Computers Elect. Engng Vol. 18, No. 3/4, pp. 291–300, 1992
Printed in Great Britain. All rights reserved

0045-7906/92 $5.00 + 0.00
Copyright © 1992 Pergamon Press Ltd

A PARALLEL IMPLEMENTATION OF THE INVERSE QR ADAPTIVE FILTER

AVINASH L. GHIRNIKAR[1], S. T. ALEXANDER[2] and R. J. PLEMMONS[3]

[1]ViTel Communications Corporation, 4699 Old Ironsides Drive, Suite 300, Santa Clara, CA 95054, [2]Electrical and Computer Engineering Department, North Carolina State University, Raleigh, NC 27695–7911 and [3]Department of Mathematics and Computer Science, Wake Forest University, Winston-Salem, NC 27109, U.S.A.

(Received 15 December 1990; accepted in final revised form 15 August 1991)

Abstract—A systolic array architecture is proposed for an inverse QR algorithm for recursive least squares signal processing. This algorithm provides a solution for the least squares filter vector which does not require the computationally intensive step of backsubstitution needed in the direct QR method. The inverse QR algorithm is therefore highly amenable to a parallel implementation, resulting in a structure which produces the N filter coefficients every $N + 1$ clock cycles. Within the paper, the individual processing elements contained in the inverse QR algorithm are outlined, after which it is demonstrated how these processing elements may be connected to implement the overall inverse QR array. Timing considerations, data rates, latency and throughput are also discussed.

1. INTRODUCTION

The area of adaptive signal processing known as recursive least squares (RLS) filtering has become extremely important in recent years [1,2]. These techniques offer the possibility of algorithms which converge much more rapidly to desired solutions than traditional gradient-based approaches, such as the LMS algorithm. However, for an adaptive filter of N coefficients, the initial derivations for RLS filters using the matrix inverse lemma required on the order of N^2 operations per time update [abbreviated as $O(N^2)$], compared to $O(N)$ for LMS. This complexity was reduced to $O(N)$ operations per update by exploiting the data shifting property of the input, as well as the geometrical properties of the RLS problem [1,2]. Unfortunately, the family of "fast" RLS algorithms derived in this manner has displayed a tendency to become numerically unstable [3].

Consequently, research began into determining algorithms which would preserve the rapid convergence of RLS while avoiding the problems of numerical instability. One of the most promising aproaches has been applying matrix techniques based on the QR decomposition [2,4]. These techniques are also sometimes called rotation-based methods since one of the distinguishing features is a set of norm-preserving Givens rotations. These methods are numerically very well-behaved and have been shown to be stable for RLS applications [5].

Another important feature of QR algorithms is that they may be constructed in parallel implementations [2,6]. Exploiting the parallel attributes of QR algorithms thus allows their consideration for application to real-time RLS problems. With this in mind, this paper presents a systolic array architecture for an inverse QR algorithm originally derived in [7]. The inverse QR algorithm allows the LS weight vector to be found by a simple vector update, thus avoiding the highly serial backsubstitution step required in the direct QR algorithm. Additionally, a preliminary theoretical analysis has shown that the finite precision properties of the inverse QR algorithm are indicative of a stable algorithm [8].

One of the main benefits of the method proposed in this paper is its use of simple rotation-based operations. A more complex alternative approach different from the one developed in this paper has also recently been proposed [9]. However, in [9] storage of the matrix elements is through an odd-even partitioning where each processor is associated with a 2×2 block of neighboring elements of the matrix. The functionality of the array blocks used in [9] also differs from those in the present paper.

In this paper, Section 2 provides a brief background of adaptive filtering using the direct QR approach and then introduces the inverse QR algorithm. The individual processing elements contained in the inverse QR algorithm are described in Section 3. Sections 4 and 5 next demonstrate how these processing elements may be connected to implement the overall inverse QR array.

Timing considerations which are necessary for the correct operation of the array, as well as the effect on the data rate, are also discussed. Section 6 then concludes with some discussion of the latency and throughput of the array introduced in this paper.

2. BACKGROUND

The development of RLS in this section is necessarily brief and the reader is directed to [1,2] for a more detailed examination. The transversal form of the RLS filter examined in this paper estimates a desired signal sequence $\{d(n)\}$ using a data sequence $\{x(n)\}$. The N-coefficient RLS filter:

$$\mathbf{w}(n) = [w_1(n), \quad w_2(n), \ldots, w_N(n)]^T \tag{1}$$

minimizes the error measure:

$$\epsilon(n) = \sum_{i=1}^{n} \lambda^{n-i} e^2(i \mid n) \tag{2}$$

where n is the total number of samples used in the minimization. In (2), $e(i \mid n)$ is the error resulting from predicting $d(i)$ using $\mathbf{w}(n)$, and is given by:

$$e(i|n) = d(i) - \sum_{k=1}^{N} w_k(n) x(i-k+1). \tag{3}$$

The parameter λ in (2) is a weighting factor (sometimes called the forgetting factor) and has a value in the range $0 \ll \lambda < 1$ to weight more recent errors more heavily.

The adaptive filtering problem may be cast as a least squares problem by first defining the n-component vectors $\mathbf{e}(n)$ and $\mathbf{d}(n)$ as:

$$\mathbf{e}(n) = [e(1 \mid n), \quad e(2 \mid n), \ldots, e(n \mid n)]^T \tag{4}$$

$$\mathbf{d}(n) = [d(1), \quad d(2), \ldots, d(n)]^T. \tag{5}$$

Substituting (1), (3) and (5) in (4) then gives:

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n) \tag{6}$$

where $\mathbf{X}(n)$ is the $n \times N$ data matrix:

$$\mathbf{X}(n) = \begin{bmatrix} x(1) & 0 & \ldots & 0 \\ x(2) & x(1) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x(n-1) & x(n-2) & \ldots & x(n-N) \\ x(n) & x(n-1) & \ldots & x(n-N+1) \end{bmatrix}. \tag{7}$$

This development has assumed that both $d(n)$ and $x(n)$ are zero for $n \leqslant 0$, which is sometimes called the prewindowed case.

Using (6), the error measure in (2) is now seen to be given by:

$$\epsilon(n) = \| \Lambda(n)\mathbf{d}(n) - \Lambda(n)\mathbf{X}(n)\mathbf{w}(n) \|^2 \tag{8}$$

where $\| \, . \, \|$ denotes the Euclidean norm and

$$\Lambda(n) = \text{Diag}[\sqrt{\lambda^{n-1}}, \quad \sqrt{\lambda^{n-2}}, \ldots, \sqrt{\lambda}, 1] \tag{9}$$

is the diagonal matrix which incorporates the weighting factor. An orthogonal matrix $\mathbf{Q}(n)$ may now be determined which operates on the exponentially weighted data matrix such that [2,4]:

$$\mathbf{Q}^T(n)\Lambda(n)\mathbf{X}(n) = \begin{bmatrix} \mathbf{R}(n) \\ \mathbf{0} \end{bmatrix} \tag{10}$$

where $\mathbf{R}(n)$ is an $N \times N$ upper triangular matrix. This same transformation may be applied to the weighted desired signal vector $\mathbf{d}(n)$ to give:

$$\mathbf{Q}^T(n)\Lambda(n)\mathbf{d}(n) = \begin{bmatrix} \mathbf{z}(n) \\ \mathbf{v}(n) \end{bmatrix} \tag{11}$$

where $\mathbf{z}(n)$ is an N length vector and $\mathbf{v}(n)$ is an $(n - N)$ length vector. It is now straightforward [2,4] to show that the set of linear equations:

$$\mathbf{R}(n)\mathbf{w}(n) = \mathbf{z}(n) \tag{12}$$

may be then be solved by backsubstitution to obtain the filter vector $\mathbf{w}(n)$.

In the direct QR method it is assumed that $\mathbf{R}(n-1)$ is available from the previous iteration. As a result, the RLS problem reduces to determining $\mathbf{R}(n)$ from $\mathbf{R}(n-1)$. It is known [10] that $\mathbf{R}(n)$ may be determined from $\mathbf{R}(n-1)$ by the transformation:

$$\mathbf{T}(n)\begin{bmatrix}\sqrt{\lambda}\mathbf{R}(n-1)\\ \mathbf{x}^T(n)\end{bmatrix} = \begin{bmatrix}\mathbf{R}(n)\\ \mathbf{0}^T\end{bmatrix} \tag{13}$$

where $\mathbf{x}^T(n) = [x(n), x(n-1), \ldots, x(n-N+1)]$ is the vector incorporating the new data sample collected at time n and $\mathbf{T}(n)$ is an $(N+1) \times (N+1)$ orthogonal matrix. Equation (13) implies the recursive rank-one update relation:

$$\mathbf{R}^T(n)\mathbf{R}(n) = \lambda\mathbf{R}^T(n-1)\mathbf{R} + \mathbf{x}(n)\mathbf{x}^T(n). \tag{14}$$

It can also be shown [10] that the same transformation in (13) may be used to update $\mathbf{z}(n-1)$, giving:

$$\mathbf{T}(n)\begin{bmatrix}\sqrt{\lambda}\mathbf{z}(n-1)\\ d(n)\end{bmatrix} = \begin{bmatrix}\mathbf{z}(n)\\ \zeta(n)\end{bmatrix}. \tag{15}$$

Once $\mathbf{R}(n)$ and $\mathbf{z}(n)$ are determined via (13) and (15), the LS solution at time index n can be easily obtained by solving (12) by back-substitution.

In the inverse QR approach, a rank-one update relationship like (14) is derived for the inverse Cholesky factor $\mathbf{R}^{-T}(n)$. Then it can be shown that a very natural consequence of updating $\mathbf{R}^{-T}(n)$ produces a vector $\mathbf{u}(n)$ which can be used to update the transversal filter parameters [7,11]. The inverse QR algorithm results from this approach and a pseudocode of the algorithm is listed as follows:

```
INITIALIZE:  $\mathbf{w}(0) = \mathbf{0}$,  $\mathbf{x}(0) = \mathbf{0}$,  $\mathbf{R}^{-T}(0) = \delta\mathbf{I}$      (16)
For $n = 1$ to $n$ final do:
    $\mathbf{x}^T(n) = [x(n), x(n-1), \ldots, x(n-N+1)]$      (17)
    $\mathbf{a}(n) = \lambda^{-1/2}\mathbf{R}^{-T}(n-1)\mathbf{x}(n)$      (18)
    $\mathbf{u}^{(0)}(n) = 0,\ b^{(0)}(n) = 1$      (19)
    For $i = 1$ to $N$ do:
        $b^{(i)}(n) = \sqrt{[b^{(i-1)}(n)]^2 + a_i^2(n)}$      (20)
        $s_i(n) = a_i(n)/b^{(i)}(n)$      (21)
        $c_i(n) = b^{(i-1)}(n)/b^{(i)}(n)$      (22)
        For $j = 1$ to $i$ do:
            $r_{ij}(n) = \lambda^{-1/2}c_i(n)r_{ij}(n-1) - s_i(n)u_j^{(i-1)}(n)$      (23)
            $u_j^{(i)}(n) = c_i(n)u_j^{(i-1)}(n) + \lambda^{-1/2}s_i(n)r_{ij}(n-1)$      (24)
        End $j$ loop
    End $i$ loop
    $\mathbf{w}(n) = \mathbf{w}(n-1) + \left[\frac{d(n) - \mathbf{w}^T(n-1)\mathbf{x}(n)}{b^{(N)}(n)}\right]\mathbf{u}^{(N)}(n)$      (25)
End $n$ loop.
```

The initialization of the inverse QR algorithm is done in (16), in which $\mathbf{R}^{-T}(0)$ is initialized to a diagonal matrix. The quantities $c_i(n)$ and $s_i(n)$ are the Givens rotation angles used to

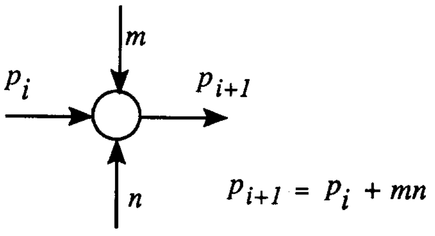


Fig. 1. Inner product cell.

update $\mathbf{R}^{-T}(n-1)$ and the $r_{ij}(n)$ are the elements of the updated matrix $\mathbf{R}^{-T}(n)$. The scalar $b^{(i)}(n)$ allows a convenient calculation of these angle parameters and the vector $\mathbf{a}(n)$ arises in the rank-one update of $\mathbf{R}^{-T}(n-1)$. The reader is referred to [7,11] for a detailed derivation of the inverse QR algorithm.

3. SYSTOLIC ARRAY COMPONENTS

The inverse QR algorithm in (16)–(25) resulted from a set of rotations which are structurally very similar to the rotations used in the direct QR method. Consequently, the rotations in the inverse QR method may also be performed by a systolic array. This section describes the operation of some of the individual components of the inverse QR systolic array. The following sections will then demonstrate how these components may be connected together to form a complete inverse QR array.

3.1. *Inner product cell*

The first cell needed will be used in computing an inner product [12]. This cell has three inputs as shown in Fig. 1 and outputs a quantity p_{i+1} given by

$$p_{i+1} = p_i + mn. \tag{26}$$

This processor will be used for computing the intermediate vector $\mathbf{a}(n)$ in (18) and the prediction of $d(n)$ given by $\mathbf{w}^T(n-1)\mathbf{x}(n)$ in (25).

3.2. *Rotation cell*

Another set of calculations are the rotations which update elements of $\mathbf{R}^{-T}(n-1)$ in (23) and compute the updated $b^{(i)}(n)$ in (20) and the components of the vector $\mathbf{u}(n)$ in (24). This set of rotations will be done in a triangular rotation array in which there are two types of cells. Figure 2 displays a type I rotation cell which will occupy the j = 1st column elements in the rotation array. This cell computes the trigonometric quantities $c_i(n)$ and $s_i(n)$, and then performs the required rotations. Since all cells in the ith row of the rotation array will use the same $c_i(n)$ and $s_i(n)$ these angle parameters are passed to the right. The updated $b^{(i+1)}(n)$ and $u_1^{(i+1)}(n)$ are passed downward to the $(i+1)$th row. The final operation of the type I cell updates the array

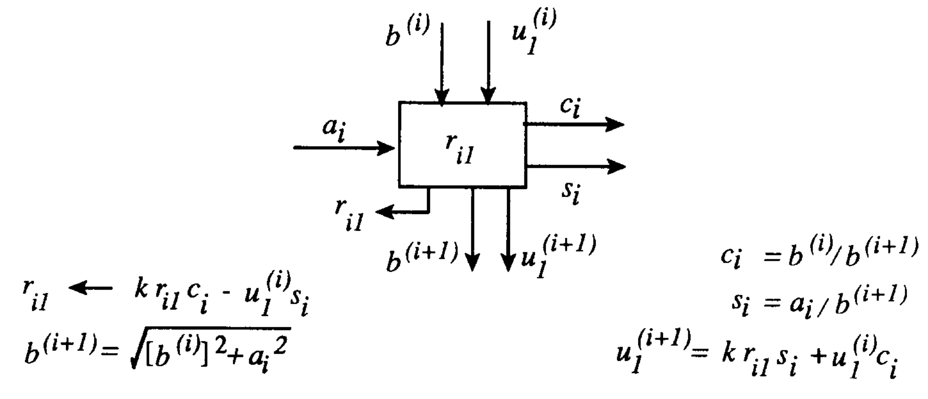
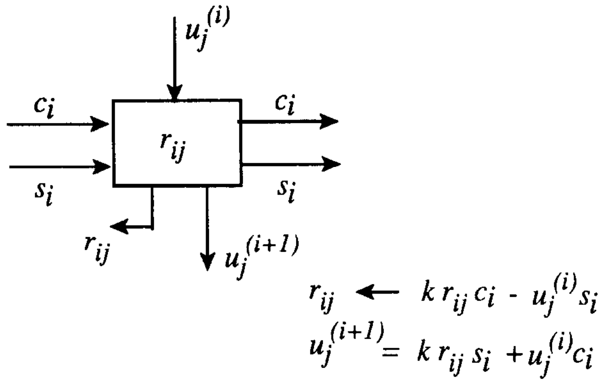

$b^{(i)}$
$u_1^{(i)}$
a_i
r_{i1}
c_i
s_i
r_{i1}
$b^{(i+1)}$
$u_1^{(i+1)}$
$r_{i1} \leftarrow k r_{i1} c_i - u_1^{(i)} s_i$
$b^{(i+1)} = \sqrt{[b^{(i)}]^2 + a_i^2}$
$c_i = b^{(i)} / b^{(i+1)}$
$s_i = a_i / b^{(i+1)}$
$u_1^{(i+1)} = k r_{i1} s_i + u_1^{(i)} c_i$

Fig. 2. Type I rotation cell.



$u_j^{(i)}$
c_i
s_i
r_{ij}
c_i
s_i
r_{ij}
$u_j^{(i+1)}$
$r_{ij} \leftarrow k r_{ij} c_i - u_j^{(i)} s_i$
$u_j^{(i+1)} = k r_{ij} s_i + u_j^{(i)} c_i$

Fig. 3. Type II rotation cell.

element $r_{i1}(n)$ as shown in Fig. 2. The constant k is just the reciprocal of the square root of the forgetting factor and is given by $k = 1/\sqrt{\lambda}$.

Figure 3 next shows a type II rotation cell which occupies the elements in the $j = 2, \ldots, N$ columns of the rotation array. This cell simply receives $c_i(n)$ and $s_i(n)$ as part of its input and then performs the required rotations. Updated parameters are computed and passed as in the type I cell.

3.3. *Weight update cell*

The final set of components are the weight update cells required for computation of the updated filter vector in (25). The type I weight update cell (shown in Fig. 4) is the computational cell for weight w_1. It receives the two inputs d and y from the left, calculates their difference $e = d - y$, and passes e to the right. In the overall inverse QR array the d input will be $d(n)$, the desired signal value at time n, and y will be the output of the transversal filter given by the inner product computation $\mathbf{w}^T(n-1)\mathbf{x}(n)$. The second operation of the type I cell uses e and the inputs $b^{(N)}$ and $u_j^{(N)}$ to update weight 1, as needed by (25) and shown by the computation in Fig. 4. Since all subsequent weights w_j, $2 \leqslant j \leqslant N$, are updated using the same error e, the remaining weight update cells (type II shown in Fig. 5) are simpler. As shown in Fig. 5, only the jth weight update recursion need be computed within the type II cell and the inputs on the left are simply passed to the next type II weight update cell on the right. In this way the weights are updated in a time sequential fashion, with $w_1(n)$ being updated first, followed by $w_2(n)$, and so on. The next section will show that the timing necessary to achieve this sequential updating is provided naturally by the rotation cells in the rotation array.

4. UPDATING THE ROTATION ARRAY

This section will demonstrate how the rotation cells described in the previous section may be connected to update the inverse Cholesky factor from $\mathbf{R}^{-T}(n-1)$ to $\mathbf{R}^{-T}(n)$. One assumption is that the elements of the $\mathbf{a}(n)$ vector enter from the left in the time-skewed schedule as shown in

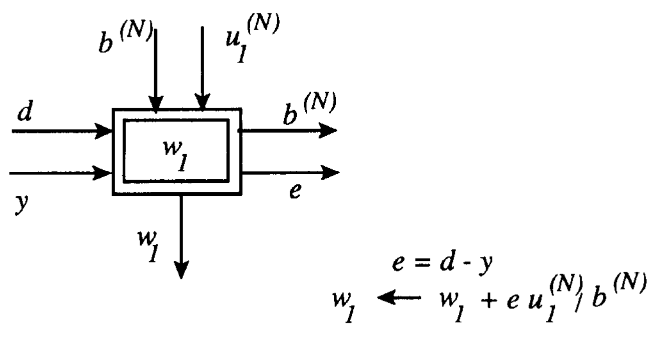


Fig. 4. Type I weight update cell.

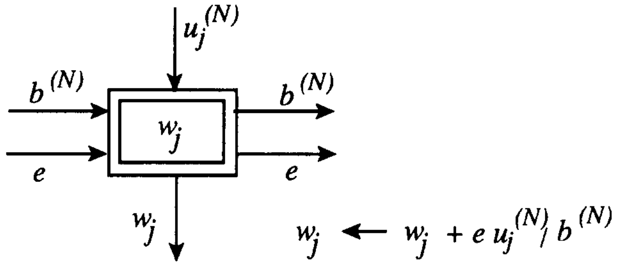


Fig. 5. Type II weight update cell.

Fig. 6. The heavy black dots represents unit clock cycle delays. Section 5 will subsequently show how the natural operation of the entire inverse QR array provides this necessary scheduling.

For example purposes, Fig. 6 shows the rotation array for the case $N = 3$. At the beginning of the nth clock cycle, the inverse Cholesky factor element $r_{ij}(n-1)$ has been previously stored in the (i, j)th element of the array. To begin the update of the elements in this array, the ith element of the j = 1st column of the array (which are all type I rotation cells) receives $a_i(n)$ and $b^{(i-1)}(n)$ as part of its input. The cell then calculates $c_i(n)$ and $s_i(n)$ according to (21) and (22) and propagates these to the next cell on the right for use in the next clock cycle. The type I rotation cell also calculates $b^{(i)}(n)$ and $u_1^{(i)}(n)$ according to (20) and (24), and then propagates these downward to the $(i+1)$th row. The last operation in the type I rotation cell then uses (23) to update the contents of the array element to $r_{i1}(n)$.

The other cells not in the first column of the rotation array are type II rotation cells. The angle parameters $c_i(n)$ and $s_i(n)$ move from left to right in the array and the $b^{(i)}(n)$ and $u_j^{(i)}(n)$ parameters from top to bottom, with the result that row $(i+1)$ is updated one cycle after row i. Each of these cells not in column 1 implements equations (20), (24) and (23). For the $N = 3$ case, the outputs of the third row in the rotation array will go directly to the weight update cells. This last row produces $u_1^{(3)}(n)$, $u_2^{(3)}(n)$ and $u_3^{(3)}(n)$ on successive clock cycles, such that the weight update cells will receive the update information in the correct schedule.

5. COMPLETE INVERSE QR ARRAY

At this point, the various components and arrays previously described may be interconnected to implement the complete inverse QR array. Figure 7 shows an example for the case $N = 3$.

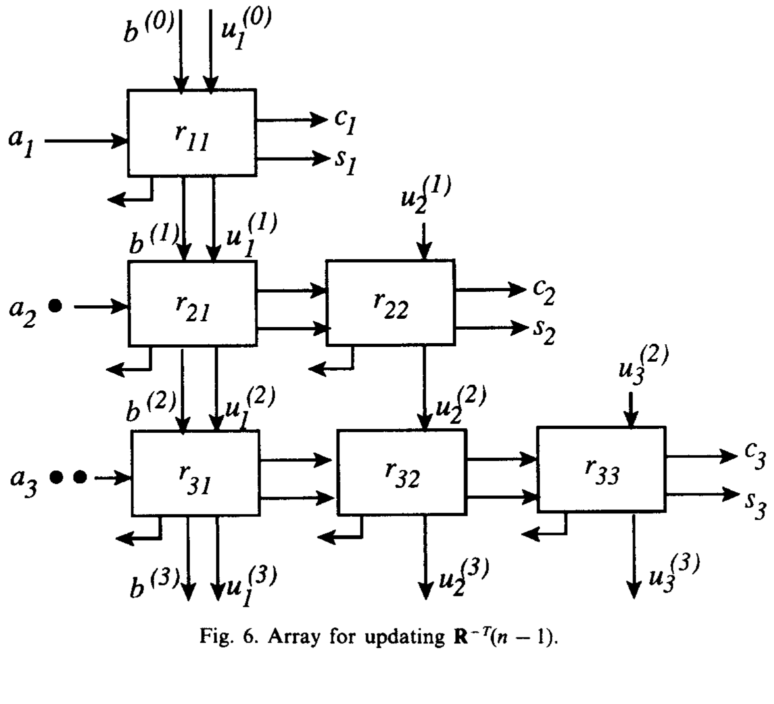


Fig. 6. Array for updating $\mathbf{R}^{-T}(n-1)$.

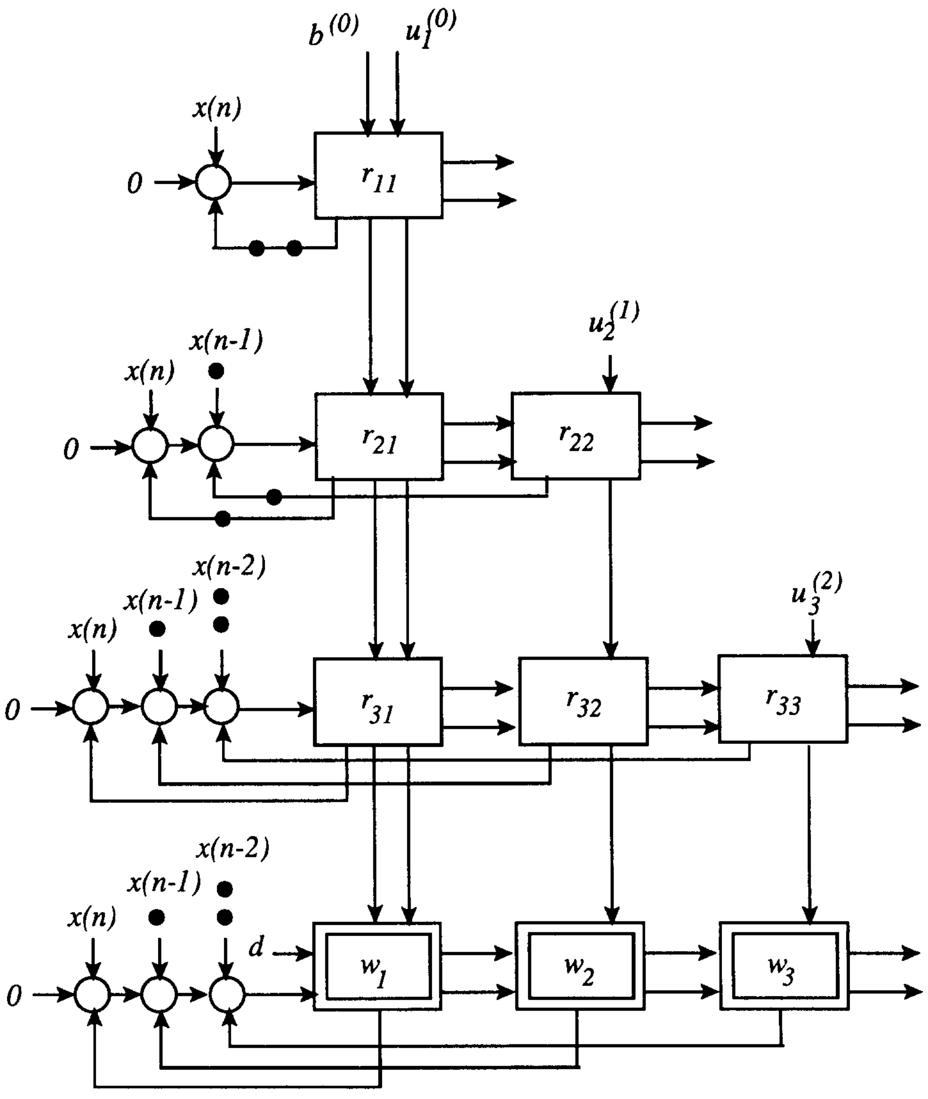


Fig. 7. Complete inverse QR array.

The operations of the array for updating $R^{-T}(u-1)$ have been described in Section 4. The new additions are the inner product arrays at the right for updating $\mathbf{a}(n)$ and at the bottom for updating $\mathbf{w}(n)$. These arrays are described in this section.

5.1. Updating a(b)

The data movement in the $\mathbf{a}(n)$ inner product array is from left to right, with the left input to the leftmost cells being zero for correct operation. Expansion of (18) shows that the ith row in the array will produce $a_i(n)$ by using i such inner product cells. It is also evident that N such rows of inner product cells are necessary to compute the N components of $\mathbf{a}(n)$. The heavy dots in the $\mathbf{a}(n)$ array are unit clock delays which have the effect of correctly aligning the r_{ij} which have been transmitted from the rotation array. This time alignment ensures that all elements within each column of $\mathbf{R}^{-T}(n-1)$ are available simultaneously in the inner product array. This allows computation of all $a_i(n)$ to begin on the same clock cycle. Similarly, the heavy dots on the $x(n-1)$ and $x(n-2)$ paths represent delays to align the input data for correct computation of the inner products. It is straightforward then to see that the $a_i(n)$ are produced on successive clock cycles. Moreover, this operation demonstrates how the update of $r_{31}(n)$ follows the update of $r_{21}(n)$ by one clock cycle and follows the update of $r_{11}(n)$ by two clock cycles. Therefore, aligning the simultaneous appearance of the first column of R^{-T} requires two delays in the transmission path for r_{11} in the first row and the single delays in the transmission paths for r_{21} and r_{22} in the second row.

5.2. Updating w(n)

Updating $\mathbf{w}(n)$ via (25) requires the desired signal estimate given by the inner product $\mathbf{w}^T(n-1)\mathbf{x}(n)$. The inner product array which computes this estimate is shown in the lower left of Fig. 7. The input data samples $x(n)$, $x(n-1)$, and $x(n-2)$ are time-skewed such that $\mathbf{w}^T(n-1)\mathbf{x}(n)$ is available at the appropriate time on the y input of the type I weight update cell which will update w_1. The arrival of this estimate on the y input must coincide with the arrival of the other data on the $b^{(3)}$ and $u_1^{(3)}$ inputs. The error e and $b^{(3)}$ outputs of the weight w_1 cell will then propagate to the right along the w update row and thus succeeding w_j will be updated one after the other.

6. DISCUSSION

This section completes the paper by making some observations concerning the latency and throughput of the inverse QR array. The latency of the inverse QR array is the amount of time required from sampling $x(n)$ until the appearance of the completely updated filter $\mathbf{w}(n)$. A direct examination of Fig. 7 shows that a total of seven clock cycles elapse before $w_3(n)$ is updated. For the general N coefficient case the latency is therefore seen to be $2N+1$ clock cycles.

The throughput of the proposed inverse QR array is ultimately limited by the update of $\mathbf{a}(n)$ because it has been assumed that computation of all $a_i(n)$ will begin on the same clock cycle. Therefore, the calculation of $a_1(n+1)$ [which is the product $r_{11}(n)x(n+1)$] can not begin until four clock cycles after $a_1(n)$ has been calculated. As a result, a new $\{x(n)\}$ data sample can be clocked into the array only every four clock cycles. In the general N coefficient case, a new data sample could be clocked in every $N+1$ clock cycles. Observe that acquiring a new data sample and beginning a cycle of new rotations at the top left of the array in Fig. 7 will not disturb the update of the $\mathbf{w}(n)$ filter. Hence, these operations of rotation and previous filter update may proceed at the same time. The result is that the inverse QR array produces N new filter coefficients every $N+1$ clock cycles. The direct QR approach requires that inputs cease while backsubstitution computes the filter weights [2]. Therefore, the inverse QR array is much more efficient than the direct QR method for recursively updating the least squares filter.

7. SUMMARY

This paper has presented a new array for parallel computation of the inverse QR algorithm. It has been seen that rotation methods may be used to update the recursive least squares filter without employing the highly serial backsubstitution step required by the direct QR method. Consequently, the inverse QR approach offers potential improvement over the direct method for many applications.

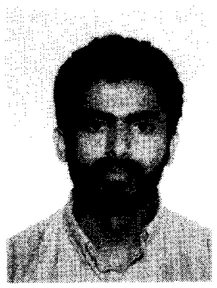
Acknowledgements—The research of authors Ghirnikar and Alexander was funded in part by the National Science Foundation, Grant No. MIP-8552571. The research of author Plemmons was funded in part by the U.S. Air Force, Grant No. AFOSR-88-0255.

REFERENCES

1. S. T. Alexander, *Adaptive Signal Processing*. Springer, Berlin (1986).
2. S. Haykin, *Adaptive Filter Theory*. Prentice-Hall, Englewood Cliffs, N.J. (1991).
3. J. M. Coiffi, Limited precision effects for adaptive filtering. *IEEE Trans. Circuits Syst.* **34,** 821–833 (1987).
4. G. H. Golub and C. Van Loan, *Matrix computations*. Johns Hopkins Press, Baltimore, Md (1983).
5. H. Leung and S. Haykin, Stability of recursive QRD-LS algorithms using finite precision systolic array implementation. *IEEE Trans. Acoust. Speech Signal Process.* **37,** 760–763 (1989).
6. J. G. McWhirter, Recursive least squares minimization using a systolic array. *Proc. SPIE, Real Time Signal Processing VI* **431,** 105–112 (1983).
7. C. T. Pan and R. J. Plemmons, Least squares modifications with inverse factorizations: parallel implications. *J. Comput. appl. Math.* **27,** 109–127 (1989).
8. A. L. Ghirnikar and S. T. Alexander, Stable recursive least squares filtering using an inverse QR decomposition. *Proc. IEEE Int. Conf. Acoust. Speech Signal Process*, Albuquerque, N.M., pp. 1623–1626 (1990).
9. M. Moonen and J. Vandewalle, A systolic array for recursive least squares computations. *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Toronto, pp. 1013–1016 (1991).

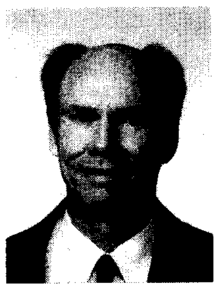
10. P. E. Gill, G. H. Golub, W. Murray and M. A. Saunders, Methods for modifying matrix factorizations. *Maths Comput.* **28**, 505–535 (1974).
11. S. T. Alexander and A. L. Ghirnikar, A method for recursive least squares filtering based upon an inverse QR decomposition. *IEEE Trans. Signal Processing.* To be published.
12. C. Mead and L. Conway, *Introduction to VLSI Systems*. Addison-Wesley, Reading, Mass. (1980).

AUTHORS' BIOGRAPHIES



Avinash Ghirnikar—Avinash Ghirnikar was born in Durgapur, India. He received his B.S. Degree in electrical engineering from the Indian Institute of Technology, Bombay in 1985. From August 1985 to October 1990 he was at North Carolina State University where he obtained his M.S. and Ph.D. degrees in electrical and computer engineering. Since November 1990, he has been with ViTel Communications Corporation in Santa Clara, California. His research interests include adaptive signal processing algorithms, their properties in finite precision, and their real-time implementations for communications applications.

S. T. Alexander—S. T. Alexander was born in Jackson, Tennessee and received a B.S. from Tennessee Tech University in 1971, an M.S. from The University of Tennessee in 1975, and the Ph.D. from North Carolina State University in electrical engineering in 1982. He is currently an Associate Professor of electrical and computer engineering at North Carolina State University in Raleigh, North Carolina, U.S.A. His main areas of research are adaptive algorithms, matrix computations, and systolic arrays. He is the author of the text *Adaptive Signal Processing* (Springer, 1986) and was an Associate Editor for the *IEEE Transactions on Acoustics, Speech, and Signal Processing* from 1986 to 1988. He was also Editor of the *IEEE ASSP Magazine* from 1988 to 1990.



Robert J. Plemmons—R. J. Plemmons received his B.S. degree in mathematics from Wake Forest University in 1962 and his Ph.D. in applied mathematics from Auburn University in 1966. He was professor of computer science and mathematics at North Carolina State University, Raleigh, North Carolina from 1981 to 1990. He also served as Director of the University of North Carolina system wide Center for Research in Scientific Computation until 1990. He works in numerical linear algebra and its applications, and in parallel processing. His research interests thus include a spectrum of topics ranging from conditioning and stability considerations to the design, implementation, and testing of parallel algorithms on currently available high performance architectures, under support from the Air Force Office of Scientific Research and the National Science Foundation. He is the author of over 100 research papers and four books, and is on the editorial boards of four journals, including service as the Associate Managing Editor of the *SIAM Journal on Matrix Analysis*. His current position is Z. Smith Professor of Mathematics and Computer Science at Wake Forest University in Winston-Salem, North Carolina.