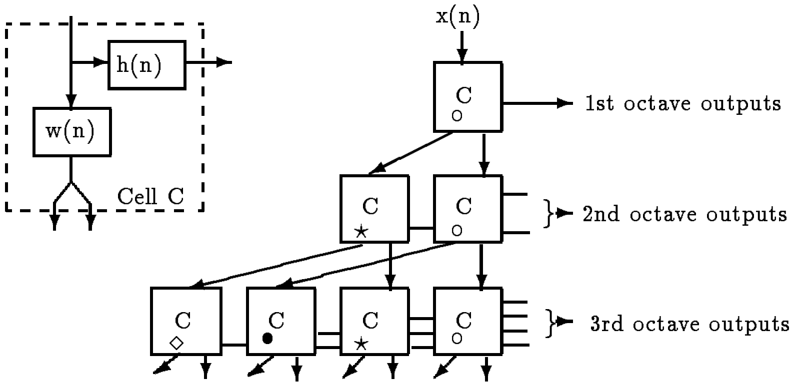

Figure 9: Reorganized 'atrous' computational structure for 1-D CWT [11]

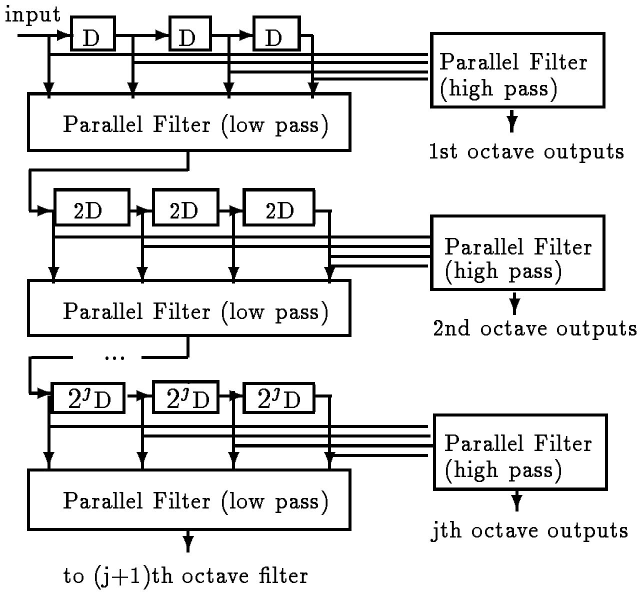

Figure 10: Parallel filter architecture for 1-D CWT when J outputs are computed every cycle. 'D' stands for unit delay. The filter size $L = 4$

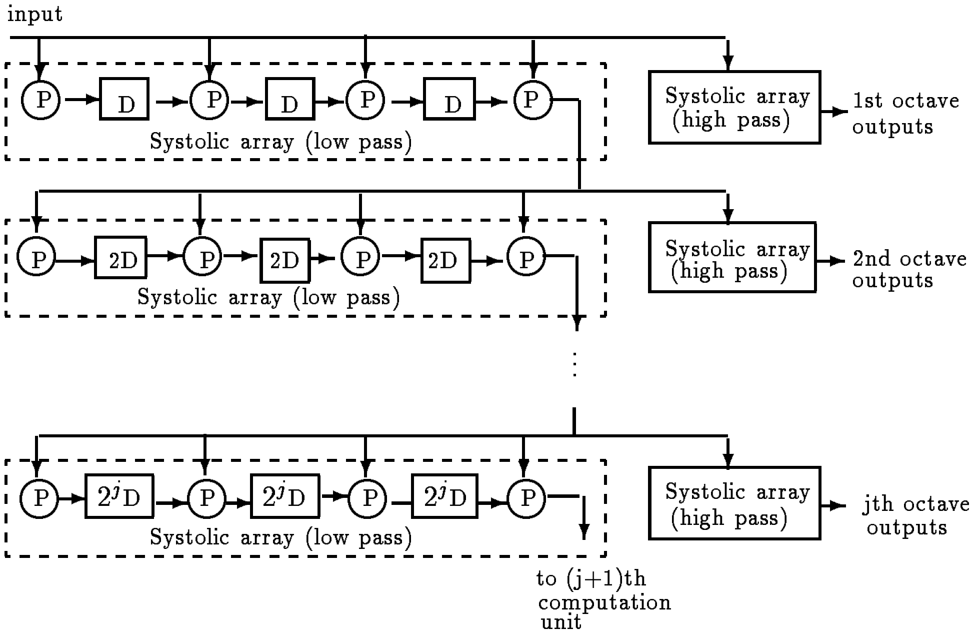

Figure 11: Systolic array architecture for 1-D CWT when J outputs are computed every cycle. 'P' stands for a processor, which is essentially a multiplier-accumulator. The filter size $L = 4$

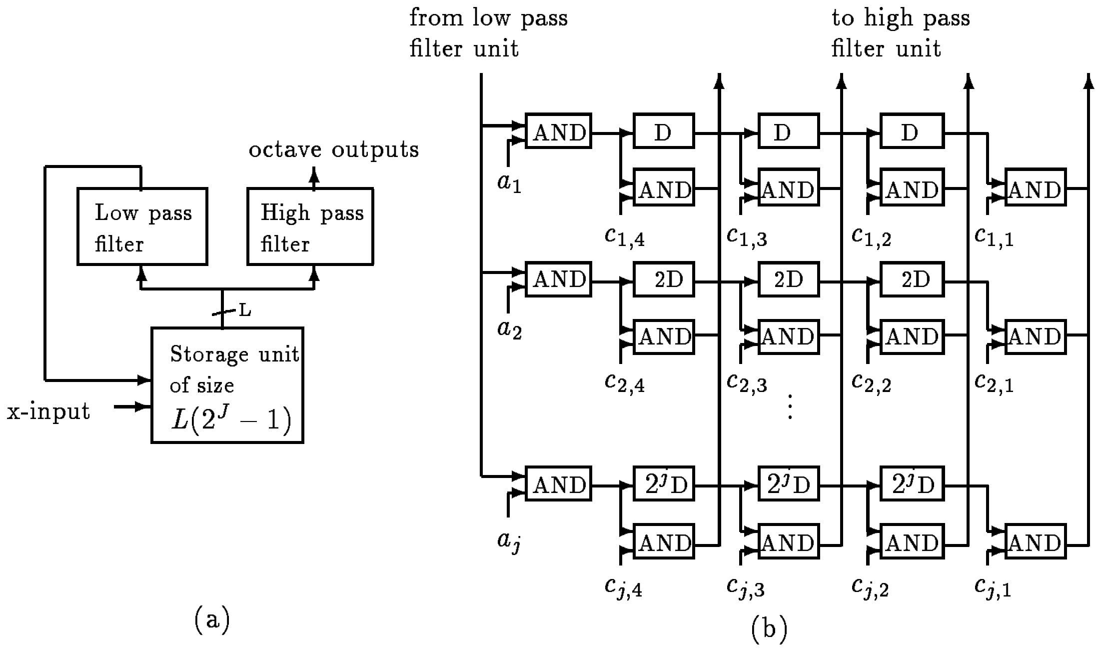

Figure 12: (a) Block diagram of the architecture for 1-D CWT, when only 1 output is computed every cycle. (b) Storage unit of the architecture when $L = 4$

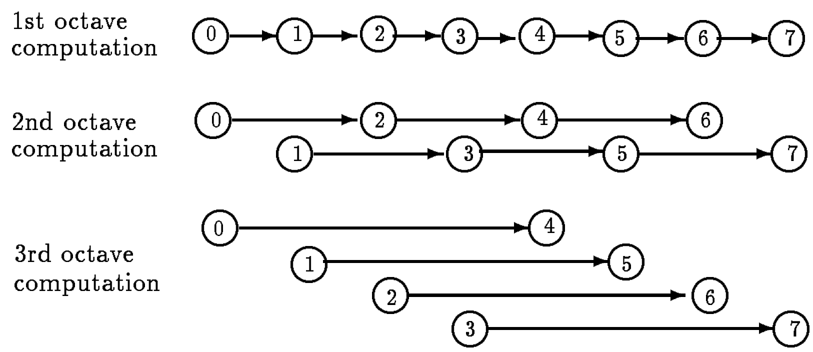

Figure 13: Interconnection among processors in a SIMD linear array for computing 1-D CWT

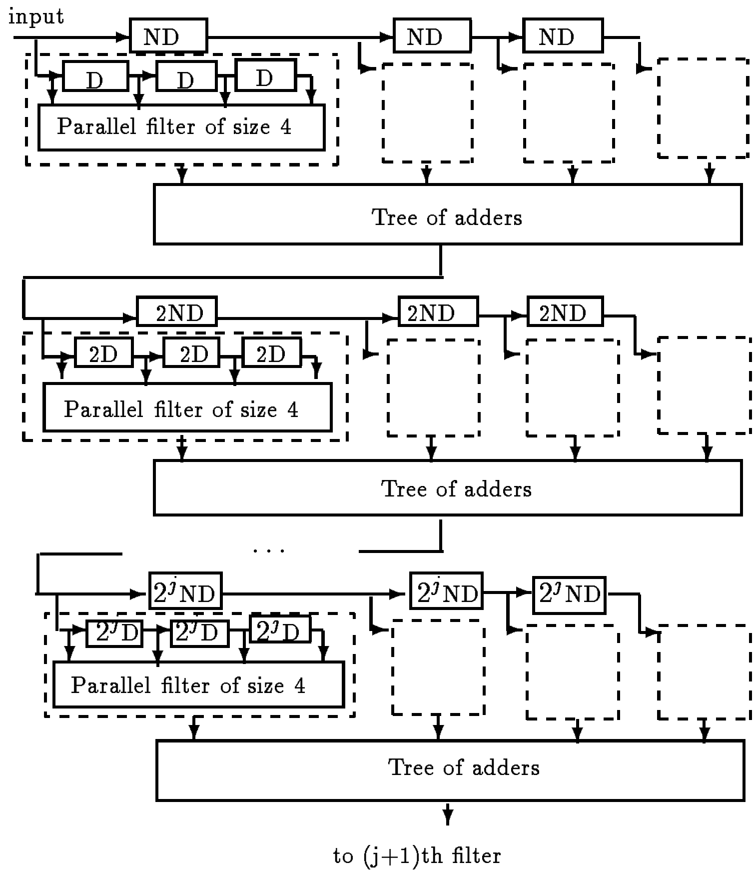

Figure 14: Parallel filter architecture for 2-D CWT when the filter is of size (4×4). The high pass filters have not been shown.

Architectures	*Multipliers*	*Area*	*Period*
Aware's WTP	4	$O(Nk)$	$O(N \log N)$
Knowles's arch	–	$O(Lk \log N(\log L + \log\log N))$	$O(N)$
Systolic array	L	$O(LJk)$	$2N$
Parallel filter	$2L$	$O(LJk)$	$\approx N$
SIMD linear array	$2N$	$O(Nk)$	LJ
SIMD multigrid	$2NJ$	$O(NJk)$	L

Table 1: Comparison of area and time complexities of 1-D DWT architectures

Architectures	*Multipliers*	*Area*	*Period*
Systolic-Parallel	$2L$	$O(NLk)$	$N^2 + N$
Parallel filter	$2L^2$	$O(NLk)$	$\approx N^2$
SIMD 2-D array	$2N^2$	$O(N^2k)$	L^2J

Table 2: Comparison of area and time complexities of 2-D DWT architectures

Architectures	*Multipliers*	*Area*	*Period*
Parallel filter 1 (J outputs/cycle)	$2JL$	$O(NLk)$	$\approx N$
Systolic array 1 (J outputs/cycle)	$2JL$	$O(NLk)$	N
Parallel filter 2 (1 output/cycle)	$2L$	$O(NLk)$	$\approx NJ$
Systolic array 2 (1 output/cycle)	$2L$	$O(NLk)$	NJ
SIMD linear array	$2N$	$O(Nk)$	$L(2^J - 1)$

Table 3: Comparison of area and time complexities of 1-D CWT architectures

Architectures	*Multipliers*	*Area*	*Period*
Parallel filter	$2JL^2$	$O(N^2Lk)$	$\approx N^2$
Systolic	$2JL^2$	$O(N^2Lk)$	N^2
SIMD 2-D array	$2N^2$	$O(N^2k)$	$L^2(4^J - 1)/3)$

Table 4: Comparison of area and time complexities of 2-D CWT architectures

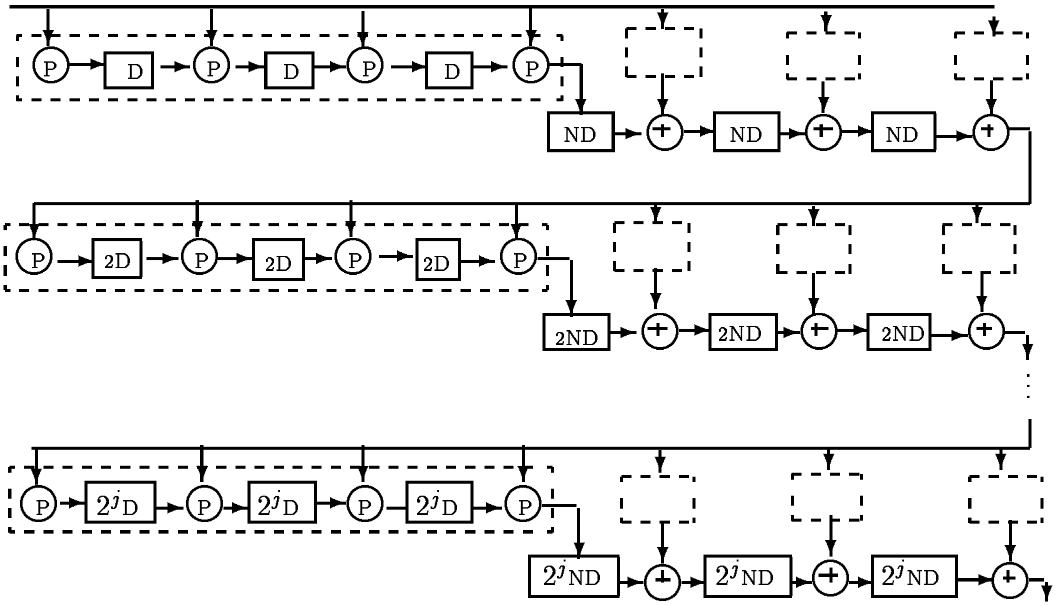

Figure 15: Systolic array architecture for 2-D CWT when the filter size is (4×4). The high pass filters have not been shown.