TABLE I
COMPARISONS OF FSA, TSS, MTSS, AND DSWA / IS WITH FOUR CIF FORMAT SEQUENCES ("CLAIRE," "MISSA," "SALESMAN," AND "SWING")

BMA		Displaced Frame Difference (DFD)	Entropy of DFD (bits/pcl)	PSNR (dB)	Average Search Positions (per frame)
$w = 7$	FSA	220 640	3.334	39.97	91 125
	TSS	224 952	3.358	39.74	10 125
	MTSS	222 277	3.341	39.83	11 793
	DSWA/IS	223 210	3.345	39.76	6626
$w = 21$	FSA	243 246	3.484	38.22	748 845
	TSS	256 106	3.541	37.14	16 605
	MTSS	249 363	3.511	37.70	18 106
	DSWA/IS	253 342	3.530	37.38	9282

TABLE II
COMPARISONS OF FSA, TSS, MTSS, AND DSWA / IS ALGORITHMS WITH "BALL," "LADY," "TRAIN," AND "BIRD"—FOUR CCIR 601 FORMAT SEQUENCES

BMA		Displaced Frame Difference (DFD)	Entropy of DFD (bits/pcl)	PSNR (dB)	Average Search Positions (per frame)
$w = 7$	FSA	2 039 845	4.766	31.12	295 924
	TSS	2 278 935	4.863	29.67	33 750
	MTSS	2 091 261	4.794	30.77	42 513
	DSWA/IS	2 125 939	4.809	30.57	24 506
$w = 21$	FSA	2 584 641	5.045	28.98	2 397 136
	TSS	3 366 908	5.337	26.18	55 350
	MTSS	3 089 561	5.241	27.14	76 351
	DSWA/IS	3 336 509	5.323	26.45	41 904

technique. In four CIF and four CCIR 601 format video sequences, each of 30 frames is used as the test picture. All entries in Tables I and II represent the average results using a total of 120 frames. The motion-vector search is based on the luminance component with the search block of 16×16 pels. Table I compares the average performances of four CIF images for maximum motion displacements of $w = 7$ and 21. As can be seen, DSWA/IS is slightly superior to TSS in performance, but the computations are evidently reduced about 35% and 44% for $w = 7$ and 21, respectively. The experiment with the CCIR 601 pictures is shown in Table II. It also shows that DSWA/IS is more efficient than TSS. Referring to our experimental results, it can be concluded that DSWA/IS, which adequately adjusts the search-window size in every stage, can perform better than TSS at a lower computational cost.

V. CONCLUSIONS

A new search algorithm DSWA/IS is proposed in this paper. This algorithm shows lower DFD and has a 24–44% savings in computations over TSS. Our experimental results also demonstrate the DSWA/IS is better than TSS, even if the search window size is large (43×43 pels). Hence, it is concluded that this algorithm can be applied to a wide range of applications such as video phone, video conference, and HDTV.

REFERENCES

[1] K. M. Yang, M. T. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, no. 10, pp. 1317–1325, Oct. 1989.
[2] T. Koga, K. Iinuma, A. Hirano, Y. Iinuma, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proc. Nat. Telecommun. Conf.* (New Orleans, LA, Nov. 29–Dec. 3, 1981), pp. G5.3.1–5.3.5.
[3] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Trans. Commun.*, vol. 38, pp. 950–953, July 1990.
[4] S. C. Kwatra, C.-M. Lin, and W. A. Whyte, "An adaptive algorithm for motion compensated color image coding," *IEEE Trans. Commun.*, vol. COM-35, pp. 747–754, July 1987.

Interleaved Pipeline Structures for Two-Dimensional Recursive Filtering

Tongxin Lu and Mahmood R. Azimi-Sadjadi

Abstract—**This paper presents new parallel and pipeline structures for real-time 2-D recursive filtering. For general scalar 2-D recursive filters, a 2-D multiple-interleaved pipeline architecture is introduced that is compatible with the nature of the image-scanning scheme. Using this new structure, the sampling period can be a fraction of the time needed for one scalar addition operation and the delay is only a few samples. In addition, this structure does not need any I / O buffers for real-time implementation.**

I. INTRODUCTION

The need for high-speed digital image processing became evident with the increasing utilization of the relevant techniques in medical, geophysical, and military applications. Many of these applications require acquisition, processing, and display of images in fractions of a second. A number of approaches have been suggested to achieve high-speed processing for 2-D recursive filtering using systolic array processors [1]–[11]. The "look-ahead computing" [4] and the "interleaved pipeline" [6] schemes have been proposed by several authors, mainly for 1-D filters. With look-ahead computing, an output sample can be computed without the need to use the results of several previous output samples. By interleaving such pipelines, several output samples can be computed concurrently.

In this paper, new parallel and pipeline architectures are introduced which can be used to perform high-speed 2-D scalar recursive-filtering operations. In Section II, a pipeline architecture for canonical implementation of 2-D recursive filters is given that requires a pixel-processing period of only one single-scalar multiplication time plus one single-scalar addition time. The delay is one multiplication time plus two addition times. By interleaving multiple look-ahead pipelines in parallel form with look-ahead computing [4], [6], [8], [9], a multiple look-ahead architecture is developed. If a quadruple look-ahead pipeline is used, the sampling period can be only one single addition time, and the delay from receiving an input pixel to delivering the corresponding output pixel can be one scalar multiplication plus two addition times. If a higher sampling rate is desired, more than four look-ahead architectures can be used. Section III gives concluding remarks and discussions on the proposed architectures.

Manuscript received February 12, 1991; revised August 27, 1992. This paper was recommended by Associate Editor Peter Pirsch.
The authors are with the Department of Electrical Engineering, Colorado State University, Fort Collins, CO 80523.
IEEE Log Number 9204662.

II. Look-Ahead Computing and Multiple-Interleaved Pipeline Structure

In this section, a high concurrent structure for 2-D recursive filtering is introduced. The conventional direct implementation of the 2-D recursive filtering is first briefly reviewed. Unlike block implementation [7]–[9], scalar implementation does not provide massive parallelism. However, the 2-D block-implementation scheme introduces a delay proportional to the number of rows in a block and hence becomes incompatible with the row-scanning process. This inevitable delay is undesirable, particularly for real-time applications. To overcome these deficiencies a multiple-interleaved look-ahead implementation scheme is introduced which is ideally suited for real-time image-filtering applications.

A. Direct Pipeline and Systolic Implementation

Consider a linear time-invariant causal 2-D recursive filter described by the following 2-D difference equation;

$$y(i,j) = \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} a_{p,q} x(i-p, j-q) + \sum_{\substack{p=0\\(p,q)\neq(0,0)}}^{N-1} \sum_{q=0}^{N-1} b_{p,q} y(i-p, j-q) \quad (1)$$

where $\{x(i,j)\}$ and $\{y(i,j)\}$ represent the input and the output arrays, respectively, and $a_{p,q}$ and $b_{p,q}$ are the coefficients of the $N \times N$ order 2-D recursive filter. The regions of support for a third-order filter are shown in Fig. 1.

Fig. 2 shows the signal flow for a third-order ($N = 3$) 2-D filter in a canonical form. In this figure, z_2^{-1} corresponds to the delay between pixels along a row and z_1^{-1} corresponds to the delay between adjacent rows during the row-scanning process. This signal flow is ideally suited for pipeline and systolic implementations. There are nine nodes in the pipeline path of this figure at which the partial sums of products are generated and propagated from left to right. The inputs to the right three nodes are $x(i,j)$ and $y(i,j-1)$, and to the left six nodes are $x(i,j-1)$ and $y(i,j-1)$. At the present moment, the three right-most nodes are working on the output pixels along the ith row, the three nodes in the middle are calculating the partial sums for the output pixels along row $i+1$, and the left-most three nodes are working on partial sums for the outputs of the row $i+2$. For example, while the right-most node performs the operations for the output $y(i,j)$, the left-most node is generating the first partial sum for the future output pixel $y(i+2, j+1)$.

The advantage of this pipeline structure over the parallel block-implementation scheme [7]–[9] is that it provides the minimum latency, which is one multiplication time plus two addition times. The latency is defined as the time interval elapsed from the time $x(i,j)$ is applied to the time $y(i,j)$ is generated. In addition, this direct scalar implementation is compatible with the raster scanning of images, while for the parallel vector-implementation schemes, such as block implementation, the scanning sequence is not followed and as a result some extra latencies are commonly introduced. The minimum achievable sampling period for this single pipelined structure is limited to only one multiplication period plus one addition period. This can be achieved by interleaving the multiplication and addition operations for the term $\{+a_{oo}x(i,j)\}$ at one addition time in advance of those needed for $\{+b_{01}y(i,j-1)\}$. The achievable sampling

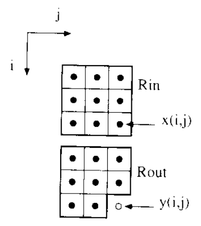

Fig. 1. Support regions for the quarter-plane causal recursive filter.

speed can, of course, be much higher for the parallel block implementation. Therefore, it is important to search for other concurrent algorithms and structures which cannot only provide the minimum latency benefit of this conventional pipelined implementation but also the high-speed capability of the block-implementation scheme. This objective can be met by using multiple-interleaved look-ahead pipeline structures, which are discussed next.

B. Look-Ahead Computing

By recursive application of (1) and substituting $y(i,j-1), y(i,j-2), \ldots, y(i,j-L+1)$ into the expression for $y(i,j)$, $y(i,j)$ can be calculated without the need for output pixels $y(i,j-L+1)$ to $y(i,i-1)$. Consequently, all L output pixels, $y(i,j-L+1), \ldots, y(i,j-1)$ and $y(i,j)$ can be calculated independently and concurrently. It can be shown that

$$\begin{aligned} y(i,j) = &\sum_{(p,q)=(1,0)}^{N-1} \sum^{L+N-1} d_{p,q} y(i-p, j-q) \\ &+ \sum_{(p,q)=(0,0)}^{N-1} \sum^{L+N-1} c_{p,q} x(i-p, j-q) \\ &+ \sum_{q=L}^{L+N-1} d_{o,q} y(i, j-q) \end{aligned} \quad (2)$$

where L is the look-ahead factor and $c_{p,q}$'s and $d_{p,q}$'s are coefficients obtained using

$$c_{p,q} = [0 \quad 0 \quad \cdots \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 & \cdots & \\ -b_{o,1} & 1 & 0 & & \\ -b_{o,2} & -b_{o,1} & 1 & & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ -b_{o,L-1} & \cdots & \cdots & -b_{o,1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} a_{p,q-L+1} \\ \vdots \\ a_{p,q-1} \\ a_{p,q} \end{bmatrix} \quad (3a)$$

$$d_{p,q} = [0 \quad 0 \quad \cdots \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -b_{o,1} & 1 & 0 & \cdots & 0 \\ -b_{o,2} & -b_{o,1} & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ -b_{o,L-1} & \cdots & \cdots & -b_{o,1} & 1 \end{bmatrix}^{-1} \begin{bmatrix} b_{p,q-L+1} \\ \vdots \\ b_{p,q-1} \\ b_{p,q} \end{bmatrix}. \quad (3b)$$

Note that $a_{i,j}$ and $b_{i,j}$ are set to zero if the (i,j) pair is outside the region of support of the filter. Using (2), the computations of the L-successive output pixels, $y(i,j-L+1), \ldots, y(i,j)$, can be performed simultaneously in L pipelines with the time difference of the $1/L$ pipeline clock cycle. The region of support

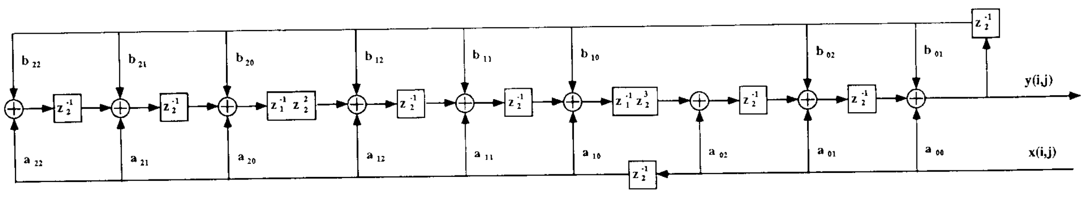

Fig. 2. Signal flow of the 2-D recursive filter.

for (2) and the corresponding signal flow graph are shown in Figs. 3 and 4, respectively.

C. Multiple-Interleaved Pipeline

In this section, the multiple-interleaved look-ahead pipeline structures and in particular the quadruple-interleaved pipeline structure are introduced. As shown in Fig. 4, the calculation of $y(i, j)$ does not depend on output pixels $y(i, j - 1)$, $y(i, j - 2)$, and $y(i, j - 3)$. This implies that any output pixel can be calculated without the three past nearest neighbor output pixels along the same row. Consequently, the four output pixels $y(i, j - 3)$, $y(i, j - 2)$, $y(i, j - 1)$, and $y(i, j)$ can be calculated concurrently using four identical structures in an interleaved fashion. Though the speed of each pipeline structure does not change, the overall sampling speed is improved by a factor of 4. Fig. 5 shows the signal flow of one of the four identical structures. In this figure, the horizontal delay is represented by $z_2'^{-1}$ which is the delay of a single pipeline and is four times the overall sampling period, i.e., $z_2'^{-1} = z_2^{-4}$. Also, $z_1'^{-1} = z_1^{-1}$ is used to represent the corresponding line delay. This delay can be implemented using arrays of linear shift registers or line memories. In this signal flow, $y(i, j - 5)$, $y(i, j - 6)$, and $y(i, j - 7)$ are assumed to be available in the appropriate pipeline periods. These are generated using similar computing structures which are not shown in Fig. 5. Fig. 6 shows the data flow of the last pipeline in Fig. 5, which is shown in the dashed-line block. The blocks marked A, B, C, and D are latches clocked by four interleaved clocks A, B, C, and D, as shown in the timing diagram of Fig. 7. As shown in this timing diagram, in one pipeline period the relevant multiplications and additions at each pipeline node are performed, and the result is added to the previous partial sum generated at other nodes along the pipeline. Thus, the speed at which these partial sums are collected is only one addition time.

Fig. 8 shows the overall configuration of the quadruple-interleaved pipeline structure with four identical pipelines as denoted by A, B, C, and D. Note that the assignment of clocks in Figs. 6 and 7 correspond to pipeline A in this figure. Pipeline A generates $y(i, j - 4)$ at the last pipeline period. Similarly, $y(i, j - 5)$ is generated from pipeline D and $y(i, j - 6)$ and $y(i, j - 7)$ are generated from pipelines C and B, respectively. In this structure, multiplications and additions for the local partial sums of products are performed outside the pipeline path. Therefore, as soon as a partial sum in the next node is available, the pipeline is able to collect all the partial sums along the entire pipeline path at the speed of a single addition time for each node. As shown in Figs. 6 and 7, the paths from $y(i, j\text{-}4)$ to $y(i, j)$ require only one multiplication plus two additions. This determines the latency and the sampling period of a single pipeline. However, because there are four identical pipelines in the interleaved fashion and each of them generates an output pixel in every time interval of one multiplication plus two addition operations, the overall sampling period is one-fourth of the

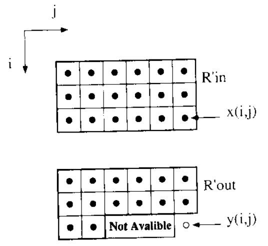

Fig. 3. Support regions for the look-ahead structure.

single pipeline period. Assuming that a multiplication takes twice the time as an addition, then the overall sampling period using the quadruple pipeline structure may be a single addition time. However, this sampling period is not the minimum achievable sampling period for the multiple-interleaved pipeline implementation. If more interleaved pipelines are used, the sampling period may be even less than an addition operation time. Note that in this case the data flow in Fig. 6 needs to be rearranged because the sampling speed exceeds the pipeline data-propagation speed. However, the arrangement of the pipeline is quite flexible.

III. Conclusion and Discussion

The interleaved look-ahead pipeline in this paper provides an extremely high-speed (both in sampling period and latency) tool for 2-D scalar recursive filtering. This is achieved by combining the look-ahead computing, pipelining, and interleaving schemes. In addition, its time complexity and processing delay are no longer functions of the filter order. With today's integrated-circuit technology, such a special-purpose structure can be directly connected to the image-acquisition system for real-time 2-D recursive-filtering operations. Using the available integrated-circuit chips [1], the quadruple-interleaved look-ahead pipeline architecture can process 1024 × 1024 size images directly at the rate of 30 frames/s independent of the filter order. However, 1024 × 1024 is not an upper bound of a possible sampling rate by the multiple-interleaved look-ahead pipelines. Let us consider the same integrated-circuit chips as listed in [1] with the following specifications:

One multiplication time:	$T_M = 45$ ns
One addition time:	$T_A = 19$ ns
One latch time:	$T_L = 6$ to 9 ns.

The quadruple look-ahead pipelines can be used to process 1024 × 1024 images at a rate of 30 frames/s. Using 16 interleaved pipelines, 2048 × 2048 images can be processed at a rate of 30 frames/s without I/O buffering.

The look-ahead hardware cost for 2-D recursive filtering can be obtained in terms of the number of required basic computa-

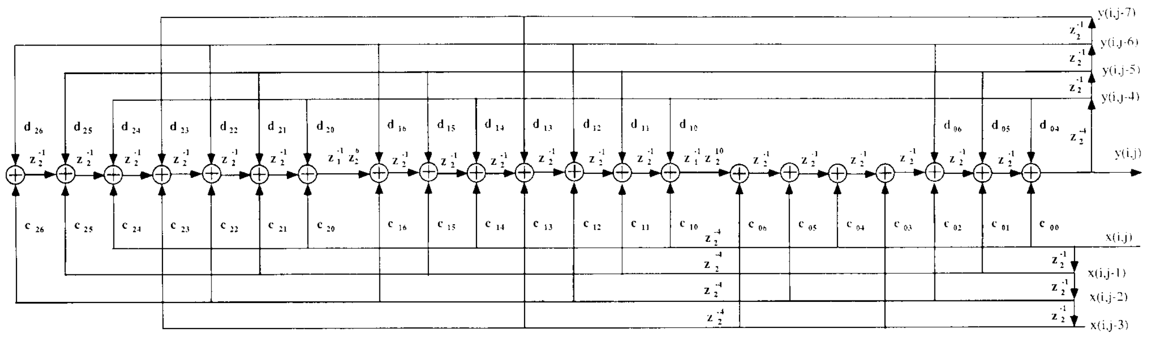

Fig. 4. Signal flow of the look-ahead structure.

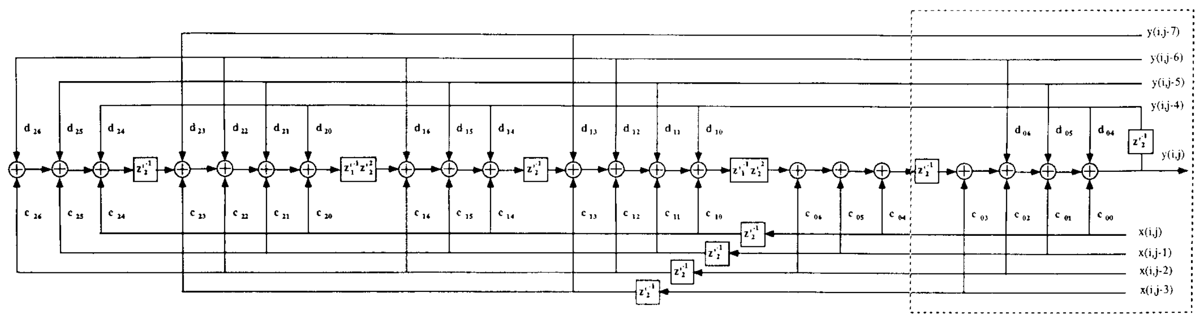

Fig. 5. Signal flow of the look-ahead structure with interleaving.

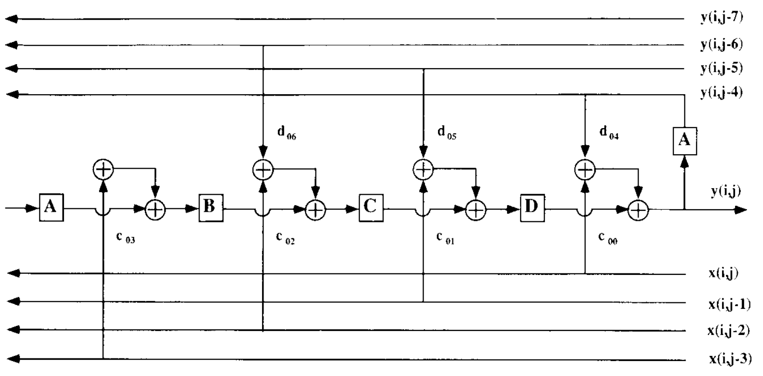

Fig. 6. Data flow of the last pipeline node in Fig. 5.

TABLE I
VALUES OF N_{CU} FOR TYPICAL VALUES OF L AND N

L	Image size (30 frames/s)	Filter order ($N \times N$)			
		2×2	3×3	4×4	8×8
1	512×512	7	17	31	127
2	512×512	20	44	76	284
4	1024×1024	64	128	208	688
8	1024×1024	224	376	640	1856
16	2048×2048	832	1472	2176	5632

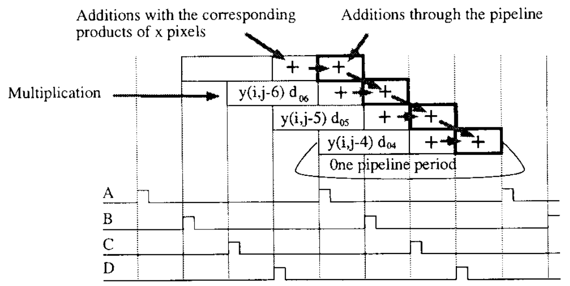
Additions with the corresponding products of x pixels
Additions through the pipeline
Multiplication
y(i,j-6) d06
y(i,j-5) d05
y(i,j-4) d04
One pipeline period
A
B
C
D

Fig. 7. Timing diagram of Pipeline A in the quadruple-interleaved pipeline structure.

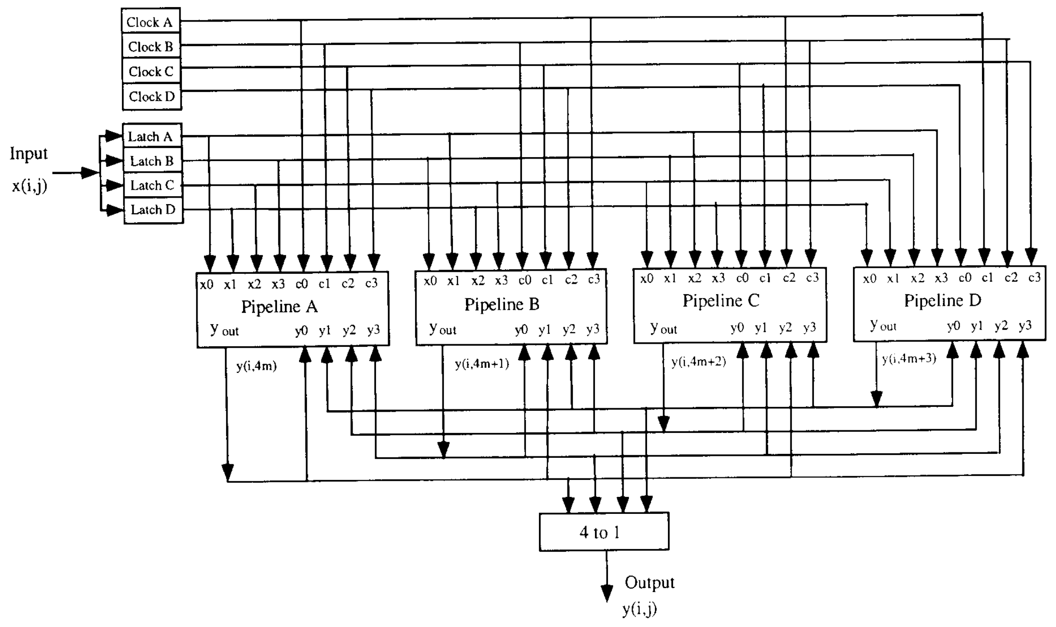

Fig. 8. The quadruple-interleaved pipeline structure.

tional units. Each computational unit is composed of one scalar multiplier. The following equation gives the expression for the number of computational units needed as a function of the filter order N and the look-ahead factor L:

$$N_{CU} = [2N^2 - 1 + (L - 1)(2N - 1)]L$$

where N_{CU} is the number of basic computational units. A list of the values of N_{CU} for some typical values of L and N is given in Table I. As a consequence, the interleaved pipelined provides an efficient architecture for moderate-size images and filter orders.

References

[1] A. N. Venetsanopoulos, K. M. Ty, and A. C. P. Loui, "High-speed architectures for digital image processing," *IEEE Trans. Circuits Syst.*, vol. CAS-34, no. 8, pp. 887–896, Aug. 1987.

[2] T. Aboulnasr and W. Steenart, "Real-time array processor for 2-D spatial filtering," *IEEE Trans. Circuits Syst.*, vol. 35, no. 4, pp. 451–455, Apr. 1988.

[3] R. A. Cohen, J. W. Woods, M. Sanya, and J. F. McDonald, "A video rate architecture for a fully recursive two-dimensional filter," in *Proc. ICASSP* (Dallas, TX, Apr. 1987), pp. 1973–1976.

[4] K. K. Parhi and D. G. Messerschmitt, "Look-ahead computation: Improving bound in linear recursions," in *Proc. ICASSP* (Dallas, TX, Apr. 1987), pp. 1855–1858.

[5] ——, "Concurrent architectures for two-dimensional recursive digital filtering," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 813–829, June 1989.

[6] ——, "Pipeline interleaving and parallelism in recursive digital filters—Parts I and II," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1099–1134, July 1989.

[7] M. R. Azimi-Sadjadi and A. R. Rostampour, "Parallel and pipeline architectures for 2-D block processing," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 443–448, Mar. 1989.

[8] T. Lu, "Implementations, algorithms and architectures for 2-D recursive filtering," M.S. thesis, Dept. Elec. Eng., Colorado State Univ., Fort Collins, 1988.

[9] T. Lu, M. R. Azimi-Sadjadi, and A. R. Rostampour, "Skew-pipeline and interleaved pipeline structures for 2-D recursive filtering," *Proc. ICASSP'90* (Albuquerque, NM, Apr. 1990), pp. 1037–1040.

[10] S. Sunder, F. El-Guibaly, and A. Antoniou, "Systolic implementations of two-dimensional recursive digital filters," in *Proc. of ISCAS* (New Orleans, LA, May 1990), pp. 1034–1037.

[11] N. R. Shanbhag, "An improved systolic architecture for 2-D digital filters," *IEEE Trans. Signal Proc.*, vol. 39, no. 5, pp. 1195–1202, May 1991.