Published in: ProRISC and IEEE Benelux Workshop on Circuits, Systems and Signal Processing,
Mierlo, The Netherlands, pp. 123-132 (March 1995).

Very High Speed Least Squares Adaptive Multichannel Filtering and System Identification

G.O. Glentis and S.H. Gerez

University of Twente, Department of Electrical Engineering
Control, Systems and Computer Engineering Group (BSC)
Laboratory for Network Theory and VLSI-Design
P.O. Box 217, 7500 AE Enschede, The Netherlands.

Abstract

In this paper an arbitrarily high speed adaptive lattice algorithm for multichannel Least Squares FIR filtering and multivariable system identification, is presented. The design procedure consists of two steps. First, a channel decomposition technique is applied and the multichannel algorithm is decomposed into multiple single channel stages. Then, look-ahead techniques are applied to expand the feedback loops inherent in the adaptive lattice recursions. Look-ahead with pipeline interleaving as well as look-ahead with vectorization are used to increase the system's overall throughput rate.

1 Introduction

Adaptive lattice algorithms update the so called error parameters, that is, the difference between system's output and a desired response signal, for all intermediate filter orders [1]-[2]. The number of error variables used as well as the operations needed for their time update, depends linearly on the dimension of system's parameters. The error variables are utilized for the computation of the *reflection coefficients* which in turn are used for the update of the error parameters themselves. Moreover, they can serve for the computation of the corresponding transversal filters, using order update recursions.

An efficient adaptive lattice algorithm for LS multichannel filtering was presented in [3], based on the so called *a posteriori* error parametrization for the time update of β variables and error power variables, α. The reflection coefficients sought were estimated by the ratio β/α. Multichannel filters with different number of delay elements per input channel are allowed. Two basic units, the lattice cell and ladder cell, are required for the implementation of the algorithm. Scalar only operations, multiplications/additions and divisions are utilized. Moreover, the inherent modulatity and the local communication enables the development of a fully pipelineable architecture.

Similarly to the single channel case, further increase of the throughput rate of an adaptive multichannel lattice LS filter is prohibited by the existence of feedback loops in the updating recursions. A possible remedy to the backward feedback bottleneck is the application of look-ahead techniques. Then, as was shown in [4]-[6], the processing rate can be increased at the expense of hardware.

2 Problem formulation

A multi-input single-output FIR filter is described by the following difference equation

$$y(n) = -\sum_{i=1}^{k}\sum_{j=1}^{p_i} x_i(n-j+1)c_{ij} \tag{1}$$

where k is the number of input channels, and $x_i(n)$, $i = 1, \ldots k$, are the input signals. Integers p_i, named *the order* of the filter with respect to input i, are, in general, different for each input signal channel, $p_i \neq p_j, i, j = 1, 2 \ldots k$, [2]-[3].

Given a desired response signal $z(n)$ and input signals $x_1(n)$, $x_2(n)$ .. $x_k(n)$, the optimum filter, in the Least Squares sense, is obtained by minimizing the total squared error over a finite data horizon, weighted by an exponentially decay factor, $0 < \lambda \leq 1$,

$$E_{\mathbf{p}_k}(N) = \langle e(n), e(n)\rangle, \quad e(n) = z(n) - y(n)$$

where $\langle x, y\rangle = \sum_{n=0}^{N} \lambda^{N-n} x y^t$. The pertinent LS filter satisfies the set of linear equations

$$\mathbf{R}_{\mathbf{p}_k}(N)\mathcal{C}_{\mathbf{p}_k}(N) = -\mathbf{d}_{\mathbf{p}_k}(N) \tag{2}$$

$\mathcal{C}_{\mathbf{p}_k}(N)$ is the vector that carries the filter coefficients, $\mathbf{R}_{\mathbf{p}_k}(N)$ is the sampled autocorrelation matrix and $\mathbf{d}_{\mathbf{p}_k}(N)$ is the sampled crosscorrelation vector defined as

$$\mathbf{R}_{\mathbf{p}_k}(N) = \langle \mathcal{X}_{\mathbf{p}_k}(n), \mathcal{X}^t_{\mathbf{p}_k}(n)\rangle, \quad \mathbf{d}_{\mathbf{p}_k}(N) = \langle \mathcal{X}_{\mathbf{p}_k}(n), z(n)\rangle, \quad \mathcal{X}_{\mathbf{p}_k}(n) = \Big[x_i(n-j+1)\Big]_{\substack{i=1\ldots k\\ j=1\ldots p_i}}$$

$\mathcal{X}_{\mathbf{p}_k}(n)$ is the regressor vector, and $\mathbf{p}_k \equiv [p_1\ p_2 \ldots p_k]$ is a multi-index that carries the individual filter orders with respect to input channels, $i = 1, \ldots k$.

An alternative representation of eq. (1) based on a set of orthogonal input signals is given by

$$y(n) = -\sum_{s=1}^{k}\sum_{j=1}^{p_s - p_{s+1}}\sum_{i=1}^{s} \epsilon^{b(i)}_{s,j-1,i}(n) K^c_{s,j,i}(N), \tag{3}$$

$K^c_{s,j,i}(N)$ are the so-called *reflection coefficients* and $\epsilon^{b(\mu)}_{s,j,i}(n)$ is a set of single channel backward predictors that satisfy the orthogonality condition

$$\langle \epsilon^{b(\mu)}_{s,j,i}(n), \epsilon^{b(\nu)}_{s1,j1,i1}(n)\rangle = \delta(s-s1, j-j1, i-i1, \mu-\nu)\alpha^{b(\mu)}_{s,j,i}(N)\rangle$$

This set of mutually orthogonal signals can be produced by processing the input data $x_i(n)$ as shown in [3]. The autocorrelation matrix corresponding to the transformed data is diagonal, and thus, the reflection coefficients are obtained as

$$K^c_{s,j,i}(N) = -\frac{\beta^c_{s,j,i}(N)}{\alpha^{b(i)}_{s,j-1,i}(N)} \quad \beta^c_{s,j,i}(N) = \langle \epsilon^{b(i)}_{s,j-1,i}(n), z(n)\rangle \tag{4}$$

3 The Adaptive Lattice Algorithm

Based on order and time updates following from the shift invariance properties of the autocorrelation matrix, an a posteriori adaptive lattice algorithm has been derived in [3]. During each iteration, a

$$\epsilon_0^c(N) \equiv \epsilon_{s,j-1,s}^c(N), \epsilon_0^{b(\mu)}(N) = \epsilon_{s,j-1,s}^{b(\mu)}(N), \epsilon_0^{f(\nu)}(N) = \epsilon_{s,j-1,s}^{f(\nu)}(N) \alpha_0^c(N) = \alpha_{s,j-1,s}^c(N),$$
$$\alpha_0^w(N) = \alpha_{s,j-1,s}^w(N), \alpha_0^{b(\mu)}(N) = \alpha_{s,j-1,s}^{b(\mu)}(N), \alpha_0^{f(\nu)}(N) = \alpha_{s,j-1,s}^{f(\nu)}(N)$$

FOR $i = 0$ TO $(s-1)$

LET $\mathbf{I} = \text{right\_rotate}[12\ldots s-1]_{i+1}$

FOR $\mu = 0$ TO $(s-1)$, LET $\nu = \mathtt{I}(\mu+1)$

THE LATTICE CELL

$$e_i^{b(\mu+1)}(N) = \epsilon_i^{b(\mu+1)}(N) \alpha_i^{w(\nu)}(N) \quad (1.1)$$
$$\beta_{i+1}^{b(\mu+1)}(N) = \lambda \beta_{i+1}^{b(\mu+1)}(N-1) + \epsilon_i^{f(\nu)}(N) Z_N^h e_i^{b(\mu+1)}(N) \quad (1.2)$$
$$k_{i+1}^{b(\mu+1)}(N) = -\beta_{i+1}^{b(\mu+1)}(N)/\alpha_i^{f(\nu)}(N) \quad (1.3)$$
$$k_{i+1}^{f(\nu)}(N) = -\beta_{i+1}^{b(\mu+1)}(N)/Z_N^h \alpha_i^{b(\mu+1)}(N) \quad (1.4)$$
$$\epsilon_{i+1}^{b(\mu+1)}(N) = Z_N^h \epsilon_i^{b(\mu+1)}(N) + \epsilon_i^{f(\nu)}(N) k_{i+1}^{b(\mu+1)}(N) \quad (1.5)$$
$$\epsilon_{i+1}^{f(\nu)}(N) = \epsilon_i^{f(\nu)}(N) + Z_N^h \epsilon_i^{b(\mu+1)}(N) k_{i+1}^{f(\nu)}(N) \quad (1.6)$$
$$\alpha_{i+1}^{f(\nu)}(N) = \alpha_i^{f(\nu)}(N) + \beta_{i+1}^{b(\mu+1)}(N) k_{i+1}^{f(\nu)}(N) \quad (1.7)$$
$$\alpha_{i+1}^{b(\mu+1)}(N) = Z_N^h \alpha_i^{b(\mu+1)}(N) + \beta_{i+1}^{b(\mu+1)}(N) k_{i+1}^{b(\mu+1)}(N) \quad (1.8)$$
$$a_{i+1}^{w(\nu)}(N) = a_i^{w(\nu)}(N) + (e_i^{b(\mu+1)}(N))^2 / Z_N^h \alpha_i^{b(\mu+1)}(N) \quad (1.9)$$

ENDFOR μ

THE LADDER CELL

$$\beta_{i+1}^c(N) = \lambda \beta_{i+1}^c(N-1) + \epsilon_i^c(N) \epsilon_i^{b(i+1)}(N) \alpha_i(N) \quad (1.10)$$
$$k_{i+1}^c(N) = -\beta_{i+1}^c(N)/\alpha_i^{b(i+1)}(N) \quad (1.11)$$
$$\epsilon_{i+1}^c(N) = \epsilon_i^c(N) + \epsilon_i^{b(i+1)}(N) k_{i+1}^c(N) \quad (1.12)$$

ENDFOR i

$$\epsilon_{s,j,s}^c(N) = \epsilon_s^c(N), \epsilon_{s,j,s}^{b(\mu)}(N) = \epsilon_s^{b(\mu)}(N), \epsilon_{s,j,s}^{f(\nu)}(N) = \epsilon_s^{f(\nu)}(N)$$
$$\alpha_{s,j,s}^c(N) = \alpha_s^c(N), \alpha_{s,j,s}^w(N) = \alpha_s^w(N), \alpha_{s,j,s}^{b(\mu)}(N) = \alpha_s^{b(\mu)}(N), \alpha_{s,j,s}^{f(\nu)}(N) = \alpha_s^{f(\nu)}(N)$$

Table 1. The a posteriori adaptive lattice algorithm. Z_N represents a delay with respect to N. It is activated by h, as: $h = 1$, if $\nu = s$, $h = 0$, otherwise.

set of reflection coefficients is updated which in turn is utilized for the order update of the error parameters themselves. The algorithm is summarized in Table 1.

A posteriori error variables for the filter as well as for the forward and backward predictors, are estimated using a recursive lattice-ladder scheme. The passage from error variables of order $(s, j-1, s)$ to the successive (s, j, s) is accomplished via a set of s single step recursions. During the first iteration $(i = 1)$ of a new step, the forward prediction errors are coupled together with the backward prediction errors in a particular way which is: $[\epsilon^{f1}, \epsilon^{bs}]$, $[\epsilon^{f2}, \epsilon^{b1}]$, $[\epsilon^{f3}, \epsilon^{b2}]$, $[\epsilon^{fs}, \epsilon^{b(s-1)}]$. When $(i = 2)$ prediction errors are coupled together as: $[\epsilon^{f1}, \epsilon^{b(s-1)}]$, $[\epsilon^{f2}, \epsilon^{bs}]$, $[\epsilon^{f3}, \epsilon^{b1}]$, $[\epsilon^{fs}, \epsilon^{b(s-2)}]$. Finally, $(i = s)$ the pairs are: $[\epsilon^{f1}, \epsilon^{b1}]$, $[\epsilon^{f2}, \epsilon^{b2}]$, $[\epsilon^{f3}, \epsilon^{b3}]$, $[\epsilon^{fs}, \epsilon^{bs}]$ (see Figure 1).

The lattice algorithm of Table 1 is depicted in Figure 2, for a special case of a three channel problem, $k = 3$, and final multi index $\mathbf{p}_3 = [p_1, p_2, p_3] = [7, 3, 1]$.

The computational complexity of each of the lattice and ladder cells are found to be $C_{Lattice} = 8\ MAD + 3\ DIV$, and $C_{Ladder} = 3\ MAD + 1\ DIV$, where MAD and DIV stand for the multiplication-and-addition and the division operation, respectively. The adaptive lattice consists

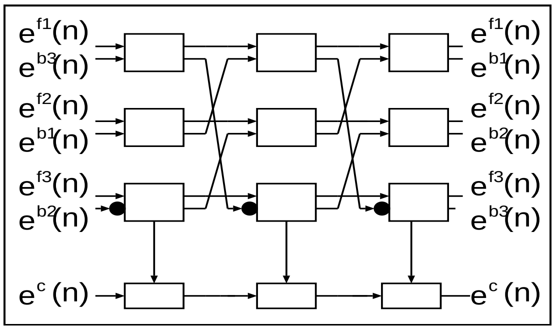

Figure 1: Error variables update ($s = 3$). $\bullet$ represents a unit time delay.

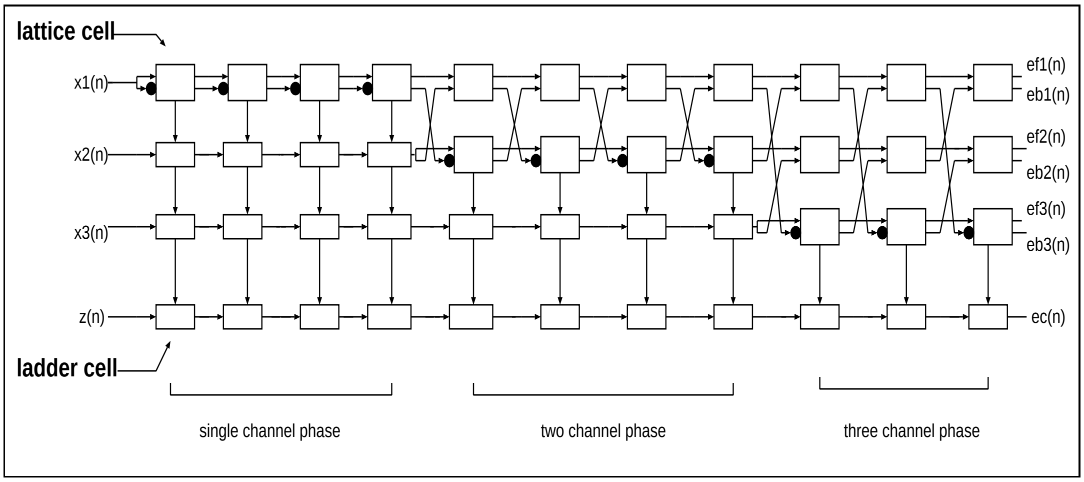

Figure 2: The highly modular normalized adaptive lattice structure for $\mathbf{m}_3 = [7, 3, 1]$.

of a grid of $k \times (k+1)$ locally interconnected cells. The total computational complexity is

$$C_{Total} = C_{Lattice}^{Total} + C_{Ladder}^{Total} \tag{5}$$

where

$$C_{Lattice}^{Total} = \Big(\sum_{s=1}^{k}(p_s - p_{s+1})s^2\Big) \times (8\ MAD + 3\ DIV) \tag{6}$$

$$C_{Ladder}^{Total} = \Big(\sum_{s=1}^{k}(p_s - p_{s+1})(k+1-s) * s\Big) \times (3\ MAD + 1\ DIV) \tag{7}$$

In the special case of equal individual filter lengths, i.e., $p_i = p_j$, the above figure is simplified to

$$C_{Total}^{Equal} = k * 2p \times (8\ MAD + 3\ DIV) + kp \times (3\ MAD + 1\ DIV) \tag{8}$$

4 Improving the Iteration Period Bound

When considering the implementation of a DSP algorithm, it is useful to represent the algorithm as an *iterative data-flow graph* (see e.g. [5]). The graphs are iterative as they are meant to be executed repetitively for each set of samples arriving at the input at regular time instants. The separation between these instants is called the *iteration period* T_0. A data-flow graph (DFG) consists of *nodes* (or vertices) that represent computations and *edges* that indicate precedence relations between nodes. An edge can incorporate zero or more *delay elements.* A delay element delays its input by one iteration period before passing it to its output. A DFG will be denoted by $G(V, E)$, with V the set of nodes and E the set of edges. The number of delay elements on an edge $e \in E$ will be given by $\Delta(e)$.

An issue that is central in the implementation of an algorithm is the speed that can be achieved, given a *library* of computational hardware elements. For the sake of simplicity, it is assumed in this paper that there is a unique mapping from the nodes in the DFG to the hardware elements. Each element requires a specific time to perform a specific operation (addition, multiplication, etc.). This time will be an integer multiple of the *system clock period.* The time necessary to compute a node $v \in V$ will be denoted by $\delta(v)$. Then, given a DSP algorithm and a hardware library, it is possible to make statements on the iteration period T_0.

In the case of an *acyclic* DFG (a graph without feedback), the T_0 of the implementation can be decreased arbitrarily by the use of *pipelining* and the multiplication of the hardware, given a certain library. This does not require any modification of the DFG. However, the same is not possible in case of a *cyclic* DFG. There, the minimum T_0, $T_{0_{min}}$, is bounded by the topology of the DFG due to feedback loops: a node n whose output propagates through a number of other nodes and then reaches an input of n, has to wait through this propagation time. Of course, a feedback loop always needs to contain an edge with a delay element in order for the DFG to be computable. The minimum T_0 is defined as follows [7, 8][1]:

$$T_{0_{min}} = \left\lceil \max_{\text{all loops } L} \frac{\sum_{v \in L} \delta(v)}{\sum_{e \in L} \Delta(e)} \right\rceil \tag{9}$$

The minimum T_0 of a DFG can only be improved by modifying the DFG itself, by application of a so-called *look-ahead* transformation [4, 6]. This will be explained by using the following first-order time-invariant recurrence taken from [6] as a vehicle:

$$x(n+1) = ax(n) + bu(n) \tag{10}$$

Here u is an input variable and x is both a state and an output variable. The DFG of this system is illustrated in Figure 3(b), making use of the symbol for a combined multiply-add operation shown in Figure 3(a).

If the number of system clock cycles to perform a multiplication and an addition is T_{ma}, the application of Equation 9 gives that $T_{0_{min}} = T_{ma}$.

Applying a look-ahead transformation means the recursive substitution of Equation 10 in itself. Doing this once gives:

$$x(n+1) = a(ax(n-1) + bu(n-1)) + bu(n) = a^2x(n-1) + abu(n-1) + bu(n) \tag{11}$$

The corresponding DFG is shown in Figure 3(c). The DFG is said to have *pipeline interleaving* [6]: all values of $x(n)$ are computed sequentially, but the computation of $x(n)$ is always based on $x(n-2)$ instead of $x(n-1)$. One should be aware that look-ahead transformations modify the behavior of DSP algorithms due to finite word-length effects.

[1] In this expression, $\lceil x \rceil$ means the smallest integer $\geq x$. Besides, the set membership predicate $\in$ is assumed to be applicable to nodes and edges in a loop (a loop is formally not a set, but an alternating sequence of nodes and edges).

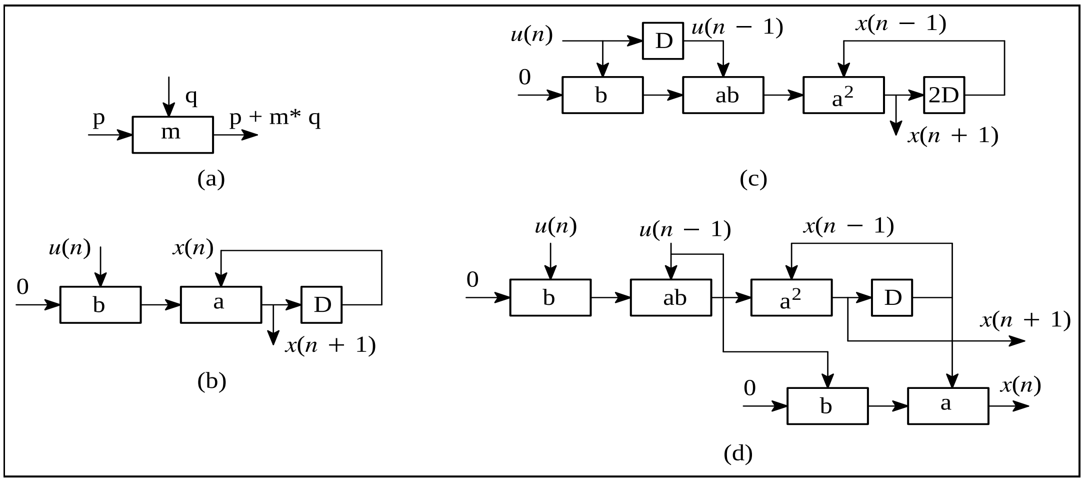

Figure 3: The symbol for the multiply-add operation (a), the DFG of a first-order recurrence (b), the application of pipeline interleaving with a look-ahead factor of 2 (c) and the application of vectorization with a look-ahead factor of 2 (d).

One now finds that $T0_{min} = \lceil \frac{T_{ma}}{2} \rceil$. This is an improvement with respect to the situation without look-ahead if $\lceil \frac{T_{ma}}{2} \rceil < T_{ma}$, which actually means that it is possible to have two pipeline stages in the hardware for a multiplication and addition.

In general, Equation 10 can be substituted M times in itself, leading to $T0_{min} = \lceil \frac{T_{ma}}{M} \rceil$. This is interesting as long as M pipeline stages can be incorporated in the multiplication-addition hardware. As there is a limit to the number of pipeline stages in the hardware, the speed-up that can be obtained by pipeline interleaving is also limited.

Further increase in the iteration period is still possible by the application of *block processing* or *vectorization*. For cyclic DFG's, first look-ahead transformations have to be applied. So, Equation 11 is the basis, for a look-ahead factor of two. In contrast to pipeline interleaving, however, two inputs ($u(n)$ and $u(n-1)$) and two outputs ($x(n+1)$ and $x(n)$) are consumed and respectively produced simultaneously. One actually gets the following equations:

$$\begin{aligned} x(n+1) &= a^2x(n-1) + abu(n-1) + bu(n) \\ x(n) &= ax(n-1) + bu(n-1) \end{aligned} \tag{12}$$

The corresponding DFG is shown in Figure 3(d). Note that there is now a single delay in the feedback loop compared to two delays in Figure 3(c), even when the physical delay in terms of the system clock is the same in both situations. This is a direct consequence of the definition of a delay in terms of the iteration period: in Figure 3(d) two inputs are processed in one iteration period. One also says that the delay element is *2-slow*.

In general the hardware costs involved with block processing are higher than with pipeline interleaving. It is therefore advisable to use pipeline interleaving until the limits of the pipelinability of the feedback loop has been reached and then to apply block processing if further speed-up is required. A detailed discussion of this topic can be found in [6].

Consider the system of Equation 10 after rewriting it for a look-ahead factor of 4:

$$x(n+1) = a^4x(n-3) + a^3bu(n-3) + a^2bu(n-2) + abu(n-1) + bu(n) \tag{13}$$

The equivalent expression:

$$x(n+1) = a^4x(n-3) + b(u(n) + a(u(n-1) + a(u(n-2)) + au(n-3)))) \tag{14}$$

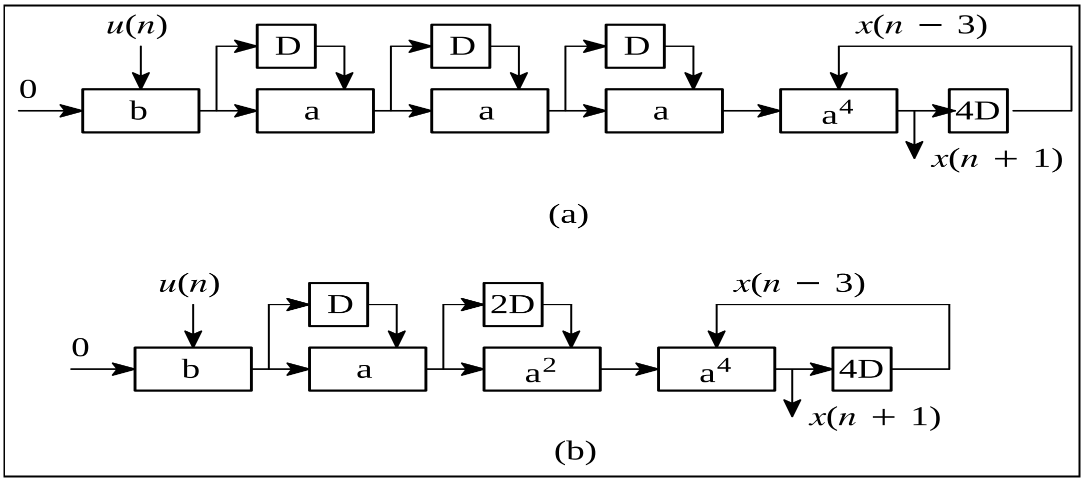

Figure 4: The pipelined interleaved DFG for a look-ahead factor of 4 without (a) and with (b) decomposition.

leads to the DFG shown in Figure 4 (using the symbol for a multiply-add operation given in Figure 3(a)). In general, a pipeline interleaved realization of a recurrent system in this way will lead to a hardware overhead that grows linearly with the look-ahead factor. However, it is possible to have an overhead that only grows logarithmically by using the *decomposition* technique [6]. In the case of the example of Equation 13, this can be seen by rewriting it as:

$$x(n+1) = a^4 x(n-3) + b(u(n) + au(n-1) + a^2(u(n-2) + au(n-3))) \tag{15}$$

which corresponds to the DFG of Figure 4(b). In general, every doubling of the look-ahead factor requires one additional multiply-add element.

5 Pipelining the multichannel LS adaptive lattice algorithm

When channel decomposition is applied to the multichannel adaptive lattice filter, a set of single channel, modular, locally connected, lattice and ladder sections, appear. Thus, the first task for increasing the system pipelineability has been accomplished. The system throughput is now equal to a single channel counterpart. Further speed up can be obtained by applying a look-ahead technique locally on each of the lattice and ladder cells.

A simple glance on Table 1 reveals that the system speed-up is prohibited by the existence of feedback loops in Eqs.(1.2) and (1.10) that update the state variables of the lattice and ladder cells. Thus, these two equations have to be processed further in such a way that the overall pipelineability is increased. Since both of them are first-order systems, the look-ahead techniques discussed in the previous section can directly be applied.

A look-ahead transformation with pipeline interleaving of length M can applied for the update of the state variables $\beta^c_{i+1}(N)$ and $\beta^{b(\mu+1)}_{i+1}(N)$. These are now estimated according to the scheme depicted in Fig. 3(c). This could be further simplified following the decomposition approach of Fig. 4. No more modifications on eqs. of Table 1 are necessary. The reorganization of the operations performed by a lattice cell when a look-ahead transformation of length $M = 4$ is utilized, is depicted in Figure 5. The hardware requirements is $log_2(M)$ extra MAD for each lattice and for each ladder cell, while the processing time for each new sample (throughput) is increased by a factor of M.

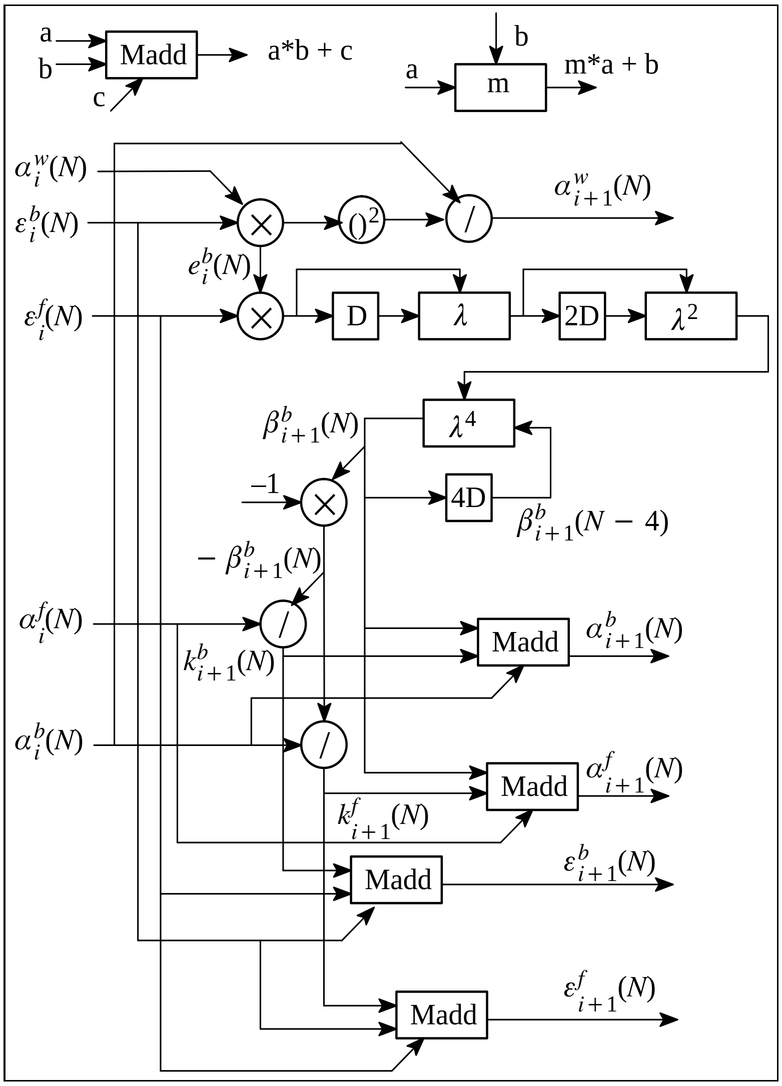

Figure 5: The Lattice cell transformation when a look-ahead of length $M = 4$ and pipeline interleaving is applied. To simplify notation, parameters have been stripped of the channel indices (μ) and (ν). The delay Z_N^h element that is required by some of the Eqs. of Table 1, is introduced outside the main lattice cell.

When a look-ahead transformation with vectorization of length L is applied for the update of the state variables $\beta_{i+1}^{c}(N)$ and $\beta_{i+1}^{b(\mu+1)}(N)$ further modifications are required. The processing scheme is now as follows

$$\beta_{i+1}^{b(\mu+1)}(N) = \mathcal{F}^{Lattice}\left[\beta_{i+1}^{b(\mu+1)}(N-L), U_i^{Lattice}(N)\right] \tag{16}$$

$$U_{i+1}^{Lattice}(N) = \mathcal{G}^{Lattice}\left[\beta_{i+1}^{b(\mu+1)}(N), U_i^{Lattice}(N)\right] \tag{17}$$

$$\beta_{i+1}^{c}(N) = \mathcal{F}^{Ladder}\left[\beta_{i+1}^{c}(N-L), U_i^{Ladder}(N)\right] \tag{18}$$

$$U_{i+1}^{Ladder}(N) = \mathcal{G}^{Ladder}\left[\beta_{i+1}^{c)}(N-L), U_i^{Ladder}(N)\right] \tag{19}$$

where

$$U_i^{Lattice}(N) = \Big[\epsilon_i^{b(\mu+1)}(N-\ell)\ \epsilon_i^{f(\nu)}(N-\ell)$$
$$\alpha_i^{b(\mu+1)}(N-\ell)\ \alpha_i^{f(\nu)}(N-\ell)\ \alpha_i^{w(\nu)}(N-\ell)\Big]_{\ell=0,1\ldots L-1}$$

and

$$U_i^{Ladder}(N) = \left[\epsilon_i^{c}(N-\ell)\right]_{\ell=0,1\ldots L-1}$$

This time L samples are consumed each iteration of the algorithm and L samples are produced at the output. $\mathcal{F}^{Lattice}(.)$ and $\mathcal{F}^{Ladder}(.)$ represent the transformation resulting by applying the look-ahead on the state equations, Eq.(1.2), and Eq.(1.10), respectively. $\mathcal{G}^{Lattice}(.)$ and $\mathcal{G}^{Ladder}(.)$ represent the transformation introduced on the output recursions. The state transformation follows Fig. 3.d. The exact form of the state, as well as the output transformations can be found in [4],[6]. The hardware requirements are roughly $L * C_{Lattice} + L\ MAD$ for the lattice cell, and $L * C_{Ladder} + L\ MAD$ for the ladder cell. Since L samples are processed in a single iteration, the computation overhead per sample is 1 MAD for each cell, while the processing speed is increased by a factor of L.

6 Conclusions

An arbitrarily high speed rate implementation of a least squares adaptive lattice filter has been developed. This task is accomplished by appling a combined channel decomposition method at the algorithmic level, and a look-ahead technique at the implementation level. The multichannel scheme is first decomposed into a set of single channel stages, that are locally interconnected in a modular way. A look-ahead transformation based on pipeline interleaving or vectorization is subsequently be applied to the individual stages. The system throughput rate can then increased, at the expense of hardware.

References

[1] N. Kalouptsidis and S. Theodoridis, (eds), Adaptive System Identification and Signal Processing, Prentice Hall, 1993.

[2] G. Glentis and N. Kalouptsidis, Efficient Order Recursive Algorithms for Multichannel Least Squares Filtering, *IEEE Transactions on Signal Processing*, SP-40, pp.1354-1374,June 1992.

[3] G. Glentis and N. Kalouptsidis, Fast Adaptive Algorithms for Multichannel Filtering and System Identification, *IEEE Transactions on Signal Processing*, SP-40,pp. 2433-2458, October 1992.

[4] T.H.Y. Meng and D.G. Messerschmitt. Arbitrarily high sampling rate adaptive filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35(4):455–470, April 1987.

[5] K.K. Parhi. Algorithm transformation techniques for concurrent processors. *Proceedings of the IEEE*, 77(12):1879–1895, December 1989.

[6] K.K. Parhi and D.G. Messerschmitt. Concurrent cellular vlsi adaptive filter structures. *IEEE Transactions on Circuits and Systems*, CAS-34(10):1141–1151, October 1987.

[7] R. Reiter. Scheduling parallel computations. *Journal of the ACM*, 15(4):590–599, October 1968.

[8] M. Renfors and Y. Neuvo. The maximum sampling rate of digital filters under hardware speed constraints. *IEEE Transactions on Circuits and Systems*, CAS-28(3):196–202, March 1981.