A Pipelined Kalman Filter Architecture

Naresh R. Shanbhag

Dept. of EE
Univ. of Minnesota
Minneapolis, MN 55455

Keshab K. Parhi

Dept. of EE
Univ. of Minnesota
Minneapolis, MN 55455

Abstract

Presented in this paper is a hardware-efficient pipelined Kalman filter architecture for recursive least-squares (RLS) identification. The proposed architecture is developed via the relaxed look-ahead technique. This technique results in an extremely low hardware overhead at the expense of a slightly degraded performance. The hardware overhead due to pipelining consists of only the pipelining latches and is therefore negligible. Convergence analysis and simulations indicate that the adaptation accuracy of the pipelined and serial architectures are identical. Speed-up of upto 15 *is demonstrated via simulations. Finally, a folded architecture for pipelined Kalman filter is presented. This folded architecture reduces the number of computational elements by half while still achieving substantial speed-up.*

1 Introduction

The design of digital signal processing (DSP) and communications algorithms, which are hardware-efficient and offer implementation flexibility, is of great interest. It is being recognized that in addition to performance, algorithm designers need to address implementation oriented issues such as throughput, power dissipation and area. In this regard, pipelined algorithms have a distinct advantage over serial algorithms.

Algorithm transformation techniques [1] such as *look-ahead* [2] have already been proposed for the pipelining of recursive DSP algorithms. Look-ahead pipelining introduces additional concurrency in serial algorithms at the expense of a hardware overhead. Recently, we have proposed the *relaxed look-ahead* technique [3-4] to reduce this overhead. This technique maintains the functionality of the algorithm instead of the exact input-output behavior.

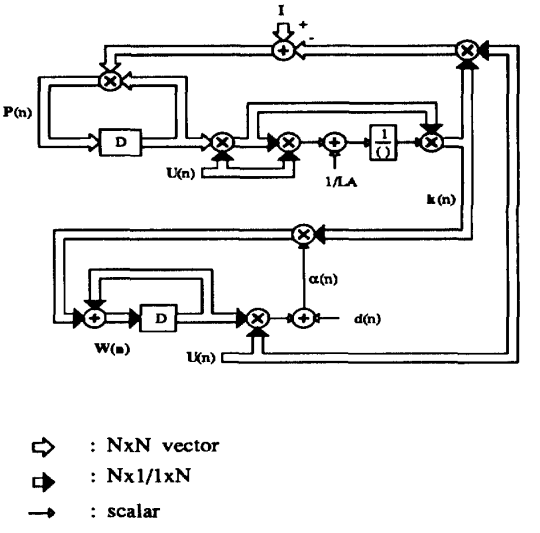


Figure 1: The serial Kalman filter architecture.

In this paper, we employ the relaxed look-ahead technique to pipeline the Kalman filter [5] for recursive least-squares estimation. We analyze the convergence behavior of the pipelined algorithm and verify the results via simulations. In addition, we employ systematic folding techniques [6] in order to trade-off speed and hardware.

This paper is organized as follows. In section 2, the pipelined Kalman filter architecture and its convergence analysis is presented. Simulation results are presented in section 3, and folded architectures are presented in section 4.

2 The Pipelined Kalman Filter

In this section, we present the piplined Kalman filter arcitecture and its convergence properties.

1058-6393/93 $03.00 © 1993 IEEE

2.1 Architecture

The serial Kalman filter (see Fig.1) for RLS estimation is described by the following equations

$$\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{P}(n-1)\mathbf{U}(n)}{1+\lambda^{-1}\mathbf{U}^T(n)\mathbf{P}(n-1)\mathbf{U}(n)} \quad (2.1)$$

$$\alpha(n) = d(n) - \mathbf{W}^T(n-1)\mathbf{U}(n) \quad (2.2)$$

$$\mathbf{W}(n) = \mathbf{W}(n-1) + \mathbf{k}(n)n) \quad (2.3)$$

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{U}^T(n)\mathbf{P}(n-1) \quad (2.4)$$

where n is the time index, $d(n)$ is the desired signal, $\mathbf{W}^T(n) = [w_1(n), w_2(n), \ldots, w_N(n)]$ is the vector of weights, $\mathbf{U}^T(n) = [u(n), u(n-1), \ldots, u(n-N+1)]$ is the input vector, $\alpha(n)$ is the error, N is the filter order, $\mathbf{K}(n)$ is the Kalman gain and $\mathbf{P}(n)$ is the inverse of the deterministic autocorrelation matrix $\Phi(n) = \sum_{i=1}^{n}\lambda^{n-i}\mathbf{U}(i)\mathbf{U}^T(i)$. Note that in Fig.1, there are two recursive inner loops corresponding to (2.3) (**W**-loop) and (2.4))(**P**-loop). These loops impose a throughput bottleneck, which will be broken by introducing pipelining latches in them.

The equations (2.1)-(2.4) recursively solve the equation

$$\mathbf{W}(n) = \Phi^{-1}(n)\Theta(n), \quad (2.5)$$

where $\Theta(n) = \sum_{i=1}^{n}\lambda^{n-i}d(i)\mathbf{U}(i)$ is the crosscorrelation vector. It can be shown that $\Phi(n)$ and $\Theta(n)$ can be computed recursively as follows

$$\Phi(n) = \lambda\Phi(n-1) + \mathbf{U}(n)\mathbf{U}^T(n) \quad (2.6)$$

$$\Theta(n) = \lambda\Theta(n-1) + d(n)\mathbf{U}(n). \quad (2.7)$$

The PIPKAL can be derived by substituting pipelined versions of (2.6) and (2.7) into (2.5). In order to pipeline (2.6), we first apply an M-step look-ahead [2] to obtain

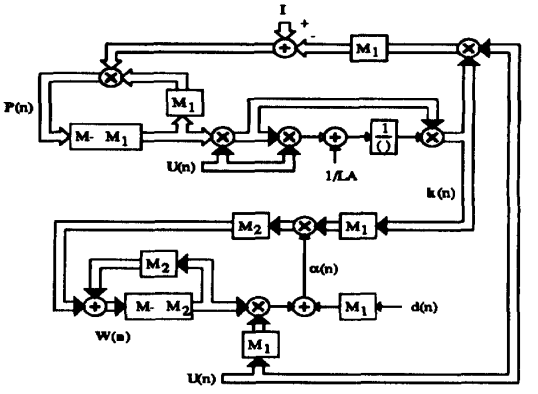
$$\Phi(n) = \lambda^M\Phi(n-M) + \sum_{i=0}^{M-1}\lambda^i\mathbf{U}(n-i)\mathbf{U}^T(n-i). \quad (2.8)$$

Application of relaxed look-ahead implies approximating the equations obtained via look-ahead. In this paper, we approximate (2.8) as follows

$$\Phi(n) = \lambda\Phi(n-M) + LA\mathbf{U}(n)\mathbf{U}^T(n), \quad (2.9)$$

where LA is referred to as the look-ahead factor and is less than equal to M. We justify this approximation under the assumption that the input $\mathbf{U}(n)$ varies slowly over M samples and λ is close to unity. In a similar fashion, we can write the pipelined version of (2.7) as

$$\Theta(n) = \lambda\Theta(n-M) + LAd(n)\mathbf{U}(n). \quad (2.10)$$



⇨ : NxN vector

⇨ : Nx1/1xN

→ : scalar

Figure 2: The PIPKAL architecture.

We can substitute (2.9) and (2.10) into (2.5) to obtain the following equations describing PIPKAL

$$K(n) = \frac{\lambda^{-1}\mathbf{P}(n-M)\mathbf{U}(n)}{\frac{1}{LA}+\lambda^{-1}\mathbf{U}^T(n)\mathbf{P}(n-M)\mathbf{U}(n)} \quad (2.11)$$

$$\alpha(n) = d(n) - \mathbf{W}^T(n-M)\mathbf{U}(n) \quad (2.12)$$

$$\mathbf{W}(n) = \mathbf{W}(n-M) + \mathbf{k}(n)\alpha(n) \quad (2.13)$$

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-M) - \lambda^{-1}\mathbf{k}(n)\mathbf{U}^T(n)\mathbf{P}(n-M). \quad (2.14)$$

The PIPKAL architecture is shown in Fig.2, where we now see that M latches have been introduced into the two inner loops. Of the M latches in the **P**-loop, M_1 latches would be employed to pipelined the computations outside the loop and the remaining $M - M_1$ latches would be employed to pipeline the **P**-loop computation. Similarly, we employ M_2 latches to pipeline the computations outside the **W**-loop and $M - M_2$ latches are used to pipeline the **W**-loop.

2.2 Convergence Analysis

As we have made an approximation in the derivation of (2.9) and (2.10), therefore some performance degradation should be expected. This also makes it necessary to subject PIPKAL to a convergence analysis. An approximate analysis of PIPKAL shows that the misadjustment of PIPKAL $\mathcal{M}$ is given by

$$\mathcal{M} = \frac{1-\lambda}{1+\lambda}N, \quad (2.15)$$

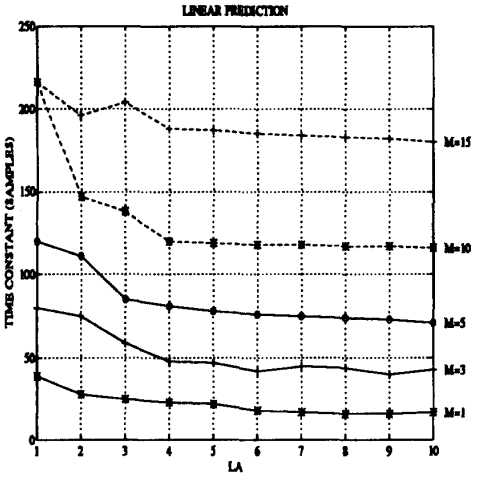
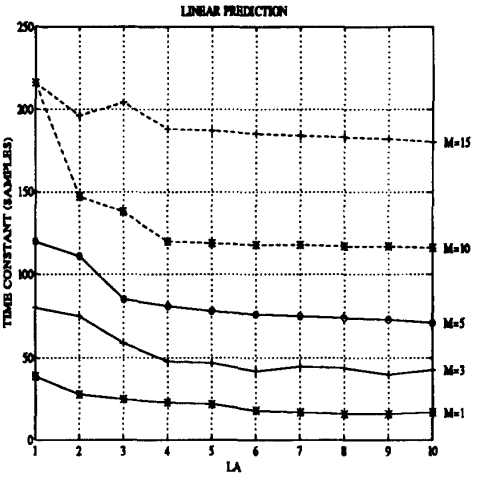


Figure 3(a)



(b)

Figure 3: Linear prediction: (a) output SNR and (b) convergence time.

which is identical to that of the serial Kalman filter. However, analysis indicates that PIPKAL would converge in approximately $2MN$ iterations. This implies that PIPKAL is M times slower than the serial algorithm.

In a practical application, where the signal environment is always non-stationary, the performance of PIPKAL would be close to that of the serial filter. This fact is verified by simulations in the next section.

3 Simulation Results

In this section, the performance of PIPKAL is compared with that of the serial filter in a linear prediction (stationary case) and channel equalization (stationary and non-stationary cases) scenarios.

3.1 Linear Prediction

In this experiment, the PIPKAL architecture has been employed for the prediction of a second-order auto-regressive (AR) process whose auto-correlation matrix has an eigenvalue spread of 10. The PIPKAL was simulated for speed-ups of $M = 1, 3, 5, 10$ and 15. The value of λ was fixed at 1.0.

In Fig.3(a), we have plotted the output signal-to-noise ratio (SNR) for different values of LA. It is clear that for a given value of M, the SNR improves until $LA = M$. This behavior can be easily explained by observing that (2.9) was obtained by approximating M terms (in (2.8)) by a single term. In any case, the maximum degradation in SNR (with $LA = M$) is about 2.6%.

From Fig.3(b), we see that the number of samples required by PIPKAL to converge is approximately M times that of the serial ($M = 1$) algorithm. This supports the conclusions of our analysis in section 2.2.

3.2 Channel Equalization

The channel (see Fig.4), with a raised cosine impulse response $h(n)$, was equalized with the serial and pipelined Kalman algorithms. The eigenvalue spread of the noiseless channel output is 21.71 and its input is a uniformly distributed ± 1 sequence. A white Gaussian noise source at the channel output generates the required channel SNR. The adaptive transversal equalizer has 11-taps and is updated via (2.11)-(2.14). As suggested in section 3.1, all simulations in this experiment were done with $LA = M$.

The output SNR (see Fig.5(a)), for a stationary channel, decreases slightly (at most by 3.8%) as the speed-up M is increased from unity to 10. Note that the maximum degradation is for high SNR channels. On the other hand, for a non-stationary channel (see Fig.5(b)), the output SNR curves for speed-ups of upto 10 are very close to each other. We obtained a non-stationary channel by varying the channel coefficients by 5% of their nominal value.

Thus, we conclude that the PIPKAL performance

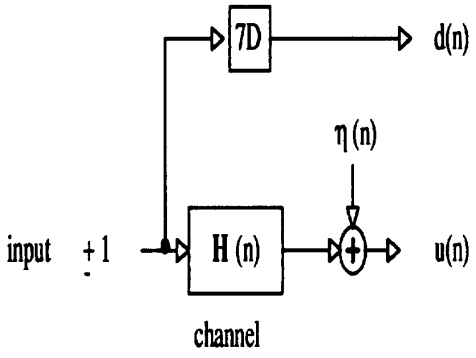


Figure 4: Channel equalization.

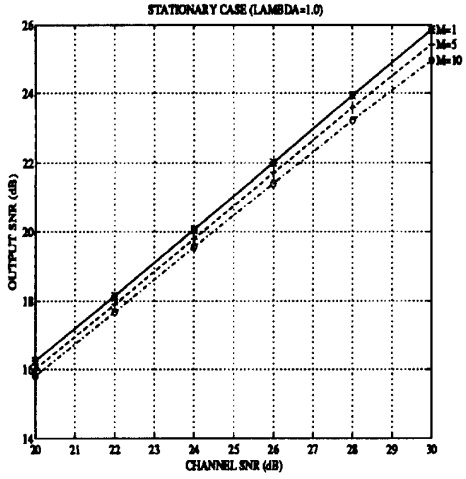
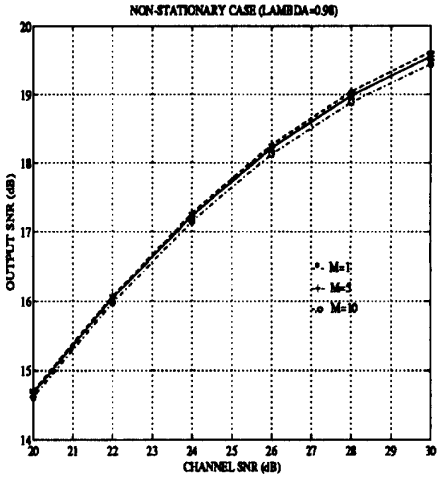


Figure 5(a)



(b)

Figure 5: Channel equalization: output SNR for (a) stationary ($\lambda = 1.0$ and (b) non-stationary ($\lambda = 0.98$) channels.

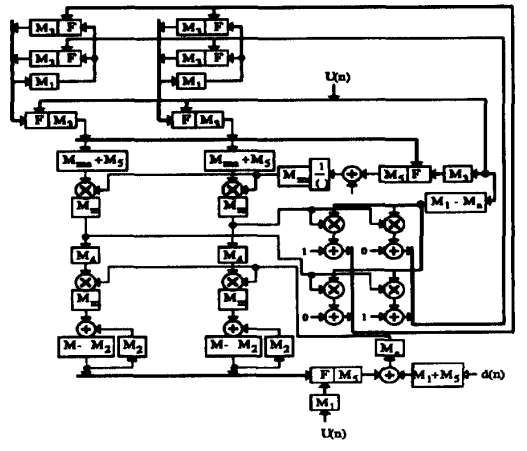
is negligibly degraded in comparison with the serial Kalman filter for the case of non-stationary environments.

4 Folded Architecture

Direct implementation of the Kalman filter is made difficult due to its inherently large hardware requirements. Hence, numerous systolic implementations using QR-decomposition [7-8] have been presented for fast recursive least-squares filtering. However, if a pipelined algorithm needs to be implemented, then we may employ the systematic folding techniques in [6] to reduce the hardware count. In the folding technique, more than one algorithmic operations (of the same type) are mapped to the same hardware operator. Thus, folding leads to a reduction in the hardware at the expense of speed. In case of pipelined algorithms, the speed-up due to pipelining is offset to some extent due to folding.

In Fig.6, we show the entire hardware for PIPKAL with order $N = 2$. The blocks **F** represent FIR filters of order N, which compute the inner product of two vectors of length N. The delay elements lumped at the output of a hardware operator are in fact employed to pipeline that operation at a fine-grain level. Thus, the **F** block, the multipliers and the adders are pipelined by M_3, M_m and M_a levels, respectively.

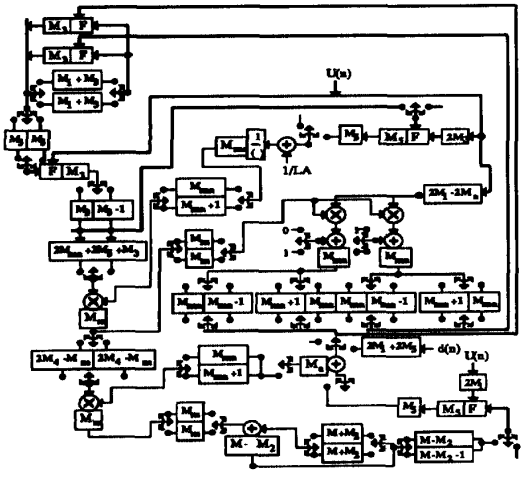
Folding by a factor of 2 results in the architecture shown in Fig.7. Note that the folded architecture has 4 **F** blocks, 4 multipliers, one divider and 5 adders. The original architecture in Fig.6, has 6 **F** blocks, 8 multipliers, one divider, and 8 adders. Thus, a substantial reduction in hardware is achieved. The speed of the folded architecture is reduced by half as compared to that of the original. However, we have shown that PIPKAL can achieve speed-up of up to 15. This implies that the folded PIPKAL in Fig.7 can achieve speed-up of up to 7.



→ : Nx1 vector

→ : scalar

Figure 6: PIPKAL for $N = 2$.



→ : Nx1 vector

→ : scalar

Figure 7: Folded PIPKAL ($N = 2$).

5 Conclusions

A pipelined Kalman filter architecture is presented. This filter has the same hardware complexity as that of the serial filter. Convergence analysis and simulation results indicate that that the degradation in performance is minimal especially in a non-stationary environment. The folding transformation [6] has been employed to reduce the hardware count of the pipelined algorithm.

The use of pipelining in combination with folding transformation [6] can be employed to develop practical architectures for complex algorithms such as those for low-bit rate speech coding.

Acknowledgment

This research was supported by the army research office by contract number DAAL-90-G-0063.

References

[1] K. K. Parhi,"Algorithm transformation techniques for concurrent processors," *Proceedings of the 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. on Acoustics, Speech and Signal Proc.*, vol. 37, pp. 1099-1117, July 1989.

[3] N. R. Shanbhag and K. K. Parhi, "A pipelined adaptive lattice filter architecture," *IEEE Trans. on Signal Processing*, vol. 41, pp. 1925-1939, May 1993.

[4] N. R. Shanbhag and K. K. Parhi, "Relaxed Look-ahead pipelined LMS adaptive filters and their application to ADPCM coder," *IEEE Trans. on Circuits and Systems*, (to appear).

[5] R. E. Kalman, "A new approach to linear and prediction problems," *J. Basic Eng.*, vol. 82, pp. 35-45, 1960.

[6] K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE J. of Solid-state Circuits*, vol. 27, no. 1, pp. 29-43, January 1992.

[7] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays," *Proc. SPIE*, vol. 298, Real-Time Signal Processing IV, pp. 298-303.

[8] J. G. McWhirter, "Recursive least-squares minimization using a systolic array," *Proc. SPIE*, vol. 431, Real-Time Signal Processing VI, pp. 105-112.