A Pipelined Adaptive Lattice Filter Architecture

Naresh R. Shanbhag, *Student Member, IEEE*, and Keshab K. Parhi, *Senior Member, IEEE*

Abstract—**The stochastic gradient adaptive lattice filter is pipelined by the application of relaxed look-ahead. This form of look-ahead maintains the functional behavior of the algorithm instead of the input–output mapping and is suitable for pipelining adaptive filters. The sum and product relaxations are employed to pipeline the adaptive lattice filter. The hardware complexity of the proposed pipelined adaptive lattice filters is the same as for the sequential filter, and is independent of the level of pipelining or speedup. Thus, the new architectures are very attractive in terms of implementation. Two pipelined architectures along with their convergence analyses are presented to illustrate the tradeoff offered by relaxed look-ahead. Simulation results supporting the conclusions of the convergence analysis are provided. The proposed architectures are then employed to develop a pipelined adaptive differential pulse-code modulation codec for video compression applications. Speedup factors up to 20 are demonstrated via simulations with image data.**

I. Introduction

THE design of concurrent signal processing algorithms for real-time applications requiring high speed is currently of interest. To this end, algorithm transformation techniques [1] have formalized the methodology for exploiting concurrency hidden in conventional digital signal processing algorithms. Of the two approaches for achieving high speed, pipelining [2] and parallel processing [3], the former is attractive due to its reduced hardware requirements.

The look-ahead pipelining [2] demonstrated the feasibility of high-speed implementation of signal processing algorithms. However, this technique achieves high speed by creating additional concurrency in nonconcurrent signal processing algorithms, at the expense of significant hardware overhead. Even though the look-ahead pipelining technique has been successfully applied to numerous problems [2], [4]–[7], its extremely high hardware complexity makes it difficult to implement. This problem is compounded further if pipelining of adaptive filters is attempted [8], [9]. Therefore, an alternative to algorithm transformation techniques [1] is to develop inherently pipelinable digital signal processing algorithms. Furthermore, the hardware requirements of these pipelinable algorithms and the traditional nonconcurrent algorithms should be the same or similar.

Manuscript received October 11, 1991; revised June 4, 1992. The associate editor coordinating the review of this paper and approving it for publication was Prof. Ed F. Deprettere. This work was supported by the Army Research Office under Contract DAAL-90-G-0063.

The authors are with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455.

IEEE Log Number 9207529.

In this paper, we present an approximate form of look-ahead referred to as relaxed look-ahead [10]. The conventional look-ahead technique [2] transforms a given serial algorithm into an equivalent (in the sense of input–output mapping) pipelined one. The relaxed look-ahead sacrifices this equivalence between the serial and pipelined algorithms at the expense of marginally altered convergence characteristics. Therefore, the relaxed look-ahead maintains the functionality of the algorithm and is well suited for adaptive filtering applications. However, unlike look-ahead, application of relaxed look-ahead requires a subsequent convergence analysis of the resulting pipelined algorithm.

The relaxed look-ahead is employed to design new concurrent stochastic gradient lattice adaptive algorithms [11], which are inherently pipelined. The adaptive lattice filters designed using relaxed look-ahead have the property that the hardware requirements for these filters are independent of the speedup or the level of pipelining. In contrast, the hardware requirements in look-ahead increase with the speedup (although logarithmically). The proposed adaptive lattice algorithms are then employed to develop a pipelined adaptive pulse-code-modulation (ADPCM) codec for image compression applications. Preliminary results on the performance of the pipelined ADPCM codec were found to be very promising [12]. As expected, the hardware requirements for the new ADPCM codec was found to be independent of the number of quantizer levels L, the order of the predictor N, and the speedup M. This is in contrast to the pipelined codec architecture developed via look-ahead in [5]. The application of look-ahead in [5] proceeded in two steps. First, the ADPCM loop was linearized by moving the quantizer outside. This step resulted in an increase in the hardware complexity of $O(L)$. Then, the look-ahead was applied to the linearized loop. This two-step process resulted in a hardware complexity which is strongly dependent on L, N, and M. The architecture in [5] was for a ADPCM codec with a transversal predictor and therefore a comparison with lattice ADPCM codec may seem unfair. However, similar conclusions were reached in [13], where a pipelined transversal ADPCM codec was developed via relaxed look-ahead. An additional advantage of the proposed codec is the fact that the output latency is usually much smaller than the level of pipelining.

In addition to the transversal ADPCM codec [13], the relaxed look-ahead has also been successfully applied to pipeline the transversal LMS algorithm [10]. The hardware increase in the pipelined architecture was only the

pipelining latches. This is a remarkable improvement over previous attempts at high-speed adaptive filtering [14]–[17].

We first explain the technique of pipelining in Section II and then describe the relaxed look-ahead in Section III. In Section IV, we develop two pipelined stochastic gradient lattice filter architectures PIPSGLA1 and PIPSGLA2 and compare the hardware requirements of the serial architecture (SSGLA) [11] and the architecture resulting from the conventional deterministic look-ahead (DLASGLA) [8]. The results of the convergence analysis of PIPSGLA1 and PIPSGLA2 are presented in Section V. The pipelined ADPCM codec is developed in Section VI. The simulation results presented in Section VII confirm the results of our analysis and demonstrate the performance of the pipelined codec.

II. Pipelining

Conventionally, pipelining has been viewed as an architectural technique for increasing the throughput of an algorithm. However, in [18] the use of pipelining for reducing power consumption in VLSI chips has been described. This fact extends the utility of pipelined algorithms from high-speed (high-power) applications such as video compression to low-speed (low-power) applications such as speech compression.

A. A High Throughput Technique

Consider an algorithm (see Fig. 1(a)) consisting of two computational blocks A and B. The throughput of this serial (or unpipelined) system is limited by

$$f_{\text{unpipe}} \leq \frac{1}{T_A + T_B} \tag{2.1}$$

where T_A and T_B are the computation times of blocks A and B, respectively.

In order to increase the throughput, we need to pipeline the system in Fig. 1(a) by introducing pipelining latches. A two-stage pipelined system is shown in Fig. 1(b). The throughput of this pipelined system is

$$f_{\text{pipe}} \leq \frac{1}{\max(T_A, T_B)} \tag{2.2}$$

which is clearly greater than that of the serial system in Fig. 1(a). This increase in throughput has been made at the expense of an increase in the output latency. For the serial system, the output corresponding to the current input is generated in the current clock cycle. However, for the pipelined system, the output is delayed by one clock cycle. In general, assuming that each stage has the same delay, the throughput of a system pipelined by M stages is M times that of the serial system, while its output latency increases by the computational delay associated with the M pipelining latches.

B. A Low Power Technique

We now show how pipelining can be employed to design low-power circuits. The dynamic power consumption (P) of a CMOS circuit is given by

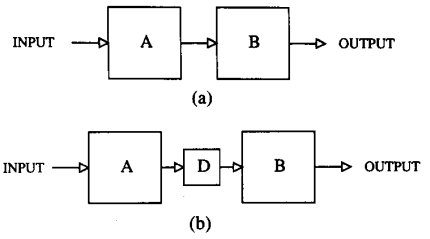


Fig. 1. Pipelining: (a) a serial system, and (b) equivalent pipelined system.

$$P = C_{\text{total}} V_{dd}^2 f \tag{2.3}$$

where C_{total} is the total switching capacitance, V_{dd} is the supply voltage and f is the clock frequency. From (2.3) it is clear that dramatic reductions in power are possible by reducing the supply voltage for a constant clock frequency.

Consider the serial algorithm of Fig. 1(a). The propagation delay at supply voltage of V_{dd} ($t_{pd,\text{unpipe}}(V_{dd})$) of the serial algorithm is given by

$$t_{pd,\text{unpipe}}(V_{dd}) = \frac{C_L V_{dd}}{\epsilon(V_{dd} - V_t)^2} \tag{2.4}$$

where C_L is the capacitance along the critical path in Fig. 1(a), V_t is the device threshold voltage, and ϵ is a constant which depends on the process parameters. In an optimally designed system the propagation delay should in fact equal the clock period.

For a M-stage pipelined system, the propagation delay $t_{pd,\text{pipe}}(V_{dd})$ is

$$t_{pd,\text{pipe}}(V_{dd}) = \frac{C_L}{M} \frac{V_{dd}}{\epsilon(V_{dd} - V_t)^2} \tag{2.5}$$

which is M times less than that of the serial algorithm. Clearly, the pipelined system can be clocked at a much higher frequency than is necessary. However, by reducing V_{dd}, we can increase the propagation delay of the pipelined system till it equals that of the serial system. This step not only matches the propagation delay to the desired clock period but also results in a reduction of power consumption.

There is however a lower limit to which the supply voltage can be reduced for a given value of V_t. Let K_1 be the factor by which the supply voltage of the pipelined system needs to be reduced for its propagation delay to equal that of the serial system. Equating $t_{pd,\text{pipe}}(V_{dd}/K_1)$ (from (2.5)) to $t_{pd,\text{unpipe}}(V_{dd})$ (2.4) we get the following equation:

$$MV_t^2 K_1^2 - (2MV_{dd}V_t + (V_{dd} - V_t)^2)K_1 + MV_{dd}^2 = 0 \tag{2.6}$$

which can be solved for K_1 given the pipelining level M. Substituting $V_{dd} = 5$ V and $V_t = 0.7$ V, we find that for a two-stage pipelined system (i.e., $M = 2$) a supply voltage of 3.08 V is necessary for the two propagation delays

to equal each other. At this supply voltage the power dissipation is reduced by a factor of 2.62. Similarly, for $M = 3$, a supply voltage of 2.43 V would be needed and the power dissipation would be reduced by a factor of 4.

This analysis indicates that pipelining by fewer levels results in reduction of power dissipation at the same speed.

III. Pipelining Using Relaxed Look-Ahead

In this section, we develop the relaxed look-ahead and point out its significance in the development of inherently concurrent adaptive algorithms. First, we apply the look-ahead to a first order recursive section and then introduce the relaxed look-ahead.

Consider the first-order recursion given by

$$x(n + 1) = ax(n) + u(n). \tag{3.1}$$

We first apply an M-step look-ahead to (3.1), which is equivalent to expressing $x(n + M)$ in terms of $x(n)$. This leads to

$$x(n + M) = a^M x(n) + \sum_{i=0}^{M-1} a^i u(n + M - 1 - i). \tag{3.2}$$

The $M - 1$ extra latches introduced into the recursive loop by this transformation can be used to pipeline the multiply and add operations in the loop. Note that the second term in (3.2) is an additional overhead due to look-ahead and is called the look-ahead overhead $f_o(M)$. The argument of $f_o(M)$ indicates its dependence on the level of pipelining (M).

The overhead in look-ahead is very hardware expensive and at times is impractical to implement. However, under certain circumstances we can substitute approximate expressions on the right-hand side (RHS) of (3.2). Depending on the application at hand, different types of approximations (or relaxations) may be employed. We now formulate two such relaxations which are called the sum and the product relaxations. These two relaxations would be employed to pipeline the stochastic gradient lattice filter [11] in the next section.

A. Sum Relaxation

In (3.2), if $a \approx 1$ and the input $u(n)$ remains more or less constant over M cycles, then we can replace the summation in $f_o(M)$ with $Mu(n)$. The resulting expression is given by

$$x(n + M) = a^M x(n) + Mu(n). \tag{3.3}$$

The a^M term in (3.3) can be precomputed if a is a constant. If $u(n)$ is close to 0, then another relaxation of (3.3) is achieved by

$$x(n + M) = a^M x(n) + u(n). \tag{3.4}$$

Thus, (3.3) and (3.4) are the results of application of M-step relaxed look-ahead with sum relaxations to (3.1).

B. Product Relaxation

If a in (3.2) is time varying (i.e., we have $a(n)$ instead of a), but the magnitude of $a(n)$ is close to unity, then we can replace $a(n)$ by $(1 - a'(n))$, where $a'(n)$ is close to zero. Then, (3.2) is approximated as

$$x(n + M) = (1 - Ma'(n))x(n) + \sum_{i=0}^{M-1} a^i u(n + M - 1 - i). \tag{3.5}$$

Hence, (3.5) is the outcome of an application of a M-step relaxed look-ahead with product relaxation to (3.1).

These relaxations (and any other) constitute the relaxed look-ahead. The relaxations may be applied individually or in combination. As an example, the application of a M-step relaxed look-ahead with sum relaxation of (3.3) and product relaxation to (3.1), with time-varying coefficients, results in

$$x(n + M) = (1 - Ma'(n))x(n) + Mu(n). \tag{3.6}$$

Thus, the relaxed look-ahead does not result in a unique final architecture. This is in contrast to look-ahead, where there is a one-to-one mapping between the resulting pipelined architecture and the original one. The mapping for relaxed look-ahead, on the other hand, is one to many. This point will be illustrated further when we develop two pipelined architectures for the stochastic gradient adaptive lattice filter.

The relaxed look-ahead is not an algorithm transformation technique in the conventional sense [1]. This is because it modifies the input–output behavior of the original algorithm. It may be called a transformation technique in a stochastic sense if the average output profile is maintained. This, however, depends upon the nature of the approximations made. The pipelined architecture resulting from the application of relaxed look-ahead can always be clocked at a higher speed than the original one.

IV. Pipelined Adaptive Lattice Architectures

The relaxed look-ahead, described in the previous section, is employed in this section to develop inherently pipelined adaptive lattice filter architectures. Even though numerous adaptive lattice algorithms exist [11], we choose a simple version of the stochastic-gradient lattice algorithm. It must be pointed out that the relaxed look-ahead can be used to pipeline any of the other adaptive lattice algorithms. The convergence analysis would, however, differ.

A. The Serial Stochastic-Gradient Lattice Architecture

We choose the stochastic-gradient lattice algorithm [11] described by the following equations:

$$\begin{aligned} k_m(n + 1) = {} & [1 - \beta_m(n)(e_f^2(n|m - 1) \\ & + e_b^2(n - 1|m - 1))]k_m(n) \\ & + 2\beta_m(n)e_f(n|m - 1) \\ & \cdot e_b(n - 1|m - 1) \end{aligned} \tag{4.1}$$

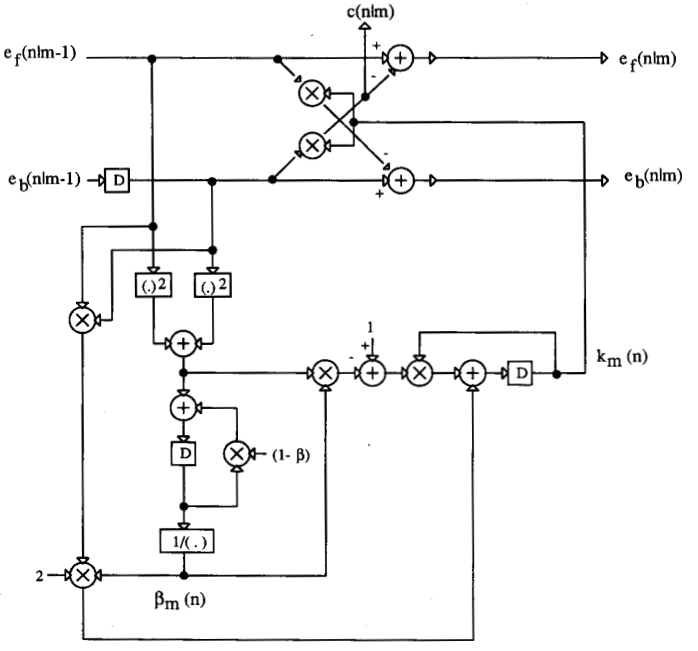


Fig. 2. The SSGLA architecture.

$$S(n+1|m-1) = (1-\beta)S(n|m-1) + e_f^2(n|m-1) + e_b^2(n-1|m-1) \quad (4.2)$$

$$\beta_m(n) = \frac{1}{S(n|m-1)} \quad (4.3)$$

$$e_f(n|m) = e_f(n|m-1) - k_m(n)e_b(n-1|m-1) \quad (4.4a)$$

$$e_b(n|m) = e_b(n-1|m-1) - k_m(n)e_f(n|m-1) \quad (4.4b)$$

where n denotes the time instance, $m = 1, \cdots, N$ denotes the lattice stage number (N being the total number of stages or the order of the filter). Equations (4.1)–(4.3) and (4.4) are referred to as the adaptation and order-update equations, respectively. The reflection coefficient $k_m(n)$ is updated according to (4.1). The energy $S(n|m-1)$ of the input signals $e_f(n|m-1)$ and $e_b(n-1|m-1)$ to the mth stage is recursively computed by (4.2). This energy is required to compute the normalized adaptation coefficient $\beta_m(n)$. This serial stochastic-gradient lattice algorithm is referred to as the SSGLA algorithm.

The SSGLA is shown in Fig. 2, where the intermediate output $c(n|m)$ would be employed later (see section VI) to develop a pipelined ADPCM codec. The computation time T_c of the critical path is equal to

$$T_c = 4T_m + (N+2)T_a \quad (4.5)$$

where T_m and T_a are the computation times of a two-operand multiplier and adder, respectively. For simplicity, we assume that squaring and the division operations take the same amount of time as multiplication, although in practice division operations may require longer computation time. In addition, we see that there are two recursive loops with computation times:

$$T_l = T_m + T_a. \quad (4.6)$$

It is now desired to pipeline the SSGLA such that the clock period (T_p) of the pipelined architecture (to be referred to as PIPSGLA) is less than T_c/M, where M is the desired speedup. If T_p is greater than T_l, then the recursive loops need not be pipelined. For this case, the speedup can be achieved by employing interstage pipelining only.

B. The Pipelined Stochastic-Gradient Lattice Architecture (PIPGLA)

The pipelining of SSGLA proceeds in two steps. First, we include the interstage pipelining of the SSGLA. If the desired speedup cannot be achieved with this level of pipelining, then finer grain loop pipelining with relaxed look-ahead is adopted. Interstage pipelining is trivial as the stages are connected in a non-recursive fashion. Let M_S and M_L denote the number of interstage and loop pipelining latches, respectively. We can now apply relaxed look-ahead to pipeline the two recursive loops in SSGLA (Fig. 2).

We first apply a M_L-step relaxed look-ahead with sum relaxation of (3.4) to (4.2), which describes the recursive computation of the input power. Like (3.1), (4.2) is a first-order recursion. Therefore, by inspection of (3.4), the final result of the application of the sum relaxation to (4.2)

is given by

$$S(n + M_L|m - 1) = (1 - \beta)^{M_L} S(n|m - 1) + e_f^2(n|m - 1) + e_b^2(n - 1|m - 1). \quad (4.7)$$

As the constant $(1 - \beta)^{M_L}$ can be precomputed, it is not necessary to apply the product relaxation.

Similarly, the application of a M_L-step relaxed look-ahead with sum relaxation of (3.3) to (4.1) results in the following equation:

$$k_m(n + M_L) = [1 - \beta_m(n)(e_f^2(n|m - 1) + e_b^2(n - 1|m - 1))]^{M_L} k_m(n) + 2M_L\beta_m(n)e_f(n|m - 1) \cdot e_b(n - 1|m - 1). \quad (4.8)$$

Note that the bracketed part of the first term in (4.8) represents a hardware module whose output equals its input raised to the power M_L. It is possible to reduce the hardware complexity of this module via decomposition. However, it is more efficient to apply the product relaxation. Before doing so, we need to confirm that the bracketed part of the first term in (4.8) is close to unity. This is shown next.

Taking the expectation of (4.7) in the limit as $n \to \infty$, we get

$$\frac{E[e_f^2(n|m - 1) + e_b^2(n - 1|m - 1)]}{E[S(n|m - 1)]} = 1 - (1 - \beta)^{M_L}. \quad (4.9)$$

If β is sufficiently small as compared to 1, then the left-hand side (LHS) of (4.9) is also very small as compared to unity. From (4.3), we see that

$$\frac{e_f^2(n|m - 1) + e_b^2(n - 1|m - 1)}{S(n|m - 1)} = \beta_m(n)(e_f^2(n|m - 1) + e_b^2(n - 1|m - 1)). \quad (4.10)$$

Comparing (4.10) and (4.9), we conclude that the bracketed term in (4.8) is indeed close unity. Thus, applying the product relaxation to (4.8) (see (3.5)), we get

$$k_m(n + M_L) = [1 - M_L\beta_m(n)(e_f^2(n|m - 1) + e_b^2(n - 1|m - 1))]k_m(n) + 2M_L\beta_m(n)e_f(n|m - 1) \cdot e_b(n - 1|m - 1). \quad (4.11)$$

Assuming that the inputs to an mth lattice stage are $e_f(n|m - 1)$ and $e_b(n|m - 1)$, the following set of equations completely describe the functional behavior of the first PIPSGLA architecture, referred to as PIPSGLA1:

$$k_m(n - M_S + M_L) = [1 - M_L\beta_m(n - M_S)(e_f^2(n - M_S|m - 1) + e_b^2(n - M_S - 1|m - 1))]k_m(n - M_S) + 2M_L\beta_m(n - M_S)e_f(n - M_S|m - 1) \cdot e_b(n - M_S - 1|m - 1) \quad (4.12)$$

$$S(n - M_S + M_L|m - 1) = (1 - \beta)^{M_L} S(n - M_S|m - 1) + e_f^2(n - M_S|m - 1) + e_b^2(n - M_S - 1|m - 1) \quad (4.13)$$

$$\beta_m(n - M_S) = \frac{1}{S(n - M_S|m - 1)} \quad (4.14)$$

$$e_f(n - M_S|m) = e_f(n - M_S|m - 1) - k_m(n)e_b(n - M_S - 1|m - 1) \quad (4.15a)$$

$$e_b(n - M_S|m) = e_b(n - M_S - 1|m - 1) - k_m(n)e_f(n - M_S|m - 1). \quad (4.15b)$$

The complete PIPSGLA1 is shown in Fig. 3, where we see that the increase in the hardware complexity equals two multipliers and $2(M_L - 1) + 2M_S$ latches for each stage. This is a remarkably low increase considering the available alternative architectures [8], [9].

An alternative pipelined architecture PIPSGLA2 can be obtained by simply introducing M_L latches into the recursive loops in Fig. 2. This corresponds to the application of the sum relaxation as defined by (3.4) (where a^M is replaced by a) to both (4.1) and (4.2). The equations describing PIPSGLA2 are

$$k_m(n - M_S + M_L) = [1 - \beta_m(n - M_S)(e_f^2(n - M_S|m - 1) + e_b^2(n - M_S - 1|m - 1))]k_m(n - M_S) + 2\beta_m(n - M_S)e_f(n - M_S|m - 1) \cdot e_b(n - M_S - 1|m - 1) \quad (4.16)$$

$$S(n - M_S + M_L|m - 1) = (1 - \beta)S(n - M_S|m - 1) + e_f^2(n - M_S|m - 1) + e_b^2 \cdot (n - M_S - 1|m - 1) \quad (4.17)$$

$$\beta_m(n - M_S) = \frac{1}{S(n - M_S|m - 1)} \quad (4.18)$$

$$e_f(n - M_S|m) = e_f(n - M_S|m - 1) - k_m(n) \cdot e_b(n - M_S - 1|m - 1) \quad (4.19a)$$

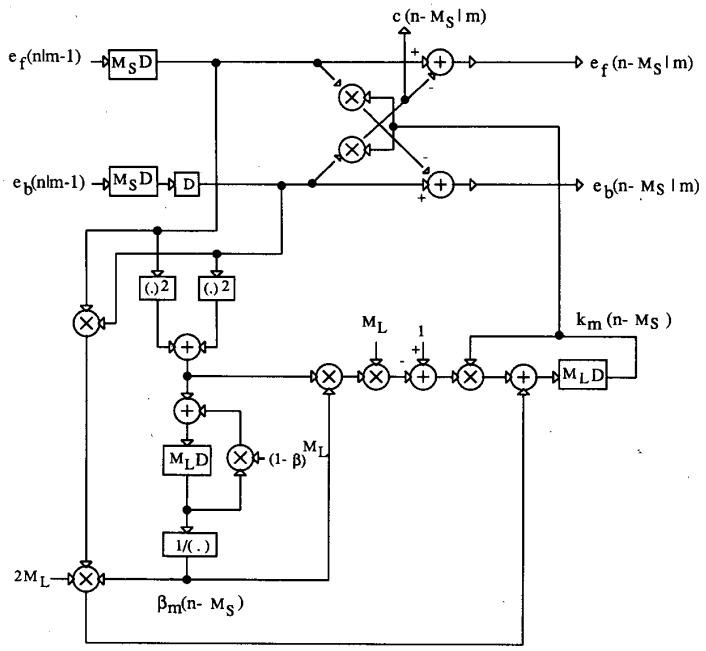


Fig. 3. The PIPSGLA1 architecture.

TABLE I
HARDWARE REQUIREMENTS

	Adders	Multipliers	Latches
SSGLA	$6N$	$10N$	$6N$
DLASGLA	$6N + 2N(M_L - 1)$	$10N + 4N(M_L - 1)$	$6N + 5N(M_L - 1) + 2NM_S$
PIPSGLA1	$6N$	$10N + 2N \operatorname{sgn}(M_L - 1)$	$6N + 2N(M_L - 1) + 2NM_S$
PIPSGLA2	$6N$	$10N$	$6N + 2N(M_L - 1) + 2NM_S$

$$e_b(n - M_S|m) = e_b(n - M_S - 1|m - 1) - k_m(n) \cdot e_f(n - M_S|m - 1). \quad (4.19b)$$

The architecture of PIPSGLA2 can be obtained from that of PIPSGLA1 (in Fig. 3) by the removal of multipliers with M_L as one of the inputs. Therefore, the increase in algorithm-level hardware for PIPSGLA2 are the $2(M_L - 1) + 2M_S$ pipelining latches only. As will be shown later via convergence analysis and simulations, the marginal complexity increase of PIPSGLA1 as compared with PIPSGLA2 is more than compensated for by the former's superior convergence time.

A comparison of the hardware requirements of SSGLA, DLASGLA (without decomposition), PIPSGLA1 and PIPSGLA2 has been done. The number of two-operand adders, two-operand multipliers and latches necessary to implement an N-stage lattice filter are shown in Table I. In Table I, the function $\operatorname{sgn}(x)$ is unity for x greater than zero and it equals zero otherwise. It can be seen that DLASGLA has the highest hardware penalty while PIPSGLA2 has the lowest. In addition, PIPSGLA1 and PIPSGLA2 have addition and multiplication complexities which are independent of M_L.

We now give an example to illustrate the speed-up due to pipelining. We assume that $T_m = 40$, $T_a = 20$, and $N = 1$. From (4.5), the clock period of SSGLA is $T_c = 220$. Using interstage pipelining with $M_S = 3$, the clock period can be reduced to 60 units. For higher speedups, we need to employ loop pipelining with relaxed look-ahead. Employing relaxed look-ahead with $M_L = 2$ and $M_S = 6$ results in a clock period of 40 units. The final retimed architecture, which can operate with a clock-period of 40 units, is shown in Fig. 4. The clock period could be reduced further to 30 units by using the pipelining latches in a uniform manner to pipeline and retime the multipliers and adders.

As mentioned before, convergence analysis needs to be done on the pipelined architectures resulting from relaxed look-ahead. The convergence analysis of PIPSGLA1 and PIPSGLA2 is presented in the next section.

V. Convergence Properties of the PIPSGLA Architectures

As the lattice stages are connected in a nonrecursive fashion, the interstage pipelining does not result in any change in the convergence behavior. This property favors

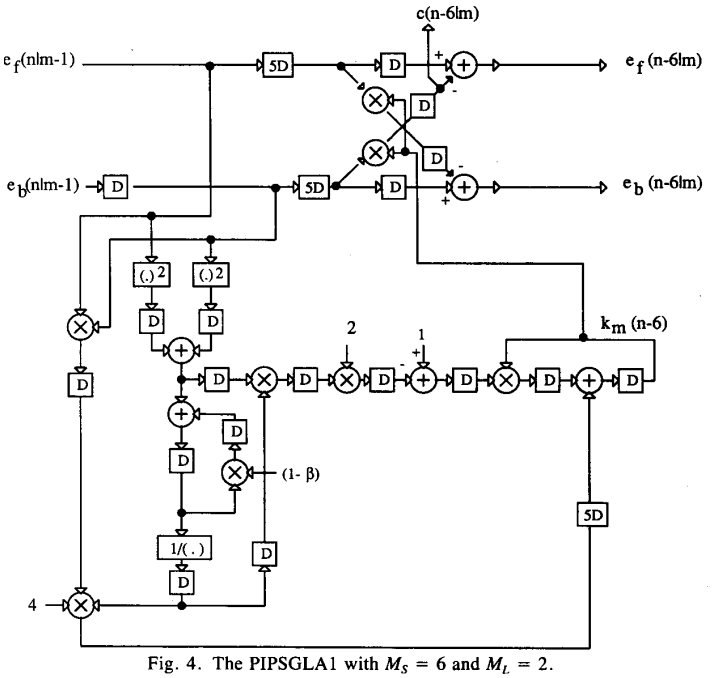


Fig. 4. The PIPSGLA1 with $M_S = 6$ and $M_L = 2$.

lattice structures as compared with transversal filters. In fact, with interstage pipelining, the optimum steady-state values of the reflection coefficients of PIPSGLA1, PIPSGLA2, and SSGLA are the same. Therefore, convergence analysis needs to be done only if the recursive loops are pipelined.

In this section, we analyze a single lattice stage assuming the inputs to be stationary. It is also assumed that the reflection coefficient $k_m(n_1)$ at time instance n_1 is independent of $e_f(n|m-1)$ and $e_b(n|m-1)$ for all $n < n_1$. This is known as the independence assumption. This assumption is more true if the number of stages in the lattice filter increases. In order to evaluate the higher order statistical expectations, we also assume that $e_f(n|m-1)$ and $e_b(n|m-1)$ are jointly Gaussian. This enables us to evaluate fourth-order statistics in terms of second-order ones. Due to the assumptions made, the analytical expressions should be used with caution. In particular, our aim for deriving these expressions is to obtain a comparative analysis of the various architectures and not to provide an absolute measure of their performance.

Most of the analysis proceeds along lines similar to those in [11]. While the details of this analysis are presented in the Appendixes, only the results are summarized here.

A. Bounds on β for Convergence

The bounds on β to guarantee the convergence of the reflection coefficients of PIPSGLA1 were found to be tighter as compared to those for SSGLA. In particular, the bounds on β for PIPSGLA1 were

$$0 \le \beta \le \frac{2}{M_L^2} \tag{5.1}$$

while those for SSGLA were

$$0 \le \beta \le 2. \tag{5.2}$$

Note that for $M_L = 1$, (5.1) and (5.2) are identical. This is in accordance with the fact that the convergence behavior of PIPSGLA1 would differ from that of SSGLA only if the recursive loops are pipelined, i.e., $M_L \ge 2$. In most practical applications, the actual value of β is much smaller than the upper bound in (5.1) or (5.2).

The bounds on β to guarantee the convergence of PIPSGLA2 are the same as that of SSGLA (see (5.2)). The details of the derivation of (5.1) and (5.2) are given in Appendix I.

For the normalized form of stochastic gradient lattice algorithms, it is difficult to find the bounds on β for the convergence of the output mean-squared error (MSE). However, it has been stated in [11] that it is empirically safe to assume the upper bound on β for the convergence of the output error to be one half of that for the convergence of the reflection coefficients. Thus, the upper bound on the values of β for the convergence of MSE should be one half of those suggested by (5.1) and (5.2).

B. Convergence Speed

The convergence time-constants (τ_{mse}) of the mean-squared error curve of SSGLA, DLASGLA, PIPSGLA1, and PIPSGLA2 are shown in Table II. The τ_{mse} for SSGLA, PIPSGLA1, and PIPSGLA2 are derived in Appendix II. In Table II, all architectures are assumed to have been pipelined at feedforward cutsets using interstage pipelining latches before applying loop pipelining using relaxed look-ahead. In addition, the clock period

TABLE II
SPEED

	τ_{mse}	t_{mse}
SSGLA	$1/2\,\beta$	$T/2\beta$
DLASGLA	$1/2\,\beta$	$T/2\,\beta\,M_L$
PIPSGLA1	$1/2\,\beta\,M_L$	$T/2\,\beta\,M_L^2$
PIPSGLA2	$M_L/2\,\beta$	$T/2\,\beta$

with interstage pipelining is denoted by T. Due to the nature of look-ahead, the τ_{mse} for DLASGLA is the same as that of SSGLA. It can be seen that the convergence speed of PIPSGLA1 is the highest while that of PIPSGLA2 is the lowest. As the level of loop pipelining M_L increases, the convergence speed of PIPSGLA1 improves further while that of PIPSGLA2 degrades. The tracking behavior in a nonstationary environment is governed largely by τ_{mse}. Therefore we should expect PIPSGLA1 to track the best.

Due to the differences in the clock period of the architectures under consideration, it is instructive to compare the convergence time in seconds t_{mse}. The latter can be obtained by multiplying the clock period of a given architecture with its τ_{mse}. In Table II, we list the t_{mse}'s for each of the architectures under consideration. Again PIPSGLA1 has the lowest t_{mse}, followed by DLASGLA, PIPSGLA2, and SSGLA.

C. Adaptation Accuracy

The adaptation accuracy of an adaptive algorithm is defined in terms of its misadjustment, which is defined below:

$$\mathfrak{M} = \frac{E(J(\infty)) - J_{\min}}{J_{\min}} \tag{5.3}$$

where $J(n)$ is the mean-squared error at time instant n, and $E(J(n))$ is its average. The notation $J_{\min}$ refers to the minimum mean-squared error, which would be obtained if the reflection coefficient $k_m(n)$ equalled the optimal value $k_{m,\text{opt}}$.

The misadjustment analysis is carried out in detail in Appendix III. The misadjustment of the SSGLA ($\mathfrak{M}_S$) and PIPSGLA2 ($\mathfrak{M}_{PIP2}$) were found to be the same. This misadjustment is given by

$$\mathfrak{M}_S = \mathfrak{M}_{PIP2} = \frac{\beta(1 - k_{m,\text{opt}}^2)}{(2 - \beta)(2 + k_{m,\text{opt}}^2)}. \tag{5.4}$$

The convergence speed of PIPSGLA2 is much slower than that of SSGLA. Therefore, PIPSGLA2 requires more iterations to attain the final adaptation accuracy. As the convergence speed of PIPSGLA1 is faster than that of SSGLA, we should expect its accuracy to be degraded as compared with SSGLA. This can be seen in the expression for misadjustment of PIPSGLA1 ($\mathfrak{M}_{PIP1}$) shown below:

$$\mathfrak{M}_{PIP1} = \frac{M_L\gamma(1 - k_{m,\text{opt}}^2)}{(2 - M_L\gamma)(2 + k_{m,\text{opt}}^2)} \tag{5.5}$$

where γ is defined as

$$\gamma = 1 - (1 - \beta)^{M_L}. \tag{5.6}$$

Note that (5.5) reduces to (5.4) for $M_L = 1$.

In order to estimate the factor by which $\mathfrak{M}_{PIP1}$ is greater than $\mathfrak{M}_S$, we assume that β is much smaller than 1. Then, the ratio α of $\mathfrak{M}_{PIP1}$ to $\mathfrak{M}_S$ can be shown to be

$$\alpha = M_L^2\left(\frac{(2 - \beta)}{(2 - M_L^2\beta)}\right) \approx M_L^2\left(1 + \frac{(M_L^2 - 1)\beta}{2}\right). \tag{5.7}$$

The value of α can be seen to equal unity for $M_L = 1$, which is to be expected. As β is very small as compared to one, α in most cases can be approximated by M_L^2. Thus, the misadjustment of PIPSGLA1 is M_L^2 times that of SSGLA.

VI. PIPELINED ADPCM VIDEO CODEC

In this section, we demonstrate an application of the pipelined architectures presented in Section IV. We employ the proposed lattice filter algorithms to develop a pipelined ADPCM codec architecture for high-speed image compression applications. Simulation results on compression of real images is given in Section VII.

Employing (4.4(a)) with $m = N$, we iteratively substitute for $e_f(n|N)$ to get

$$e_f(n|N) = e_f(n|0) - \sum_{i=1}^{N} k_i(n)e_b(n - 1|i - 1) = s(n) - \sum_{i=1}^{N} c(n|i - 1). \tag{6.1}$$

As $e_f(n|0) = s(n)$ is the input to the lattice filter and $e_f(n|N)$ is the Nth order prediction error, therefore the summation term in (6.1) is the predicted value $\hat{s}(n)$ of the input $s(n)$.

With the predicted value of the input signal thus available, we can construct a pipelined ADPCM coder architecture. Before doing so it is essential to decide which image pixels to employ for prediction. The input is assumed to be in a row-by-row raster-scan format. Denoting an image pixel in the ith row and the jth column by $x(i, j)$, we depict a conventional prediction topology in Fig. 5(a). The current pixel $x(i, j)$ (dark circle) is predicted from $x(i, j - 1)$, $x(i - 1, j)$, and $x(i - 1, j - 1)$. Employing $x(i, j - 1)$ for prediction of $x(i, j)$ is detrimental to pipelining as these two pixels are input consecutively. For pipelining by M_S levels, it is necessary to predict $x(i, j)$ with $x(i, j - M_S - 1)$, $x(i - 1, j)$, and $x(i - 1, j - 1)$ as shown in Fig. 5(b) (for $M_S = 2$). As M_S increases the correlation between $x(i, j - M_S - 1)$ and $x(i, j)$ is reduced, which results in inaccurate prediction. In order to achieve the dual objectives of accurate prediction and high pipelining levels, we employ the pixels $x(i - 1, j + 1)$, $x(i - 1, j)$, and $x(i - 1, j - 1)$ for predicting $x(i, j)$ (see Fig. 5(c)). Note that all three pixels have been pro-

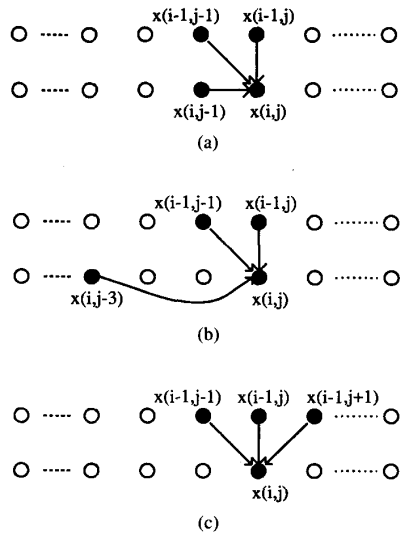


Fig. 5. Pixels employed for prediction.

cessed much before $x(i, j)$ and therefore do not place any limitation on the level of pipelining. In addition, they also are the topological neighbors of $x(i, j)$ and therefore would accurately predict $x(i, j)$.

The equations describing the pipelined Nth-order (N is assumed to be odd) ADPCM coder can now be formulated as follows:

$$\hat{s}(n + M_S) = \sum_{i=1}^{N} c\left(n - N\,cols + M_S + \frac{N-1}{2}\middle| i - 1\right) \quad (6.2a)$$

$$c(n|m) = k_m(n)e_b(n-1|m-1) \quad (6.2b)$$

$$e(n) = s(n) - \hat{s}(n) \quad (6.2c)$$

$$e_q(n) = Q[e(n)] \quad (6.2d)$$

$$\tilde{s}(n) = \hat{s}(n) + e_q(n) \quad (6.2e)$$

where $s(n)$ is the raster scan input formed from the image pixels, $N\,cols$ is the number of pixels in each row of the image, $\hat{s}(n)$ is the reconstructed signal, $e(n)$ is the prediction error, and $e_q(n)$ is the quantized value ($Q[.]$ representing the quantization operator with R bits). Note that M_S now denotes the sum of all the interstage pipelining latches. The order updates and the adaptation equations can be obtained from (4.12)–(4.15) (if PIPSGLA1 is employed) or (4.16)–(4.19) (if PIPSGLA2 is employed). In both cases, the time index for the order updates would be $n - N\,cols + M_S + (N-1)/2$, while that of the adaptation equations would be $n - N\,cols + M_S + (N-1)/2 + M_L$. These time indices reflect the fact that the image pixels employed for prediction belong to the previous row. The equations for the serial ADPCM coder can be obtained by substituting $M_S = 0$ and $M_L = 1$ in (6.2), the order updates and the adaptation equations.

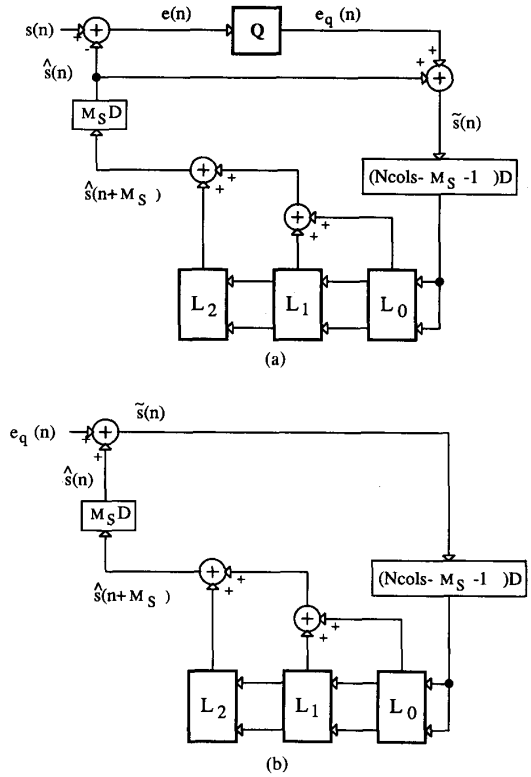


Fig. 6. Pipelined ADPCM codec: (a) coder and (b) decoder.

TABLE III
SPEEDUP VERSUS M_S AND M_L

Speed-up	M_S	M_L	L_c
1	0	1	0
2	4	1	1
4	13	1	2
5	16	2	2
10	29	3	4
13	36	3	5
15	44	4	6
20	57	5	8

The pipelined ADPCM coder architecture for $N = 3$ is shown Fig. 6(a), where Q represents the quantizer and the blocks L_0, L_1, and L_2 are the lattice stages. The $c(n|m)$ outputs of each stage are tapped (from the top of each block in Fig. 6(a)) and summed according to (6.2(a)). In a practical implementation, the latches M_S would be retimed to pipeline the lattice stages, the adders, and the quantizer. The decoder architecture is shown in Fig. 6(b).

An interesting fact to be noted is that the latency of the coder (L_c) is equal to the number of latches required to pipeline the quantizer and the input adder. This number is usually much less than the speedup M. This can be seen in Table III, where the values of M_S and M_L necessary for achieving a given speedup M (with a third-order

PIPSGLA2 predictor) are shown. From Table III, we can also see that for speedups greater than 4, loop pipelining with relaxed look-ahead is necessary.

VII. Simulation Results

We present the results of simulations carried out on SSGLA an PIPSGLA architectures. In the first experiment (Experiment A), we consider an AR process as the input to the lattice predictor. In Experiment B, we employ the pipelined ADPCM video codec developed in Section VI for compression of real images.

A. Experiment A: Stationary Case

The purpose of this experiment is to qualitatively verify the analytical expression describing the convergence behavior, presented in Section V. As mentioned in Section V, absolute quantitative verification is not possible due to the restrictive nature of the assumptions made during the analysis.

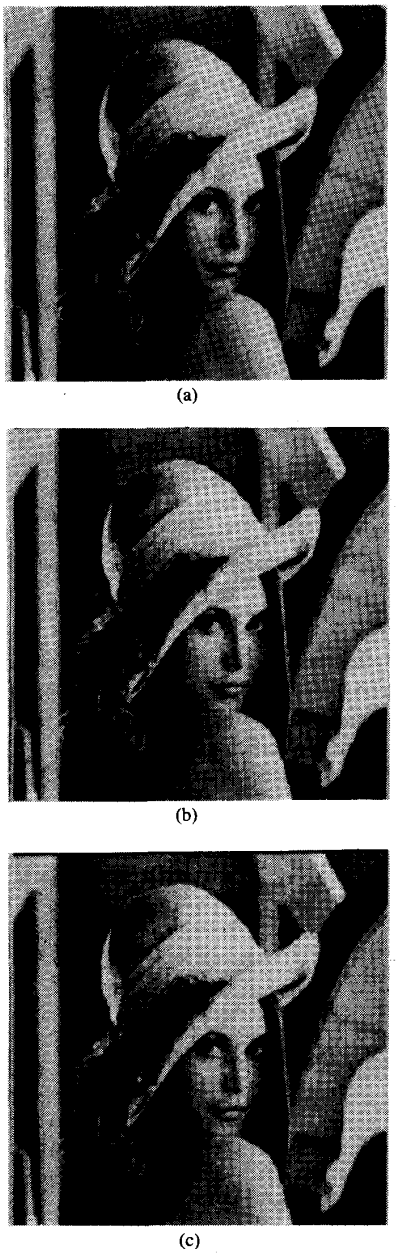
In this experiment, the PIPSGLA1 and PIPSGLA2 architectures have been simulated with $M_S = 6$ and $M_L = 2$. The input to the filter is taken as a second-order AR process with generating filter poles at $0.4875 \pm j0.8440$. The mean squared error (MSE), which is taken as the sum of forward and backward prediction error powers, is averaged over 32 independent trials and 400 iterations. The value of β is equal to 0.005. The convergence time-constants τ_{mse} for SSGLA, PIPSGLA1, and PIPSGLA2 were found to be 90, 47, and 108 iterations, respectively. As predicted by the expressions in Table I, the τ_{mse} for SSGLA is twice that of PIPSGLA1. Comparison of the convergence time in absolute units t_{mse} shows that PIPSGLA1 has the fastest convergence with $t_{\text{mse}} = 1880$ units. However, the t_{mse} for PIPSGLA2 is 4320.

The misadjustment for PIPSGLA1 was found to be 9 times that of SSGLA. The convergence analysis predicts (see (5.7)) an increase by a factor of 4 (i.e., $M_L^2 = 4$). This discrepancy would be reduced if the number of stages of the filter increases.

B. Experiment B: Pipelined ADPCM Codec

In this experiment, we have chosen the PIPSGLA2 as the lattice predictor. The input is scaled to lie between $+1$ and -1. The quantizer is assumed to be uniform and fixed with a dynamic range of 0.4. The value of β is 0.009 for all simulations. The order of the predictor is 3.

The original image has a frame-size of 256×256, with 8 b per pixel (bpp) (see Fig. 7(a)). The pipelined ADPCM codec was employed to code and then reconstruct this image for different values of the speed-up. As discussed in Section V, the PIPSGLA2 has a degraded convergence speed as compared to SSGLA. This implies that as the speedup increases the signal-to-noise ratio at the codec output would decrease. Two sets of simulations with $R = 3$ and $R = 4$ were done for different speedups. The values for M_S and M_L were obtained from Table III. In Fig. 8, we show the trend in SNR as the speedup increases. It can be seen that for a speedup as high as 20, the loss in SNR (for $R = 3$) is 0.1 dB. For $R = 4$, this loss is 0.27 dB. Thus, we conclude that the SNR loss due to the application of relaxed look-ahead is minimal and leads to no perceptual degradation of image quality.



(a) (b) (c)

Fig. 7. Images: (a) original, (b) serial codec output, and (c) pipelined codec output.

In Fig. 7(b), we show the reconstructed image ($R = 3$) for the serial ADPCM codec, which is equivalent to the pipelined ADPCM codec with a speedup of unity. From Fig. 8, the SNR for the serial ADPCM codec is 23.4 dB. The reconstructed image for a speedup of 20 is shown in Fig. 7(c), which has an SNR of 23.3 dB. Note that there

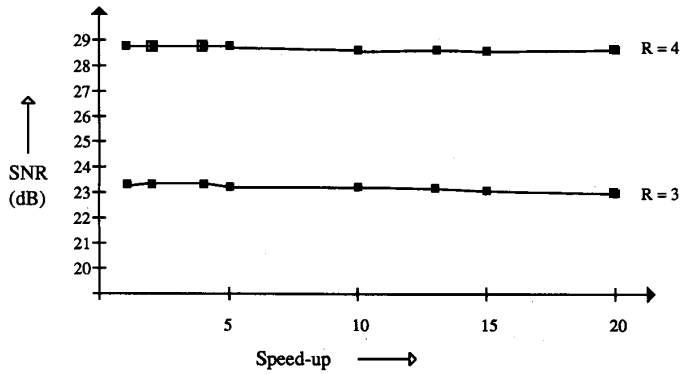


Fig. 8. SNR versus speedup.

is hardly any perceptual difference between the outputs of the serial (Fig. 7(b)) and the pipelined architectures (Fig. 7(c)).

VIII. Conclusions

The relaxed look-ahead [10] is presented as an attractive technique for pipelining adaptive filters. The stochastic gradient lattice filter [11] has been pipelined via the application of relaxed look-ahead. As the relaxed look-ahead is a one-to-many mapping, therefore two pipelined architectures PIPSGLA1 and PIPSGLA2 are proposed. The convergence analysis indicates minimal degradation in the convergence behavior. A pipelined lattice ADPCM codec is then developed. Simulations verifying the convergence analysis results and application to video predictive coding demonstrate the usefulness of high-speed or low-power lattice adaptive filtering.

We have shown that different forms of relaxations result in different adaptation characteristics. While two examples of relaxed look-ahead pipelined adaptive filter architectures have been proposed and analyzed, many other architectures can be systematically derived using other relaxations. For example, use of the coefficient update equation (4.16) and energy update equation (4.13) results in an architecture with same time constant as the SSGLA and misadjustment error M_L times higher than SSGLA. If we use the coefficient adaptation equation (4.16) and the energy update equation obtained by replacing $(1-\beta)^{M_L}$ by $(1-\beta)^{\sqrt{M_L}}$ in (4.13), then both the time constant and the misadjustment error of the resulting architecture are $\sqrt{M_L}$ times higher than the SSGLA architecture. By using sum relaxation either in the form of (3.3) or (3.4) or other forms and by using product relaxation in different forms, several architectures with varying adaptation characteristics can be designed. It can be noted that the product of the time constant and the misadjustment error in any pipelined architecture is M_L times higher than the SSGLA. Thus, an appropriate pipelined architecture can be selected for desired tradeoff in time constant and misadjustment error.

This work is a continuation of our endeavor to develop design methodologies for inherently pipelinable digital signal processing algorithms. Work in fixed-coefficient, inherently concurrent direct-form and lattice recursive digital filters [19] has been successful. Further work is being directed towards the application of relaxed look-ahead to the adaptive decision-feedback equalizers and predictive vector quantizers. Modifying the relaxed look-ahead to convert it into a one-to-one mapping, where the final architecture is optimized in some sense (i.e., with respect to the hardware overhead, or the convergence speed or the adaptation accuracy) would be of interest.

Appendix I
Derivation of the Bounds on β for Convergence

In order to make this analysis tractable we invoke the independence assumption, i.e., the reflection coefficient $k_m(n)$ is independent of the inputs $e_f(j|m-1)$ and $e_b(j|m-1)$ for all $j = 0, \cdots, n-1$.

A. Bounds on β for PIPSGLA2

For PIPSGLA2, we take the expectation of (4.16), with the independence assumption, to get

$$\begin{aligned} E[k_m(n+M_L)] &= [1 - E[\beta_m(n)(e_f^2(n|m-1) + e_b^2(n-1|m-1))]]E[k_m(n)] \\ &\quad + 2E[\beta_m(n)e_f(n|m-1)e_b(n-1|m-1)] \\ &\approx \left[1 - \frac{E[e_f^2(n|m-1) + e_b^2(n-1|m-1)]}{E[S(n|m-1)]}\right] E[k_m(n)] \\ &\quad + 2\frac{E[e_f(n|m-1)e_b(n-1|m-1)]}{E[S(n|m-1)]} \end{aligned} \tag{A1.1}$$

where $E[.]$ represents the expectation operator.

Starting with the order-update equations (4.19), we can compute the gradient of $E[e_f^2(n|m-1) + e_b^2(n|m-1)]$ with respect to $k_m(n)$. The optimum value of $k_m(n)$ (denoted as $k_{m,\text{opt}}$) obtained by equating this gradient to zero is given by

$$k_{m,\text{opt}} = \frac{2E[e_f(n|m-1)e_b(n-1|m-1)]}{E[e_f^2(n|m-1) + e_b^2(n-1|m-1)]}. \quad \text{(A1.2)}$$

Taking the expectation as $n \to \infty$ of (4.17), we get

$$\beta = \frac{E[e_f^2(n|m-1) + e_b^2(n-1|m-1)]}{E[S(n|m-1)]}. \quad \text{(A1.3)}$$

Using (A1.2) and (A1.3), it is easy to write (A1.1) as follows:

$$E[k_m(n+M_L)] - k_{m,\text{opt}} = [1-\beta](E[k_m(n)] - k_{m,\text{opt}}). \quad \text{(A1.4)}$$

Hence, for convergence, it is sufficient that

$$-1 \le 1-\beta \le 1 \quad \text{(A1.5)}$$

which is equivalent to

$$0 \le \beta \le 2. \quad \text{(A1.6)}$$

Note that (A1.6) also gives the bounds on β for SSGLA. This is because substitution of $M_L = 1$ in (A1.4) results in the equation for the convergence of the reflection coefficients of the SSGLA.

B. Bounds on β for PIPSGLA1

Taking the expected value of both sides of (4.12), we get

$$\begin{aligned} E[k_m(n+M_L)] &= (1 - M_L E[\beta_m(n)(e_f^2(n|m-1) + e_b^2(n-1|m-1))])E[k_m(n)] \\ &\quad + 2M_L E[\beta_m(n)e_f(n|m-1)e_b(n-1|m-1)] \\ &\approx \left(1 - M_L \frac{E[e_f^2(n|m-1) + e_b^2(n-1|m-1)]}{E[S(n|m-1)]}\right) E[k_m(n)] \\ &\quad + 2M_L \frac{E[e_f(n|m-1)e_b(n-1|m-1)]}{E[S(n|m-1)]} \end{aligned} \quad \text{(A1.7)}$$

where $E[.]$ represents the expectation operator.

As $k_{m,\text{opt}}$ for PIPSGLA1 is also given by (A1.2), therefore, we may write (A1.7) as

$$E[k_m(n+M_L)] - k_{m,\text{opt}} = [1 - M_L(1-(1-\beta)^{M_L})](E[k_m(n)] - k_{m,\text{opt}}). \quad \text{(A1.8)}$$

For the LHS of (A1.8) to converge to zero it is sufficient that

$$-1 \le 1 - M_L(1-(1-\beta)^{M_L}) \le 1 \quad \text{(A1.9)}$$

simplifying the inequality (A1.9) results in the following bounds on β:

$$0 \le \beta \le \frac{2}{M_L^2}. \quad \text{(A1.10)}$$

Appendix II
Derivation of the Convergence Time Constants

We first derive the convergence time-constants τ_k for the convergence of the reflection coefficients.

A. τ_k for SSGLA

Rewriting (A1.4) with $M_L = 1$, we get

$$E[k_m(n+1)] - k_{m,\text{opt}} = [1-\beta](E[k_m(n)] - k_{m,\text{opt}}). \quad \text{(A2.1)}$$

Denoting $E[k_m(n)] - k_{m,\text{opt}}$ by $\nu(n)$, we get the following difference equation:

$$\nu(n+1) = (1-\beta)\nu(n). \quad \text{(A2.2)}$$

Iterating (A2.2) n times, we get

$$\nu(n+1) = (1-\beta)^{n+1}\nu(0). \quad \text{(A2.3)}$$

Equating the RHS (right-hand side) of (A2.2) to a decaying exponential with time-constant τ_k, we get

$$\tau_k = \frac{-1}{\ln(1-\beta)} \approx \frac{1}{\beta}. \quad \text{(A2.4)}$$

Thus, the reflection coefficient learning curve is inversely proportional to β.

B. τ_k for PIPSGLA2

Replacing $E[k_m(n)] - k_{m,\text{opt}}$ by $\nu(n)$ in (A1.4), we get

$$\nu(n+M_L) = (1-\beta)\nu(n). \quad \text{(A2.5)}$$

The characteristic equation $F(\gamma)$ of (A2.5) is

$$\gamma^{M_L} - (1-\beta) = 0. \quad \text{(A2.6)}$$

Hence, the solution to (A2.5) is given by

$$\nu(n) = \sum_{i=0}^{M_L-1} (\nu)_i^n C_i \quad \text{(A2.7)}$$

where C_i's are constants to determined by the specified initial conditions, and $(\nu)_i (i = 0, 1, \cdots, M_L - 1)$ are the roots of (A2.6), given by

$$(\nu)_i = (1-\beta)^{1/M_L} \exp\left\{\frac{2\pi i}{M_L}\right\}. \quad \text{(A2.8)}$$

By fitting (A2.7) with a decaying exponential with time-

constant τ_k, we get

$$\tau_k = \frac{-M_L}{\ln(1-\beta)} \approx \frac{M_L}{\beta}. \quad \text{(A2.9)}$$

C. τ_k for PIPSGLA1

Repeating the analysis done for PIPSGLA2 in the previous subsection on (A1.8) we get

$$\nu(n+M_L) = (1 - M_L(1-(1-\beta)^{M_L}))\nu(n). \quad \text{(A2.10)}$$

The solution to the characteristic equation of (A2.10) is given by

$$\nu(n) = \sum_{i=0}^{M_L-1} (\nu)_i^n C_i \quad \text{(A2.11)}$$

where again C_i's are to determined by the initial conditions, and $(\nu)_i (i = 0, 1, \cdots, M_L - 1)$ are the roots of the characteristic equation of (A2.10), and are given by

$$(\nu)_i = (1 - M_L(1-(1-\beta)^{M_L}))^{1/M_L} \exp\left\{\frac{2\pi i}{M_L}\right\}. \quad \text{(A2.12)}$$

Fitting a decaying exponential envelope we get the required time constant as

$$\tau_k = \frac{-M_L}{\ln(1 - M_L(1-(1-\beta)^{M_L}))} \approx \frac{-M_L}{\ln(1-M_L^2\beta)} \approx \frac{1}{\beta M_L}. \quad \text{(A2.13)}$$

In order to get the convergence time constants for the output error we square the order-update equation (4.4(a)) or (4.15(a)) or (4.19(a)). Then taking the expectation of both sides, we get

$$\begin{aligned} E[e_f^2(n|m)] &= [1 + E[k_m^2(n)] - 2k_{m,\text{opt}}E[k_m(n)]] \cdot E[e_f^2(n|m-1)] \\ &= [1 + (E[k_m(n)] - k_{m,\text{opt}})^2 - k_{m,\text{opt}}^2 \\ &\quad + \text{var}(k_m(n))]E[e_f^2(n|m-1)] \\ &= [1 - k_{m,\text{opt}}^2 + \text{var}(k_m(n)) + \nu^2(n)] \\ &\quad \cdot E[e_f^2(n|m-1)]. \end{aligned} \quad \text{(A2.14)}$$

Assume that the variance of $k_m(n)$ $(\text{var}(k_m(n)))$ has a constant value equal to var $(k_m(\infty))$ and that the input to the lattice stage is stationary. Therefore, we see from (A2.14) that the convergence speed of $E[e_f^2(n|m)]$ is governed by the rate at which $\nu^2(n)$ approaches zero. Thus, the mean-squared error-time-constant τ_{mse} would be half of that of τ_k.

Appendix III
Derivation of the Misadjustment Expressions

In order to get closed form expressions for the misadjustment of SSGLA, PIPSGLA1, and PIPSGLA2, we need to make certain simplifying assumptions. In particular, the independence assumptions outlined in Section V, are often invoked. In addition, let

$$\sigma_e^2 = E[e_f^2(n|m-1)] = E[e_b^2(n|m-1)] \quad \text{(A3.1)}$$

represent the input signal power. Before the misadjustment expressions are derived, we develop some preliminary expressions for fourth-order statistics in terms of second-order ones. All the expectations in this Appendix are in the limit as $n \to \infty$.

A. Preliminaries

If X_1, X_2, X_3, and X_4 are jointly Gaussian variables then the following is true:

$$E[X_1X_2X_3X_4] = E[X_1X_2]E[X_3X_4] + E[X_1X_3]E[X_2X_4] + E[X_1X_4]E[X_2X_3]. \quad \text{(A3.2)}$$

Employing (A3.2) and the expression for $k_{m,\text{opt}}$ ((A1.2)), it is trivial to show the validity of the following:

$$E[e_f(n|m-1)e_b(n-1|m-1)] = k_{m,\text{opt}}\sigma_e^2 \quad \text{(A3.3a)}$$

$$E[e_f^2(n|m-1)e_b^2(n-1|m-1)] = (1 + 2k_{m,\text{opt}}^2)\sigma_e^4 \quad \text{(A3.3b)}$$

$$E[e_f^3(n|m-1)e_b(n-1|m-1) + e_b^3(n|m-1) \cdot e_f(n-1|m-1)] = 6k_{m,\text{opt}}\sigma_e^4 \quad \text{(A3.3c)}$$

$$E[(e_f^2(n|m-1) + e_b^2(n-1|m-1))^2] = (8 + 4k_{m,\text{opt}}^2)\sigma_e^4. \quad \text{(A3.3d)}$$

If we consider (A2.14) in the limit as $n \to \infty$, we get

$$\begin{aligned} E[e_f^2(n|m)] &= [1 - k_{m,\text{opt}}^2 + \text{var}(k_m(\infty))]\sigma_e^2 \\ &= (1 - k_{m,\text{opt}}^2)\sigma_e^2 + \text{var}(k_m(\infty))\sigma_e^2 \\ &= J_{\min} + E[J_{ex}(\infty)]. \end{aligned} \quad \text{(A3.4)}$$

Thus, the misadjustment $\mathfrak{M}$ for a lattice stage is given by

$$\mathfrak{M} = \frac{E[J_{ex}(\infty)]}{J_{\min}} = \frac{\text{var}(k_m(\infty))}{(1 - k_{m,\text{opt}}^2)}. \quad \text{(A3.5)}$$

Hence, all that remains to be done is to calculate the steady-state variance of the reflection coefficient var $(k_m(\infty))$. To simplify notation in the following analysis, we represent $e_f(n|m-1)$ by e_f, $e_b(n|m-1)$ by e_b and $S(n|m-1)$ by S.

B. Misadjustment Expression for SSGLA

Taking the expectation of (4.2) in the limit as $n \to \infty$, it can be shown that

$$\frac{E[(e_f^2 + e_b^2)]}{E(S)} = \beta. \tag{A3.6}$$

Finally, we need to derive the expression for $E[S^2(n|m - 1)]$ by squaring (4.2) first and then taking its expectation. This gives

$$E[S^2] = (1 - \beta)^2 E[S^2] + 2(1 - \beta)E[S(e_f^2 + e_b^2)] + E[(e_f^2 + e_b^2)^2]. \tag{A3.7}$$

Multiplying (4.2) by $S(n|m - 1)$ and taking its expectation, we get

$$E[S(e_f^2 + e_b^2)] = \beta E[S^2]. \tag{A3.8}$$

If we now substitute (A3.8) and (A3.3(d)) into (A3.7). we get the desired expression

$$E[S^2] = \frac{(8 + 4k_{m,\text{opt}}^2)}{\beta^2}\sigma_e^4. \tag{A3.9}$$

We are now in a position to compute the steady-state variance of the reflection coefficient. Subtracting $k_{m,\text{opt}}$ from both sides of (4.1) and squaring, we get

$$\begin{aligned}(k_m(n + 1) - k_{m,\text{opt}})^2 &= \left(1 - \frac{e_f^2 + e_b^2}{S}\right)^2 k_m^2(n) + \frac{4e_f^2 e_b^2}{S^2} + k_{m,\text{opt}}^2 \\ &+ \frac{4e_f e_b}{S}\left(1 - \frac{e_f^2 + e_b^2}{S}\right) k_m(n) \\ &- 2k_{m,\text{opt}}\left(1 - \frac{e_f^2 + e_b^2}{S}\right) \\ &\cdot k_m(n) - \frac{4k_{m,\text{opt}} e_f e_b}{S}.\end{aligned} \tag{A3.10}$$

We take the expectation of (A3.10) to get

$$\begin{aligned}\text{var}\,(k_m(\infty)) &= (1 - 2\beta + \beta^2)E[k_m^2(n)] \\ &+ \frac{4\beta^2(1 + 2k_{m,\text{opt}}^2)}{(8 + 4k_{m,\text{opt}}^2)} \\ &- k_{m,\text{opt}}^2 - \frac{24\beta^2 k_{m,\text{opt}}^2}{(8 + 4k_{m,\text{opt}}^2)} \\ &+ 2\beta k_{m,\text{opt}}^2.\end{aligned} \tag{A3.11}$$

Adding and subtracting $(1 - 2\beta + \beta^2)k_{m,\text{opt}}^2$ on the RHS of (A3.11), we get

$$\begin{aligned}\text{var}\,(k_m(\infty)) &= (1 - 2\beta + \beta^2)\,\text{var}\,(k_m(\infty)) + \beta^2 k_{m,\text{opt}}^2 \\ &+ \frac{4\beta^2(1 + 2k_{m,\text{opt}}^2)}{(8 + 4k_{m,\text{opt}}^2)} - \frac{24\beta^2 k_{m,\text{opt}}^2}{(8 + 4k_{m,\text{opt}}^2)}.\end{aligned} \tag{A3.12}$$

It is now possible to simplify (A3.12) to get the analytical expression for var $(k_m(\infty))$ shown below:

$$\text{var}\,(k_m(\infty)) = \frac{\beta(1 - k_{m,\text{opt}}^2)^2}{(2 - \beta)(2 + k_{m,\text{opt}}^2)}. \tag{A3.13}$$

From (A3.5) and (A3.13), the misadjustment of SSGLA is given by

$$\mathfrak{M}_S = \frac{\beta(1 - k_{m,\text{opt}}^2)}{(2 - \beta)(2 + k_{m,\text{opt}}^2)}. \tag{A3.14}$$

As all the expectations are taken in the limit as $n \to \infty$, therefore, the analysis for SSGLA is also applicable to PIPSGLA2. Hence, the misadjustment expression for PIPSGLA2 is the same as that for SSGLA.

C. Misadjustment Expression for PIPSGLA1

The deviation of $\mathfrak{M}_{\text{PIP1}}$ proceeds in a manner analogous to that of $\mathfrak{M}_S$. The expectation of (4.13) in the limit as $n \to \infty$ gives

$$\frac{E[(e_f^2 + e_b^2)]}{E(S)} = 1 - (1 - \beta)^{M_L} = \gamma. \tag{A3.15}$$

The expression for $E[S^2(n|m - 1)]$ is given by

$$E[S^2] = \frac{(8 + 4k_{m,\text{opt}}^2)}{\gamma^2}\sigma_e^4. \tag{A3.16}$$

The rest of the derivation is identical to that shown in the preceding subsection, and is not repeated. Subtracting $k_{m,\text{opt}}$ from both sides of (4.13), squaring and then taking the expectation, we get

$$\text{var}\,(k_m(\infty)) = \frac{M_L\gamma(1 - k_{m,\text{opt}}^2)^2}{(2 - M_L\gamma)(2 + k_{m,\text{opt}}^2)}. \tag{A3.17}$$

Therefore, the adjustment of PIPSGLA1 is given by

$$\mathfrak{M}_{\text{PIP1}} = \frac{M_L\gamma(1 - k_{m,\text{opt}}^2)}{(2 - M_L\gamma)(2 + k_{m,\text{opt}}^2)}. \tag{A3.18}$$

Thus, (A3.14) and (A3.18) are the desired expressions for the misadjustments of SSGLA and PIPSGLA1, respectively. The misadjustment for PIPSGLA2, as mentioned before, is the same as that of SSGLA.

ACKNOWLEDGMENT

The authors acknowledge the constructive criticisms of an anonymous reviewer which were instrumental in improving the contents of this paper.

REFERENCES

[1] K. K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proc. IEEE*, vol. 77, pp. 1879–1895, Dec. 1989.

[2] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters—Part I: Pipelining using scattered look-ahead and decomposition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1099–1117, July 1989.

[3] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters—Part II: Pipelined incremental block filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1118–1134, July 1989.

[4] K. K. Parhi, "Pipelining in dynamic programming architectures," *IEEE Trans. Signal Processing*, vol. 39, pp. 1442–1450, June 1991.

[5] K. K. Parhi, "Pipelining in algorithms with quantizer loops," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 745–754, July 1991.

[6] K. K. Parhi, "High-speed VLSI architectures for Huffman and Viterbi decoders," *IEEE Trans. Circuits Syst. II: Analog and Digital Signal Processing*, vol. 39, pp. 385–391, June 1992.

[7] H.-D. Lin and D. G. Messerschmitt, "Finite state machine has unlimited concurrency," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 465–475, May 1991.

[8] K. K. Parhi and D. G. Messerschmitt, "Concurrent cellular VLSI adaptive filter architectures," *IEEE Trans. Circuits Syst.*, vol. 34, pp. 1141–1151, Oct. 1987.

[9] T. Meng and D. G. Messerschmitt, "Arbitrarily high sampling rate adaptive filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 35, pp. 455–470, Apr. 1987.

[10] N. R. Shanbhag and K. K. Parhi, "A pipelined LMS adaptive filter architecture," in *Proc. 25th Asilomar Conf. Signals, Syst., Comput.*, Pacific Grove, CA, Nov. 1991, pp. 668–672.

[11] M. L. Honig and D. G. Messerschmitt, *Adaptive Filters: Structures, Algorithms, and Applications*. Boston: Kluwer, 1984.

[12] N. R. Shanbhag and K. K. Parhi, "A pipelined adaptive lattice filter architecture: Theory and applications," in *Proc. EUSIPCO '92*, Brussels, Belgium, Aug. 1992.

[13] N. R. Shanbhag and K. K. Parhi, "A high-speed architecture for ADPCM coder and decoder," in *Proc. 1992 IEEE Int. Symp. Circuits Syst.*, San Diego, CA, pp. 1499–1502.

[14] G. A. Clark, S. K. Mitra, and S. R. Parker, "Block implementation of adaptive digital filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 29, pp. 744–752, June 1981.

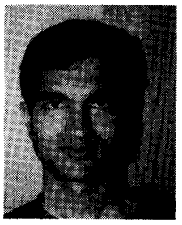
[15] S. Theodoridis, "Pipeline architecture for block adaptive LS FIR filtering and prediction," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 81–90, Jan. 1990.

[16] J. M. Cioffi, P. Fortier, S. Kasturia, and G. Dudevoir, "Pipelining the decision feedback equalizer," presented at the *IEEE DSP Workshop*, 1988.

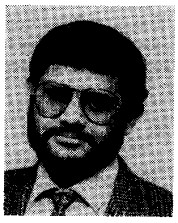
[17] A. Gatherer and T. H.-Y. Meng, "High sampling rate adaptive decision feedback equalizer," in *Proc. IEEE ICASSP '90*, pp. 909–912.

[18] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, pp. 473–484, Apr. 1992.

[19] J.-G. Chung and K. K. Parhi, "Design of pipelined lattice IIR digital filters," in *Proc. 25th Asilomar Conf. Signals, Syst., Comput.*, Pacific Grove, CA, Nov. 1991, pp. 1021–1025.



Naresh R. Shanbhag (S'86) was born on July 2, 1966 in India. He received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, New Delhi, India, in 1988, and the M.S. degree in electrical engineering from Wright State University, Dayton, OH, in 1990. Currently, he is working towards the Ph.D. degree in electrical engineering at the University of Minnesota, Minneapolis. His research interests are in the general area of VLSI architectures and algorithms for signal processing which includes the design of high-speed algorithms for speech and image compression, adaptive filtering, and channel equalization. In addition, he is also interested in efficient VLSI implementation methodologies for digital signal processing algorithms.



Keshab K. Parhi (S'85–M'88–SM'91) received the B. Tech. (Honors) degree from the Indian Institute of Technology, Kharagpur, India, in 1982, the M.S.E.E. degree from the University of Pennsylvania, Philadelphia, in 1984, and the Ph.D. degree from the University of California, Berkeley, in 1988.

He is currently an Associate Professor of Electrical Engineering at the University of Minnesota, Minneapolis. He has held short-term positions at the NEC C&C Research Laboratories, Kawasaki, Japan, AT&T Bell Laboratories, Holmdel, NJ, and the IBM T. J. Watson Research Center, Yorktown Heights, NY, and has been a consultant to U.S. West Advanced Technologies. His research interests include concurrent algorithm and architecture designs for communications, signal and image processing systems, digital integrated circuits, VLSI digital filters, computer arithmetic, high-level DSP synthesis, and multiprocessor prototyping and task scheduling for programmable software systems. He has published over 90 papers in these areas.

Dr. Parhi received the 1992 Young Investigator Award of the National Science Foundation, the 1992–1994 McKnight-Land Grant Professorship of the University of Minnesota, the 1991 Paper Award from the IEEE Signal Processing Society (for his papers on pipeline interleaving and parallelism in recursive digital filters published in the July 1989 issue of the IEEE TRANSACTIONS ON SIGNAL PROCESSING), the 1991 Browder Thompson Prize Paper Award of the IEEE (for his paper on algorithm transformations published in the December 1989 issue of the PROCEEDINGS OF THE IEEE), the 1989 Research Initiation Award of the National Science Foundation, and the 1987 Eliahu Jury Award for Excellence in Systems Research and the 1987 Demetri Angelakos Award for altruistic activities afforded fellow graduate students of the University of California (UC), Berkeley, where he was a UC Regents Fellow and an IBM Graduate Fellow. He is a former Associate Editor for Image Processing and VLSI Applications for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, and is currently an Editor for the *Journal of VLSI Signal Processing*. He is a member of Eta Kappu Nu and the Association for Computing Machinery.