ELSEVIER

Signal Processing 51 (1996) 167–190

SIGNAL PROCESSING

Architectures for block Toeplitz systems

Ilias Bouras[a], George-Othon Glentis[b], Nicholas Kalouptsidis[c,*]

[a]*Institute of Microelectronics, DEMOKRITOS, Athens 153 10, Greece*
[b]*Department of Electrical Engineering, University of Twente, EL-BSC, P.O. Box 217, 7500 AE Enschede, The Netherlands*
[c]*Department of Informatics, University of Athens, Panepistimiopolis, TYPA Buildings, 157 71 Athens, Greece*

Received 8 April 1994; revised 22 June 1995 and 14 December 1995

Abstract

In this paper efficient VLSI architectures of highly concurrent algorithms for the solution of block linear systems with Toeplitz or near-to-Toeplitz entries are presented. The main features of the proposed scheme are the use of scalar only operations, multiplications/divisions and additions, and the local communication which enables the development of wavefront array architecture. Both the mean squared error and the total squared error formulations are described and a variety of implementations are given.

Zusammenfassung

In dieser Arbeit werden effiziente VLSI Architekturen für hochgradig parallele Algorithmen zur Lösung von blocklinearen Systemen vorgestellt, die eine Töplitzstruktur oder Fast-Töplitz-struktur aufweisen. Die Hauptmerkmale des vorgeschlagenen Schemas sind die Verwendung ausschließlich skalarer Operationen, und zwar sowohl hinsichtlich der Multiplikationen/Divisionen als auch der Additionen, sowie eine lokale Kommunikationsstruktur, die die Entwicklung einer Wellenfrontfeld-Architektur ermöglicht. Es wird sowohl ein Ansatz für den mittleren quadratischen Fehler als auch ein Ansatz für den gesamten quadratischen Fehler formuliert, und eine Vielzahl von Implementierungen wird angegeben.

Résumé

Dans cet article sont présentées des architectures VLSI efficaces d'algorithmes hautement concurrents pour la résolution de systèmes linéaires par blocs avec des entrées Toepliz ou presque Toepliz. Les caractéristiques principales des méthodes proposées sont l'utilisation d'opérations scalaires (multiplications/divisions et additions) uniquement, ainsi que la communication locale, qui permet le développement d'une architecture en réseau de front d'onde. Les formulations des erreurs quadratiques moyenne et totale sont décrites, et plusieurs implémentations distinctes sont données.

Keywords: Block Toeplitz matrices; Multichannel Schur algorithms; Parallel processing; VLSI implementation

*Corresponding author. Fax: 301 72 28 981.

0165-1684/96/$15.00 © 1996 Elsevier Science B.V. All rights reserved
PII S0165-1684(96)00041-2

1. Introduction

In this paper efficient VLSI architectures for block Toeplitz and Toeplitz like linear systems are developed. Block Toeplitz solvers are encountered in a wide range of applications. Typical examples include multichannel spectral estimation, multivariable system identification and realization, multichannel one dimensional (1-D) and two dimensional (2-D) Wiener filtering and smoothing, nonlinear filtering using truncated Volterra series [5, 10, 13, 15, 18, 19].

High performance massively parallel architectures are required in most real time digital signal processing applications. Advanced algorithms used for adaptive antennas and adaptive beam-forming have a data throughput of hundreds of millions of operations per second. The requirements of real time image processing are even more demanding, since several billions of operations per second are needed. VLSI array processors provide a low cost solution to high speed, heavily loaded real time applications [11].

Real time operation of block Toeplitz solvers is an extremely heavy task. Since block Toeplitz systems result from multichannel and multidimensional applications the computational load is expected to be tremendous even for moderate cases. Computation savings can be obtained by applying the efficient Levinson–Wiggins–Robinson (LWR) type algorithms that take into account the special structure of block Toeplitz systems [8, 17, 22, 23]. The computational complexity of these techniques is $O(k^3p^2)$, where k is the number of channels (or the block order of the block system) and p is the number of taps associated with each input channel (or the dimensions of each entry Toeplitz matrix).

A major drawback of the LWR type algorithms is the need of inner products computations, thus prohibiting full parallelism. A remedy to this bottleneck is the engagement of Schur type algorithms, which bypass the need for inner product estimation, thus being suitable for parallel implementation on a general purpose parallel machine or on dedicated VLSI hardware, using a systolic or a wavefront array architecture [1–4, 6, 7, 9, 12].

In this paper several VLSI architectures are proposed for the Schur type block Toeplitz solvers recently developed in [1, 3, 4, 9]. Different throughput rates can be achieved depending on the available hardware. A planar array processor is first designed that computes the desired solution in $O(kp)$ time units utilizing $O(k^2p)$ processing elements. A reduced processor architecture is then derived that computes the output in $O(k^2p)$ time units, requiring however $O(kp)$ processing elements only. Finally, a superfast highly pipelineable three dimensional structure is developed which in a full pipeline mode has a constant throughput rate.

These algorithms offer significant advantages over their predecessors, since inner products are now bypassed, and they are free of matrix operations. On the contrary, the parallel structures presented in [6, 7, 21] require additional parts that implement matrix inversion and multiplication, as well as inner products, of order k. Moreover, they suffer from communication, since matrices of order k must be transmitted.

The proposed implementation of the highly concurrent Schur type algorithms involves the following steps. (i) Derivation of dependence graph showing the flow of computations. (ii) Employment of the canonical mapping methodology and the default schedule, to derive a signal flow graph. (iii) Algorithm reorganization to ensure local flow of data and event-driven local control. (iv) Minimization of the number of the over-all processing elements as well as computational and communication tasks. (v) Programming in Occam and algorithm validation.

The paper is organized as follows. First, the block Toeplitz and near-to-Toeplitz solvers of [1–4] are briefly discussed in Section 3. New systolic and wavefront architectures are proposed in Section 4, and some hardware implementation issues are presented to give directions to hardware designers. The estimation of the correlation lags used for the initialization of the algorithms is also discussed. An alternative, more efficient from the hardware point of view, architecture is described in Section 5 and a description of the main processor unit is given. Finally, in Section 6 a highly pipelined 3-D wavefront array, suitable for adaptive processing, is proposed.

2. Problem formulation

Let us consider a multichannel FIR (finite impulse response) filter of k input channels and a single output. The extension to the multi-output case is straightforward. Let k be the number of filter input channels; $x_i(n)$, $i = 1, \ldots, k$, be the corresponding input. Then, the filter output $y(n)$ is estimated by the discrete time equation

$$y(n) = -\sum_{i=1}^{k}\sum_{l=1}^{p_i} c_i(l)x_i(n-l+1), \tag{1}$$

where $c_i(l)$, $l = 1, \ldots, p_i$, are the filter coefficients assigned to the ith channel. The number of filter coefficients p_i, $i = 1, \ldots, k$, is in general different for each input channel, i.e., $p_i \neq p_j$, $i, j = 1, \ldots, k$. The multichannel filter is completely characterized by the parameter set $(\boldsymbol{p}_k, \boldsymbol{c}_{p_k})$, where

$$\boldsymbol{p}_k = [p_1, p_2, \ldots, p_k]$$

is a multi-index that specifies the lengths of the input registers [1–4] and $\boldsymbol{c}_{p_k}$ is a block vector that carries the filter coefficients

$$\boldsymbol{c}_{p_k} = [\boldsymbol{c}_{p_1}^{1\mathrm{T}}\ \boldsymbol{c}_{p_2}^{2\mathrm{T}}\ \ldots\ \boldsymbol{c}_{p_k}^{k\mathrm{T}}]^{\mathrm{T}},$$

where $\boldsymbol{c}_{p_i}^{i\mathrm{T}} = [c_i(1)\ c_i(2)\ \ldots\ c_i(p_i)]$, $i = 1, 2, \ldots, k$. Superscript T means transpose. Eq. (1) can be compactly written as a linear regression

$$y(n) = -\boldsymbol{x}_{p_k}^{\mathrm{T}}(n)\boldsymbol{c}_{p_k}. \tag{2}$$

The regressor vector $\boldsymbol{x}_{p_k}(n)$ consists of blocks, each one being formed by successive samples and of varying dimension

$$\boldsymbol{x}_{p_k}(n) = [\boldsymbol{x}_{p_1}^{1\mathrm{T}}(n)\ \boldsymbol{x}_{p_2}^{2\mathrm{T}}(n)\ \ldots\ \boldsymbol{x}_{p_k}^{k\mathrm{T}}(n)]^{\mathrm{T}},$$

where $\boldsymbol{x}_{p_i}^{i\mathrm{T}}(n) = [x_i(n)\ x_i(n-1)\ \ldots\ x_i(n-p_i+1)]$, $i = 1, 2, \ldots, k$.

Let

$$e_{p_k}^c(n) = z(n) - y(n) = z(n) + \boldsymbol{x}_{p_k}^{\mathrm{T}}(n)\boldsymbol{c}_{p_k}$$

be the instantaneous output error between the desired response signal $z(n)$ and the filter's output $y(n)$. The error signal $e_{p_k}^c(n)$ is forced to be orthogonal to the regressor vector $\boldsymbol{x}_{p_k}(n)$. This can be expressed as

$$\langle \boldsymbol{x}_{p_k}(n), e_{p_k}^c(n)\rangle = \boldsymbol{0}, \tag{3}$$

where $\langle\cdot\rangle$ is a suitable orthogonalization operator. The choice of $\langle\cdot\rangle$ gives rise to several algorithms for the determination of the optimal filter $\boldsymbol{c}_{p_k}$. Two particular cases are considered in this paper.

(a) *The mean square error (MSE).* Let $\mathscr{E}(.)$ denote the expectation operator. Then, Eq. (3) takes the form

$$\mathscr{E}(\boldsymbol{x}_{p_k}(n)e_{p_k}^c(n)) = \boldsymbol{0}. \tag{4}$$

The mean squared error filter is then estimated by the solution of the linear system of equations

$$\boldsymbol{R}_{p_k}\boldsymbol{c}_{p_k} = -\boldsymbol{d}_{p_k}, \tag{5}$$

where

$$\boldsymbol{R}_{p_k} = \mathscr{E}(\boldsymbol{x}_{p_k}(n)\boldsymbol{x}_{m_k}^{\mathrm{T}}(n)), \quad \boldsymbol{d}_{p_k} = \mathscr{E}(\boldsymbol{x}_{p_k}(n)z(n)) \tag{6}$$

are the autocorrelation matrix and the crosscorrelation vector, respectively.

$\boldsymbol{R}_{p_k}$ is a symmetric positive definite block matrix of block order k with Toeplitz entry matrices. This means that $\boldsymbol{R}_{p_k}$ consists of k^2 Toeplitz submatrices $\boldsymbol{R}_{ij}$ of dimensions $p_i \times p_j$, $i, j = 1, \ldots, k$. Thus,

$$\boldsymbol{R}_{p_k} = [\boldsymbol{R}_{ij}]_{\substack{i=1,\ldots,k\\ j=1,\ldots,k}} = \begin{bmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} & \cdots & \boldsymbol{R}_{1k} \\ \boldsymbol{R}_{12}^{\mathrm{T}} & \boldsymbol{R}_{22} & \cdots & \boldsymbol{R}_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{R}_{1k}^{\mathrm{T}} & \boldsymbol{R}_{2k}^{\mathrm{T}} & \cdots & \boldsymbol{R}_{kk} \end{bmatrix}, \tag{7}$$

where each entry $\boldsymbol{R}_{ij}$ is a Toeplitz matrix

$$\boldsymbol{R}_{ij} = \mathscr{E}(\boldsymbol{x}_{p_i}^{i}(n)\boldsymbol{x}_{p_j}^{j\mathrm{T}}(n)), \quad i = 1, \ldots, k,\ j = 1, \ldots, k. \tag{8}$$

In a similar way $\boldsymbol{d}_{p_k}$ is a block vector that contains k subvectors of dimensions $p_i \times 1$, each.

$$\boldsymbol{d}_{ij} = \mathscr{E}(\boldsymbol{x}_{p_i}^{i}(n)z_j(n)), \quad i = 1, \ldots, k,\ j = 1, \ldots, q. \tag{9}$$

(b) *The total squared error (TSE).* Operator $\langle\cdot\rangle$ now takes the form $\sum_{n=0}^{N}(\cdot)$. The orthogonality condition (3) is expressed by

$$\sum_{n=0}^{N}(\boldsymbol{x}_{p_k}(n)e_{p_k}^c(n)) = \boldsymbol{0}. \tag{10}$$

The corresponding least squared filter is determined by the linear system

$$\boldsymbol{R}_{p_k}(N)\boldsymbol{c}_{p_k}(N) = -\boldsymbol{d}_{p_k}(N), \tag{11}$$

where

$$\boldsymbol{R}_{p_k}(N) = \sum_{n=0}^{N} \boldsymbol{x}_{p_k}(n)\boldsymbol{x}_{m_k}^{\mathrm{T}}(n), \quad \boldsymbol{d}_{p_k}(N) = \sum_{n=0}^{N} \boldsymbol{x}_{p_k}(n)z(n) \tag{12}$$

are the sampled autocorrelation matrix and the crosscorrelation vector, respectively.

Block matrix $\boldsymbol{R}_{p_k}(N)$ consists of k^2 near-to-Toeplitz entry matrices, $\boldsymbol{R}_{ij}(N)$, of dimensions $p_i \times p_j$, given by

$$\boldsymbol{R}_{ij}(N) = \sum_{n=0}^{N} \boldsymbol{x}_{p_i}^{i}(n)\boldsymbol{x}_{p_j}^{j\mathrm{T}}(n), \quad i,j = 1, \ldots, k.$$

Similarly, block vector $\boldsymbol{d}_{p_k}(N)$ consists of k subvectors

$$\boldsymbol{d}_i(N) = \sum_{n=0}^{N} \boldsymbol{x}_{p_i}^{i}(n)z(n)(n), \quad i = 1, \ldots, k.$$

3. Algorithm description

Linear systems (5) and (11) can efficiently be solved by the algorithms recently proposed in [1–4], using a channel decomposition technique. The main advantage of this novel approach is the need of scalar operations only. This is an attractive feature when VLSI implementation is under consideration.

Eq. (5) determines the optimal FIR filter in the MSE sense. The pertinent filter is obtained as the solution of a block linear system of equations with Toeplitz matrices as entries. While a standard linear system solver can be utilized, the structure of matrix $\boldsymbol{R}_{p_k}$ enables the development of efficient algorithms for the computation of the optimal filter $\boldsymbol{c}_{p_k}$. The derivation of such fast algorithms is based on the nesting properties of matrix $\boldsymbol{R}_{p_k}$ that permit the order recursive estimation of the optimal filter, starting from $\boldsymbol{c}_1$ and going up to the final filter $\boldsymbol{c}_{p_k}$. A novel algorithmic family that recursively solves the block Toeplitz linear system of Eq. (5) has been derived in [1–4]. Several fast order recursive algorithms have been presented, possessing the following desired properties:

- Multichannel FIR filters with different input orders, $p_i \neq p_j$ can be easily handled.
- The new algorithms offer a significant advantage over their well-known predecessors [6–8, 12, 17, 22, 23] since they manage to get free of matrix operations altogether.
- Very highly concurrent structures can be extracted using a Schur type format, involving scalar operations only.

A set of k single channel forward and backward predictors is utilized by both the MSE and the TSE algorithm. Single channel predictors are assigned to each input channel. The forward v-channel predictors $\boldsymbol{a}_{p_k}^{v}$, $v = 1, \ldots, k$, are determined by the orthogonality condition

$$\langle \boldsymbol{x}_{p_k}^{v}(n), e_{p_k}^{fv}(n) \rangle = \boldsymbol{0},$$

where

$$e_{p_k}^{fv}(n) = x_v(n) + \boldsymbol{x}_{p_k}^{v\mathrm{T}}(n)\boldsymbol{a}_{p_k}^{v}$$

is the forward prediction error and $\boldsymbol{x}_{p_k}^{v}(n)$ is an alternative regressor vector defined by

$$\boldsymbol{x}_{p_k}^{v}(n) = \begin{bmatrix} [\boldsymbol{x}_{p_i}(n-1)]_{i=1,\ldots,v} \\ [\boldsymbol{x}_{p_i}(n)]_{i=v+1,\ldots,k} \end{bmatrix}. \tag{13}$$

The v first vector components of $\boldsymbol{x}_{m_k}^{v}(n)$ contain the elements of the channels delayed by one unit. Clearly, $\boldsymbol{x}_{p_k}^{0}(n) = \boldsymbol{x}_{p_k}(n)$, $\boldsymbol{x}_{p_k}^{k}(n) = \boldsymbol{x}_{p_k}(n-1)$.

The backward single channel predictors are obtained from the orthogonality condition

$$\langle \boldsymbol{x}_{p_k}^{\mu}(n), e_{p_k}^{b(\mu+1)}(n) \rangle = \boldsymbol{0},$$

where

$$e_{p_k}^{b(\mu+1)}(n) = x_{\mu+1}(n - p_{\mu+1}) + \boldsymbol{x}_{p_k}^{(\mu)\mathrm{T}}(n)\boldsymbol{b}_{p_k}^{\mu+1},$$

$$\mu = 0, 1, \ldots, k-1.$$

In this paper we will concentrate on the Schur type fast order recursive algorithms developed in [1–4] for the solution of linear systems (5) and (11). For presentation reasons filters with equal number of delay elements for all input channels are considered, i.e., $p_i = p_j = p$, $i,j = 1, \ldots, k$. The MSE multichannel Schur algorithm is summarized in Table 1.

The solution vector $\boldsymbol{c}$ is recursively computed by the algorithm, using Eq. (3) of Table 1. Two sets of auxiliary vectors are utilized, $\boldsymbol{a}^{v}$, $v = 1, 2, \ldots, k$, and $\boldsymbol{b}^{\mu}$, $\mu = 1, 2, \ldots, k$, corresponding to single channel forward and backward predictors, respectively. The

Table 1
The highly parallel algorithm for block Toeplitz systems (MSE case)

FOR $j = 0$ TO $p-1$
FOR $i = 0$ TO $k-1$

$$k^{c}_{m_k+i+1} = -e^{ci+1}_{m_k+i}(j)/e^{b(i+1,i+1)}_{m_k+i}(j) \quad (1)$$

IN PARALLEL DO 1
  FOR $n = j$ TO $p-1$ IN PARALLEL DO 2

$$e^{ch}_{m_k+i+1}(n) = e^{ch}_{m_k+i}(n) + e^{b(i+1,h)}_{m_k+i+1}(n)k_{m_k+i+1} \quad (2)$$

  END PAR 2

$$S^{i+1}_{m_k+i+1}c_{m_k+i+1} = \begin{pmatrix} c_{m_k+i} \\ 0 \end{pmatrix} + \begin{pmatrix} b^{i+1}_{m_k+i} \\ 1 \end{pmatrix} k_{m_k+i+1} \quad (3)$$

  END PAR 1
$I = \text{left\_rotate}[1\ 2\ \ldots\ k]_{i+1}$
FOR $v = 1$ TO k IN PARALLEL DO 3
$\mu = I(v)$
  IN PARALLEL DO 4

$$k^{fv}_{m_k+i+1} = -e^{f(v,\mu)}_{m_k+i}(j+l_1)/e^{b(\mu,\mu)}_{m_k+i}(j+l_2) \quad (4)$$

$$k^{b\mu}_{m_k+i+1} = -e^{f(\mu,v)}_{m_k+i}(j+l_1)/\alpha^{fv}_{m_k+i+1} \quad (5)$$

END PAR 4
FOR $n = j$ TO $p-1$ IN PARALLEL DO 6

$$e^{f(v,h)}_{m_k+i+1}(n) = e^{f(v,h)}_{m_k+i}(n) + z^q_n e^{b(\mu,h)}_{m_k+i+1}(n)k^{fv}_{m_k+i+1} \quad (6)$$

$$e^{b(\mu,h)}_{m_k+i+1}(n) = z^q_n e^{b(\mu,h)}_{m_k+i}(n) + e^{f(v,h)}_{m_k+i+1}(n)k^{b\mu}_{m_k+i+1} \quad (7)$$

  END PAR 6
  IN PARALLEL DO 7

$$S^{\mu}_{m_k+i+1}a^{v}_{m_k+i+1} = \begin{pmatrix} a^{v}_{m_k+i} \\ 0 \end{pmatrix} + \begin{pmatrix} b^{\mu}_{m_k+i} \\ 1 \end{pmatrix} k^{fv}_{m_k+i+1} \quad (8)$$

$$T^{\vee}_{m_k+i+1}b^{\mu}_{m_k+i+1} = \begin{pmatrix} 0 \\ b^{\mu}_{m_k+i} \end{pmatrix} + \begin{pmatrix} 1 \\ a^{v}_{m_k+i} \end{pmatrix} k^{b\mu}_{m_k+i+1} \quad (9)$$

$$\alpha^{fv}_{m_k+i+1} = \alpha^{fv}_{m_k+i} + e^{f(v,\mu)}_{m_k+i}(j+l_1)k^{f(v)}_{m_k+i+1} \quad (10)$$

  END PAR 7
END PAR 3
END FOR i
END FOR j

algorithm computes the solution in p steps. Each step consists of k phases. During each phase the solution $\boldsymbol{c}$ is augmented by a new element. Thus, after the completion of a single step, $\boldsymbol{c}$ has been increased by k elements. S^{v} and T^{μ} that appear in Tables 1 and 2 are suitable permutation matrices [1–4].

A notable feature of the algorithm is the cyclic way in which the forward predictors, $\boldsymbol{a}^{v}$, and the backward predictors, $\boldsymbol{b}^{\mu}$, are updated. At the first phase ($i = 0$) of a new step, the forward predictors are coupled together with the backward predictors in a particular way which is $[\boldsymbol{a}^1,\boldsymbol{b}^k]$, $[\boldsymbol{a}^2,\boldsymbol{b}^1]$, $[\boldsymbol{a}^3,\boldsymbol{b}^2]$, $[\boldsymbol{a}^k,\boldsymbol{b}^{k-1}]$. At the second phase ($i = 1$), the predictors are coupled together as $[\boldsymbol{a}^1,\boldsymbol{b}^{k-1}]$, $[\boldsymbol{a}^2,\boldsymbol{b}^k]$, $[\boldsymbol{a}^3,\boldsymbol{b}^1]$, $[\boldsymbol{a}^k,\boldsymbol{b}^{k-2}]$. Finally, at the last phase ($i = k-1$), the pairs are $[\boldsymbol{a}^1,\boldsymbol{b}^1]$, $[\boldsymbol{a}^2,\boldsymbol{b}^2]$, $[\boldsymbol{a}^3,\boldsymbol{b}^3]$, $[\boldsymbol{a}^k,\boldsymbol{b}^k]$. The same coupling holds for the forward and backward error variables, $e^{f(v,h)}$ and $e^{b(\mu,h)}$. The cyclic nature of the parameters coupling is nicely described by introducing the index sequence $[1\ 2\ \ldots\ k]$ and the rotation operator

$$I = \text{left\_rotate}[1\ 2\ \ldots\ k]_{i+1},$$

which rotates $[1\ 2\ \ldots\ k]$ $(i+1)$ times to the left. In this way, the backward predictors $\boldsymbol{b}^{\mu}$ associated with the forward predictors $\boldsymbol{a}^{v}$ at phase i are found if we set $\mu = I(v)$, where $I(v)$ means the vth element of the index sequence I. The operator z^q_n that appears in Eqs. (6) and (7) of Table 1 denotes a shift with respect to n. It is activated by the superscript q, $q = 1$ if $v = k$, otherwise $q = 0$. Variables l_1 and l_2 denote a delay with respect to n, and are defined as $l_1 = 1$ if $\mu \leqslant i$, otherwise $l_1 = 0$, and $l_2 = 1$ if $\mu < i$, otherwise $l_2 = 0$. The algorithm is initialized as

$$e^{ch}(n) = d^{h}(n),$$

$$e^{f(v,h)}(n) = r_{v,h}(n), \qquad e^{b(v,h)}(n) = e^{f(v,h)}(n),$$

where

$$d_h(n) = \mathscr{E}[x_h(l-n)z(l)],$$

$$r_{v,h}(n) = \mathscr{E}[x_v(l-n)x_h(l)]$$

are cross-correlation and autocorrelation lags.

The overall complexity of the proposed algorithm is $O(2k^2p^2 + kp^2)$ MADS (multiplications and divisions). On the contrary, Cholesky's decomposition method requires $O(k^3p^3/6)$ MADS. The Wiggins–Levinson–Robinson (WLR) algorithm requires roughly the same amount of operations; however extra linear system solvers of order k are required.

In a parallel processing environment, the proposed algorithm enables full parallelism using

Table 2
The highly parallel algorithm for block near to Toeplitz systems (TSE case)

FOR $j = 0$ TO $p - 1$
FOR $i = 0$ TO $k - 1$
IN PARALLEL DO 1

$$k^{c}_{m_k+i+1}(N) = -e^{ci+1}_{m_k+i}(j\,|\,N)/e^{b(i+1,i+1)}_{m_k+i}(j\,|\,N) \tag{1}$$

$$k^{w}_{m_k+i+1}(N) = -\varepsilon^{i+1}_{m_k+i}(j\,|\,N)/e^{b(i+1,i+1)}_{m_k+i}(j\,|\,N-1) \tag{2}$$

$$\tilde{k}^{b}_{m_k+i}(N) = -\varepsilon^{(i+1)}_{m_k+i}(j\,|\,N)/\alpha_{m_k+i}(N) \tag{3}$$

END PAR 1
IN PARALLEL DO 2
FOR $n = j$ TO $p - 1$ IN PARALLEL DO 3

$$e^{ch}_{m_k+i+1}(n\,|\,N) = e^{ch}_{m_k+i}(n\,|\,N) + e^{b(i+1,h)}_{m_k+i+1}(n\,|\,N)k_{m_k+i+1}(N) \tag{4}$$

$$e^{b(i+1)}_{m_k+i}(n-1\,|\,N-1) = e^{b(i+1)}_{m_k+i}(n-1\,|\,N) + \varepsilon^{(i+1)}_{m_k+i}(n-1\,|\,N)\tilde{k}^{b}_{m_k+i}(N) \tag{5}$$

END PAR 3
FOR $n = j$ TO $p - 1$ IN PARALLEL DO 4

$$\varepsilon^{h}_{m_k+i+1}(n\,|\,N) = \varepsilon^{h}_{m_k+i}(n\,|\,N) + e^{b(i+1,h)}_{m_k+i}(n\,|\,N)k^{w}_{m_k+i+1}(N) \tag{6}$$

END PAR 4

$$S^{i+1}_{m_k+i+1}\boldsymbol{c}_{m_k+i+1}(N) = \begin{pmatrix}\boldsymbol{c}_{m_k+i}(N)\\ 0\end{pmatrix} + \begin{pmatrix}\boldsymbol{b}^{i+1}_{m_k+i}(N)\\ 1\end{pmatrix}k^{c}_{m_k+i+1}(N) \tag{7}$$

$$\boldsymbol{b}^{i+1}_{m_k+i}(N-1) = \boldsymbol{b}^{i+1}_{m_k+i}(N) + \boldsymbol{w}_{m_k+i}(N)\tilde{k}^{b}_{m_k+i}(N) \tag{8}$$

$$S^{i+1}_{m_k+i+1}\boldsymbol{w}_{m_k+i+1}(N) = \begin{pmatrix}\boldsymbol{w}_{m_k+i}(N)\\ 0\end{pmatrix} + \begin{pmatrix}\boldsymbol{b}^{i+1}_{m_k+i}(N-1)\\ 1\end{pmatrix}k^{w}_{m_k+i+1}(N) \tag{9}$$

$$\alpha_{m_k+i+1}(N) = \alpha_{m_k+i}(N) - \varepsilon^{(i+1)}_{m_k+i}(m_{i+1}\,|\,N)k^{w}_{m_k+i+1}(N) \tag{10}$$

END PAR 2
I = left_rotate$[1\ \ 2\ \ \ldots\ \ k]_{i+1}$
FOR $v = 1$ TO k IN PARALLEL DO 3
$\mu = I(v)$
IN PARALLEL DO 4

$$k^{fv}_{m_k+i+1}(N) = -e^{f(v,\mu)}_{m_k+i}(j + l_1\,|\,N)/(z^{q}_{N})e^{b(\mu,\mu)}_{m_k+i}(j + l_2\,|\,N) \tag{11}$$

$$k^{b\mu}_{m_k+i+1}(N) = -e^{f(\mu,v)}_{m_k+i}(j + l_1\,|\,N)/\alpha^{fv}_{m_k+i+1}(N) \tag{12}$$

END PAR 4
FOR $n = j$ TO $p - 1$ IN PARALLEL DO 6

$$e^{f(v,h)}_{m_k+i+1}(n\,|\,N) = e^{f(v,h)}_{m_k+i}(n\,|\,N) + (z_N z_n)^{q}e^{b(\mu,h)}_{m_k+i+1}(n\,|\,N)k^{fv}_{m_k+i+1}(N) \tag{13}$$

$$e^{b(\mu,h)}_{m_k+i+1}(n\,|\,N) = (z_n z_N)^{q}e^{b(\mu,h)}_{m_k+i}(n\,|\,N) + e^{f(v,h)}_{m_k+i+1}(n\,|\,N)k^{b\mu}_{m_k+i+1}(N) \tag{14}$$

END PAR 6

Table 2 (continued)

IN PARALLEL DO 7

$$S^{\mu}_{m_k+i+1}a^{\nu}_{m_k+i+1}(N) = \begin{pmatrix} a^{\nu}_{m_k+i}(N) \\ 0 \end{pmatrix} + \begin{pmatrix} z^{q}_{N} b^{\mu}_{m_k+i}(N) \\ 1 \end{pmatrix} k^{f\nu}_{m_k+i+1}(N) \tag{15}$$

$$T^{\nu}_{m_k+i+1}b^{\mu}_{m_k+i+1}(N) = \begin{pmatrix} 0 \\ z^{q}_{N} b^{\mu}_{m_k+i}(N) \end{pmatrix} + \begin{pmatrix} 1 \\ a^{\nu}_{m_k+i}(N) \end{pmatrix} k^{b\mu}_{m_k+i+1}(N) \tag{16}$$

$$\alpha^{f\nu}_{m_k+i+1}(N) = \alpha^{f\nu}_{m_k+i}(N) + e^{f(\nu,\mu)}_{m_k+i}(j + l_1 \mid N) k^{f(\nu)}_{m_k+i+1}(N) \tag{17}$$

END PAR 7
END PAR 3
END FOR i
END FOR j

scalar operations only. The multichannel implementation of the parallel structure presented in [12, 21] requires additional parts that implement matrix inversion and multiplication, as well as inner products, of order k. Moreover, it suffers from communication, since matrices of order k must be transmitted.

The total least squared error (TSE) FIR filter is determined by Eq. (11). Matrices that compose the block matrix $\boldsymbol{R}_{p_k}(N)$ are no longer Toeplitz. It does however have a Toeplitz like structure which enables the development of highly concurrent algorithms. The TSE Schur algorithm is summarized in Table 2. It has the same structure as the MSE counterpart of Table 1. Some extra variables are now introduced to compensate for the close-to-Toeplitz structure of the data matrix. The rotation operator $\boldsymbol{I}$ is used again in order to couple suitably the various variables. Operator z^{q}_{N} denotes a shift with respect to N. The algorithm is initialized as

$$e^{ch}(n) = d^{h}(n),$$

$$e^{f(v,h)}(n) = r_{v,h}(n), \qquad e^{b(v,h)}(n) = e^{f(v,h)}(n),$$

where

$$d_h(n) = \sum_{l=0}^{N} x_h(l-n)z(l),$$

$$r_{v,h}(n) = \sum_{l=0}^{N} x_v(l-n)x_h(l). \tag{14}$$

4. Systolic/wavefront architectures

In this section several systolic/wavefront implementations of the highly parallel Schur type algorithms of Tables 1 and 2 are derived. The estimation task is naturally decomposed into separate submodules. First, autocorrelation and crosscorrelation variables are estimated, as are required for the initialization of both algorithms. In the MSE case, autocorrelation and crosscorrelation lags are assumed to be known in advance. Otherwise, they are estimated on the basis of the available data record [15]

$$d_h(n) = \frac{1}{N}\sum_{l=0}^{N} x_h(l-n)z(l),$$

$$r_{v,h}(n) = \frac{1}{N}\sum_{l=0}^{N} x_v(l-n)x_h(l).$$

In the TSE case, initialization variables are estimated according to Eq. (14). Both cases can be handled by a similar array processor.

4.1. Systolic estimation of correlation lags

A systolic/wavefront array that estimates autocorrelation and crosscorrelation lags for the initialization phase is first developed. The variables $r_{v,h}(n)$ and $d_h(n)$ of Eq. (14) are fed into the algorithm. These are computed by the systolic array of Fig. 1(a). The arcs of the graph illustrate the

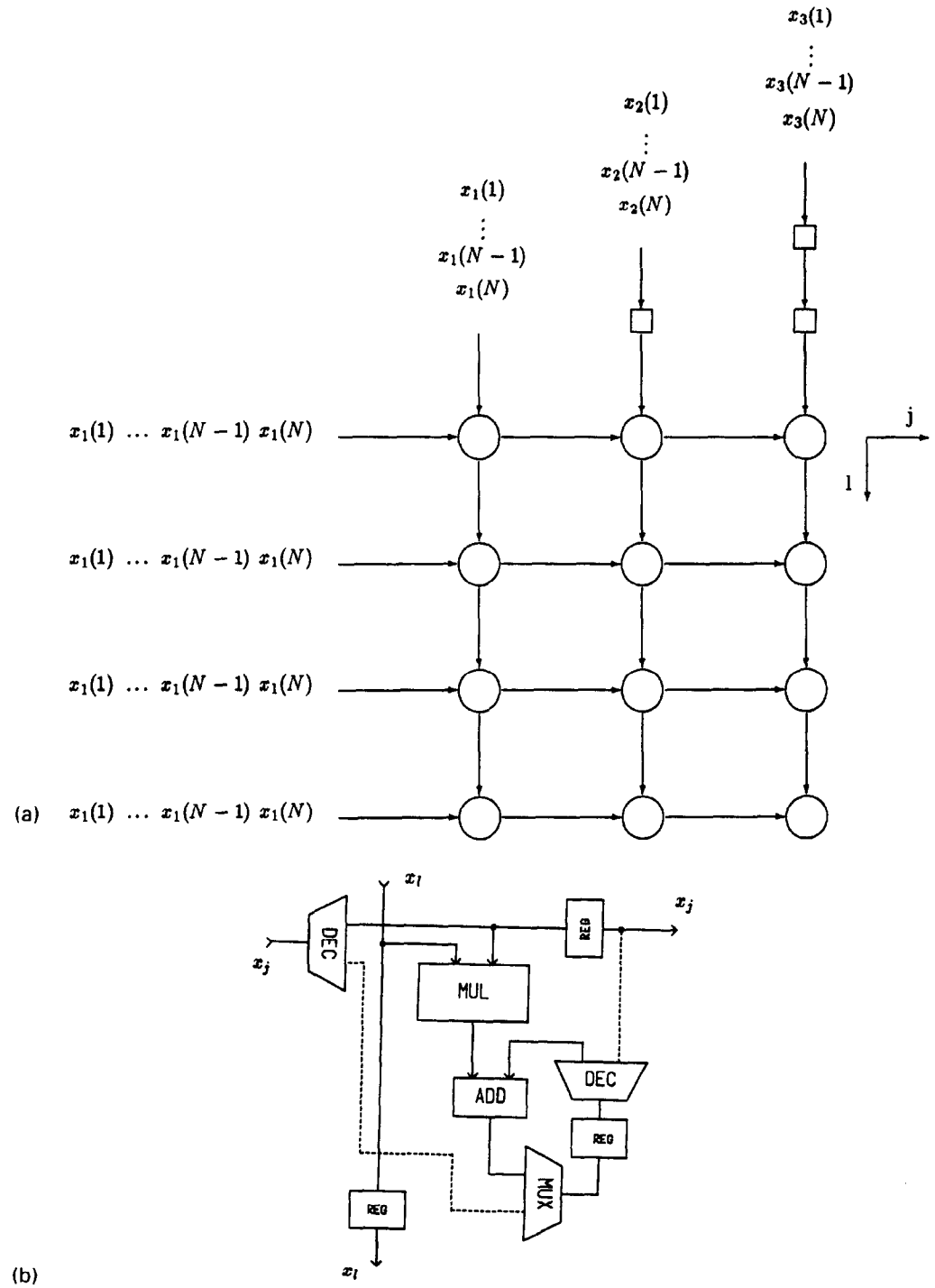


Fig. 1. (a) A 2-D systolic/wavefront array for the computation of the initial values (3-channel case with filters of order 3). Boxes denote delay elements and circles denote Processor Elements that perform multiplication and division. (b) Functional block for each node of the array. MUL, REG, MUX, DEC and ADD denote a multiplier, a register, a multiplexer, a decoder and an adder, respectively.

computational flow among the various processor elements. Circles denote processor elements executing multiplication and addition and boxes denote delay elements. The functional operation of the *processor element* (PE) used in this array is illustrated in Fig. 1(b). In the sequel abbreviation PE is utilized to denote a processor element.

Each array utilizes kp PEs, where k is the number of channels and p is the order of the filter. The computation scheme is terminated after $N + k$ time units, N being the number of available data. The time unit consists of the time needed to perform a multiplication and an addition. The absolute value of this time unit is dependent on the specific circuits that will be used. When all necessary computations have been finished, initial values for e^f, e^b and e^c are located at the nodes of the array. At every time step each processor (j, l), $j = 1, \ldots, k$, $l = 0, \ldots, p$, is fed with inputs $x_1(n)$, $x_j(n)$, $n = N, \ldots, 1$, from the left and from the top, respectively. The result of the multiplication between the two inputs is added to the resulting output of the previous computation step. At the end of computation ($N + k$ time units) the outputs $r_{1,j}(l) = e^{b(1,j)}(l) = e^{f(1,j)}(l)$ are located at nodes (j, l). We can activate the data path created by the multiplexer and the decoders (denoted with a dashed line in Fig. 1(b)) in order to obtain the results serially from the rows of the array of Fig. 1(a) and pass them to controller for further manipulation. The overall architecture is depicted in Fig. 2.

A column of dividers is used inside the controller in order to divide the outputs with the number of available data (in the MSE case). The array of Fig. 1 needs $k(p + 1)$ PEs and $k(k - 1)/2$ registers as delay units. Each PE is composed of one multiplier, one adder, three registers, two decoders and one multiplexer. The overall time of the array function is

$$\text{Tall} = \text{Tcomp} + \text{Tres},$$

where Tcomp is the computation time and Tres is the time needed to obtain the results. $\text{Tcomp} = Nkt_c + (k - 1)t_c$, where t_c is the cycle time of a PE. $t_c = t_{\text{mult}} + t_{\text{add}}$, where t_{mult} is the delay of a multiplier and t_{add} is the delay of an adder. $\text{Tres} = kt_{\text{res}}$, where t_{res} is the time that a result variable is obtained from every PE.

Completion of the initial values computations requires either $k + 1$ such arrays (k for the computation of $e^b(n)$ and $e^f(n)$ and 1 for the computation

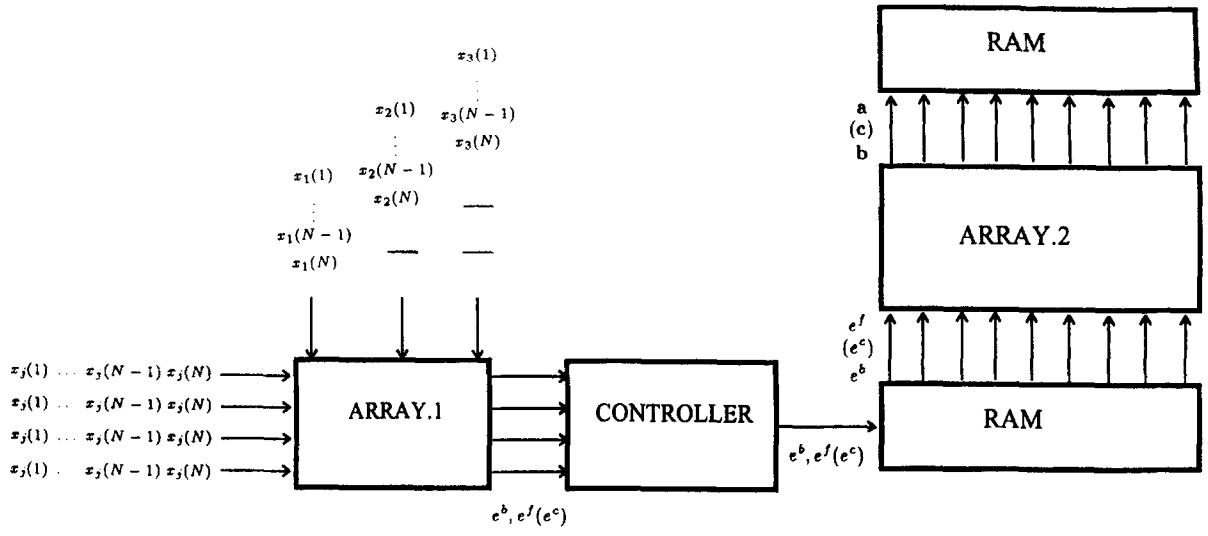


Fig. 2. The overall architecture for the solution of block Toeplitz or Toeplitz like systems (3-channel case with filters of order 3). ARRAY.1 is a 2-D systolic array for the estimation of the correlation lags used for the initialization of the algorithms. CONTROLLER is an interface for the rearrangement of the initial values. ARRAY.2 is a 2-D wavefront array which implements the proposed algorithms.

of $e^c(n)$) or a sequential implementation of the same array $k+1$ times.

A suitable overall system architecture is depicted in Fig. 2. The computed initial values of e^f, e^b and e^c are obtained serially from every row of the ARRAY.1 of Fig. 2. Then they pass through a controller composed of a column of $(p+1)$ dividers, a column of $p+1$ FIFOs of length k and glue logic, that places them in an appropriate order. The results are stored in a RAM of size $2(k+1)\times kp$ words, in the order presented in Fig. 7, and can directly be the input of input of ARRAY.2 of Fig. 2. ARRAY.2 is depicted in detail in Fig. 7 and analytically presented in Section 4.2. After the computation of the filter parameters the results are serially obtained from the columns of the above-mentioned array and are stored in a multiport RAM of size $2(k+1)\times kp$ words.

4.2. Main algorithm. A two dimensional approach

The main array processor architecture that implements the proposed Schur algorithms is developed. Since the algorithms dependence graph (DG) representation is complicated we will only give the DG for some equations of the first algorithm (MSE case). In a similar way all the other equations of the algorithms are handled. Consider Eqs. (1), (2) and (3) of Table 1. Eqs. (1), (2) are represented by the DG of Fig. 3. The DG of Fig. 4 represents Eq. (3). In order to simplify the exposition the 3-channel case with filter order 3 is depicted. Nodes of the arrays denoted by boxes perform division, while nodes denoted by circles perform multiplication and addition. The last index of variables e^f, e^b, b, c and the index of reflection coefficient k^c are for representation purposes and denote the phase of the algorithm. As is obvious from Tables 1 and 2 the algorithms are composed of two parts, one for the computation of the various errors and another for the computation of the various predictors and the resulting filter. The variables of the first part are decreased by one element in every phase of the algorithm while the variables of the second part are augmented by one element in the same phase. As we see from Fig. 3, the node responsible for the computation of the reflection coefficients is not necessary for subsequent computations. These nodes are located in such array positions that if a projection along any axis of the array is made, nodes performing division are used. Using a rearrangement of the input variables a triangular array results with the nodes performing division located at the first column. In conjunction with the array of Fig. 4 we finally have the DG graph, presented in Fig. 5 for all the equations ((1), (2), (3)) of Table 1. A projection along the vertical axis in conjunction with the consideration of the reflection coefficient being transferred rather than broadcasted, in order to ensure locality, results in the DG of Fig. 6(a). Using the canonical methodology [11, 16], we finally have the wavefront array of Fig. 6(b). The depicted array consists of appropriate PEs provided with handshaking ports. The functional block of such a PE is illustrated in Fig. 8(b).

In a similar way using the DGs for the remaining equations we obtain the main array processor architecture.

The final configuration for the 3-channel case with filters of order 3 is depicted in a two dimensional (2-D) graph in Fig. 7. The array discussed above is located in the last row of the array with its PEs denoted by an asterisk. $O((k+1)kp)$ processor cells are required to cope with the k channel FIR filtering. The overall processing time is $O(kp)$ time units.

Each node of the graph is represented by indices (v,n) where $v=1,\ldots,(k+1)$ and $v=0,\ldots,((kp)-1)$. Two types of processors are utilized. The first is a multiply and add unit and is denoted by a circle. The second, denoted by a box, apart from multiplication and addition, performs a division as well. Their functional operation is illustrated in Fig. 8. The arcs of the graph illustrate the computational flow among the various processor elements.

The function of the array can be divided in three phases. First, the PEs are serially loaded with the initial values, as illustrated in Fig. 7. These values in the MSE case are assumed to be known in advance. Otherwise they are computed in the array presented in Section 4.1 and are fed in the array as depicted in Fig. 2. As illustrated in Fig. 8 we have created a data path inside the PEs (indicated with dashed lines) using multiplexers and decoders,

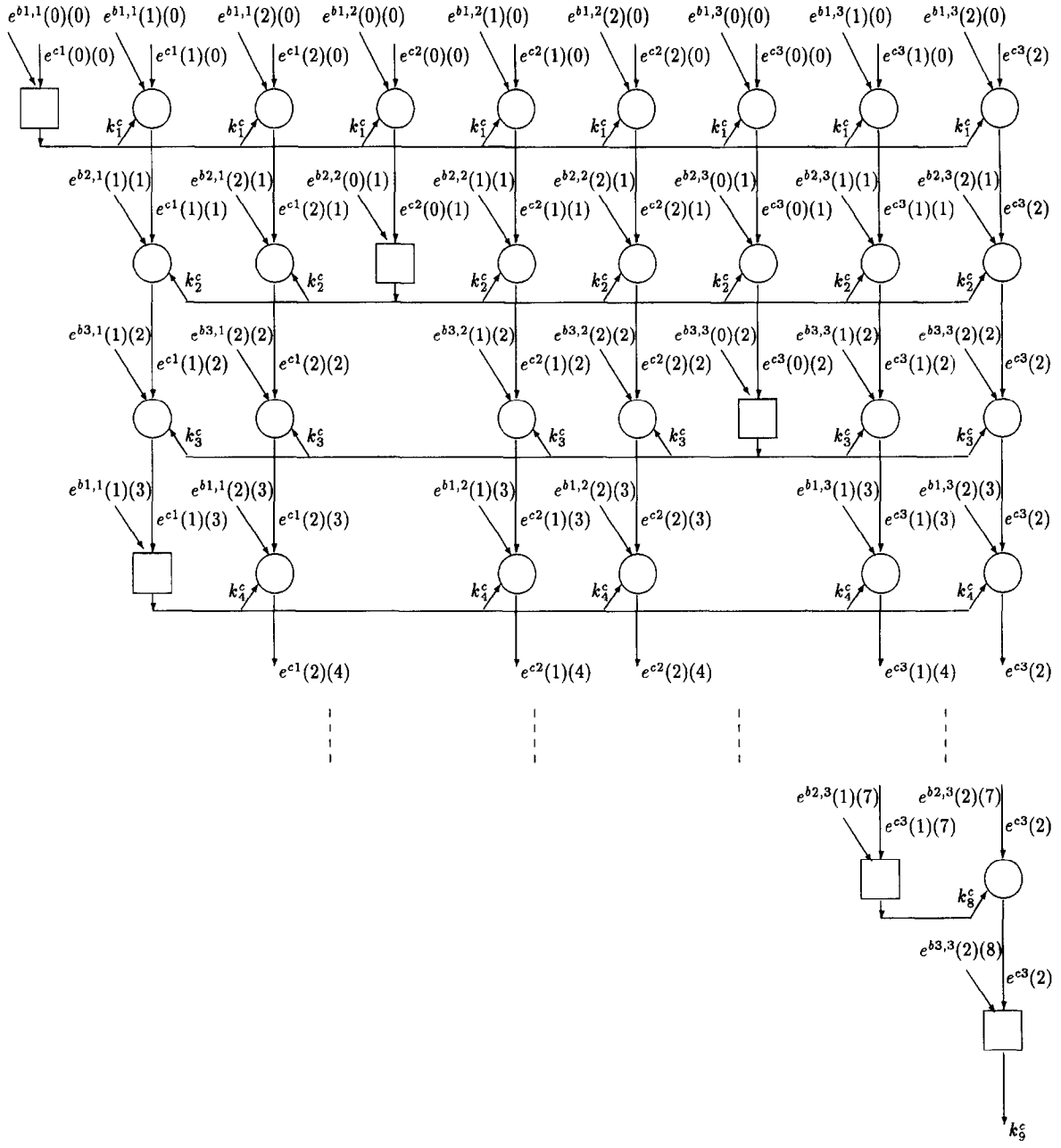


Fig. 3. The Dependence Graph for Eqs. (1) and (2) of Table 1 (3-channel case with filters of order 3). The circles denote nodes that perform multiplication and addition, while boxes denote nodes that perform division.

through which the data input occurs. These variables have been properly rearranged in order to ensure local communication as well as to minimize the use of division cells. Indeed, the proposed architecture involves $k + 1$ PEs that execute division.

The PEs located at nodes of the form $(v, 0)$, $v = 1, \ldots, k$, compute variables k^{fv} and $k^{b\mu}$, while

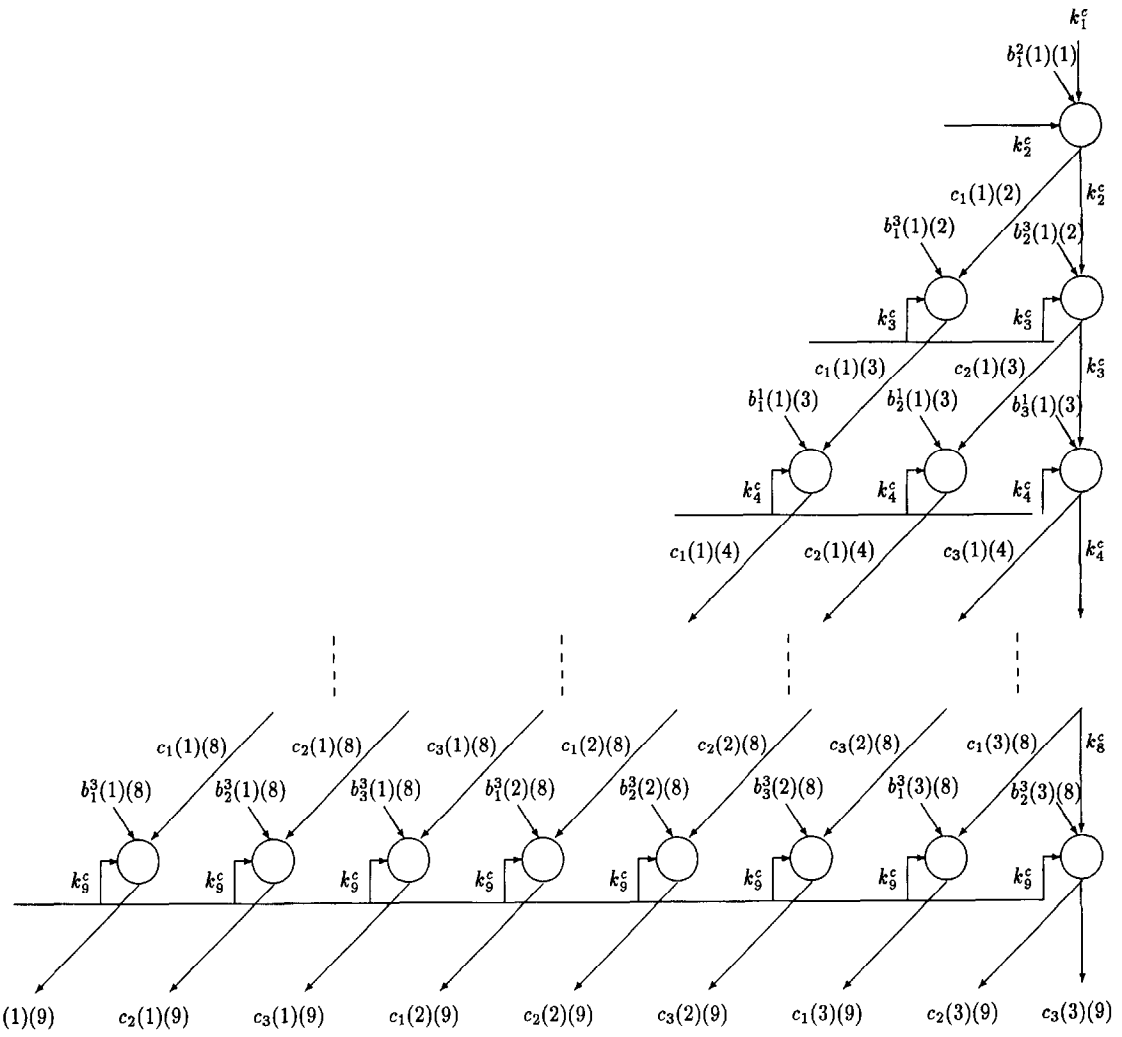


Fig. 4. The Dependence Graph for Eq. (3) of Table 1 (3-channel case with filters of order 3). The circles denote nodes that perform multiplication and addition.

the first processor of the last row $(k+1,0)$ (PE denoted by a box with an asterisk) computes k^c. The resulting outputs are serially transmitted to the neighboring nodes $(v,1)$. All other processors (v,n) receive the reflection coefficients, denoted by k, from the $(v,n-1)$ processor and pass them to $(v,n+1)$. Once the values of reflection coefficients k^{fv} and $k^{b\mu}$ reach processor (v,n), they are used for the computation of $e^{b\mu,h}(n)$ and $e^{fv,h}(n)$ (or $\boldsymbol{b}_h^{\mu}(|n-kp|)$ and $\boldsymbol{a}_h^{v}(|n-kp|)$), since the new values of $\boldsymbol{b}$ and $\boldsymbol{a}$ are created at the places of e^b and e^f that are not necessary for subsequent computations. PEs $(k+1,n)$ (PEs denoted by a circle with an asterisk) compute e^c (or $\boldsymbol{c}$). The PEs $(v,0)$ do not compute the variables e^f and e^c. When computations are completed, the new values of $e^{b\mu}$ (or $\boldsymbol{b}^{\mu}$) are passed to the $(v-1,n)$ processor in order to meet the appropriate e^{fv} (or $\boldsymbol{a}^{v}$) for the next phase. Concerning the new values of e^{fv} (or $\boldsymbol{a}^{v}$) and e^c (or $\boldsymbol{c}$), they are passed to $(v,n-1)$ and $(k+1,n-1)$ processors, respectively. The circulation of e^f (or $\boldsymbol{a}$) and e^c (or $\boldsymbol{c}$) is necessary to ensure that the

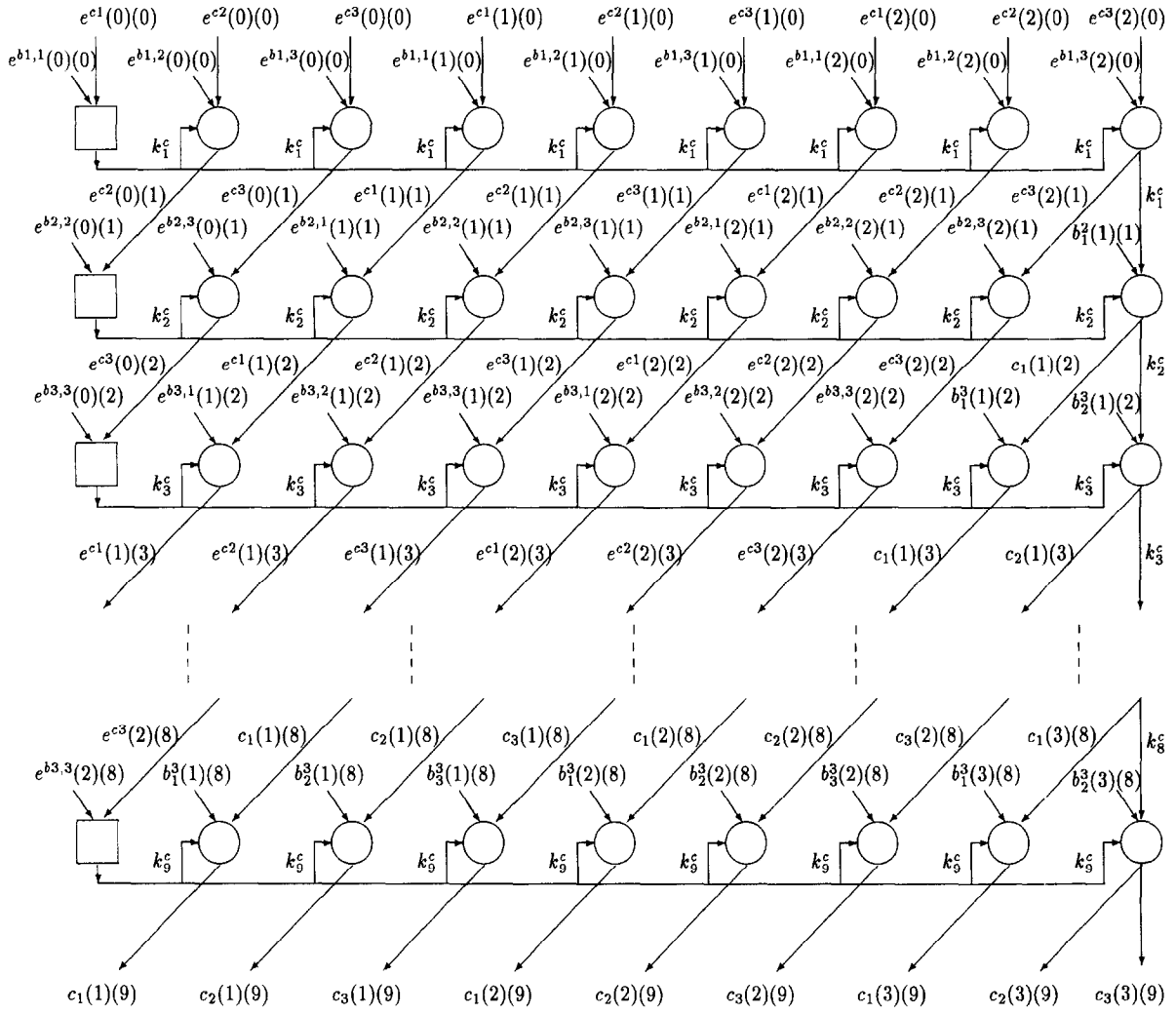


Fig. 5. The Dependence Graph for Eqs. (1), (2) and (3) of Table 1, resulting from the rearrangement of input data (3-channel case with filters of order 3). The circles denote nodes that perform multiplication and addition, while boxes denote nodes that perform division.

computation of the reflection coefficients (k^c, k^b, k^f) is always performed by $(v, 0)$ processors and at the same time create an empty place at the $(v, kp-1)$ processors. In a similar fashion variables e^c flow from processors $(k+1, n)$ to processors $(k+1, n-1)$, leaving an empty place at $(k+1, kp-1)$ PE.

In each phase of the algorithm there are empty places where the new values of $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ are created from the values of k^f, k^b and k^c respectively. At the end, the resulting values of $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{c}$ are placed at corresponding places where initialization variables e^f, e^b, e^c were stored.

The new values of $\boldsymbol{a}^v$ are always created at the $(v, kp-1)$ PE, $v = 1, \ldots, k$. The new values of $\boldsymbol{c}$ are created at the $(k+1, kp-1)$ PE. The new values of $\boldsymbol{b}^\mu$ appear at PE $(v, kp-1-m)$, $m = 0, \ldots, kp$. A 1-bit control signal is used to indicate the PE where the new values of $\boldsymbol{b}^\mu$ are created. Initially all PEs stay in a 'false' mode except the ones with indices $(v, kp-1)$. Thus, the first values of $\boldsymbol{b}^\mu$ are generated at these nodes. Once this has been completed, the

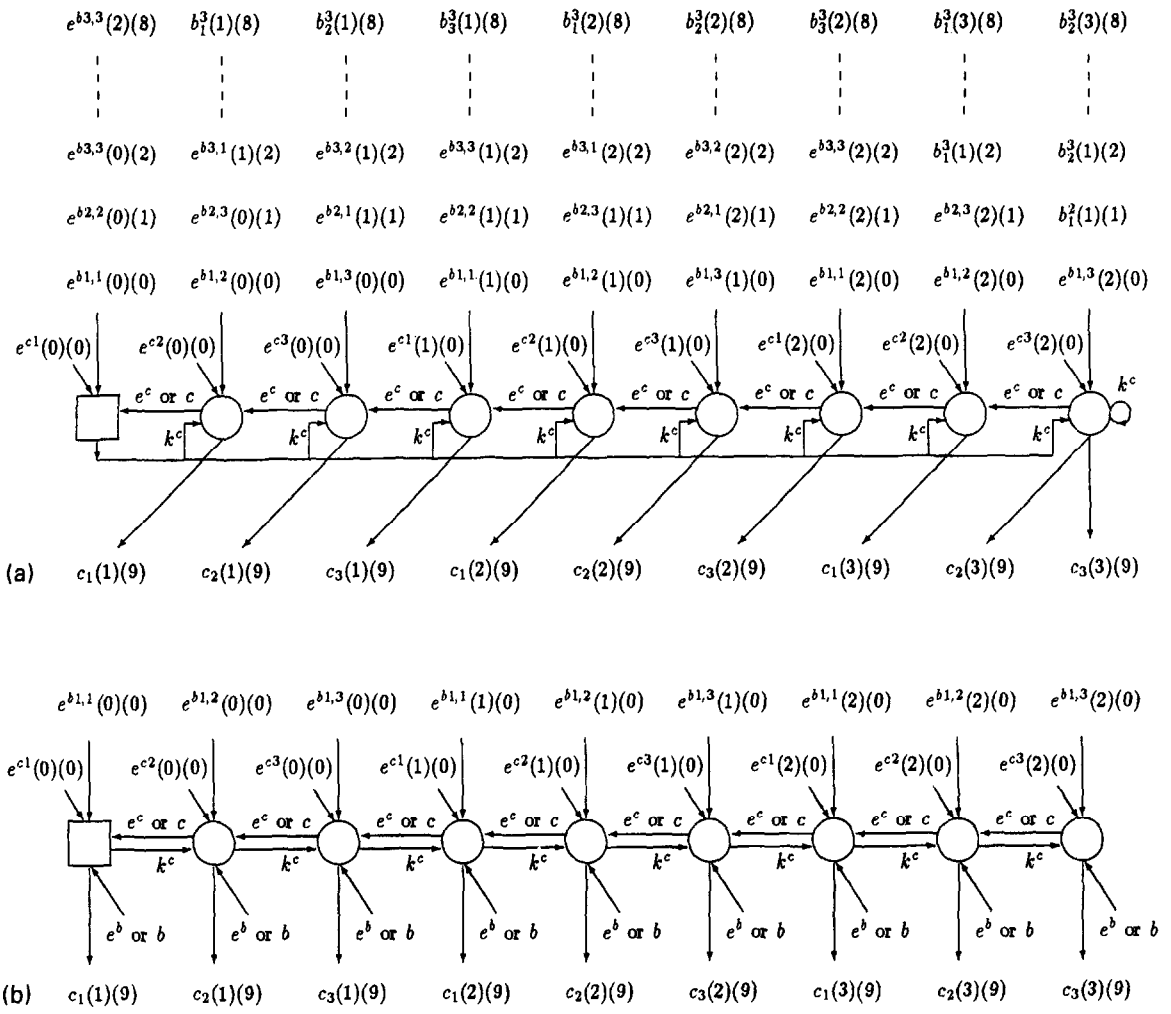


Fig. 6. (a) The Dependence Graph of Eqs. (1), (2) and (3) of Table 1 resulting from the projection along the vertical axis of the DG of Fig. 5 (3-channel case with filters of order 3). (b) The wavefront array that implements Eqs. (1), (2) and (3) of Table 1 (3-channel case with filters of order 3). Circles denote PEs that perform multiplication and addition, while boxes denote PEs that perform division.

PEs $(v, kp-1)$ are deactivated, passing the control signal to their neighboring PEs $(v, kp-2)$ making them active and so on. When the PEs $(v, 0)$ become active and the last values of $\boldsymbol{b}^{\mu}$ are generated, another control signal is activated, which is sent together with the last values of reflection coefficients and signals the end of the computation. As is obvious from the above description both the control and data flow is local leading to a wavefront array. In the final phase of the array function the resulting values of $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ are placed at corresponding places where initialization variables e^f, e^b, e^c were loaded as described earlier. We can now employ the data path used in the initial phase, to obtain the results serially from the columns of the array and store them in a multiport RAM of size $2(k+1) \times kp$ words.

As we can conclude from Fig. 7 the array needs $(k+1)kp$ PEs. Each of $k^2(p-1)$ PEs (denoted by a circle in Fig. 7) is composed of two multipliers, two adders, five handshaking modules, four registers, two decoders, three multiplexers and a control

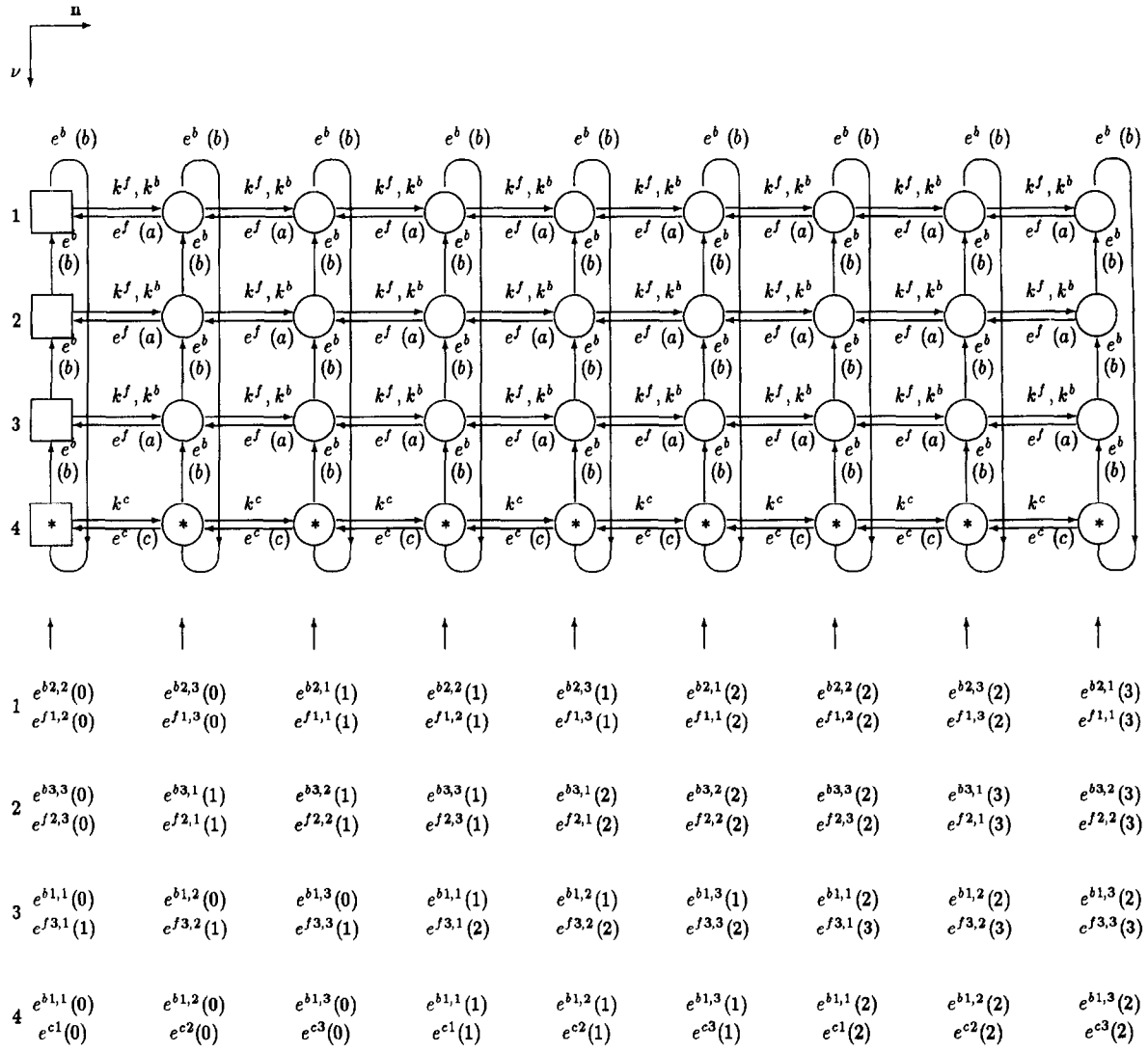


Fig. 7. A 2-D wavefront array for the 3-channel case with filters of order 3. The circles denote PEs that perform multiplication and addition while boxes denote PEs that apart from multiplication and addition perform division as well. At the end of the computation the results are located at the PEs of the array.

unit of about ten gates of glue logic. Their functional block is depicted in Fig. 8(a). The k PEs (denoted by a box in Fig. 7) have the same amount of hardware with the above-mentioned PEs plus two dividers in order to perform the computation of the reflection coefficients. The $k(p-1)$ PEs (denoted by a circle with an asterisk in Fig. 7) are composed of one multiplier, one adder, five handshaking modules, three registers, two decoders, two multiplexers and a control unit of about ten gates of glue logic, each. Their functional block is depicted in Fig. 8(b). One PE (denoted by a box with an asterisk in Fig. 7) is composed of the hardware mentioned above and one divider.

The overall time of the array function is

$$\text{Tall} = \text{Tinit} + \text{Tcomp} + \text{Tres},$$

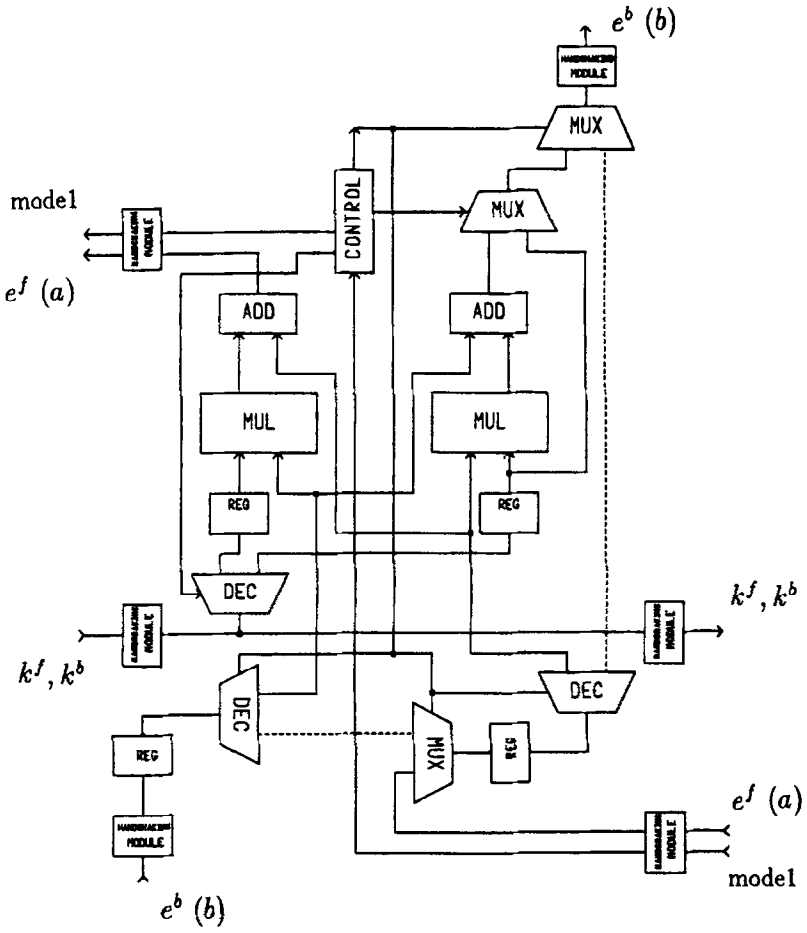


Fig. 8. Functional blocks for the nodes of the array of Fig. 7. (a) PE denoted by circle (MSE and TSE cases).

where Tinit is the time needed for the initialization of the array, Tcomp is the computation time and Tres is the time needed to obtain the results. Note that Tinit $= 2(k+1)t_{\text{load}}$, where t_{load} is the time in which every variable e is loaded in a PE.

Tcomp = Trcomp + Tkdel.

Trcomp $= kpt_c$ where kp are the needed cycles of a PE for the overall computation and t_c is the cycle time of a PE. $t_c = 2t_{\text{kload}} + t_{\text{mult}} + t_{\text{add}}$ where t_{kload} is the time for a reflection coefficient to be stored in a register in the PE. Tkdel $= k(p-1)t_{\text{kdel}}$ where t_{kdel} is the time that a reflection coefficient needs to pass from a PE to its neighbor. Tkdel is the time that a reflection coefficient needs to arrive at the last PEs of the array $(v, k(p-1))$. Tres $= 2(k+1)t_{\text{res}}$ where t_{res} is the time in which every variable of the results $\boldsymbol{a}$ (or $\boldsymbol{b}$, or $\boldsymbol{c}$) is obtained from a PE.

OCCAM description of the basic processor units is provided in Fig. 9. Readers unfamiliar with OCCAM language may look at the detailed description of the code of Fig. 9(a) given in Appendix A.

A similar array processor can be utilized for the implementation of the total least squared Schur algorithm tabulated in Table 2. The only difference is the functional operation of the PEs with indices $(k+1, n)$ $(n = 0, \ldots, kp-1)$. The functional block of such a PE is illustrated in Fig. 10). The PE $(k+1, 0)$ computes the values of k^c, k^w, k^b and

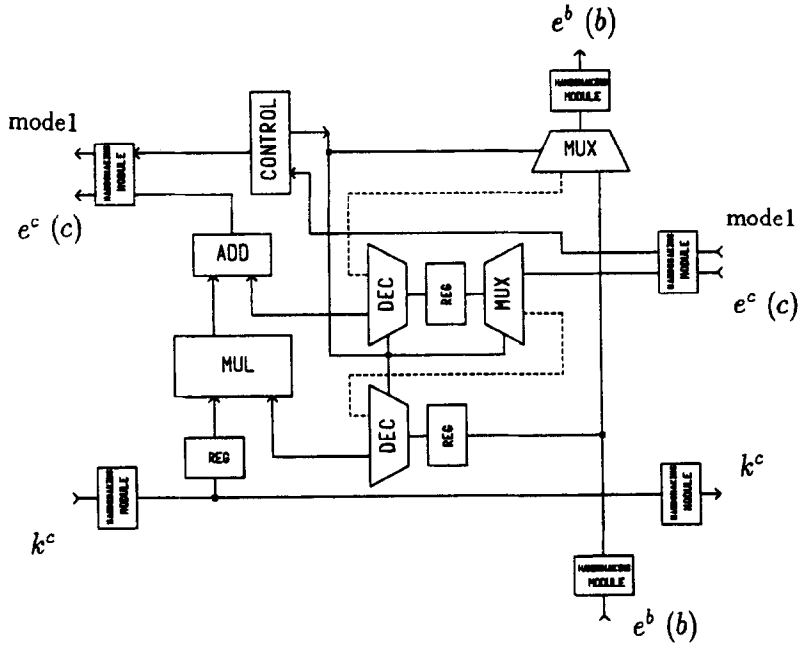


Fig. 8. (b) PE denoted by circle with an asterisk (MSE case). MUL, ADD, REG, DEC and MUX denote a multiplier, an adder, a register, a decoder and a multiplexer, respectively.

passes them to its neighbor $(k+1,1)$. Each PE $(k+1,n)$ receives the values of reflection coefficients from PE $(k+1,n-1)$ and forwards them to the next PE $(k+1,n+1)$. When these values reach a PE they are used for the computation of the $e^c(N)$ (or $\boldsymbol{c}(N)$), $\varepsilon(N)$ (or $\boldsymbol{w}(N)$) and $e^b(N-1)$ (or $\boldsymbol{b}(N-1)$). When the above computations have been completed, PEs $(k+1,n)$ forward the computed values of $e^c(N)$ (or $\boldsymbol{c}(N)$) and $\varepsilon(N)$ (or $\boldsymbol{w}(N)$) to PEs $(k+1,n-1)$ while they receive the computed values of $e^c(N)$ (or $\boldsymbol{c}(N)$) and $\varepsilon(N)$ (or $\boldsymbol{w}(N)$) from PEs $(k+1,n+1)$. The new values of $\boldsymbol{c}(N)$ and $\boldsymbol{w}(N)$ are generated at PE $(k+1,kp-1)$. $e^b(N-1)$ (or $\boldsymbol{b}(N-1)$) are forwarded to PEs (k,n) while $e^b(N-1)$ (or $\boldsymbol{b}(N-1)$) are received from PEs $(1,n)$. All the other processors of the array have the same functionality as the ones discussed in the MSE case. The data and control flow remain the same as well. Variables e^f (or $\boldsymbol{a}$) and e^b (or $\boldsymbol{b}$) have been replaced by $e^f(N)$ (or $\boldsymbol{a}(N)$) and $e^b(N)$ (or $\boldsymbol{b}(N)$), respectively.

The amount of hardware is augmented since the PEs of the last row of the array (denoted with an asterisk in Fig. 7) have the following extra circuits each: one multiplier, one adder, three handshaking modules and two registers. The size of the off chip RAM is now $(2k+3)\times kp$ words.

The overall time of the array function is again

$$\text{Tall} = \text{Tinit} + \text{Tcomp} + \text{Tres},$$

where Tinit is the time needed for the initialization of the array, Tcomp is the computation time and Tres is the time needed to obtain the results of the computation from the array. $\text{Tinit} = 3(k+1)t_{\text{load}}$, where t_{load} is the time in which every variable e is loaded in a PE.

Tcomp = Trcomp + Tkdel.

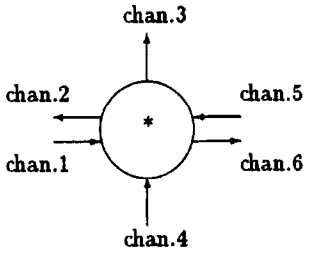
Trcomp = kpt_c where kp are the needed cycles of a PE for the overall computation and t_c is the cycle time of a PE. $t_c = 3t_{\text{kload}} + 2t_{\text{mult}} + t_{\text{add}}$ where t_{kload} is the time for a reflection coefficient to be stored in a register in the PE. $\text{Tkdel} = k(p-1)t_{\text{kdel}}$ where t_{kdel} is the time that a reflection coefficient needs to pass from a PE to its neighbor. Tkdel is the time that a reflection coefficient needs to arrive at the last PEs of the array $(v,k(p-1))$. $\text{Tres} = 3(k+1)t_{\text{res}}$ where t_{res} is the time in which every variable of the results $\boldsymbol{a}$ (or $\boldsymbol{b}$, or $\boldsymbol{c}$, or $\boldsymbol{w}$) is obtained from a PE.

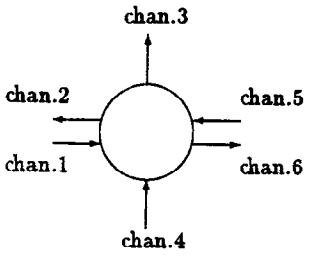
```
WHILE NOT stop
  SEQ
    chan.1 ? kf ; kb ; stop
    chan.6 ! kf ; kb ; stop
    IF
      NOT model
        PAR
          SEQ
            ef := tempef + (tempeb*kf)
            chan.2 ! ef ; model
          eb := tempeb + (tempef*kb)
      model
        PAR
          SEQ
            ef := tempef + (tempeb*kf)
            chan.2 ! ef ; model
          eb := kb
          model := FALSE
    PAR
      chan.3 ! eb
      chan.5 ? tempef ; mode
    chan.4 ? tempeb
```

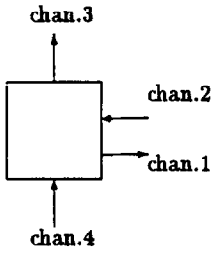
a)

```
WHILE NOT stop
  SEQ
    chan.1 ? kc ; stop
    chan.6 ! kc ; stop
    ec := ec + (eb*kc)
    chan.3 ? eb
    chan.5 ! eb
    chan.2 ! ec ; mode
    chan.5 ? ec ; mode
```

b)








```
WHILE NOT stop
  SEQ
    PAR
      kf := -(tempef/tempeb)
      kb := -(tempeb/af)
      IF
        model
          stop := TRUE
        TRUE
          SKIP
    chan.1 ! kf ; kb ; stop
    IF
      NOT model
        PAR
          af := af + (tempef*kf)
          eb := tempeb + (tempef*kb)
      model
        eb := kb
    PAR
        chan.2 ? tempef ; model
        chan.3 ! eb
        chan.4 ? tempeb
```

c)

Fig. 9. OCCAM description for some of the PEs of the array of Fig. 7. The boundary processors $n = (kp - 1)$ have some modifications since they do not compute the a or c but assign to them the new values of reflection coefficients. (a) PE denoted by a circle (MSE and TSE cases). (b) PE denoted by a circle with an asterisk (MSE case). (c) PE denoted by a box (MSE and TSE case).

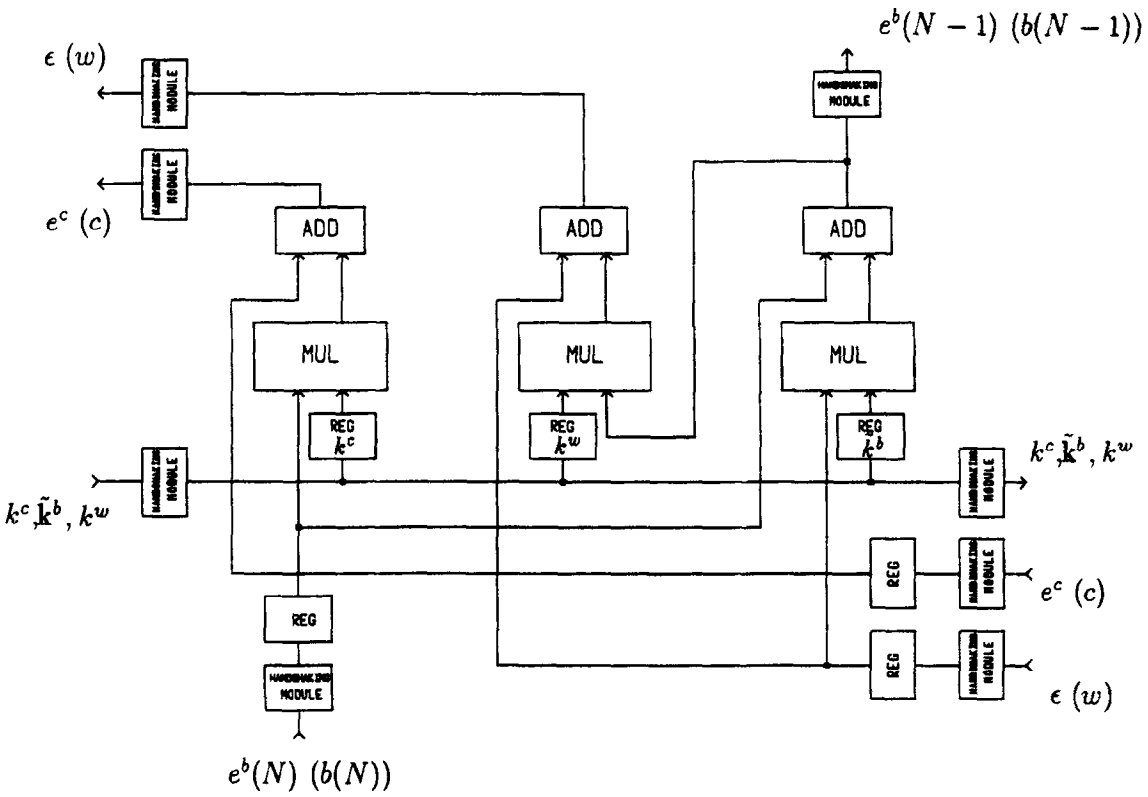


Fig. 10. Functional block for the PE of the array depicted in Fig. 7, which is denoted by a circle with an asterisk (TSE case). MUL, ADD and REG denote a multiplier, an adder and a register, respectively.

4.3. *Implementation issues*

The wavefront arrays discussed so far are the two dimensional, thus being suitable for VLSI implementation. Although the number of PEs is extremely large the increasingly diminished dimensions of the devices in current technologies along with new technologies as wafer scale integration (WSI) or multiple chip modules (MCM) allow for a possible implementation of the algorithm in a single chip, in the near future.

Dedicated PEs must be designed in order to maximize performance. Simulations have shown that both algorithms are well behaved using fixed-point arithmetic with word-length greater than 16-bits. Thus fixed-point arithmetic will be used for hardware implementation of the arrays. All PEs have a very simple structure. They all consist of a multiplier, an adder, a couple of registers and some glue logic for the control. Since wave-front arrays are employed, the ports must be implemented by handshaking circuits. Notice that the PEs assigned to $(v,0)$ include extra circuitry to perform division.

Multiple data paths can be utilized to achieve full parallelism. Even in the case of PEs assigned to $(v,0)$, more than one divider must be used, when we have to compute more than one reflection coefficient. However, the large number of PEs makes it prohibitive within the existing technology. Many choices could be made for the implementation of the multiplier and the adder, such as bit-serial or full parallel architectures, having trade-offs between area and speed. Since the area is the most restrictive factor in our case, 'slow' but 'small' circuits must be used in order to implement the

algorithm in a single chip. The scheduling of the operations is not so vital for the speed of the algorithms except the priority given to the computation of the $\boldsymbol{b}(N-1)$ (or $e^b(N-1)$) in the TSE case.

5. An architecture with reduced number of processors

In this section a more realistic approach towards hardware implementation is described. Grouping of the PEs of Fig. 7 along the 'channel' axis is performed. We map every group of k PEs (k is the number of channels) to one PE of the new architecture. The resultant architecture is illustrated in Fig. 11 (3-channel case with filters of order 3).

Each PE in the new scheme has to handle the operations performed by k PEs of Fig. 7. The input data are serially loaded in the array in a similar fashion as in the array of Fig. 7. Every PE of Fig. 11 has to be loaded with all the input data of k PEs of Fig. 7, increasing the time needed for the initialization of the array by a factor of k. The same increase in time is also there at the end of the computation, when we have to output serially the results located at the PEs of the array.

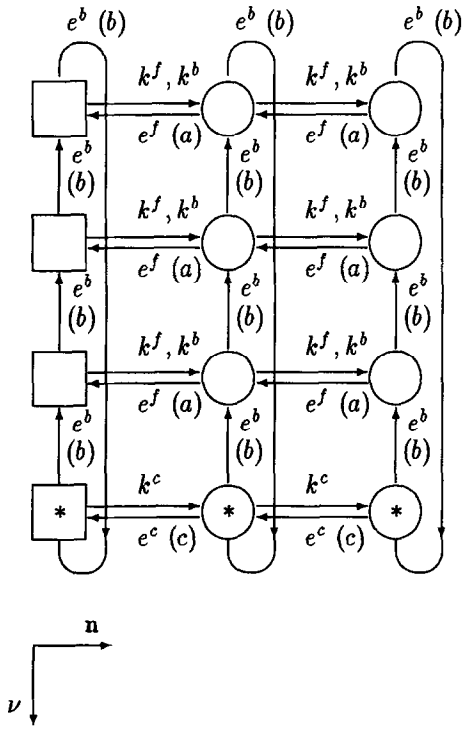


Fig. 11. A 2-D array for the 3-channel case with channel length 3, resulting from the folding of the array of Fig. 7. Input and output of data are performed in a similar fashion to the one described for the array of Fig. 7.

Following the array processor of Fig. 7, multiply and add processors are represented by a circle, while units equipped with division circuitry are denoted by a box. This 2-D array is of primary interest when VLSI implementation is considered, due to the reduced number of PEs that are utilized. Indeed only $(k+1)p$ processors are employed. This significant hardware reduction is traded off by an increase of computation time. The computational load of each PE is increased by a factor of k, resulting in an overall time of $O(k^2p)$ time units.

Implementation of the algorithm in a massively parallel machine composed of standard DSP processors is more effectively carried out by the above architecture. Indeed parallel machines have large communication cost because their processors are intended for general use. The architecture allows grouping of the variables and thus reduction of the communication overhead. The efficient architecture depicted in Fig. 6, for the implementation of the multichannel Schur algorithm, was programmed on Parsytec's GC512 highly parallel machine, consisting of 512 transputers. A particular problem encountered was the excessive time needed for input and output of data, since only one processor was available for the communication between the machine and the host computer.

A suitable dedicated processor for wavefront implementation is shown in block diagram form in Fig. 12. A double data path is utilized to increase processor speed, to simplify control and to take advantage of the specific structure of the pertinent algorithms. Also using multiplexers and decoders we create a data path in order to load the initial values to the array and to obtain the results from it. The circuit that mostly affects the PE speed is the multiplier. This is also the case in most DSP applications. At present a wide range of fast multiplication algorithms are available to the designer [14, 20]. A parallel multiplier based on the modified Booth algorithm seems to be appropriate in our

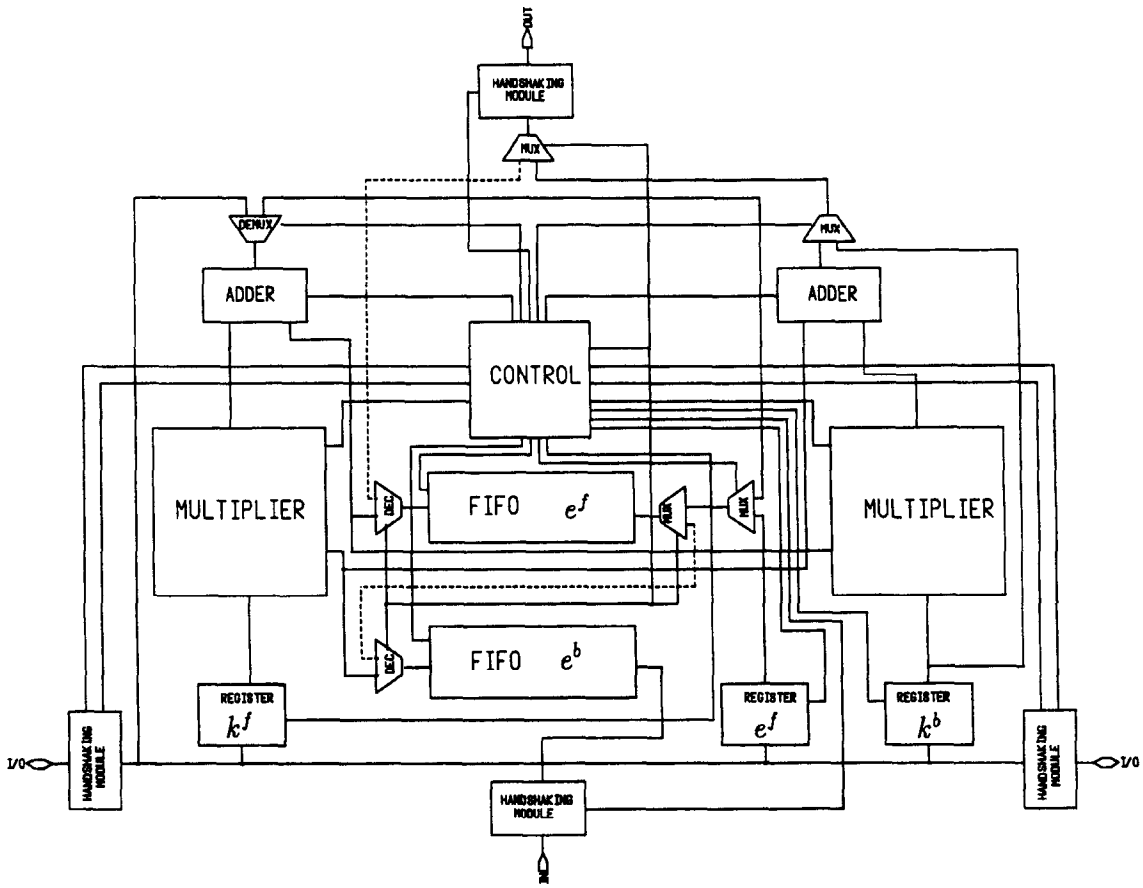


Fig. 12. The block diagram of an ASIC processor suitable for the 2-D array of Fig. 11.

case. It assures a high speed performance and a compact layout as well. In our case there is an extra advantage since in each phase of the algorithm a value of variable k is multiplied with a number of variable e values. Thus, the delay due to the modified Booth algorithm encoding, is counted only once, for a number of multiplications equal to the number of channels. It should be pointed out that good performance could be achieved even with the use of single-level-metal CMOS technology [14, 20]. This enables the utilization of the remaining metal levels for the interconnections of the whole array. The processor also includes two fast adders (e.g., CLA adders), two FIFOs, each one equal to at least the number of channels, four ports suitable for handshaking, some registers and a simple control unit. The FIFOs may need some extra space to cope with any communication bottleneck.

The two FIFOs are initially loaded with initial values of $e^{fv,h}$ and $e^{b\mu,h}$ by activating the appropriate data path. At each phase the new values of k^f, k^b arrive at the processor. These in turn pass to its neighbor and simultaneously are stored in the two registers. After the computation of the new values e^f (or $\boldsymbol{a}$) and e^b (or $\boldsymbol{b}$), the first value of e^f is passed to the neighboring processor while the other ones return to the FIFO. After the end of the phase the value of e^f that arrives at the processor is placed in the last position of the FIFO. All values of e^b (or $\boldsymbol{b}$) pass to the neighboring processor as indicated in Fig. 11. Similar operations are performed by the processors that compute e^c (or $\boldsymbol{c}$).

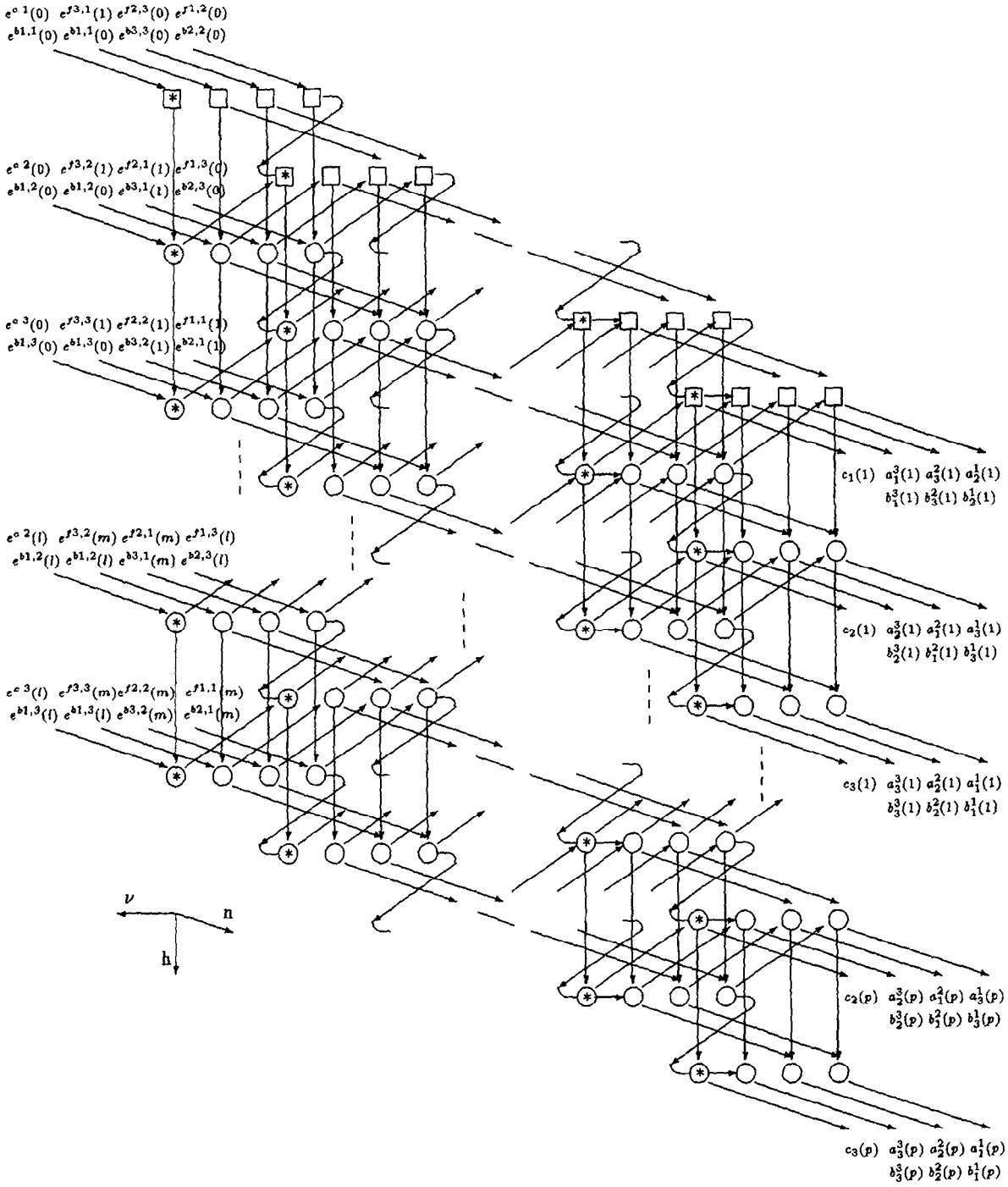


Fig. 13. A 3-D wavefront array for the 3-channel case, suitable for both the MSE and TSE case. PEs denoted by circles perform multiplication and addition. PEs denoted by boxes apart from multiplication and addition, perform division as well. The indices m and l used to describe the input data are $m = kp$ and $l = kp - 1$.

In the total squared error case of Table 2, the structure of the algorithm is essentially the same, i.e., the dependence graph does not change apart from some of the nodes which perform some extra multiply–add operations. More precisely, the processors computing the vector $\boldsymbol{c}$ have now to perform three operations at a time. One design alternative is to increase the data paths. A possible remedy is to allocate the third operation to one of the existing data paths slightly increasing the overall computation time of the array. In this case a careful scheduling of the operations must be accomplished giving priority to the computation of $\boldsymbol{b}(N-1)$ (or $e^b(N-1)$). In both cases a third FIFO must be used to store the additional variables.

6. A highly pipelined 3-D array

An alternative architecture for the implementation of the proposed algorithms is depicted in Fig. 13. It is a three dimensional array consisting of $(k+1)k^2p^2$ PEs, needed to handle the k channel setup. As with the previous arrays, the circles represent multiply–add processors while boxes perform division as well. The PEs are identified by a triplex (v,h,n) $(v=1,\ldots,k+1,\ h=1,\ldots,kp,\ n=1,\ldots,kp)$. The PEs $(k+1,h,n)$ compute the solution vector $\boldsymbol{c}$ (and vectors $\boldsymbol{w}$ and $\boldsymbol{b}(N-1)$ in the TSE case), while the rest compute the forward and backward predictors $\boldsymbol{a}$ and $\boldsymbol{b}$. The functional operations of each PE are similar to that utilized by the 2-D array counterparts. The input and output data are illustrated in Fig. 13. To clarify the computational flow, input and output data are indicated in the figure. The main advantage of this array is that it is susceptible to pipeline with latency kp time units. This attractive feature compensates for the excessive number of PEs. This particular implementation is suitable for block adaptive processing [21, 24]. Indeed, it is more efficient than the triangular array processors developed in [21] for the adaptive Schur algorithm, since matrix operations are replaced now by scalar counterparts.

7. Conclusions

Efficient VLSI architectures of highly concurrent algorithms for the order recursive solution of block Toeplitz like systems have been presented. The proposed structures can be implemented either on 3-D or on 2-D arrays of VLSI processors, compromising between execution time and number of processors.

Appendix A

All processes in Fig. 9(a) with the same indentation belong to the same process. For explanation reasons we give a number at the right of each process. All the subprocesses with the same number at their left belong to the same process. The symbols kf, kb, eb, ef denote the variables k^f, k^b, e^b (or $\boldsymbol{b}$), e^f (or $\boldsymbol{a}$), respectively, the symbols tempeb, tempef denote temporal variables for the e^b (or $\boldsymbol{b}$), e^f (or $\boldsymbol{a}$), the stop and model are logical variables and chan.1, chan.2, chan.3, etc., are the six channels of the processor element.

```
WHILE NOT stop
  SEQ(1)
    (1.a)chan.1 ? kf; kb; stop
    (1.b)chan.6 ! kf; kb; stop
    IF
      (1.c)NOT model
        PAR(2)
          (2.a)SEQ(3)
            (3.a)ef := tempef + (tempeb*kf)
            (3.b)chan.2 ! ef; model
          (2.b)eb := tempeb + (tempef*kb)
      (1.c)model
        PAR(4)
          (4.a)SEQ(5)
            (5.a)ef := tempef + (tempeb*kf)
            (5.b)chan.2 ! ef; model
          (4.b)eb := kb
          (4.c)model := FALSE
    (1.d)PAR(6)
      (6.a)chan.3 ! eb
      (6.b)chan.5 ? tempef; model
    (1.e)chan.4 ? tempeb
```

While the logical variable stop has the false Do sequentially (1)
(1.a) *The variables* kf, kb, stop *take serially their new values from the channel* chan.1.
(1.b) *The variables* kf, kb, stop *output their values serially to the channel* chan.6.

(1.c) *If the logic variable* model *has the value false do in parallel* (2)
(2.a) *Do sequentially* (3)
(3.a) *Add the result of the multiplication of* tempeb *and* kf *to the* tempef *and assign the result to the variable* ef.
(3.b) *The variables* ef *and* model *output their values to the channel* chan.2.
(2.b) *Add the result of the multiplication of* tempef *and* kb *to the* tempeb *and assign the result to the variable* eb.
(1.c) *If the logic variable* model *has the value false do in parallel* (4)
(4.a) *Do sequentially* (5)
(5.a) *Add the result of the multiplication of* tempeb *and* kf *to the* tempef *and assign the result to the variable* ef.
(5.b) *The variables* ef *and* model *output their values to the channel* chan.2.
(4.b) *Assign the value of the variable* kb *to the variable* eb.
(4.c) *Assign the value false to logic variable* model.
(1.d) *Do in parallel* (6)
(6.a) *Output the value of variable* eb *to the channel* chan.3.
(6.b) *The variables* tempef *and* model *take their new values from the channel* chan.5.
(1.e) *The variable* tempeb *takes its new value from the channel* chan.4.

References

[1] G. Glentis and N. Kalouptsidis, "Efficient order recursive algorithms for multichannel least squares filtering", *IEEE Trans. Signal Process.*, June 1992, pp. 1354–1374.
[2] G. Glentis and N. Kalouptsidis, "Efficient algorithms for the solution of block linear systems with Toeplitz entries", *Linear Algebra Applications*, January 1993.
[3] G. Glentis and N. Kalouptsidis, "Solution of block linear systems with Toeplitz entries using a channel decomposition technique", *Signal Processing*, Vol. 37, No. 1, May 1994, pp. 15–60.
[4] G.-O. Glentis and N. Kalouptsidis, "Efficient multichannel FIR filtering using a step versatile order recursive algorithm", *Signal Processing*, Vol. 37, No. 3, June 1994, pp. 437–462.
[5] A.K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[6] I.C. Jou, Y.H. Hu and W.S. Feng, "Novel implementation of pipelined Toeplitz system solver", *Proc. IEEE*, Vol. 74, 1986, pp. 1463–1464.
[7] T. Kailath, *Signal Processing in the VLSI Era*, Prentice-Hall, Englewood Cliffs, NJ, 1985.
[8] N. Kalouptsidis, G. Carayannis, D. Manolakis and E. Koukoutsis, "Efficient recursive in order LS FIR filtering and prediction", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 33, October 1985, pp. 1175–1187.
[9] N. Kalouptsidis and S. Theodoridis, "Parallel implementation of efficient LS algorithms for filtering and prediction", *IEEE Trans. Acoust. Speech Signal Processing*, Vol. 35, November 1987, pp. 1565–1569.
[10] N. Kalouptsidis and S. Theodoridis, eds., *Adaptive System Identification and Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[11] S.Y. Kung, *VLSI Array Processors*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[12] S.Y. Kung and Y.H. Hu, "A highly concurrent algorithm and pipelined architecture for solving Toeplitz systems", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 31, February 1983, pp. 66–76.
[13] L. Ljung and T. Söderström, *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA, 1982.
[14] G.K. Ma and F.J. Taylor, "Multiplier policies for digital signal processing", *IEEE ASSP Mag.*, January 1990, pp. 6–20.
[15] S.L. Marple, *Digital Spectral Analysis with Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1987.
[16] D.I. Moldovan and J. Fortes, "Partitioning and mapping algorithms onto fixed size arrays", *IEEE Trans. Comput.*, Vol. 35, No. 1, January 1986, pp. 1–12.
[17] M. Morf, B. Dickinson, T. Kailath and A. Viera, "Efficient solution of covariance equations for linear prediction", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 25, 1977, pp. 429–433.
[18] J.G. Proakis, *Digital Communications*, McGraw-Hill, New York, 1983.
[19] E. Robinson, *Multichannel Time Series Analysis with Digital Computer Programs*, Holden-Day, San Francisco, CA, 1967.
[20] R. Sharma et al., A 6.75-ns 16×16-bit multiplier in single-level-metal CMOS technology", *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, August 1989, pp. 922–927.
[21] P. Strobach, "Recursive triangular array ladder algorithms", *IEEE Trans. Signal Process.*, Vol. 39, January 1991, pp. 122–136.
[22] S. Treiter, "Principles of digital multichannel filtering", *Geophysics*, Vol. 35, No. 5, 1970, pp. 785–811.
[23] R. Wiggins and E.A. Robinson, 'Recursive solution to multichannel filtering problem", *J. Geophys. Res.*, Vol. 70, 1965, pp. 1885–1891.
[24] Xiao-Hu Yu and Zhen-Ya He, "Efficient block implementation of exact sequential least squares problems", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 36, 1988, pp. 392–399.