Parallel and Pipeline Architectures for 2-D Block Processing

M. R. AZIMI-SADJADI AND A. R. ROSTAMPOUR

Abstract —**This paper is concerned with the development and design of parallel and pipeline architectures for 2-D recursive and nonrecursive block digital filter. In this regard, several high speed structures using single-instruction multiple-data stream (SIMD) machines have been developed. These structures are designed based upon the specific nature of the block convolution processor, block recursive processor and block state-space processor both at the block and scalar levels.**

I. Introduction

VLSI architectures are being increasingly used in real-time signal processing applications. High-speed performance can be gained in VLSI structures when the algorithms possess concurrent characteristic. In contrast to non-recursive or FIR filters which inherently benefit from this concurrency and can efficiently be implemented with high parallelism, recursive or IIR filters can only be implemented in sequential manner. This obviously imposes an upper limit on the speed of the operation and in addition limits the effective use of large number of processing elements (PE's). Lu, Lee and Messerschmitt [1] have proposed several systolic architectures for block implemented 1-D FIR and IIR digital filters which offer considerably high sampling and throughput rates as compared with the single processing element. However, in order to utilize such advantages the state matrix in the block state-space equation should be transformed to a triangular or a quasi-triangular form using unitary or orthogonal similarity transformations. Parhi and Messerschmitt [2], introduced a look-ahead computation scheme for parallel implementation of the block state-space equation. This technique utilizes pipelining at bit level and allows higher sampling rate. They have also proposed an incremental block state-space structure [3] which offers lesser computational complexities when compared to the standard and parallel block state-space structures [4].

In this paper, several parallel and pipeline architectures for 2-D block implemented FIR and IIR digital filters described by block convolution sum, block recursive equation and block state-space formulations [5], [6] are presented. These structures make use of a class of parallel computers which is known as single-instruction multiple-data stream (SIMD) machine [7], [8]. This type of machine is particularly useful when similar operations on a large set of data are to be performed.

II. Implementation of FIR Filters Using SIMD Architectures

A 2-D nonrecursive or FIR filter is described by a convolution summation over a finite window of size $N_1 \times N_2$. Since for these filters each output pixel is dependent only on a limited portion of the input image data, they can be implemented on a fully parallel architecture without requiring any recursion or sequencing. One such structure is given in this section for 2-D block implemented FIR filters using SIMD machine.

Manuscript received January 4, 1988; revised June 12, 1988. This paper was recommended by Associate Editor H. Gharavi.

The authors are with the Department of Electrical Engineering, Colorado State University, Fort Collins, CO 80523.

IEEE Log Number 8825767.

Consider an image of size $M \times M$ which is partitioned into a number of nonoverlapping blocks of size $K \times L$. Now, if the size of the blocks are chosen to be greater or equal to the order of the FIR filter, the convolution summation can be arranged into the following block convolution equation (causal quarter-plane filters) [5]

$$Y_{i,j} = H_{00}U_{i,j} + H_{01}U_{i,j-1} + H_{10}U_{i-1,j} + H_{11}U_{i-1,j-1} \quad (1)$$

where $U_{i,j}$ represent the (i, j)th block of the input image defined in a column vector of size $KL \times 1$ such that

$$U_{i,j} = \left[U_{iK}^{(j)} \quad U_{iK+1}^{(j)} \cdots U_{(i+1)K-1}^{(j)}\right]^t \quad (2a)$$

and

$$U_m^{(j)} = \left[u_{m,jL} \quad u_{m,jL+1} \cdots u_{m,(j+1)L-1}\right] \quad (2b)$$

where $\{u_{m,n}\}$ represents the input sequence. $Y_{i,j}$ which represents the (i, j)th block of the output image can be defined in a similar manner. Matrices $H_{i,j}$'s are doubly Toeplitz matrices with blocks that contain impulse response, $h_{m,n}$ elements [5]. Let us for simplicity assume $K = L$. Then, this block equation can be implemented using N^2 PE's ($N \triangleq M/K$, number of blocks in each strip of the image). Each PE which performs the required operations to calculate a block of the output image requires blocks of input image from its North, West, and North–West PE's as well as its input block from the buffers. This requirement dictates an interconnection network as shown in Fig. 1(a). Since there are as many PE's as there are blocks of data, the entire operation can be carried out in parallel. The sequence of operations depends on the internal architecture of each PE. In Fig. 1(b), the complex architecture is traded in favor of speed. In this scheme, four matrix–vector multipliers and one vector adder are used. The sequence of events are:

a) after loading blocks of the input image to the corresponding PE's, all PE's transfer a copy of their input blocks to South, South-East, and East PE's in parallel;
b) all PE's will perform the block computation in parallel.

Therefore, the total computation time is

$$\tau_{\text{total}} = \tau_{\text{mult}} + \tau_{\text{add}} + \tau_{\text{transfer}}. \quad (3)$$

where τ_{mult} is the time to perform a matrix–vector multiplication; τ_{add} is the time to perform the addition operation over the entire block and τ_{transfer} is the time to transfer a block of data to adjacent PE's. If longer computation time is to be traded in favor of less complex hardware, a structure can be designed using only one matrix–vector multiplier and one vector adder–accumulator.

Note that in general there are two methods used for the construction of SIMD (systolic) machines. In one, PE's are programmable hardware capable of data manipulation. All these PE's will operate under the control of a central controller which is responsible for instruction decoding and broadcasting. The entire system is then connected to a host computer which is used for program development. The host is connected to the system through the use of a dual port memory. The second method involves the construction of the PE using dedicated hardware which performs a prescribed task. In this case, there is no need to broadcast instructions, thus the central controllers can greatly be simplified or altogether eliminated. However, the host is needed for data exchange. In this situation, the memory resident in each

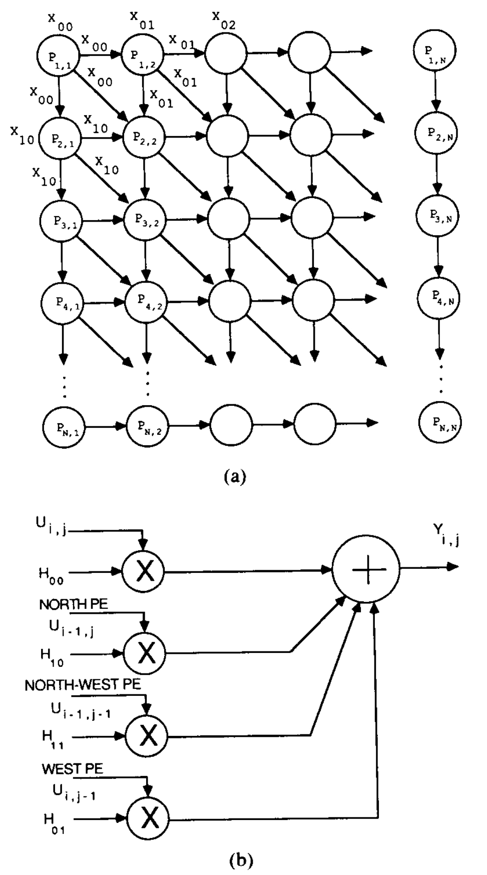


Fig. 1. (a) The SIMD implementation of FIR filters. (b) The PE architecture using four-matrix multipliers.

PE is regarded as a dual port memory and the host has access to each memory modules in the PE's. The image blocks are loaded into the PE's memory by the host.

A structure can also be designed to accomplish all the matrix vector operations in each PE with a minimum number of multiplications. In order to arrive at such a structure, consider the special form of matrices H_{ij}'s. If the constituent block matrices of H_{00} and H_{10} are $H_n^{(0)}$'s and those of H_{01} and H_{11} are $H_n^{(1)}$'s, then the block equation in (1) can be decomposed into a number of (K) equations each describing the row operations required for the output block (i, j). This row operation is given by

$$\begin{aligned} Y_{iK+l}^{(j)} &= \sum_{n=0}^{K} \left[H_n^{(0)} H_n^{(1)} \right] \begin{bmatrix} U_{iK+l-n}^{t(j)} \\ U_{iK+l-n}^{t(j-1)} \end{bmatrix} \\ &= \sum_{n=0}^{K} \overline{H}_n \overline{U}_{iK+l-n}^{(j)}, \qquad l \epsilon [0, K-1] \end{aligned} \tag{4}$$

where

$$\overline{H}_n = \left[\begin{array}{cccc:cccc} h_{n,0} & & \bigcirc & & h_{n,K} & \cdots & h_{n,1} \\ h_{n,1} & h_{n,0} & & & 0h_{n,K} & & h_{n,2} \\ \vdots & & \ddots & & & \ddots & \vdots \\ h_{n,K-1} & \cdots & \cdots & h_{n,0} & \bigcirc & & h_{n,K} \end{array} \right] \tag{5a}$$

and

$$\overline{U}_m^{(j)} = \left[U_m^{(j)} U_m^{(j-1)} \right]^t \tag{5b}$$

Let us consider the worst case when $N_1 = N_2 = K$. Note that this can always be satisfied by zero padding $\{h_{m,n}\}$ to make it of size $K \times K$. If K is divisible by 2, the matrices $\overline{H}_n$'s can be partitioned into blocks of 2×2. The operations on these partitioned matrices which are of special type circulants with noncircular blocks, can be carried out using length-2 short convolution algorithm [5], [9]. Each length-2 operator can be accomplished using only 3 multiplications and 5 additions as

$$\begin{bmatrix} h_0 & h_{-1} \\ h_1 & h_0 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \quad \text{or} \quad \tilde{h}\tilde{u} = \tilde{y} \tag{6}$$

can be performed by defining the intermediate variables

$$g_0 = (h_0 + h_{-1}) u_1 \tag{7a}$$

$$g_1 = h_0 (u_0 - u_1) \tag{7b}$$

$$g_2 = (h_1 + h_0) u_0 \tag{7c}$$

and then

$$g_0 + g_1 = y_0 \tag{8a}$$

$$g_2 - g_1 = y_1. \tag{8b}$$

The operations in (4) can then be implemented on the structure shown in Fig. 2(a). The internal structure of each block of Fig. 2(a) is shown in Fig. 2(b). In this structure the initial inputs to the multiplexers are the subblocks of length 2 of $U_{iK+l-n}^{t(j-1)}$ that are

$$\tilde{u}_{iK+l-n}^{(j-1)}(m), \qquad m \in [0, (K/2) - 1].$$

The subblocks of $U_{iK+l-n}^{t(j)}$ that are

$$\tilde{u}_{iK+l-n}^{(j)}(m), \qquad m \in [0, (K/2) - 1]$$

enter serially to the row processors. Fig. 2(c) shows the structure of each $\tilde{h}$ operator. The total number of multipliers per output sample for this structure is found to be

$$N_{\text{total}} = 3/4(K+2)(K+1) \tag{9}$$

which is considerably smaller than that of the direct matrix-vector implementation.

These architectures are independent of the type of scanning and can provide real-time FIR filtering on temporal image data provided that the temporal sampling period $T_s > \tau_{\text{total}}$. In what follows, SIMD and pipeline structures are introduced for 2-D block implemented IIR digital filters.

III. Implementation of IIR Filters Using SIMD Machines and Pipeline Processors

3.1. 2-D Block Recursive Implementation

Consider a causal quarter-plane 2-D recursive (IIR) digital filter described by a 2-D difference equations. If we divide the input output arrays into non-overlapping blocks of size $K \times L$ the following 2-D block recursive equations can be obtained [5]

$$\begin{aligned} Y_{i,j} = E_{10} Y_{i-1,j} + E_{01} Y_{i,j-1} + E_{11} Y_{i-1,j-1} + F_{00} U_{i,j} + F_{10} U_{i-1,j} \\ + F_{01} U_{i,j-1} + F_{11} U_{i-1,j-1} \end{aligned} \tag{10}$$

where matrices E_{ij}'s and F_{ij}'s are similar in structure to H_{ij}'s in (1), but they are defined in terms of the coefficients of the 2-D difference equation [5]. Since each output block is dependent not only on past and present input blocks but also on the past output

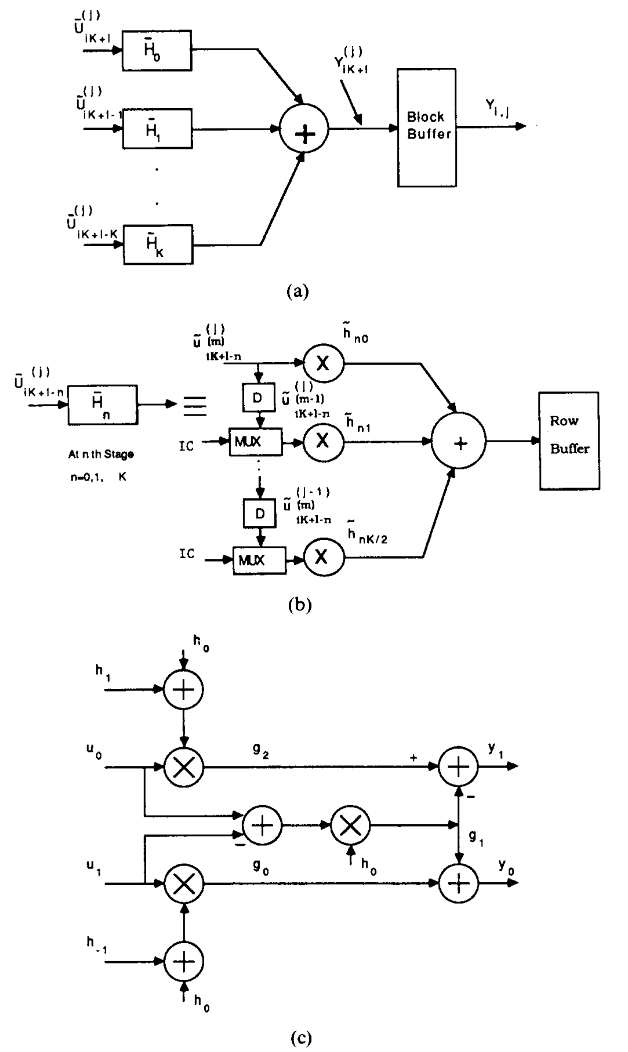


Fig. 2. (a) The generation of output block $Y_{i,j}$ using the row processing scheme. (b) The internal structure of each row processor. (c) The internal structure of each two-by-two operator.

blocks, the operations for IIR filters should be carried out in sequential manner. In this section a pipeline mechanism for block recursive equation is introduced making use of SIMD machines and the fact that IIR filters can be realized as a cascaded of an all-pole and a nonrecursive filter, i.e.,

$$W_{i,j} = F_{00}U_{i,j} + F_{10}U_{i-1,j} + F_{01}U_{i,j-1} + F_{11}U_{i-1,j-1} \quad \text{(11a)}$$

$$Y_{i,j} = E_{10}Y_{i-1,j} + E_{01}Y_{i,j-1} + E_{11}Y_{i-1,j-1} + W_{i,j}. \quad \text{(11b)}$$

Now these operations for 2-D block recursive filtering on multiple frames of digital images can be organized in pipeline fashion on an array of N^2 PE's as shown in Fig. 3. During one pipeline clock, all the PE's along each diagonal work on the same image and their computed results are delivered to the output buffers and also to the immediate neighboring PE's in the next diagonals; while the PE's in the other diagonals are working on different images. As a result at every clock pulse, $2N-1$ images are being processed once the pipe is full; and it takes $2N-1$ clock periods to complete the processing of one image. The PE

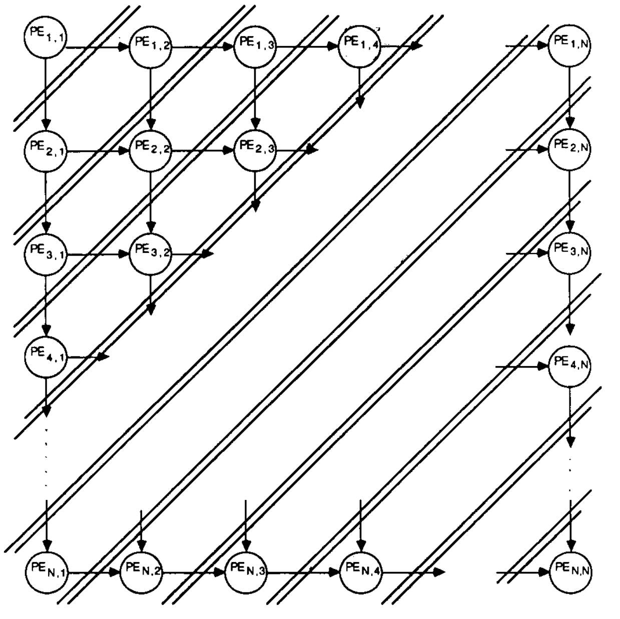


Fig. 3. The pipelined SIMD implementation of IIR filters.

computation time τ_{PE} is

$$\tau_{\mathrm{PE}} = \tau_Y + \tau_W, \quad (12)$$

where τ_W and τ_Y are the time required to compute $W_{i,j}$ and $Y_{i,j}$, respectively. The sampling rate will then be $1/\tau_{\mathrm{PE}}$.

The operations in each PE can be divided into two steps, assuming that $\mathrm{PE}_{i,j}$ is processing a block of the kth frame.

(1) During the time interval τ_W, the following four operations can be accomplished simultaneously:

a) compute

$$W^k_{i,j} = F_{00}U^k_{i,j} + F_{10}U^k_{i-1,j} + F_{01}U^k_{i,j-1} + F_{11}U^k_{i-1,j-1}; \quad (13)$$

b) deliver $Y^{k-1}_{i,j}$ to the output buffer;

c) send $Y^{k-1}_{i,j}$ to three immediate neighbors, $\mathrm{PE}_{i,j+1}$, $\mathrm{PE}_{i+1,j+1}$ and $\mathrm{PE}_{i+1,j}$.

d) receive $Y^k_{i-1,j}$ from $\mathrm{PE}_{i-1,j}$, $Y^k_{i,j-1}$ from $\mathrm{PE}_{i,j-1}$ (kth frame), and receive $Y^{k+1}_{i-1,j-1}$ of frame $k+1$ from $\mathrm{PE}_{i-1,j-1}$ and store it for future processing on the $(k+1)$th frame. Note that, $Y^k_{i-1,j-1}$ of frame k is stored in $\mathrm{PE}_{i,j}$ one clock in advance.

(2) During the time interval of τ_Y, the following four operations are accomplished simultaneously:

a) compute

$$Y^k_{i,j} = E_{10}Y^k_{i-1,j} + E_{01}Y^k_{i,j-1} + E_{11}Y^k_{i-1,j-1} + W^k_{i,j}; \quad (14)$$

b) send $U^k_{i,j}$ to the three of its neighboring PE's.

c) get $U^{k+1}_{i,j}$ from the input buffer.

d) receive $U^{k+1}_{i-1,j}$ and $U^{k+1}_{i-1,j-1}$ of frame $k+1$ from $\mathrm{PE}_{i-1,j}$ and $\mathrm{PE}_{i,j-1}$; receive $U^{k+1}_{i-1,j-1}$ from $\mathrm{PE}_{i-1,j-1}$ and store them for future processing at frame $k+1$. Note that the operations in 2c) and 2d) are needed for future stages.

Therefore, the processing time for one complete image is

$$\tau_{\mathrm{total}} = (2N-1)\tau_{\mathrm{PE}}. \quad (15)$$

Note that at every τ_{PE}, the operations on the last block of an image frame will be completed at $\mathrm{PE}_{N,N}$, whereas the first block of a new image frame will enter $\mathrm{PE}_{1,1}$. The temporal sampling

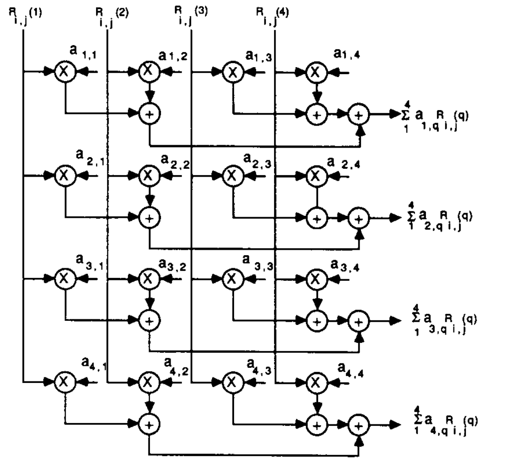


Fig. 4. A typical 4×4 matrix-vector multiplier utilizing recursive doubling method ($n1 = n2 = K = L = 2$).

interval for this pipelined architecture should be greater than or equal to τ_{PE}. The main advantage of this partitioning is to share the hardware for calculating $W_{i,j}^k$ and $Y_{i,j}^k$ in different time intervals τ_W and τ_Y. This consequently reduces the amount of hardware needed to implement an IIR filter.

3.2. 2-D Block State-Space Implementation

Recursive digital filters can alternatively be described by a state-space realization. Let us define the "block state" vectors $\tilde{R}(i,j)$ and $\tilde{S}(i,j)$ which, respectively, refer to the states associated with the boundary elements of the (i,j)th block along the horizontal and vertical directions. Having defined the "block state" vectors, the 2-D block state-space equation is given by [5]

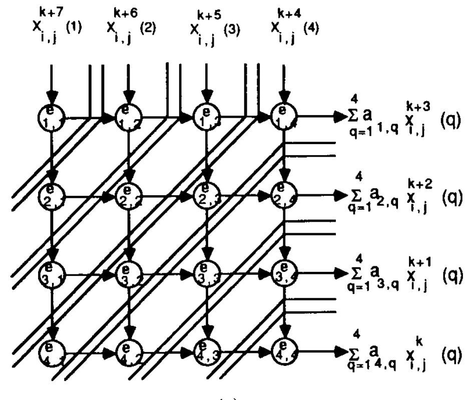
$$\tilde{R}(i+1,j) = \tilde{A}_1\tilde{R}(i,j)+\tilde{A}_2\tilde{S}(i,j)+\tilde{B}_1U(i,j) \quad (16a)$$

$$\tilde{S}(i,j+1) = \tilde{A}_3\tilde{R}(i,j)+\tilde{A}_4\tilde{S}(i,j)+\tilde{B}_2U(i,j) \quad (16b)$$

$$Y(i,j) = \tilde{C}_1\tilde{R}(i,j)+\tilde{C}_2\tilde{S}(i,j)+\tilde{D}U(i,j). \quad (16c)$$

The matrices in this multiinput multioutput (MIMO) structure are obtained [5] in terms of those of the single-input single output (SISO) structure. The block state vectors $\tilde{R}(i,j)$ and $\tilde{S}(i,j)$ propagate vertically and horizontally to generate $\tilde{R}(i+1,j)$ and $\tilde{S}(i,j+1)$, respectively. The global initial conditions are assumed to be zero.

Similar to the pipelined structure in the previous case, a pipelined SIMD structure can be devised for the 2-D block state-space model. The interconnection network needed for this implementation is simpler than the previous architecture due to the fact that each PE only requires one vector $\tilde{R}$ from the North block and one vector $\tilde{S}$ from the East block. This scheme not only reduces the number of data path to two (diagonal connections are eliminated) as opposed to three in previous design, but also each path is considerably smaller in size i.e., (n_1L) and (n_2K) as compared to (KL) and (KL) in the previous implementation, where n_1 and n_2 are the order of the 2-D IIR system. As mentioned above, $PE_{i,j}$ can perform the output, $\tilde{R}$ and $\tilde{S}$ calculations only when the $PE_{i-1,j}$ and $PE_{i,j-1}$ have already finished their calculations of the required $\tilde{R}$ and $\tilde{S}$. Therefore, during the period in which $PE_{i-1,j}$ and $PE_{i,j-1}$ are calculating their $\tilde{R}$ and $\tilde{S}$, the $PE_{i,j}$ would be idle. To avoid such a waste of resource, the system will be setup to work in a pipelined fashion. In this form, while the $PE_{i-1,j}$ and $PE_{i,j-1}$ are working on


(a)

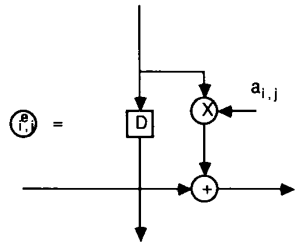

(b)

Fig. 5. (a) The structure of fully pipelined PE. (b) The structure of each subprocess element $e_{i,j}$.

frame $k+1$, the $PE_{i,j}$ can be working on image frame k. For this architecture, there will be $(2N-1)$ images being processed simultaneously. The operations in each PE, say $PE_{i,j}$ (assuming $PE_{i,j}$ is processing a block of the kth image, i.e., $U^k(i,j)$) include: calculations of $\tilde{R}^k(i+1,j)$, $\tilde{S}^k(i,j+1)$ and $Y^k(i,j)$ at $\tau_{\text{calc.}}$ using

$$\tilde{R}^k(i+1,j) = \tilde{A}_1\tilde{R}^k(i,j)+\tilde{A}_2\tilde{S}^k(i,j)+\tilde{B}_1U^k(i,j) \quad (17a)$$

$$\tilde{S}^k(i,j+1) = \tilde{A}_3\tilde{R}^k(i,j)+\tilde{A}_4\tilde{S}^k(i,j)+\tilde{B}_2U^k(i,j) \quad (17b)$$

$$Y^k(i,j) = \tilde{C}_1\tilde{R}^k(i,j)+\tilde{C}_2\tilde{S}^k(i,j)+\tilde{D}U^k(i,j) \quad (17c)$$

transferring $\tilde{R}^k(i+1,j), \tilde{S}^k(i,j+1)$ to $PE_{i+1,j}$ and $PE_{i,j+1}$ in τ_{transfer}.

For this implementation, τ_{PE} is given by

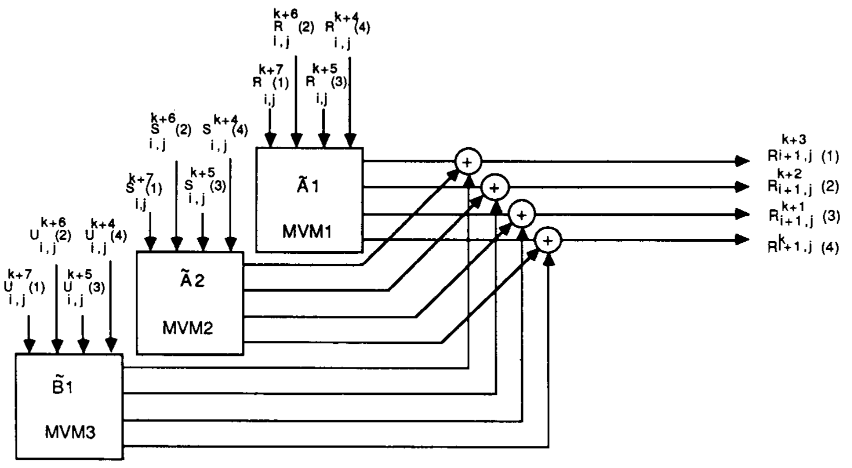
$$\tau_{PE} = \tau_{\text{calc.}} + \tau_{\text{transfer}} \quad (18a)$$

$$\tau_{\text{calc}} = \tau_M + 2\tau_A \quad (18b)$$

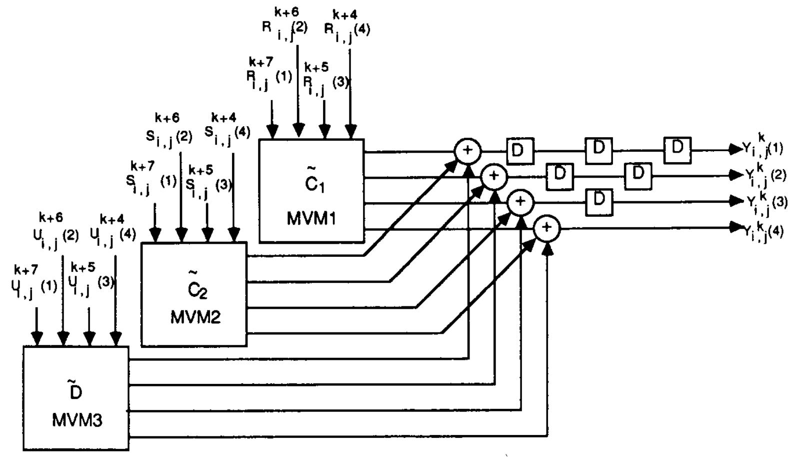
where τ_M is the time for a vector matrix multiplication and τ_A is the time for a vector addition. The temporal sampling rate will then be $1/\tau_{PE}$ and the latency is $(2N-1)\tau_{PE}$. To obtain higher sampling rate, τ_{PE} has to be reduced which in turn requires the reduction of τ_{calc}. To achieve smaller τ_{calc}, we may design matrix-vector multipliers in which all the additions are done using recursive doubling procedure. In this case, the computation time in each PE would become

$$\tau_{\text{calc.}} = \tau_m + \tau_a[2+\log_2 KL] \quad (19)$$

where τ_m is the time for a scalar multiplication and τ_a is the time for a scalar addition. A typical 4×4 matrix-vector multiplier utilizing recursive doubling is shown in Fig. 4.



(a)



(b)

Fig. 6. (a) Computation of the state $R(i+1, j)$ by fully pipelined MVM's. (b) Computation of the (i, j)th output block by fully pipelined MVM's.

3.2.1. A Fully Pipelined PE Architecture

To increase the processing speed of the PE's even further, a fully pipelined matrix-vector multiplier (MVM) is designed in this section. For simplicity, it is assumed that $K = L = n_1 = n_2$. A typical 4×4 fully pipelined MVM with an arbitrary input sequence $[x_{i,j}(1),\cdots,x_{i,j}(4)]$ of four consecutive frames is shown in Fig. 5, where k represents the frame number. As shown in Fig. 5, while the last output of image frame k,

$$\sum_{q=1}^{4} a_{4,q}x_{i,j}^{k}(q)$$

is leaving the lower and rightmost subprocess unit $e_{4,4}$ of the $\mathrm{MVM}_{i,j}$, the first input of image frame $k+7$ to $\mathrm{MVM}_{i,j}$, $x_{i,j(1)}^{k+7}$, is entering to the upper and left most subprocess unit. Therefore, there are data from $2KL-1$ image frames that are being processed simultaneously in one MVM.

In order to implement the operations in (17) in each PE a total of nine MVM's and six vector-adders are needed. Figs. 6(a) and (b) show the implementation of (17a) and (17c), respectively. Equation (17b) is implemented similar to (17a). As shown in Fig. 6(b), there are six latches (delays) for output synchronization in each PE. Let τ_m and τ_a denote the required time for one scalar multiplication and one scalar addition, respectively. Then, using this structure the total computation in one PE becomes

$$\tau_{\text{calc.}} = \tau_m + 3\tau_a \tag{20}$$

which permits even higher sampling rate.

IV. Conclusion

In this paper several SIMD and pipeline architectures are proposed for 2-D block processing. An SIMD architecture for 2-D FIR filters is presented which offer a choice of speed/cost tradeoff. The matrix-vector operations in this structure can be implemented using length-2 convolution processor with a minimum number of multiplications. This SIMD architecture can further be modified to process $2N-1$ image frames simultaneously in a pipeline fashion for direct 2-D block recursive and 2-D block state-space implementations. To yield higher throughput-rate, pipeline mechanism has also been adopted at the PE level. The structures introduced in this paper provide a symmetric design for implementing near real to real-time filtering operation.

References

[1] H. H. Lu, E. A. Lee, and D. G. Messerschmitt, "Fast recursive filtering with multiple slow processing elements," *IEEE Trans. Circuits Syst.*, vol. CAS-32, pp. 1119–1129, November 1985.

[2] K. K. Parhi and D. G. Messerschmitt, "Look-ahead computation: improving iteration bound in linear recursions," in *Proc. IEEE Int. Conf. on Acoust., Speech, Signal Processing*, Dallas, TX, 1987.

[3] K. K. Parhi and D. G. Messerschmitt, "Block digital filtering via incremental block-state structure," *Proc. 1987 IEEE Int. Symp. on Circuits and Syst.*, Philadelphia, PA, May 1987.

[4] C. L. Nikias, "Fast block data processing via a new IIR digital filter structure," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 770–779, Aug. 1984.

[5] M. R. Azimi-Sadjadi and R. A. King, "Two-dimensional block processors-structures and implementations," *IEEE Trans. Circuits Syst.*, vol. CAS-33, pp. 42–50, Jan. 1986.

[6] M. R. Azimi-Sadjadi, A. R. Rostampour, and T. Lu, "Parallel architectures for 2-D block processing," in *Proc. IEEE RIM Conf. Commun., Computers and Signal Processing*, pp. 455–460, Victoria B.C. Canada, June 1987.

[7] A. W. Burks (Editor), *Essays on Cellular Automata*. Urbana, IL, Univ. of Illinois Press, 1970.

[8] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stockes, "The ILLIAC IV computer," *IEEE Trans. Computers*, pp. 746–757, Aug. 1968.

[9] R. C. Agarwal and C. S. Burrus, "Fast one-dimensional digital convolution by multi-dimensional techniques," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 1–10, Feb. 1974.

Synthesis of Recursive LTV Digital Filters Using the Frozen-Time Approximation

S. DOUGLAS PETERS AND MOUSTAFA M. FAHMY

Abstract —**This paper reexamines the frozen-time approximation (FTA) for the synthesis of recursive linear time-varying (LTV) digital filters. While the use of the FTA has been previously discouraged, analysis shows that this approach is still valuable. Improvements on the FTA are also proposed. These improvements take advantage of the simplicity of the frozen time approach, being based on the assumption that for any desired LTV filter there exists a related filter whose FTA produces a better realization of the desired filter than its own FTA. Methods of finding such a related filter are discussed. The resulting filters are superior in performance to those obtained by the currently available optimization techniques at a fraction of the computational load.**

I. Introduction

It has been shown that linear time-varying (LTV) digital filters have application to non-stationary systems [1], [2]. Recursive structures for these filters are desirable due to the prohibitive memory requirements of even short-duration FIR LTV filters. Unfortunately, the simplicity of time-invariant recursion does not carry over to time-varying systems. In fact, many of the assumptions with which we approach linear time-invariant (LTI) systems do not hold for LTV systems.

In LTI systems, any filter described by a rational transfer function may be realized by using the coefficients of that rational function in a difference equation. In LTV systems, however, only filters with stationary poles (neither a very large nor useful subset of LTV filters) can be realized by a difference equation [3]. Therefore, in LTV systems, we must either obtain a set of difference equation coefficients to approximately realize a desired filter or avoid the use of a difference equation realization altogether [4]. In any event, it may be very difficult, and sometimes even impossible, to satisfactorily synthesize a fast varying LTV filter.

Manuscript received January 4, 1988; revised July 8, 1988. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant A4149. This paper was recommended by Associate Editor Y. C. Jenq.

The authors are with the Department of Electrical Engineering, Queen's University, Kingston, Ont., Canada K7L 3N6.

IEEE Log Number 8825766.

The simplest solution to the problem of LTV filter synthesis is known as the frozen-time approximation (FT). The FTA procedure uses the rational function coefficients in a difference equation in the same manner as an LTI filter. This procedure does not, in general, realize the desired filter, but simply produces an approximate realization of it. The approach has been shown to perform well only for slowly varying filters [3]. The term slowly varying, however, remains rather nebulous. In fact, most of the proposed applications for LTV filters have also been described as slowly varying. Therefore, the FTA should not be ruled out on these grounds. Further, the FTA approach has been recently discouraged by way of numerical example [5]. This same example will be used in the present work to reach the opposite conclusion. Indeed, the negative conclusion that was reached was based on the performance of the FTA for only one structure. Before the FTA is dismissed, its performance should be assessed using many different structures, since, in contrast to LTI filters, there is no simple relationship between the coefficients used for different structures in LTV systems.

The objective of this paper is to reestablish the frozen-time approximation as a viable option for the synthesis of recursive LTV filters. Improvements on the frozen-time approach will also be introduced. It will be shown that filters can be synthesized using these techniques which compare in performance to those resulting from time-consuming optimization techniques. In Section II an example demonstrates the accuracy of the frozen-time approximation for different structures. Section III will describe the improvement techniques that have been developed based on the FTA. Examples will be presented to demonstrate the effectiveness of the present approach.

II. The FTA and Different Structures

In general, LTV filters have been described in the time domain by an impulse response $h(n,m)$, which is defined as the response of the filter at the sampling instant n to an impulse applied at the sampling instant m. The output of this filter may then be written:

$$y(n) = \sum_{m=-\infty}^{\infty} h(n,m)x(m).$$

The generalized transfer function (GTF), $H(z,n)$, is then defined as the z-transform of the impulse response with respect to the variable $(n-m)$:

$$H(z,n) = \sum_{m=-\infty}^{\infty} h(n,m)z^{-(n-m)}. \tag{1}$$

As an example of this, consider the rational function

$$H(z,n) = \frac{\sum_{i=0}^{K_1} a_i(n)z^{-i}}{1+\sum_{i=1}^{K_2} b_i(n)z^{-i}} \tag{2}$$

where the coefficients $a_i(n)$ and $b_i(n)$ are functions of the filter characteristics at the sampling instant n. The present paper