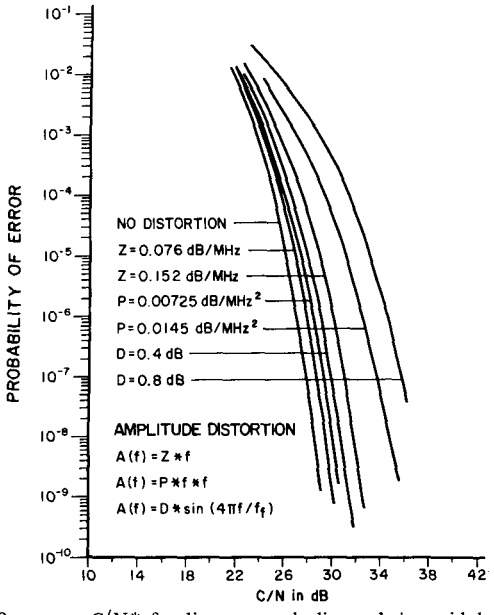


Fig. 9. P_e versus C/N* for linear, parabolic, and sinusoidal amplitude distortions (* in the double-sided Nyquist bandwidth).

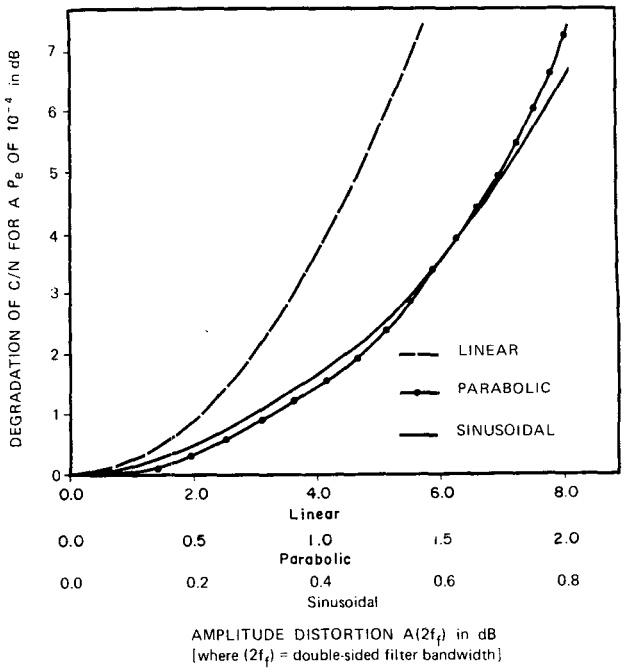


Fig. 10. Degradation of C/N for linear, parabolic, and sinusoidal amplitude distortions. *Note*—For a 90 Mbit/s system with $\alpha = 0.4$ raised cosine filters,

$A(2f_f) = Z \times 21$ dB (linear)
$= P \times 10.5 \times 10.5$ dB (parabolic)
$= D$ dB (sinusoidal).

tude distortion, the C/N in a double-sided Nyquist bandwidth is 25.33 dB for a P_e of 10^{-4}. However, with Z equal to 0.152 dB/MHz, or 3.2 dB maximum amplitude distortion in the double-sided filter bandwidth, the C/N is degraded by 2.3 dB.

For parabolic amplitude distortion, the amplitude distortion in dB is equal to Pf^2, where P is in dB/MHz^2. From Fig. 9, taking the case where P equals 7.25×10^{-3} dB/MHz^2, or 0.8 dB maximum amplitude distortion in the double-sided filter bandwidth, the resulting degradation of C/N for a $P_e = 10^{-4}$ is on the order of 1.0 dB.

In the case of sinusoidal amplitude distortion, the amplitude distortion in dB is represented as $D \sin(2\pi Kf/2f_f)$, where D is the sinusoid amplitude in dB, K is the number of periods of the sinusoid in the filter bandwidth, and $2f_f$ is the filter bandwidth. For the case in which D is equal to 0.8 dB and K is equal to 4, the degradation of C/N is on the order of 6.4 dB for a $P_e = 10^{-4}$.

Degradation data for the systems with the three different amplitude distortion characteristics are given in Fig. 10. Note that the degradation shown is of C/N in the double-sided Nyquist bandwidth for a P_e of 10^{-4} relative to the case with no amplitude distortion. For a given value of maximum amplitude distortion in the filter bandwidth, linear amplitude distortion causes the least degradation, followed in order of increasing degradation by parabolic and sinusoidal.

These results, combined with the results of the preceding section on the effects of group delay distortion, provide an indication of the effects of frequency selective fading on 64-state QAM. Although these results indicate significant degradation would accompany frequency selective fades, adaptive equalization techniques could considerably improve a system's performance during such fades.

VI. SUMMARY

The design and evaluation of our NLA 64-state QAM modem demonstrates that the effect of power level imperfections in the modulator can be easily reduced by changing the receiver's decision thresholds. The study of the effects of channel (selective fading) and hardware imperfections including amplitude and group delay distortions on the P_e performance demonstrate that future generations of radio systems may operate with a spectral efficiency of over 5 bits/s Hz. However, for reasonable C/N degradation, adaptive equalization techniques will be required.

REFERENCES

[1] K. Miyauchi, S. Seki, and H. Ishio, "New techniques for generating and detecting multilevel signal formats," *IEEE Trans. Commun.*, vol. COM-24, Feb. 1976.

[2] D. H. Morais and K. Feher, "NLA-QAM: A new method for generating high power QAM signals through nonlinear amplification," *IEEE Trans. Commun.*, vol. COM-30, Feb. 1982.

[3] M. Subramanian, K. C. O'Brien, and P. J. Puglis, "Phase dispersion characteristics during fade in a microwave line-of-sight radio channel," *Bell Syst. Tech. J.* vol. 52, Dec. 1973.

[4] G. M. Babler, "Selectively faded nondiversity and space diversity narrowband microwave radio channels," *Bell Syst. Tech. J.*, vol. 52, Feb. 1973.

[5] K. Feher, *Digital Communications Microwave Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

Multiprocessor Implementation of Adaptive Digital Filters

V. B. LAWRENCE, MEMBER, IEEE, AND S. K. TEWKSBURY, MEMBER, IEEE

***Abstract*—Adaptive filters, employing the transversal filter structure and the least mean square (LMS) adaptation algorithm, or its variations, have found wide application in data transmission equalization, echo cancellation, prediction, spectral estimation, on-line system**

Paper approved by the Editor for Signal Processing and Communication Electronics of the IEEE Communications Society for publication without oral presentation. Manuscript received September 17, 1982.

The authors are with American Bell, Holmdel, NJ 07733.

0090-6778/83/0600-0826$01.00 © 1983 IEEE

identification, and antenna arrays. Recently, in response to requirements of fast start-up, or fast tracking of temporal variations, fast recursive least squares (FRLS) adaptation algorithms for both transversal and lattice filter structures have been proposed. These algorithms offer faster convergence than is possible with the LMS/transversal adaptive filters, at the price of a five-to-tenfold increase in the number of multiplications, divisions, and additions. Here we discuss architectures and implementations of the LMS/transversal, fast-converging FRLS filter, and lattice filter algorithms which minimize the required hardware speed. We show how each of these algorithms can be partitioned so as to be realizable with an architecture based on multiple parallel processors.

I. INTRODUCTION

Adaptive filtering techniques, using transversal filters, have many applications, such as equalization for data transmission [1], echo and noise cancellation [2], prediction in waveform coding, spectral estimation, on-line system identification, and adaptive antenna arrays. Most current coefficient updating algorithms are either the well-known least mean square (LMS) algorithm or appropriate variations [1], [2]. In addition to transversal filters, adaptive lattice filters have received recent attention [3]-[5], and there has also been considerable interest in developing computationally efficient least squares adaptation algorithms [5]-[7] for the rapid adaptation of the transversal filter coefficients. These recursive algorithms are the sequential-updating counterparts of least squares block processing techniques, such as the Levinson algorithm [8]. They determine that set of filter coefficients which minimizes a weighted cumulative sum of squared errors, given all available observed data up to that time. Thus, in a least squares sense, they offer the fastest possible convergence. This advantage, which is of importance in many applications, is achieved at a price of greater complexity than that of the LMS-type algorithms.

The hardware complexity of adaptation algorithms is usually given as the number of multiplications (or equivalently the multiplication rate). This reflects the dominant role of multiplier hardware at earlier stages of large scale integrations (LSI) [9]. Contemporary, high-density digital circuits (particularly the programmable processors [10]) place considerable emphasis on memory speed and size. Multiprocessors (i.e., arrays of interconnected processors) are particularly important in this regard. By using several processors, rather than one, to obtain the overall arithmetic rate, the overall memory size remains about the same but distributed processor techniques allow memory speeds to be greatly reduced.

For signal processing applications, multiprocessors are normally in a pipelined configuration (i.e., a cascade array of processors). This leads to input-output delays much greater than the sample period. This long latency generally does not affect nonadaptive filters. However, adjustment of filter coefficients in adaptive filters uses the filter output to derive a signal that is returned to correct all coefficients before the next input sample is processed. As a result, the total processing latency must be less than one sample period. It is therefore not clear that multiprocessing can be applied to adaptive algorithms, particularly those algorithms where the latency must be kept to one sample period. If such realizations are not possible, then several high-speed multipliers and high-speed memories will be needed for adaptive algorithms.

The purpose of this paper is to show that latency can be kept to a minimum so that multiprocessing and slower arithmetic and storage can indeed be utilized for three major varieties of adaptation algorithms: 1) the fast recursive least square (FRLS) algorithm (the so-called "fast Kalman" algorithm [11]), 2) the simple LMS algorithm [2], and 3) an adaptive lattice algorithm [4]. An architecture based on multiple parallel processors is advantageous because it permits the order of the filter to be easily increased without affecting the overall structure of the hardware [12]. The conditions under which multiprocessing can be applied to algorithms with overall output-to-input feedback are clarified. Finally, some general characteristics of arithmetic for adaptation algorithms, distinct from conventional nonadaptive filter hardware, are considered. Since the computational structure of two of the adaptation algorithms must be modified to allow multiprocessing, we begin with a review of the algorithms in the next section.

II. REVIEW OF ADAPTIVE ALGORITHMS

In this section we introduce the FRLS and the lattice algorithms, and we compare the complexities of these algorithms with the simple LMS/transversal algorithm. The comparison is in terms of the number of multiplications, divisions, additions, and subtractions.

A. A Recursive Least Squares Adaptation Algorithm and the LMS Algorithm

In this section we review a recursive least squares algorithm, introduced in [6] and [11], as applied to a prediction problem. The notation is that of [11], where the adaptive predictor was part of an adaptive equalization algorithm. The predictor's input is a sequence of samples $\cdots, y(n-1), y(n), y(n+1), \cdots$ occurring at time instants $\cdots, n-1, n, n+1, \cdots$ with $y(n) = 0$ for $n \leqslant 0$. The predictor has the task of estimating each new sample from a linear combination of the N immediately preceding samples. Successive prediction errors between the actual and predicted sample values are used in updating the filter tap coefficients. At iteration n, the N latest inputs, denoted by a vector

$$X_N{}^T(n) \triangleq (y(n-1),\ y(n-2), \cdots, y(n-N))^T, \tag{1}$$

are stored by the predictor. The dimensionality of vectors and matrices is denoted by subscripts; the absence of a subscript indicates a scalar quantity, and vector transpose is indicated by superscript T.

The predictor's tap coefficients at iteration n are denoted by the N-dimensional vector $-A_N(n-1)$, with components $(-a(1, n-1), -a(2, n-1), -a(N, n-1))$. The predictor output $O(n)$ is

$$O(n) = A_N{}^T(n-1)X_N(n) = \sum_{k=1}^{N} a(k, n-1)y(n-k). \tag{2}$$

The prediction error at iteration n is defined as

$$\epsilon(n) \triangleq y(n) + A_N{}^T(n-1)X_N(n). \tag{3}$$

It can be shown that the exponentially weighted sum of squared errors $S(n)$ is

$$S(n) = \sum_{k=1}^{N} \lambda^{n-k}[y(k) - A_N{}^T(n)X_N(k)]^2 \tag{4}$$

for $0 < \lambda \leqslant 1$ *at any time instant* n is minimized, with respect

to $A_N(n)$, by the following recursive algorithm [11]. Starting at $n = 1$, the predictor coefficient adaptation and predictor output generation takes place in the following sequence of operations for each new input $y(n)$:

$$\epsilon(n) = y(n) + A_N{}^T(n-1)X_N(n) \tag{5}$$

$$A_N(n) = A_N(n-1) - k_N(n)\epsilon(n) \tag{6}$$

$$\epsilon(n)' = y(n) + A_N{}^T(n)X_N(n) \tag{7}$$

$$E(n) = \lambda E(n-1) + \epsilon(n)\epsilon(n)' \tag{8}$$

$$\bar{k}_{N+1}(n) = \begin{bmatrix} \dfrac{E(n)^{-1}\epsilon(n)'}{E(n)^{-1}\epsilon(n)'A_N(n) + k_N(n)} \end{bmatrix}. \tag{9a}$$

To proceed further, the $N+1$-dimensional vector $\bar{k}_{N+1}(n)$ is partitioned into an N-dimensional part $m_N(n)$ and a one-dimensional part $\mu(n)$:

$$\bar{k}_{N+1}(n) = \begin{bmatrix} m_N(n) \\ \hline \mu(n) \end{bmatrix}. \tag{9b}$$

The prediction algorithm is completed then as

$$\eta(n) = y(n-N) + D_N{}^T(n-1)X_N(n+1) \tag{10}$$

$$D_N(n) = [D_N(n-1) - m_N(n)\eta(n)][1-\mu(n)\eta(n)]^{-1} \tag{11}$$

$$k_N(n+1) = m_N(n) - D_N(n)\mu(n). \tag{12}$$

Initially, N-dimensional vectors $A_N(0)$, $D_N(0)$, $k_N(1)$ are all set to zeros, and a scalar quantity $E(0)$ is preset to a small positive number δ.[1] Scalar quantities are set as follows: $\epsilon(0)' = 0$, $\epsilon(1) = y(1)$, and $\eta(0) = 0$. Vectors $m_N(n)$, $\bar{k}_{N+1}(n)$ and a scalar $\mu(n)$ need not be initialized.

This recursive prediction algorithm requires $8N + 4$ multiplications, two divisions, and $7N + 5$ additions or subtractions for each new input sample. This compares with $2N + 1$ multiplications and $2N + 1$ additions or subtractions for the simple LMS algorithm, described by (13) and (14) below for the prediction error $\epsilon(n)$ and the new coefficients $A_N(n)$.

$$\epsilon(n) = y(n) + A_N(n-1)^T X_N(n) \tag{13}$$

$$A_N(n) = A_N(n-1) - g\epsilon(n)X_N(n) \tag{14}$$

where g is a suitably chosen small constant. In some versions of the LMS algorithm [13], g is replaced by $g/|X_N(n)|^2$, in which case the number of multiplications increases to $3N + 1$, the number of additions increases to $3N + 1$, and there is now one division.

Note that $5N + 6$ words of storage are required by the FRLS algorithm: three blocks of N words each for the N-dimensional vectors $A_N(n)$, $k_N(n)$, and $D_N(n)$, two blocks of $N + 1$ words each for $\bar{k}_{N+1}(n)$ and for the merge of $X_N(n)$ and $X_N(n+1)$, and single-word storage for the $\epsilon(n)$, $\epsilon'(n)$, $E(n)$, and $\eta(n)$. The simple LMS algorithm requires $2N + 2$ words of storage for $A_N(n)$, $X_N(n)$, $\epsilon(n)$, and g.

[1] A nonzero value for δ guarantees nonsingularity. The behavior of the algorithm is not very sensitive to the choice of δ.

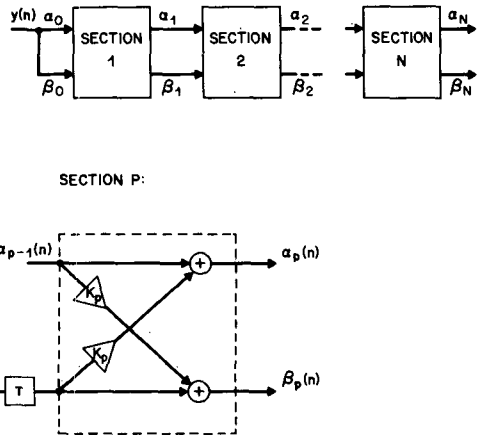


Fig. 1. Lattice predictor.

B. Adaptive Lattice Algorithm

A number of adaptation algorithms have been developed for filters with the lattice structure shown in Fig. 1, including a recent FRLS algorithm [7]. Here we consider an adaptive prediction algorithm given in [4], which has also been applied to adaptive equalization [3].

As shown in Fig. 1, the outputs of the pth section of the lattice are forward and backward prediction residuals $\alpha_p(n)$ and $\beta_p(n)$, given respectively by

$$\alpha_0(n) = \beta_0(n) = y(n) \qquad \text{(input)} \tag{15}$$

$$\alpha_p(n) = \alpha_{p-1}(n) + k_p\beta_{p-1}(n-1) \tag{16}$$

$$\beta_p(n) = k_p\alpha_{p-1}(n) + \beta_{p-1}(n-1). \tag{17}$$

The filter parameters, the so-called reflection coefficients $\{k_p\}$, are updated as follows:

$$k_p(n+1) = k_p(n) - \frac{\alpha_{p-1}(n)\beta_p(n) + \alpha_p(n)\beta_{p-1}(n-1)}{D(n)} \tag{18}$$

where

$$D(n) = (1-c)D(n-1) + \alpha_{p-1}{}^2(n) + \beta_{p-1}{}^2(n-1) \tag{19}$$

and c is a small constant.

This algorithm requires eight multiplications, one division, and six additions per lattice section. This arithmetic complexity is roughly equal to that of the fast Kalman adaptive prediction algorithm. However, since $D(n)$ plays the role of a proportional control, updating and division may be simplified, at the expense of a slight decrease in rate of convergence. The four multiplications and one division involving $D(n)$ can be done with low precision numbers or perhaps simple shift operations.

III. HARDWARE IMPLEMENTATION CONSIDERATIONS

Only through the design of a specific hardware unit can the complexity of an adaptive filter realization be precisely specified, since any given algorithm can be implemented with a wide variety of hardware realizations. Furthermore, a given algorithm may operate at different signal sample rates and/or may have characteristics of particular importance for a given application. Consequently, general comparisons of realization

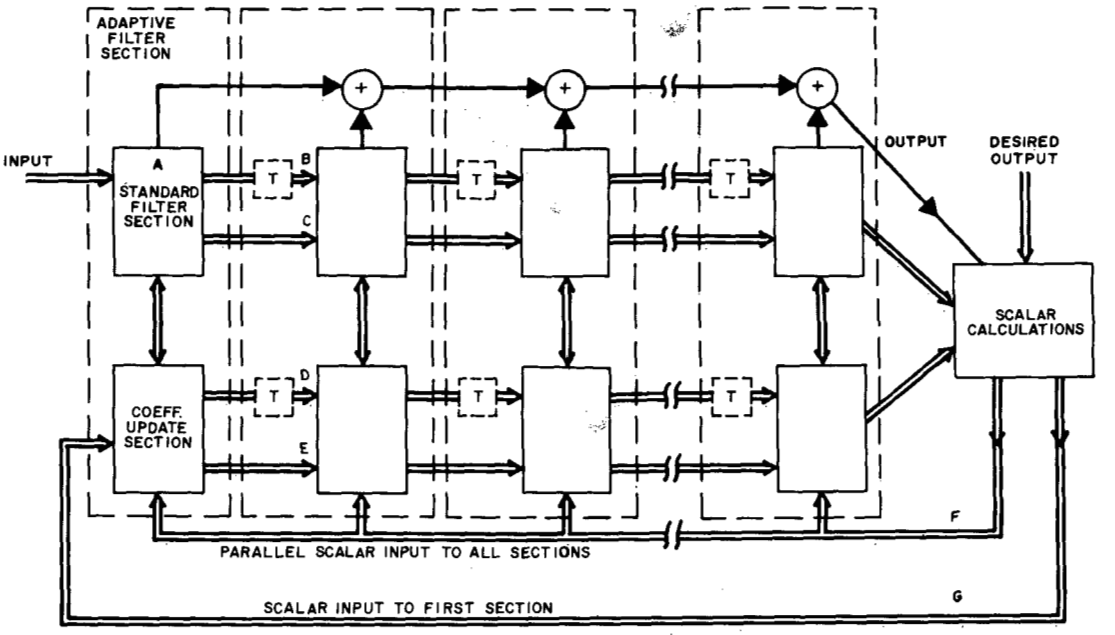


Fig. 2. General structure for adaptation algorithms.

complexity among various adaptive algorithms are generally difficult, although the traditional measures (the numbers of multiplies, adds, and divides, and memory requirements) do give a rough indication.

The discussion below does not seek detailed complexity measures for the various algorithms. Instead, we concentrate here on the implications of the computational structures of the algorithms. Our approach basically contrasts adaptive and nonadaptive filters. Compared to the general topic of data processing, adaptive and nonadaptive filters are generally similar; they are both arithmetic intensive, and perform real-time signal processing tasks based on finite difference equations. As a result, much of the literature on nonadaptive filter realization can be applied to adaptive filters [14], [15]. However, there are significant differences in the structures of the algorithms and the type of processing operations, and therefore the literature on linear, nonadaptive filters must be applied judiciously.

There are three major differences in the computational structures of adaptive and nonadaptive filters. First, adaptive filters often feed their output back to the input for coefficient adjustment, suggesting a potential need for higher speed arithmetic than with nonadaptive filters where such feedback does not normally occur. Second, sum-of-products structures, which have proven so useful for nonadaptive filter hardware, are not as dominant in adaptive filters. Third, adaptive filters use a larger set of arithmetic operations than linear nonadaptive filters. These differences are considered in more detail below.

A. General Structure of Adaptation Algorithms

We use the structure of Fig. 2 as a general representation of an adaptive filter. The overall structure is organized as cascaded "adaptive filter sections." Each adaptive filter section contains a standard filter section (e.g., one tap of a transversal filter for the LMS algorithm) and a coefficient update section. The coefficient update section contains two varieties of computation. The first are the computations which specifically update the coefficient(s) of the associated standard filter section. The second are those computations [e.g., vector product terms as in (10)] using signal samples from the corresponding standard filter section, to obtain values combined with those of other sections into the calculation of scalars.

Several potential forms of signal flow internal to the algorithm are shown (although not all signal flow paths would normally occur for a given adaptation algorithm). If a signal flow path connecting successive sections includes a sample period delay (labeled T in Fig. 2), that sample period delay is shown explicitly in the general structure of Fig. 2 (i.e., paths B and D in Fig. 2). Otherwise, no delays are shown (i.e., paths C and E). We shall see later that the presence or absence of these delays has a major impact on computation speed requirements. These speed requirements originate from paths F and/or G, which produce feedback signals having to be computed before the next external input is applied (i.e., within one sample period). Signal path A illustrates a sequence of calculations which can be executed in parallel (e.g., using an adder tree) and are only needed for the last section (the scalar computations). For reasons which will be clear below, the absence of sample period delays along this particular path does not limit computation speeds.

The following section describes the influence of the feedback paths on arithmetic speed. We then discuss the conditions under which multiprocessor realizations are possible (and arithmetic speeds are therefore not set by the overall feedback path). By reorganizing the fast Kalman and lattice algorithms, we were able to impose the desired overall structure, making them compatible with multiprocessing.

B. Latency Considerations

As illustrated in Fig. 3, computations around any feedback loop must be completed before the next input sample is applied to the loop. In particular, the processing performed for the current input must be completed before the next input sample is processed. Each computation introduces a latency T_j for the jth computation and the sum of these latencies must be less than the sample period τ_s. This realizability constraint establishes a minimum computation time for the

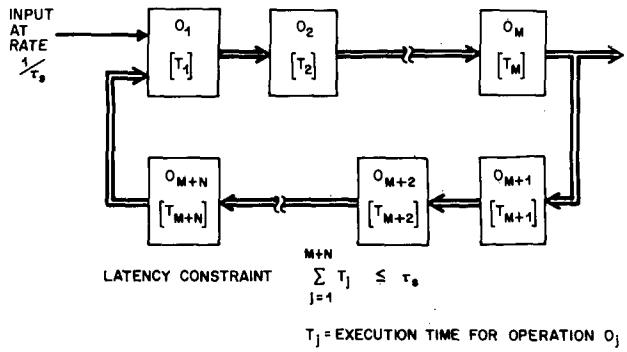


Fig. 3. Feedback loop and lower limit on arithmetic speed.

arithmetic unit, with the computation rate given by the number of arithmetic operations in the loop multiplied by the sample rate. Typically, nonadaptive feedforward filters transform an input sequence into an output sequence without returning data from the output of the overall filter back to the input. Instead, feedback of signals is present generally only internal to a single filter section. As a result, arithmetic speeds are set (in terms of minimum speeds) by the section rather than the overall filter. This is potentially not the case with adaptive algorithms, where a loop around the entire algorithm may exist. However, if the general structure shown in Fig. 2 has the proper form, then the speed requirements implied by this overall loop can be avoided with the use of parallel processing. We consider the general case and then show that the speed limits are in fact quite relaxed for the three algorithms being considered here.

C. Multiprocessor Realization of Adaptive Filters

We assume an N-section filter and processors whose arithmetic units can execute a number of computations equivalent to the number of computations in N/m (m is an integer ≥ 2) adaptive filter sections in one sample period. To realize the overall filter, m processors must be used with each processor having completed N/m sections at the end of the sample period. However, if the overall filter output is fed back to the input, the m processors must generate the output within one sample period. This requires that each processor execute, in parallel, computations corresponding to the same sample period (i.e., corresponding to the same sample index n in the algorithm equations). This is only possible if sample period delays occur on all feedforward signal paths between processors. In this case, the results of the given processor become available near the end of the sample period, but the sample period delay means that they are not needed until the next sample period. The sample period delays between sections, in this sense, allow up to one sample period of computational latency during execution. Fig. 4 shows the implied constraint that sample period delays must appear on all feedforward signal paths between successive filter sections. (This constraint can be relaxed somewhat to the requirement that sample period delays must appear on all feedforward signal paths between processors, i.e., at the partitioning points. This relaxed constraint is necessary for the lattice algorithm.) Fig. 5(a) shows the partitioning of the N-section filter among the m processors. No actual storage is associated with the sample period delay elements in dashed boxes, the delay being provided by the latency of the processor generating the corresponding signal. The organization of the sequence of sums in Fig. 5(a) is seen to be consistent with multiprocessing (as suggested earlier) even though sample period delays do not appear in the sum path. The arrangement

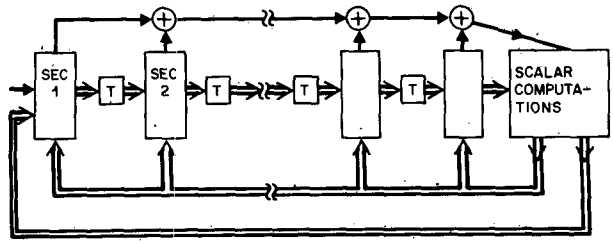
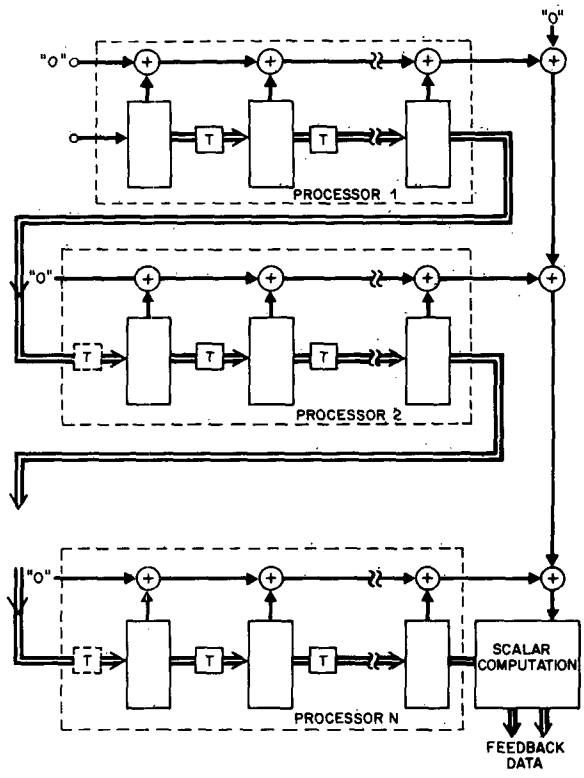
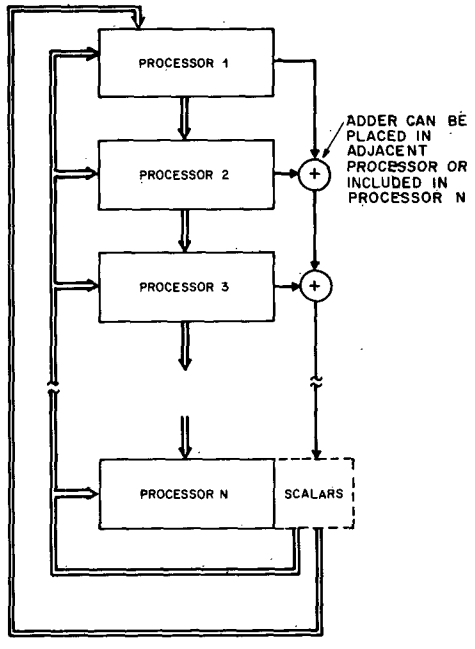


Fig. 4. General structure (with feedback) compatible with multiprocessing.



(a)



(b)

Fig. 5. (a) Algorithm partition for multiprocessing. (b) Hardware realization for partition of algorithm.

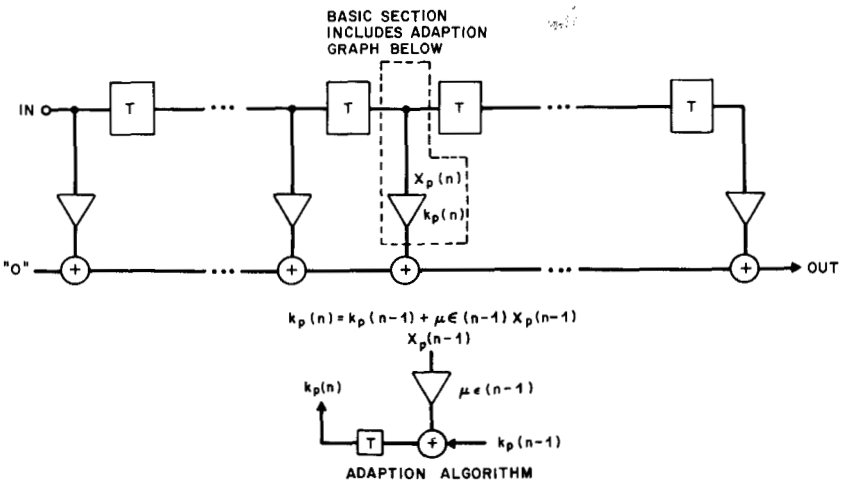


Fig. 6. Signal flow graph of LMS algorithm.

of processors is shown in Fig. 5(b). Finally, the feedback signals are generated before the start of the next sample period. They may therefore be entered into each of the processors simultaneously and do not impose additional constraints.

Each of the algorithms described in Section II is consistent with multiprocessing, as defined above, but each for a different reason. We consider the three algorithms separately.

D. Standard LMS Algorithm

Fig. 6 shows the signal flow graph for the standard LMS algorithm [see (3), (4)]. Comparing Fig. 6 to Fig. 4, we see that the LMS algorithm directly satisfies the requirements for multiprocessing. The basic section shows that the multiplier speed requirement (i.e., execution of one section per sample period) is two multipliers per sample period. In particular, a sample period delay appears on all data paths between filter sections, while the sum of products path (without delays) is allowed on the assumption of a fast adder. (The fast addition is not essential here since some delays in the input delay line can be moved to the sum of products line in an obvious manner.)

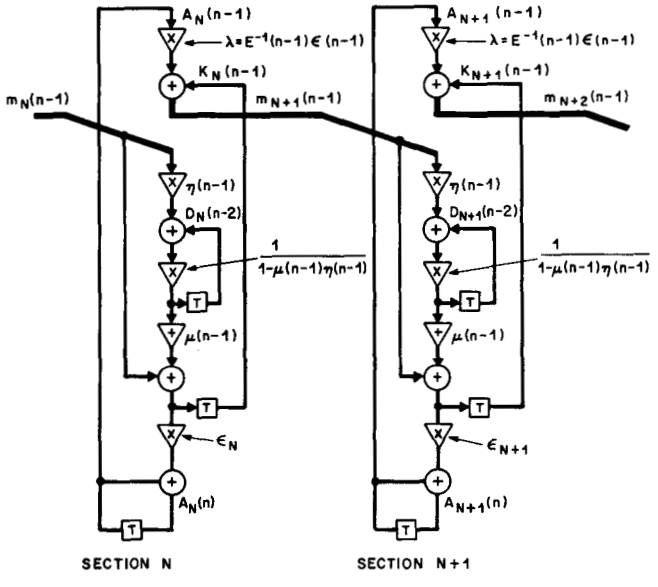
E. Kalman Algorithm

Fig. 7 shows a section of the standard fast Kalman algorithm described by (5)–(12). In this case, the basic multiprocessor requirements given by Fig. 4 are not satisfied. The heavily outlined feedforward path does not contain a delay operation. However, it is possible to reorganize the equations such that a sample period delay is imposed on the data paths interconnecting separate filter sections without affecting the performance and convergence of the algorithm. The reorganized equations are given below.

Starting at $n = 1$, the prediction and predictor coefficient adaptation takes place in the following sequence of operations for each new input $y(n)$:

$$\bar{k}_{N+1}(n-1) = \begin{bmatrix} E(n-1)^{-1}\epsilon(n-1)' \\ \hline E(n-1)^{-1}\epsilon(n-1)'A_N(n-1) + k_N(n-1) \end{bmatrix}. \tag{20a}$$

Partition $\bar{k}_{N+1}(n-1)$ into N-dimensional and one-dimensional parts:



HEAVY LINE SHOWS PATH VIOLATING CONSTRAINTS
ALGORITHM EQUATIONS ARE THOSE GIVEN BY FALCONER, AND LJUNG TRANS-COMMUN. COM-26, NO. 10 PP 1439-1446 OCT 1978

Fig. 7. Adaptation section for fast Kalman algorithm.

$$\bar{k}_{N+1}(n-1) = \begin{bmatrix} m_N(n-1) \\ \hline \mu(n-1) \end{bmatrix} \tag{20b}$$

$$D_N(n-1) = [D_N(n-2) - m_N(n-1)\eta(n-1)] \cdot [1 - \mu(n-1)\eta(n-1)]^{-1} \tag{21}$$

$$k_N(n) = m_N(n-1) - D_N(n-1)\mu(n-1) \tag{22}$$

$$A_N(n) = A_N(n-1) - k_N(n)\epsilon(n) \tag{23}$$

$$\epsilon(n)' = y(n) + A_N{}^T(n)X_N(n) \tag{24}$$

$$\epsilon(n+1) = y(n+1) + A_N{}^T(n)X_N(n+1) \tag{25}$$

$$\eta(n) = y(n-1) + D_N{}^T(n-1)X_N(n+1) \tag{26}$$

$$E(n) = \lambda E(n-1) + \epsilon(n)\epsilon(n)'. \tag{27}$$

The equivalence of (5)–(12) and (20)–(27) may be established by changing the sample time parameter and reordering the original equations. Fig. 8 shows a section of the fast Kalman algorithm based on the reorganized defining equations. The multiprocessor constraints are now satisfied and the minimum multiply speed is 5 multiplications per sample period (considering only multiplications).

F. Lattice Algorithm

Fig. 9 shows a stage of the lattice algorithm and Fig. 10 shows an equalizer application [3]. For simple prediction, no output-to-input feedback occurs and multiprocessing in the conventional cascade form is directly allowed. In equalizer applications, however, output-to-input feedback is required. In this case, the basic lattice equalizer section is not compatible with multiprocessing due to the $\alpha_p(n)$ path. However, the lattice equalizer can be compatible if the flow graph is modified. The flow graph can be modified by replicating some computations such that intersection transfers have a full sample period delay. In particular, the $\beta_p(n)$ samples used to compute the $\alpha_p(n)$ can be taken before the sample period delay, allowing computation of much of the $\alpha_p(n)$ path one sample period earlier. (Only the initial input is not yet available.) To illustrate this procedure, we assume N sections are implemented by a single processor. We then break the $\alpha_p(n)$ path at the interprocessor connection and replace that connection by the "advanced" $\alpha_p(n)$ values as shown in Fig. 11. Before the "advanced" $\alpha_p(n)$ value can be used in the next processor, it must be delayed one sample period to align it in time with the $\beta_p(n)$ connection. The initial missing term in the "advanced" $\alpha_p(n)$ value is added at the next processor.

The procedure shown in Fig. 11 uses redundant multiplications. A more efficient approach may be to use redundant storage. Fig. 12 shows an organization in which the multiplications are not increased but for which the number of sample period delays per section is doubled. In either case, multiprocessing is possible again.

It is tempting to conclude that all adaptive filter algorithms can be transformed to be consistent with multiprocessing. We know that this is not the case for nonadaptive filters, e.g., wave digital filters [16] cannot be transformed to satisfy multiprocessing requirements unless the number of reflecting ports between reflectionless ports is restricted. In general (for data processing) and for the specific topic of adaptive filters, it must be recognized that the ability of multiprocessing to increase algorithm execution speed depends on the specific algorithm. This is particularly true for real-time processing tasks, as considered here. The discussion above shows that three important adaptive filter algorithms are compatible with multiprocessing, and general conditions for this compatibility have been specified. Therefore, it is doubtful that general claims of multiprocessor compatibility for arbitrary adaptive algorithms are allowed.

IV. ARITHMETIC CONSIDERATIONS

In this section we discuss the structure and type of arithmetic that is efficient for multiple parallel processors executing adaptive filter algorithms. A distinction is made between the arithmetic for adaptive and nonadaptive filter algorithms.

A. Arithmetic Operation Sequences

Much of the literature on nonadaptive filter implementation presumes either a standard biquad recursive section or a

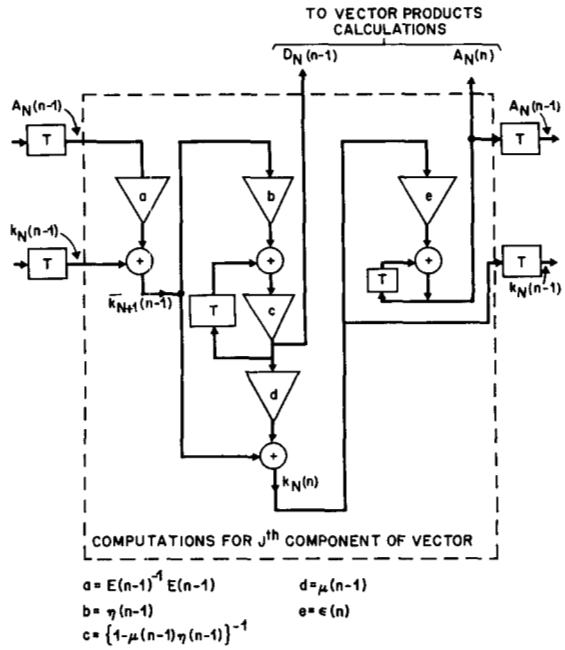


Fig. 8. New adaptation section for fast Kalman algorithm.

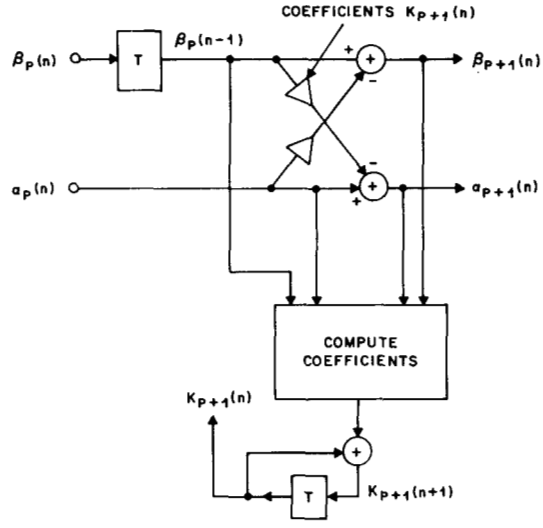


Fig. 9. Standard lattice algorithm.

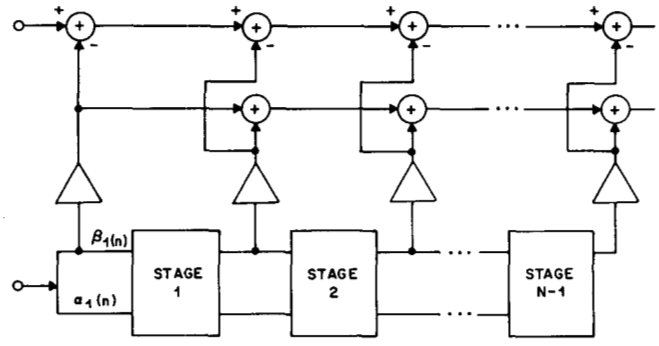


Fig. 10. Lattice structure for equalizer.

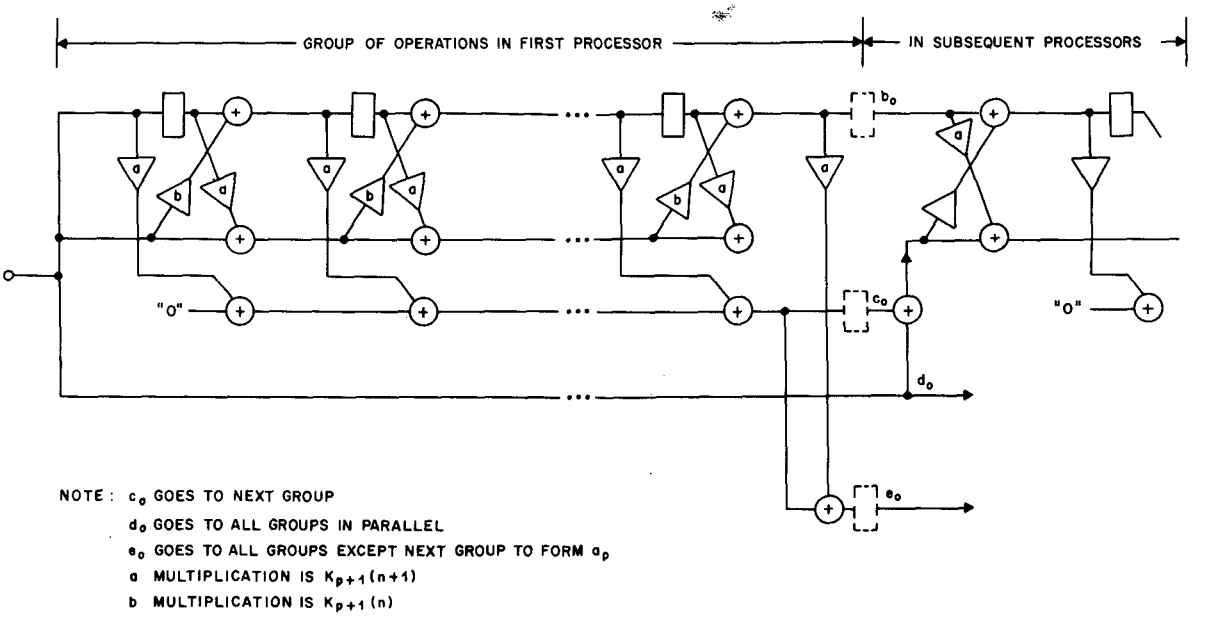


Fig. 11. Lattice algorithm realization using replicated multiplications.

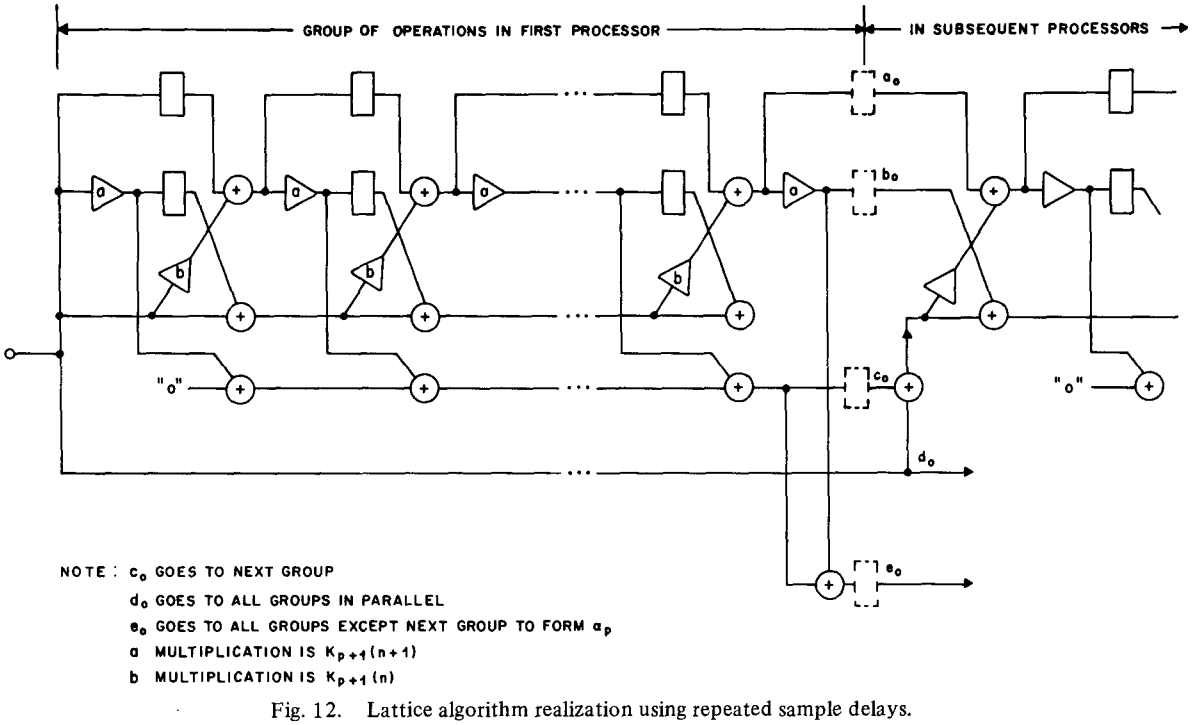


Fig. 12. Lattice algorithm realization using repeated sample delays.

transversal filter. Sum-of-products computations dominate these two cases, and arithmetic units (AU's) performing the basic function $AX + Y \rightarrow Y$ are very common [10], [17]. However, sum-of-products forms do not dominate adaptation algorithms, since they are normally found only in vector product terms. A full multiply time is generally required to execute an isolated addition $Y = A + B$ in the sum-of-products AU. Since an operation's execution time is the basic hardware measure of an operation in a general purpose, programmable digital signal processor, such isolated additions are equivalent in "complexity" to multiplications. This is at variance with the usual notion that adds are of minor importance relative to multiplies.

With sum-of-products AU's, the complexity comparisons given earlier must be modified. We have therefore generated the results shown in Table I for the three algorithms considered here. The results in the first column are those given earlier (corresponding to custom hardware). The second column shows the results for a sequential processor using a sum-of-products AU as above. Both columns show similar relative complexities among the three algorithms. The results are given for purposes of illustration and actual values would depend on the actual hardware design.

B. Arithmetic Operation Set

Nonadaptive filters are dominated by multiplications and additions (with limiting, rectification, and inversion usually also provided). The multiplications of standard filters involve data and fixed coefficients, allowing physical separation of data and coefficient paths. Adaptive filters require multiplica-

TABLE I
PREDICTION ALGORITHM COMPLEXITIES

Adaptation Algorithm	Traditional Meaures (e.g., Custom Hardware)			$Y + A.X \rightarrow Y$ (AU)	
	Multiplication	Addition Subtraction	Division	Multiply Cycles(1)	Division
LMS	$2N + 1$	$2N + 1$	0	$3N + 1$	0
Fast Kalman	$8N + 4$	$7N + 5$	2	$12N + 9$	2
Lattice	$8N$	$6N$	N	$14N$	N

Note(1): Vector products formed using the Accumulate mode of the AU to reduce isolated additions. The entries assume a full multiply cycle for each remaining isolated addition.

tion of data with data, making necessary a merging of signal and coefficient paths. Nonadaptive filters using automatic gain control (AGC) generally require data times data multiplications, and therefore, this capability should also be present in some programmable signal processors.

Adaptive filters also require, for some of the more recent algorithms [11], division of data by data. The "optimized" multipliers used for standard digital filters do not normally provide for division. Either approximations to division must be used or the AU must be extended to allow division for such adaptive algorithms. A convenient approximation scheme would separate the number into a high-order portion and a low-order portion and use a series expansion to approximate the reciprocal of the number. Table look-up can be used on the high-order portion. Such table look-up schemes do, however, imply some means of generating an address from the data word, a capability that may not be provided by a programmable signal processor optimized for nonadaptive filters. Custom hardware could easily provide such tables. Floating point arithmetic [13] instead of fixed-point arithmetic makes divisions easier at the expense of additions and subtractions.

Finite arithmetic effects can be significantly different in adaptive filters (where noise sources feed both coefficient and signals) than in nonadaptive filters (where noise sources feed only the signals).

C. Multimode Operations

Most digital filters either run continually or have a finite memory of previous inputs, making special initialization often unnecessary. Initialization of state variables for adaptive algorithms is necessary in the fast Kalman algorithms. Furthermore, some adaptive filter algorithms employ adaptation only during start-up and then hold the filter response fixed. Therefore, three distinct modes exist—initialization, convergence of the adaptation, and holding the filter response fixed. Providing such modes is relatively straightforward for programmable DSP's since separate programs can exist for each mode. The amount of program is not excessive since the program for each section of the adaptive filter is basically repeated (suggesting an efficient subroutine-like software construction).

V. CONCLUSION

We have shown that adaptive algorithms are compatible with multiprocessing only under specific constraints if feedback exists from the output back to the input. Such feedback requires that the total input-to-output delay be less than one sample period. This total latency constraint causes a cascaded (and pipelined) chain of processors to be unsuited for adaptive algorithms (although such a multiprocessor structure is useful for many nonadaptive filter algorithms). We demonstrated that a parallel configuration (in contrast to a cascade configuration) does allow multiprocessing by providing the output within one sample period. However, to be compatible with such a parallel organization of processors, the algorithm must satisfy certain structural constraints. These constraints were specified and it was shown that the standard LMS algorithm is directly compatible with the parallel multiprocessor organization. Standard fast Kalman and lattice algorithms violate the requirements. We demonstarated that the signal flow graphs of both the fast Kalman and lattice algorithms can be transformed to satisfy the multiprocessor constraints.

The results are significant in that implementation of these adaptive algorithms does not require arithmetic units significantly faster than those used for nonadaptive filters. This does not imply that processors for adaptive and nonadaptive filters are equivalent since adaptive filters may require an extended set of arithmetic operations and may require higher accuracy arithmetic. However, the ability to use moderate speed arithmetic and memory does permit consideration of integrated programmable processors for adaptive algorithms.

ACKNOWLEDGMENT

We acknowledge the valuable discussions we have had with D. D. Falconer.

REFERENCES

[1] R. W. Lucky, "Automatic equalization for digital communication," *Bell Syst. Tech. J.*, vol. 44, pp. 547–588, Apr. 1965.
[2] B. Widrow and M. E. Hoff, Jr., "Adaptive switching circuits," in *IRE Wescon Conv. Rec.*, Aug. 1960, pt. 4, pp. 96–104.
[3] E. H. Satorius and S. T. Alexander, "Channel equalization using adaptive lattice algorithms," *IEEE Trans. Commun.*, vol. COM-27, pp. 899–905, June 1979.
[4] J. Makhoul, "A class of all-zero lattice digital filters: Properties and applications," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, pp. 304–314, Aug. 1978.
[5] R. Wiggins and L. Brantingham, "Three-chip system synthesizes human speech," *Electronics*, pp. 109–116, Aug. 31, 1978.
[6] M. Morf, L. Ljung, and T. Kailath, "Fast algorithms for recursive identification," in *Proc. IEEE Conf. Decision Contr.*, Clearwater Beach, FL, Dec. 1976.
[7] E. H. Satorius and M. J. Shensa, "On the application of recursive least squares methods to adaptive processing," presented at the Int. Workshop Appl. Adaptive Contr., Yale Univ., New Haven, CT, Aug. 23–25, 1979.
[8] N. Levinson, "The Weiner rms (root mean square) error criterion in filter design and prediction," *J. Math. Phys.*, pp. 261–278, 1947.
[9] S. L. Freeny *et al.*, "Design of digital filters for an all digital frequency division multiplex-time division multiplex translator," *IEEE Trans. Circuit Theory*, vol. CT-18, pp. 702–711, 1971.
[10] Special Session on VLSI-Digital Signal Processing, in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, Apr. 1980, pp. 383–405.
[11] D. D. Falconer and L. Ljung, "Application of fast Kalman estimation to adaptive equalization," *IEEE Trans. Commun.*, vol. COM-26, pp. 1439–1446, Oct. 1978.
[12] D. D. Falconer *et al.*, "Processor-hardware considerations for

adaptive digital filter algorithms," in *Proc. Int. Conf. Commun.*, June 1980, pp. 57.5.1–57.5.6.
[13] D. L. Duttweiler, "A twelve channel digital echo canceler," *IEEE Trans. Commun.*, vol. COM-26, pp. 647–693, May 1978.
[14] S. K. Tewksbury *et al.*, "Digital signal processor architecture," *IEEE Commun. Mag.*, Jan. 1978.
[15] S. L. Freeny, "Special-purpose hardware for digital filtering," *Proc. IEEE*, vol. 63, Apr. 1975.
[16] A. Antoniou, *Digital Analysis and Design*. New York: McGraw-Hill, 1979, ch. 12.
[17] J. R. Boddie *et al.*, "A digital signal processor for telecommunications applications," in *ISSCC Dig.*, Feb. 1980.

Distributions of the Two-Dimensional DCT Coefficients for Images

RANDALL C. REININGER AND JERRY D. GIBSON, MEMBER, IEEE

***Abstract*—For a two-dimensional discrete cosine transform (DCT) image coding system, there have been different assumptions concerning the distributions of the transform coefficients. This paper presents results of distribution tests that indicate that for many images the statistics of the coefficients are best approximated by a Gaussian distribution for the DC coefficient and a Laplacian distribution for the other coefficients. Furthermore, from a simulation of the DCT coding system it is shown that the assumption that the coefficients are Laplacian yields a higher actual output signal-to-noise ratio and a much better agreement between theory and simulation than the Gaussian assumption.**

I. INTRODUCTION

In image coding systems which use a two-dimensional discrete cosine transform (DCT) [1], there have been several different assumptions on the distributions of the transform coefficients. Pratt [2] conjectured that the DC coefficient should have a Rayleigh distribution since it was the sum of positive values, and that, based on the central limit theorem, the other coefficients should be Gaussian. Netravali and Limb [5] agreed with the above assumption and also stated that the histograms of the non-DC coefficients were roughly bell-shaped. On the other hand, Tescher [3] indicated that the non-DC coefficients were not Gaussian, but Laplacian, and most recently, Murakami *et al.* [4] assumed that the DC coefficient was Gaussian and that the non-DC coefficients were Laplacian. These different assumptions have led the authors to perform goodness-of-fit tests on the transform coefficients in order to identify the distribution that best approximates the statistics of the coefficients. In the tests, the Gaussian, Laplacian, gamma, and Rayleigh distributions were considered.

This paper shows that for many images the DC coefficient is best approximated by a Gaussian distribution and non-DC coefficients are best approximated by Laplacian distributions, and that by using Laplacian quantizers for the non-DC transform coefficients, the quality of the reconstructed image can be improved as compared to Gaussian quantizers. This paper is organized as follows. Section II describes the goodness-of-fit test and how it was used with the transform coefficients, and Section III describes the results of the tests. In Section IV, comparisons between the theoretical and actual quantization error for a two-dimensional DCT system are made for different assumptions on the distribution of the coefficients.

Paper approved by the Editor for Signal Processing and Communication Electronics of the IEEE Communications Society for publication without oral presentation. Manuscript received April 16, 1982; revised September 23, 1982.

The authors are with the Department of Electrical Engineering, Texas A&M University, College Station, TX 77843.

II. GOODNESS-OF-FIT TEST

A well-known test for goodness of fit of distributions is the Kolmogorov–Smirnov (KS) test [8], [9]. For a given set of data $X = (x_1, x_2, \cdots, x_M)$, the KS test compares the sample distribution function $F_X(\cdot)$ to a given distribution function $F(\cdot)$. The sample distribution function is defined by

$$F_X(z) = \begin{cases} 0, & z < x_{(1)} \\ \dfrac{n}{M}, & x_{(n)} \leqslant z < x_{(n+1)}, \quad n = 1, 2, \cdots, M-1 \\ 1, & z \geqslant x_{(M)} \end{cases} \tag{1}$$

where $x_{(n)}$, $n = 1, \cdots, M$, are the order statistics of the data X. The KS test statistic t is then defined by

$$t = \max_{i=1,2,\cdots,M} |F_X(x_i) - F(x_i)|. \tag{2}$$

The KS test statistic is a distance measure between the sample distribution function and the given distribution function, with the distance defined by the maximum difference between $F_X(\cdot)$ and $F(\cdot)$ evaluated at the sample points x_i. When testing the data against several distributions, the distribution that yields the smallest KS statistic is the best fit for the data.

The KS test was used to test the distributions of the DCT coefficients with block sizes 8, 16, and 32 computed for the five images ("Girl," "Couple," "Moon," "X-Ray," and "Aerial") shown in Fig. 1. These images have size 256 × 256 pels, with the gray levels PCM encoded at 8 bits/pel. For each image and block size, the KS goodness-of-fit test was performed on the ten high-energy coefficients in the upper left-hand corner of the transform block c_{00}, c_{01}, c_{02}, c_{03}, c_{10}, c_{11}, c_{12}, c_{20}, c_{21}, and c_{30}. The data for a given coefficient "ij" consisted of the points $c_{ij}(k)$, $k = 1, \cdots, M$, where the index k represents the position of the block in the image, and the number of blocks M in a 256 × 256 image is related to the block size N by $M = (256/N)^2$. For these data points the sample mean and variance $\overline{c_{ij}}$ and S_{ij}^2 were calculated

0090-6778/83/0600-0835$01.00