A Highly Concurrent Algorithm and Pipelined Architecture for Solving Toeplitz Systems

SUN-YUAN KUNG, MEMBER, IEEE, AND YU HEN HU, STUDENT MEMBER, IEEE

Abstract—The design of VLSI parallel processors requires a fundamental understanding of the parallel computing algorithm and an appreciation of the implementational constraint on communications. Based on such consideration, this paper develops a highly concurrent Toeplitz system solver, featuring maximum parallelism and localized communication. More precisely, a highly parallel algorithm is proposed which achieves $O(N)$ computing time with a linear array of $O(N)$ processors. This compares very favorably to the $O(N \log_2 N)$ computing time attainable with the traditional Levinson algorithm implemented in parallel. Furthermore, to comply with the communication constraint, a pipelined processor architecture is proposed which uses only localized interconnections and yet retains the maximum parallelism attainable.

I. Introduction

With rapidly growing microelectronics technology leading the way, modern signal processor architectures are undergoing a major revolution. The availability of low cost, fast VLSI (very large scale integration) devices promises the practice of cost-effective, high-speed parallel processing of large volumes of data. This will make possible an ultra-high throughput rate and, therefore, indicates a major technological breakthrough for real-time signal processing applications. On the other hand, it has become more critical than ever to have a fundamental understanding of the algorithm structure, architecture, and implementation constraints in order to realize the full potential of VLSI computing power [1]. In this paper, the two most critical issues—the parallel computing algorithm and the VLSI architectural constraint—will be considered.

Traditionally, computational complexity is measured in terms of number of the arithmetic operations required in an algorithm. With the emergence of inexpensive (VLSI) parallel computing capability, this criterion will soon become obsolete since the most efficient algorithms for sequential machines are not necessarily the most efficient for parallel machines. Taking into account the parallelism, an up to date and more practical criterion appears to be the processing throughput rate attainable in parallel computation [2], [3]. Therefore, in formulating a parallel algorithm, the first important question should be: *How can we structure the algorithm to achieve the maximum parallelism and, therefore, the maximum throughput rate?*

Communication constraint represents another fundamental impact of VLSI device technology on the architecture design and implementation consideration. In VLSI systems, communication tends to be very restrictive as it accounts for most time, area, and energy consumption. Therefore, architectures which require only localized communication, such as a *systolic array* [4] and a *wavefront array processor* [5]-[7], become very attractive for VLSI implementation. Thus, the second important question should be: *How can we cope with the communication constraint so as to compromise least in processing throughput rate?*

Obviously, the above two questions are mutually dependent, and their answers often affect each other iteratively throughout the design process. Therefore, they deserve an integrated solution which should provide general guidelines for designing VLSI parallel processing architectures. The motivation of this paper, therefore, is to demonstrate a design methodology following these general guidelines. Nevertheless, we shall limit ourselves to the specific task of solving Toeplitz systems [8] which, we consider, is one of the most important signal processing problems.

A Toeplitz system is a set of linear system equations

$$(Tx = y) \tag{1.1}$$

with T being a Toeplitz matrix, i.e., the (i, j)th element $t_{ij} = t_{i-j} = t_k$, $-N \leq k \leq N$. (Throughout this paper, we assume that T is an $(N+1) \times (N+1)$ real matrix.) This system arises in numerous widespread applications ranging from speech, image, and neurophysics to radar, sonar, geophysics, and astronomical signal processing [9]-[11]. The contribution of this paper lies in the development of a highly concurrent algorithm and pipelined architecture which is able to solve a Toeplitz system in $O(N)$ processing time in an array processor as opposed to $O(N^3)$ for the general (sequential) Gauss elimination procedure or $O(N^2)$ for the (sequential) Levinson algorithm (cf. Section II). In addition, the design methodology demonstrated in this paper should also help answer some fundamental problems faced in designing VLSI parallel processor architectures.

The organization of this paper is as follows. In Section II, a conventional (Levinson) algorithm for solving a Toeplitz system and its inherent limitation for parallel processing are examined. In Section III, a highly concurrent algorithm is developed. In Section IV, to naturally map this algorithm onto a parallel computing system, a lattice-connected processors array is proposed. The complete Toeplitz system solver is discussed in Section V, and lastly, some system applications, implementations, and multichannel extensions are included in Section VI.

II. Preliminary Review

It is known that the conventional approach (e.g., Gauss elimination procedure) for solving a linear system takes $O(N^3)$ arithmetic operations with each operation containing one mul-

Manuscript received March 5, 1982; revised August 3, 1982. This work was supported in part by the Office of Naval Research under Contract N00014-80-C-0457, N00014-81-K-0191 and by the National Science Foundation under Grant ECS-80-16581.

The authors are with the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089.

0096-3518/83/0200-0066$01.00 © 1983 IEEE

tiplication and one addition. By making use of the Toeplitz structure, several fast algorithms are now available to solve a Toeplitz system in $O(N^2)$ operations or even less [12]-[17]. Among them, the most popular is the Levinson algorithm [12].

The Levinson algorithm was originally proposed by Norman Levinson in 1947, and since then, a number of variants have been proposed [18]-[23]. Basically, the Levinson algorithm recursively solves for the (kth-order) solution $\{\boldsymbol{a}_k, \boldsymbol{b}_k\}$ in the equation depicted below:

$$T_k \begin{bmatrix} 1 & b_{kk} \\ a_{1k} & \vdots \\ \vdots & b_{1k} \\ a_{kk} & 1 \end{bmatrix} = \begin{bmatrix} E_k & 0 \\ 0 & \vdots \\ \vdots & 0 \\ 0 & E_k \end{bmatrix} = \begin{bmatrix} E_k & \mathbf{0}_k \\ \mathbf{0}_k & E \end{bmatrix} \qquad (2.1)$$

$\boldsymbol{a}_k \quad \boldsymbol{b}_k$

where T_k denotes the $(k+1) \times (k+1)$ leading principal minor[1] of T and E_k is some nonzero number. The recursive procedure efficiently utilizes $\{\boldsymbol{a}_k, \boldsymbol{b}_k\}$ to derive the solution for the $(k+1)$th-order equation. Then, by induction, the Nth-order equation can be solved. This recursive procedure can be explained in three steps.

Step 1: Let

$$T_{k+1} \begin{bmatrix} \boldsymbol{a}_k & 0 \\ 0 & \boldsymbol{b}_k \end{bmatrix} = \begin{bmatrix} E_k & S_k \\ \mathbf{0}_k & \mathbf{0}_k \\ Q_k & E_k \end{bmatrix} \qquad (2.2)$$

where Q_k and S_k are obtained via *inner product operation*:

$$Q_k = [t_{k+1}, \cdots, t_1]\, \boldsymbol{a}_k \qquad (2.3a)$$

$$S_k = [t_{-k-1}, \cdots, t_{-1}]\, \boldsymbol{b}_k. \qquad (2.3b)$$

Step 2: $\{\boldsymbol{a}_{k+1}, \boldsymbol{b}_{k+1}\}$ are derived as

$$[\boldsymbol{a}_{k+1} \quad \boldsymbol{b}_{k+1}] = \begin{bmatrix} \boldsymbol{a}_k & 0 \\ 0 & \boldsymbol{b}_k \end{bmatrix} \begin{bmatrix} 1 & Kr^{(k+1)} \\ Ke^{(k+1)} & 1 \end{bmatrix} \qquad (2.4)$$

where[2]

$$Kr^{(k+1)} = -S_k/E_k \quad \text{and} \quad Ke^{(k+1)} = -Q_k/E_k. \qquad (2.5)$$

Step 3: Consequently, we have

$$T_{k+1}[\boldsymbol{a}_{k+1} \quad \boldsymbol{b}_{k+1}] = T_{k+1} \begin{bmatrix} E_{k+1} & \mathbf{0}_k \\ \mathbf{0}_k & E_{k+1} \end{bmatrix} \qquad (2.6)$$

where

$$E_{k+1} = E_k + Ke^{(k+1)} S_k = E_k + Kr^{(k+1)} Q_k$$
$$= E_k(1 - Ke^{(k+1)} Kr^{(k+1)}) \qquad (2.7)$$

and we are ready for the next recursion.

The computation should stop at the end of the Nth recursion. A simple calculation shows that the total number of operations required is approximately $2N^2$.

[1] Throughout this paper, we shall assume that each leading principle minor of the matrix T is nonsingular.

[2] In the literature, $\{Ke, Kr\}$ are called "reflection coefficients"; see, e.g., [13].

In the case of a symmetric Toeplitz system, by symmetry, we have $Ke^{(i)} = Kr^{(i)}$ and that $a_{ik} = b_{ik}$ [cf. (2.1)], and therefore the number of operations can be reduced to one half, that is, N^2.

To explicitly solve the Toeplitz system, we note that the $\{\boldsymbol{a}_k, \boldsymbol{b}_k\}$ vectors produced in the Levinson algorithm constitute a *UDL* decomposition of the T^{-1} matrix, namely,

$$T^{-1} = \begin{bmatrix} 1 & b_{11} & \cdots & b_{NN} \\ 0 & 1 & & \\ & & \ddots & \\ & & & 1 \quad b_{1N} \\ 0 & & 0 & 1 \end{bmatrix} \begin{bmatrix} E_0 & & & 0 \\ & E_1 & & \\ & & \ddots & \\ 0 & & & E_N \end{bmatrix}^{-1}$$
$$\cdot \begin{bmatrix} 1 & 0 & & 0 \\ a_{11} & 1 & & \\ & & \ddots & \\ & & & 0 \\ a_{NN} & \cdots & a_{1N} & 1 \end{bmatrix} \qquad (2.8)$$
$$\triangleq U_B D_E L_A.$$

Therefore, the solution of the Toeplitz system can be computed as $\boldsymbol{x} = U_B D_E L_A \boldsymbol{y}$.

In order to meet the extremely high throughput rate requirement in many real-time signal processing applications, the potential of parallel computing has to be utilized. However, for parallel execution of the Levinson algorithm on a linear processor array with N processing elements, the parallelism will be severely hampered by the presence of the inner product operations (2.3). More precisely, in each recursion step, the inner product operation will be the bottleneck of the computation, since it requires a minimum execution time of $\log_2 k$ units [3]. Consequently, to compute all the N recursions, the total parallel computing time amounts to $O(N \log_2 N)$ on a linear processor array. This not only unnecessarily slows down the parallel processing speed, but also accounts for considerable waste of processors. In the next section, we shall develop an algorithm which avoids the inner product operations, and therefore achieves much higher parallelism. More precisely, this algorithm will attain a processing time of $O(N)$ time units [as opposed to $O(N \log_2 N)$] on a vector array of $O(N)$ processing elements.

III. A Highly Concurrent Algorithm

In this section, we shall present a highly concurrent algorithm. Mathematically, this algorithm can find its roots back to the now classical Schur's algorithm [24] as first pointed out by Dewilde *et al.* [25]. The matrix formulation used in the algorithm bears a strong similarity with an earlier work by Bareiss [26] and later, in a different fashion, by Rissanen [27] and Morf [16]. However, our derivation is independent of the previous works. Moreover, in our formulation, 1) the parallelism and the (localized) date dependency inherent in the algorithm are explicitly exposed, and 2) there surfaces a natural topological mapping from the mathematical algorithm to the computing structure to be discussed in the next section. Therefore, our formulation is included below for the purpose of a clearer and easier presentation.

A. Main Algorithm

The function of the proposed algorithm is to perform a triangular decomposition on matrix T, that is,[3]

$$T = \bar{L}D\bar{U} = \bar{L}U (U = D\bar{U}) \tag{3.1}$$

where D is a diagonal matrix. Then the solution $\boldsymbol{x}$ of (1.1) can be solved explicitly with back substitution:

$$\boldsymbol{x} = T^{-1}\boldsymbol{y} = U^{-1}\bar{L}^{-1}\boldsymbol{y}. \tag{3.2}$$

To set up for a new recursive procedure, we consider an *augmented* Toeplitz matrix of T, say $\tilde{T}$, which is a natural extension of T as illustrated by the (4 × 4) example below:

$$\tilde{T} = \left[\begin{array}{cccc:ccc} t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} \\ 0 & t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} \\ 0 & 0 & t_3 & t_2 & t_1 & t_0 & t_{-1} \\ 0 & 0 & 0 & \underbrace{t_3 \quad t_2 \quad t_1 \quad t_0}_{T} & & & \end{array}\right]$$

Substituting T by $\tilde{T}$ in (3.3), we have

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix}}_{\tilde{L}} \tilde{T} = \left[\begin{array}{ccc:ccc} X & X & X & u_{11} & u_{12} & u_{13} & u_{14} \\ X & X & X & 0 & u_{22} & u_{23} & u_{24} \\ X & X & X & 0 & 0 & u_{33} & u_{34} \\ X & X & X & 0 & 0 & 0 & u_{44} \end{array}\right] \tag{3.4}$$

where $\tilde{L} \triangleq \bar{L}^{-1}$ and the X's denote DON'T CARE entries. Our strategy is then to construct the matrices $\tilde{L}$ and U by creating all the "zeros" below the diagonal in the U matrix. Let us consider the following equation:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tilde{T} = \left[\begin{array}{ccc:ccc} t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} & t_{-3} \\ 0 & t_3 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} \end{array}\right]. \tag{3.5}$$

Now perform row operations on both sides of (3.5) such that

$$\begin{bmatrix} 1 & Kr^{(2)} \\ Ke^{(2)} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tilde{T} = \begin{bmatrix} 1 & Kr^{(2)} & 0 & 0 \\ Ke^{(2)} & 1 & 0 & 0 \end{bmatrix} \tilde{T}$$

$$= \begin{bmatrix} \boldsymbol{v}^{(2)} \\ \boldsymbol{u}^{(2)} \end{bmatrix} = \left[\begin{array}{cccc:ccc} v_3^{(2)} & v_2^{(2)} & v_1^{(2)} & v_0^{(2)} & 0 & v_{-2}^{(2)} & v_{-3}^{(2)} \\ u_3^{(2)} & u_2^{(2)} & u_1^{(2)} & 0 & u_{-1}^{(2)} & u_{-2}^{(2)} & u_{-3}^{(2)} \end{array}\right] \tag{3.6}$$

where[4]

$$Ke^{(2)} = -t_1/t_0; \quad Kr^{(2)} = t_{-1}/t_0. \tag{3.7}$$

Compare the second row of the RHS (right-hand side) of (3.4) to that of (3.6), i.e., $\boldsymbol{u}^{(2)}$; it is clear that a zero is created by the row operation and the desired second rows of $\tilde{L}$ and U are obtained:

$$\tilde{L} = [1_{21} \quad 1 \quad 0 \quad 0] = [Ke^{(2)} \quad 1 \quad 0 \quad 0]$$

$$U_2 = [0 \quad u_{22} \quad u_{23} \quad u_{24}] = [0 \quad u_{-1}^{(2)} \quad u_{-2}^{(2)} \quad u_{-3}^{(2)}].$$

To compute the third rows of the $\tilde{L}$ and U matrices, the same strategy can be used. For this purpose, we first *right-shift* $[Ke^{(2)} \; 1 \; 0 \; 0]$ such that $[Ke^{(2)} \; 1 \; 0 \; 0] \rightarrow [0 \; Ke^{(2)} \; 1 \; 0]$. By the Toeplitz structure of the $\tilde{T}$ matrix, $\boldsymbol{u}^{(2)}$ in (3.6) will also be right-shifted accordingly. Thus, we have the following equation:

$$\begin{bmatrix} 1 & Kr^{(2)} & 0 & 0 \\ 0 & Ke^{(2)} & 1 & 0 \end{bmatrix} \tilde{T}$$

$$= \left[\begin{array}{cccc:ccc} v_3^{(2)} & v_2^{(2)} & v_1^{(2)} & v_0^{(2)} & 0 & v_{-2}^{(2)} & v_{-3}^{(2)} \\ 0 & u_3^{(2)} & u_2^{(2)} & u_1^{(2)} & 0 & u_{-1}^{(2)} & u_{-2}^{(2)} \end{array}\right]. \tag{3.8}$$

Note that through this shift operation, the two 0's created in the previous recursion on the RHS are realigned into the same column so as to remain uneffected by the row operations in the next recursion. With this precaution, a similar procedure as in the previous recursion can be repeated:

$$\begin{bmatrix} 1 & Kr^{(3)} \\ Ke^{(3)} & 1 \end{bmatrix} \begin{bmatrix} 1 & Kr^{(2)} & 0 & 0 \\ 0 & Ke^{(2)} & 1 & 0 \end{bmatrix} \tilde{T}$$

$$= \begin{bmatrix} 1 & X & Kr^{(3)} & 0 \\ Ke^{(3)} & X & 1 & 0 \end{bmatrix} \tilde{T} \tag{3.9a}$$

$$= \begin{bmatrix} \boldsymbol{v}^{(3)} \\ \boldsymbol{u}^{(3)} \end{bmatrix} = \left[\begin{array}{cccc:ccc} v_3^{(3)} & v_2^{(3)} & v_1^{(3)} & v_0^{(3)} & 0 & 0 & v_{-3}^{(3)} \\ u_3^{(3)} & u_2^{(3)} & u_1^{(3)} & 0 & 0 & u_{-2}^{(3)} & u_{-3}^{(3)} \end{array}\right] \tag{3.9b}$$

where

$$Ke^{(3)} = -u_1^{(2)}/v_0^{(2)} \quad \text{and} \quad Kr^{(3)} = -v_{-1}^{(2)}/u_{-2}^{(2)}. \tag{3.10}$$

By comparing $\boldsymbol{u}^{(3)}$ to the third row on the RHS of (3.4), clearly, the third rows of the $\tilde{L}$ and U matrices are obtained:

$$\tilde{L}_3 = [l_{31} \quad l_{32} \quad 1 \quad 0] = [Ke^{(3)} \quad (Ke^{(3)}Kr^{(2)} + Ke^{(2)}) \quad 1 \quad 0]$$

$$U_3 = [0 \quad 0 \quad u_{33} \quad u_{34}] = [0 \quad 0 \quad u_{-2}^{(3)} \quad u_{-3}^{(3)}].$$

This completes the second recursion. By induction, future recursions can be carried out in the same manner until all the rows of the $\tilde{L}$ and U matrices are computed. Summarizing the above procedure, several observations can be made.

1) *The shift operation in each recursion is natural due to the Toeplitz structure of the $\tilde{T}$ matrix since it retains the zeros produced in the previous recursion.* The purpose of the shift is to realign these zeros with those of the auxiliary vector $\boldsymbol{v}$ [cf. (3.8)] such that these zeros will remain unaffected by the upcoming row operations. This also explains the purpose and the necessity of computing for the auxiliary vectors $\boldsymbol{v}$ in each recursion.

2) Note that $\tilde{L}$ is nothing but the L_A matrix in (2.8) since from (2.8) we have

[3] The overbar "‾" of the triangular matrix L (or U) indicates that L (or U) has 1's along its diagonal.

[4] These $\{Ke, Kr\}$ play the same role as those in the Levinson algorithm, and therefore will also be called *reflection coefficients* in the following discussion.

$$T = L_A^{-1} D_E^{-1} U_B^{-1}.$$

Comparing the above equation to (3.1), and noting that the LDU factorization of a given matrix is unique,[5] we have $\bar{L} = L_A^{-1}$, $\bar{U} = U_B^{-1}$, and $D = D_E^{-1} = \text{diag}\,[E_0, \cdots, E_N]$. In this sense, the algorithm proposed can be regarded as a generalization of the conventional Levinson algorithm. More precisely, the $\{a_k, b_k\}$ vectors are actually embedded in the $2 \times N$ matrix on the LHS in the kth recursion. For example, in the first recursion [cf. (3.6)], we have $a_{11} = Ke^{(2)}$, $b_{11} = Kr^{(2)}$. In the second recursion [cf. (3.9a)], we have

$$\begin{bmatrix} 1 & Kr^{(3)} \\ Ke^{(3)} & 1 \end{bmatrix} \begin{bmatrix} 1 & a_{11} & 0 & 0 \\ 0 & b_{11} & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & a_{12} & a_{22} & 0 \\ b_{22} & b_{12} & 1 & 0 \end{bmatrix}. \tag{3.11}$$

3) This new formula *completely avoids the need of inner product operations;* therefore, the bottleneck incurred in parallel execution of the Levinson algorithm no longer exists. Furthermore, the reflection coefficient computation and row operations in the formulation are very suitable for parallel execution; hence, this new algorithm is inherently highly concurrent. As a consequence, it has the following parallel formulation.

Main Algorithm

INITIAL CONDITIONS:

$$v_k^{(1)} = u_k^{(1)} = t_k \qquad (-N \leq k \leq N) \tag{3.12a}$$

FOR $i = 1$ UNTIL N DO BEGIN

IN PARALLEL DO BEGIN[6]

$$Ke^{(i+1)} = -u_1^{(i)} [v_0^{(i)}]^{-1} \tag{3.12b}$$

$$Kr^{(i+1)} = -v_{-i}^{(i)} [u_{-i+1}^{(i)}]^{-1} \tag{3.12c}$$

END IN PARALLEL DO;

IN PARALLEL FOR $-N \leq K \leq N$ DO BEGIN

$$v_k^{(i+1)} = v_k^{(i)} + Kr^{(i+1)} u_{k+1}^{(i)} \tag{3.12d}$$

$$u_k^{(i+1)} = u_{k+1}^{(i)} + Ke^{(i+1)} v_k^{(i)} \tag{3.12e}$$

END IN PARALLEL DO;

OUTPUT;

END FOR LOOP;

4) Based on the above formulation, it is clear that *with $O(N)$ processing elements connected in a linear array, the parallel computing time for each recursion can take as little as two time units* (one for reflection coefficient computation, the other for row operation). *For N recursions, this amounts to a total parallel processing time of $O(N)$ time units as opposed to $O(N \log_2 N)$ in the Levinson algorithm.*

B. Duality of the Main Algorithm

1) *Duality:* In order to execute the back substitution step (3.2) to solve a Toeplitz system, both the $\tilde{L}$ and U matrices have to be computed. However, the equation for computing the $\tilde{L}$ $(= \bar{L}^{-1})$ matrix is purposely left out in the above main algorithm. *This is because each column of the L $(= \bar{L}D)$ matrix is also provided by the vector v while each row of the U matrix is provided by the vector u.* More precisely, it is proved in Appendix A that

$$T = \begin{bmatrix} v_0^{(1)} & 0 & \cdots & 0 \\ v_1^{(1)} & v_0^{(2)} & & \\ \vdots & \vdots & \ddots & \\ v_N^{(1)} & v_{N-1}^{(2)} & \cdots & v_0^{(N+1)} \end{bmatrix} \begin{bmatrix} E_0^{-1} & & & 0 \\ & E_1^{-1} & & \\ & & \ddots & \\ 0 & & & E_N^{-1} \end{bmatrix} \begin{bmatrix} u_0^{(1)} & u_{-1}^{(1)} & \cdots & u_{-N}^{(1)} \\ 0 & u_{-1}^{(2)} & \cdots & u_{-N}^{(2)} \\ & & \ddots & \vdots \\ 0 & & 0 & u_{-N}^{(N+1)} \end{bmatrix} \tag{3.13a}$$

$$= LD^{-1}U. \tag{3.13b}$$

Consequently, (3.12) alone is sufficient for $\bar{L}D\bar{U}$ factorization of T matrix and the computation of the $\tilde{L}$ matrix can be saved (see also [28]).

2) Symmetric Toeplitz Systems: Additional computational savings are possible in solving a *symmetric* Toeplitz system. In this case, $L = U$, and hence [from (3.12)] $Ke^{(i)} = Kr^{(i)}$ for each i. Consequently, the computations of $v_k^{(i)}$ and $u_k^{(i)}$ for $k > 0$ become redundant and may be omitted. *This leads to a savings of one half of the computations as compared to the nonsymmetrical case.*

In practice, symmetric Toeplitz systems arise much more often than nonsymmetric ones, and are therefore of much greater importance from an applicational point of view. In the next section, a pipelined computing structure for concurrent processing of symmetric Toeplitz systems solutions will be discussed.

IV. Pipelined Lattice Processor

A. Parallel Lattice Computing Structure

In this section, we consider the implementation of the parallel algorithm on a VLSI chip. The major computation in the algorithm lies in the linear combinations of two vectors in each recursion. Therefore, a parallel computing structure with a linear processor array as depicted in Fig. 1 is proposed.

The configuration consists of a series of modular processing cells, termed *lattice cells*, to perform row operations. Each cell is composed of an upper and a lower processing element (PE) with lattice connections. The only exception is the upper PE in the ⟨0⟩ cell which is a divider cell for the computation of reflection coefficients.

During each recursion, the reflection coefficient is first computed in the divider cell and then broadcast to all the lattice cells through the global (horizontal) interconnections. Then the row operations are performed simultaneously in all the lattice cells. Upon completion, the result in each upper PE is left-shifted to its immediate left neighbor, preparing for the next recursion.[7] Meanwhile, the contents of the lower PE's,

[5] This is true provided that all the leading principal minors of the given matrix are nonsingular.

[6] IN PARALLEL DO indicates that the operations within this block can be executed in parallel.

[7] This corresponds to the shift operation discussed in Section III. Note that since only the relative position between the data in the upper and lower PE's is of importance, the left-shift for the upper PE's (v vector) is equivalent to the right-shift for the lower PE's (u vector), as described earlier.

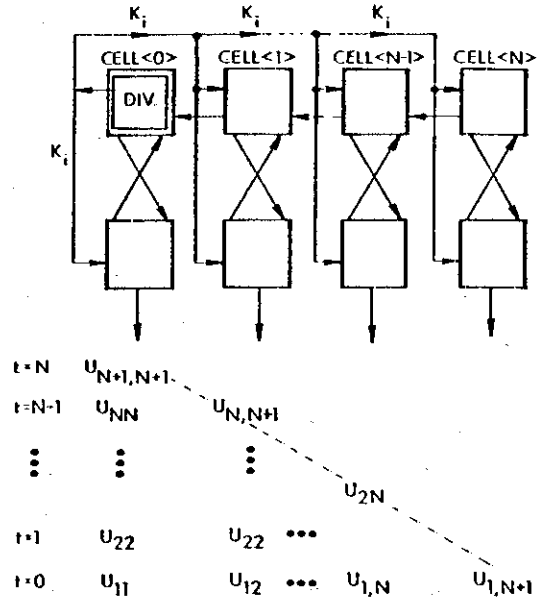

Fig. 1. Parallel lattice computing structure.

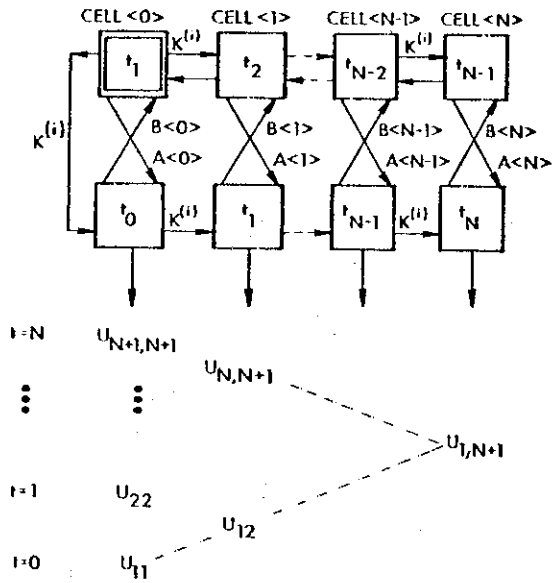

Fig. 2. Pipelined lattice computing structure.

which correspond to a row of the U matrix, can be output, and the recursion is thus completed.

The operation is completed after N such recursions. Let τ_1 denote the time interval needed for division, and let τ_2 denote the time interval for each lattice operation (multiplication and addition); then the total computing time will be $N(\tau_1 + \tau_2)$.

B. Pipelined Lattice Computing Structure

To accomplish maximal parallelism, the parallel lattice computing structure just proposed relies heavily on global communication. As mentioned earlier, this may cause certain difficulties on, for example, synchronization, longer delay, larger power, and chip area consumption in a VLSI system. Therefore, in the following, we shall propose a modified version of the lattice computing structure which eliminates the need for global communication without compromising the parallelism.

The general configuration (Fig. 2) of the modified lattice computing structure remains largely resembling the one in Fig. 1, except that the global communication links are replaced by nearest neighbor interconnections. To achieve maximal parallelism in this locally connected computing network, we must resort to a pipelined operation which renders efficient and smooth data flow. This leads to a useful notion of computational wavefront [5], [6], [29]-[31].

C. Computational Wavefront

Roughly speaking, a computational wavefront in a computing structure corresponds to the computational activity incurred in one recursion step in a recursive (parallel) algorithm. As an example, the computational wavefront of the first recursion is examined below.

Suppose that the data are initially placed in the registers of the PE's such that $B_{\langle 0\rangle} = t_0$ and $A_{\langle m-1\rangle} = B_{\langle m\rangle} = t_m$ for $m = 1, 2, \cdots, N$ where the $A_{\langle m\rangle}$ register is in the mth upper PE and $B_{\langle m\rangle}$ in the mth lower PE (cf. Fig. 2). The process starts with the $\langle 0\rangle$ cell (divider cell) where the reflection coefficient is computed and stored in register $C_{\langle 0\rangle}$:

$$C_{\langle 0\rangle} \leftarrow A_{\langle 0\rangle}/B_{\langle 0\rangle}. \tag{4.1}$$

The computation activity then propagates to the $\langle 1\rangle$ cell (after propagating $C_{\langle 0\rangle}$ to $C_{\langle 1\rangle}$) where the following computations are executed simultaneously in upper and lower PE's:[8]

$$A_{\langle 1\rangle} \leftarrow A_{\langle 1\rangle} - C_{\langle 1\rangle} \times B_{\langle 1\rangle}$$

(in the upper PE of the $\langle 1\rangle$ cell) (4.2a)

$$B_{\langle 1\rangle} \leftarrow B_{\langle 1\rangle} - C_{\langle 1\rangle} \times A_{\langle 1\rangle}$$

(in the lower PE of the $\langle 1\rangle$ cell). (4.2b)

Upon completion of the execution, the $\langle 1\rangle$ cell propagates its new content in the upper PE, $A_{\langle 1\rangle}$, to its left neighbor, $A_{\langle 0\rangle}$, to prepare for the next recursion. Meanwhile, it also sends the content of $C_{\langle 1\rangle}$ to $C_{\langle 2\rangle}$ so that the computation activity continues propagating to the $\langle 2\rangle$ cell. The next front of activity will be at the $\langle 3\rangle$ cell, then the $\langle 4\rangle$ cell, and so on. As a consequence, a computational wavefront is created traveling across the linear processor array. (It may be noted that the wave propagation implies localized data flow.) Once the wavefront sweeps through all the cells, the first recursion is completed. The content in $B_{\langle k\rangle}$ ($k = 0, 1, \cdots, N-1$) is output downward as soon as it is available.

As the first computational wavefront propagates, the second recursion can be executed concurrently *by pipelining a second wavefront as soon as the content in $A_{\langle 1\rangle}$ is sent to the $A_{\langle 0\rangle}$ register in the $\langle 0\rangle$ cell.* Therefore, the time interval between the first and second wavefronts is estimated to be $\tau_1 + \tau_2$. (Note that it takes a $\tau_1 + \tau_2$ time interval before $A_{\langle 1\rangle}$ is made available if data transfer time is considered negligible.) The second wavefront strongly resembles that of the first one. For example, the $\langle 0\rangle$ cell first computes the second reflection coefficient according to (4.1), then forward $C_{\langle 0\rangle}$ to the $\langle 1\rangle$ cell where (4.2) will be performed. After this, the second wavefront will propagate to cells $\langle 2\rangle$, $\langle 3\rangle$, and so on. Once the wavefront arrives and executes at the $(N-1)$th cell, the second recursion is completed. Again, the content in $B_{\langle k\rangle}$, which is nothing but $u^{(3)}_{-k-2}$ ($k = 0, 1, \cdots, N-2$), is output downward as soon as it is available.

[8] The lower PE of the $\langle 0\rangle$ cell should also perform the operation $B_{\langle 0\rangle} \leftarrow B_{\langle 0\rangle} - C_{\langle 0\rangle} X A_{\langle 0\rangle}$ at this moment. This accounts for the compu-

The same pipelining scheme can be repeated for the third, and eventually all the recursions. When all the N wavefronts are generated and operations are executed, the parallel algorithm is completed. Since the wavefronts are generated consecutively at a rate of one wavefront per $\tau_1 + \tau_2$ time interval, the total computing time will be $N(\tau_1 + \tau_2)$.[9] Therefore, the pipelined operation takes the processing time $N(\tau_1 + \tau_2)$, which is the same as that of the parallel operation with global communication. From now on, the proposed pipelined computing structure will be called the pipelined lattice processor (PLP). We note that the PLP has accomplished the design goal of using only local interconnections and yet not sacrificing any degree of parallelism attainable.

D. Programming Aspects

In the above, we have successfully exploited the notion of the computational wavefront to portray the pipelined operations on the lattice array. However, in general, it is even more preferable to have an appropriate language to program pipelined algorithms.

In a recent paper [5], [6], it is noted that the wavefront notion has indeed much more widespread applications. Namely, it is applicable to a large class of parallel matrix algorithms, especially to those with the so-called *locality* and *recursivity* nature. Therefore, this same notion has been further extended to a wavefront-oriented language suitable for programming pipelined algorithms on programmable (pipelined) array processors [5], [6], [29]–[31]. Although further elaboration on the structure of the language is beyond the scope of this paper, for illustration purpose, we have included in Appendix B one such programming example for the above pipelined algorithm.

The wavefront language does not only facilitate programming most pipelined algorithms, but also offers a convenient tool for the subsequent simulation and verification tasks. The simulation results of the PLP yield a series of snapshots of the computation wavefronts in the PLP which match with the result theoretically predicted [31].

V. Complete Toeplitz System Solvers

So far, we have used the PLP to decompose a Toeplitz system into lower and upper triangular matrices, i.e., $T = U^t D^{-1} U$ in $O(N)$ time units (assume that T is symmetric). To completely solve the Toeplitz system, however, an explicit solution $\boldsymbol{x}$ has to be derived. Therefore, subsequent operations are needed and should also be performed in $O(N)$ time units. Based on different inversion formulas for the Toeplitz matrix T, we shall present three different organizations—corresponding to three different methods for the subsequent computations—of a complete Toeplitz system solver. All of these utilize linear processor array and achieve $O(N)$ units of processing time.

A. Back-Substitution Method

This method involves computing $\boldsymbol{x}$ via (3.2) (for the symmetric case):

$$\boldsymbol{x} = T^{-1}\boldsymbol{y} = U^{-1}D(U^t)^{-1}\boldsymbol{y} \tag{5.1}$$

which can be separated in two back-substitution steps:

$$\boldsymbol{g} = D(U^t)^{-1}\boldsymbol{y} \tag{5.2a}$$

and

$$\boldsymbol{x} = U^{-1}\boldsymbol{g}. \tag{5.2b}$$

Back substitution is a standard matrix operation for solving linear systems, which also enjoys a pipelined operation. Therefore, it can be implemented with a locally connected linear processor array. As it is a rather well-known procedure [4], [7], the detail is omitted here.

A complete Toeplitz system solver depicted in Fig. 3 is constituted by a PLP and a (pipelined) back-substitution processor. The processor in the upper part is the PLP where the outputs are fed into the lower part—the back-substitution processor. Upon receiving the data, the back-substitution processor (initially stores the $\boldsymbol{y}$ vector) will perform the first back substitution (i.e., $\boldsymbol{g} = D(U^t)^{-1}\boldsymbol{y}$). The elements of the U^t matrix (output from PLP) are stored in the LIFO (last-in-first-out) memory stack which serves as a matrix transposer. The scaling operator $D = \text{diag}[u_{00}, \cdots, u_{NN}]$ will operate on $(U^t)^{-1}\boldsymbol{y}$ to obtain the $\boldsymbol{g}$ vector which then is stored in the G-LIFO stack. After the first back substitution, the second step (5.2b) starts immediately with the U matrix and $\boldsymbol{g}$ vector fetched from the memory stacks. Finally, the output $\boldsymbol{x}$ is obtained from the left end of the back-substitution processor. Note that the first back substitution can be executed concurrently with the LU decomposition in order to save processing time.

B. A Method Based on LU Decomposition of T^{-1}

Recall that in Section II, the $\{\boldsymbol{a}_k, \boldsymbol{b}_k\}$ vectors computed from the Levinson algorithm constitute a *UDL* decomposition of the T^{-1} matrix, and thus facilitate an explicit solution for the Toeplitz system (2.8). Now, this formula can be utilized to provide a different organization of the Toeplitz system solver. In doing so, we have to compute the $\{\boldsymbol{a}_k, \boldsymbol{b}_k\}$ vectors. However, in Section III-A, we note that the Levinson procedure is inherently imbedded in the new algorithm, and that the computation of $\{\boldsymbol{a}_k, \boldsymbol{b}_k\}$ requires virtually the same kind of row operations [see, e.g., (3.11)]. Therefore, the PLP processor can also be utilized to produce these vectors. For this, a duplicate PLP (without a divider cell) can be attached to the original one. The new PLP will have different initial conditions: all of its PE's are stored with "0," except the ⟨1⟩ cell where "1" is stored. During execution, the new PLP is triggered by the reflection coefficients sending from the original PLP for performing lattice operations. Then the results, $\boldsymbol{a}_k$ vectors [cf. (3.11)] in the lower PE's, will be output to a linear processor array for matrix-vector multiplication: $\boldsymbol{x} = L_A^t D L_A \boldsymbol{y}$. This is obtained from (2.8) with $U_B = L_A^t$ (i.e., the symmetric case). Details of this multiplication operation are omitted here since they are again accessible in the literature [1], [4], [7].

Cybenko [32] has shown that the Levinson algorithm (in the sequential computation scheme) is numerically comparable to the Cholesky factorization method which is known to be numerically stable. Since (2.8) is a formula based on the Levinson algorithm, the numerical stability of the above method in the pipelined computation scheme is expected.

[9] Note that the Nth recursion will be initiated at the time $(N-1)(\tau_1 + \tau_2)$, and by that time there will be only one computation [i.e., (4.1)] to be performed in the last recursion (cf. Section III).

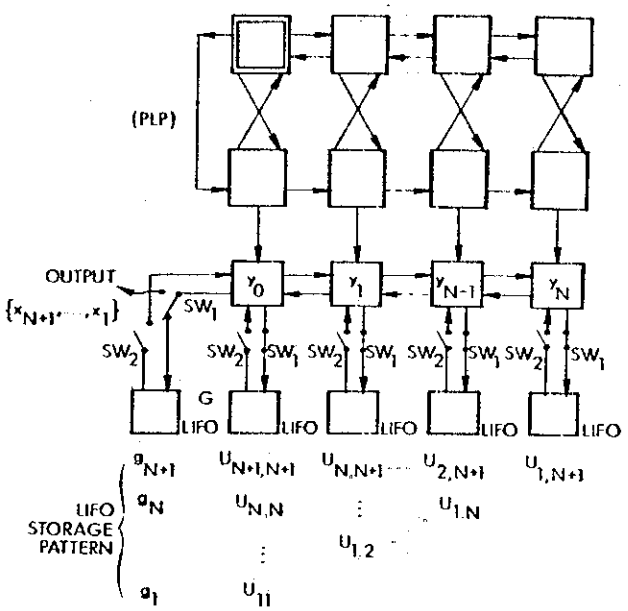

Fig. 3. Complete Toeplitz system solver.

C. A Different Organization Based on Gohberg's Formula

In addition to the *UDL* decomposition, another Toeplitz inversion formula proposed by Gohberg [33] can also be applied for an alternative organization. This formula is based only on the knowledge of $\{a_N, b_N\}$ and E_N:

$$T^{-1} = \frac{1}{E_N}\begin{bmatrix} 1 & & & \\ a_{1N} & 1 & & 0 \\ \vdots & \ddots & \ddots & \\ a_{NN} & a_{1N} & & 1 \end{bmatrix}\begin{bmatrix} 1 & b_{1N} & \cdots & b_{NN} \\ 0 & 1 & \ddots & \vdots \\ \vdots & & \ddots & b_{1N} \\ 0 & \cdots & & 1 \end{bmatrix}$$

$$- \frac{1}{E_N}\begin{bmatrix} 0 & & & \\ a_{NN} & 0 & & 0 \\ \vdots & \ddots & \ddots & \\ a_{1N} & a_{NN} & & 0 \end{bmatrix}\begin{bmatrix} 0 & b_{NN} & \cdots & b_{1N} \\ \vdots & 0 & \ddots & \vdots \\ \vdots & & \ddots & b_{NN} \\ 0 & \cdots & & 0 \end{bmatrix} \tag{5.4a}$$

$$= L_a U_a + L_b U_b. \tag{5.4b}$$

Hence, x can be solved explicitly from

$$x = T^{-1}y = L_a U_a y + L_b U_b y = L_a w + L_b z \tag{5.5}$$

where $w = U_a y$, $z = U_b y$. Note that due to the particular triangular Toeplitz structure, (5.5) can be essentially regarded as linear convolution operations.

The appended PLP in Section V-B is again applicable to produce the needed $\{a_N, b_N\}$. On completion of the lattice operation, these vectors will be ready in the upper and lower PE's for the convolution operation according to Gohberg's formula. For this convolution operation, two linear array processors can be utilized or, when the length of the vector is large, than the FET (fast Fourier transform) processor may be used for fast processing.

As a brief comparison of the three different organizations proposed, note that the first organization utilizes only one set of PLP; hence, it needs fewer PE's. On the other hand, in the third organization, no memory modules (LIFO) are needed. As for the processing speed, it depends very much on the size of the processor array and the applications. Generally speaking, the first method is perhaps the fastest for moderate matrix size. When N is large and the FFT is worth utilizing, the third method will become more favorable.

VI. Applications, Implementations, and Extensions

A. Applications to Array Signal Processing

From a practical point of view, the proposed Toeplitz system solver has many immediate signal processing applications. For example, it can be applied to maximum likelihood (ML) and maximum entropy (ME) adaptive array signal processing[10] problems [9]. In the ML method, the wave-vector spectrum is computed as

$$\sigma_G^2 = [E^t(f, v)S_x^{-1}(f)E(f, v)]^{-1}$$

where $S_x(f)$ is an $N \times N$ cross-covariance matrix and $E(f, v)$ is an $N \times 1$ vector denoting the array phasing vector corresponding to the wave direction [9]. Suppose that the input sensor array is uniformly spaced, and that the statistical environment is stationary, then the $S_x(f)$ matrix to be inverted in the above equation will be a Toeplitz matrix. Then the PLP can be applied to efficiently compute the spectrum σ_G^2.

For a better illustration, we use the following sample system specifications. Suppose that the number of the sensors $N = 32$ and that the temporal frequency bandwidth is $F = 50$ Hz and the frequency resolution desired is $\delta F = 0.1$ Hz. Then for one set of input data, the total computation time required is

$$(F/\delta F) \times 2Nt_0 = 32\,000\, t_0.$$

Now, we must have $32\,000\, t_0 \leqslant T$ where $T = 1/\delta F = 10$ s is the processing time allowed for one set of data, provided a real-time processing rate is required. Hence, each arithmetic operation must take no more than 10 s/32 000 = 312.5 μs (that is, $t_0 \leqslant 312.5$ μs). This time frame can be easily complied by a commercially available ALU, and therefore, a real-time signal processing rate is trivially attainable.

Obviously, the specifications quoted above are oversimplified. In practical situations, one will probably face a larger sensor array with higher bandwidth; hence, many more operations will probably be needed. On the other hand, in terms of VLSI technology, the size of the PLP can be made much larger, and the multiplication speed can be 10^3–10^4 times faster than that given above. Therefore, for most cases, the real-time processing rate can still be achievable, and with VLSI implementation of PLP, it is also rather affordable.

B. Implementation

In a joint effort between the Hughes Research Laboratory (at Malibu, CA) and the University of Southern California, Los Angeles, a VLSI Toeplitz system solver chip is implemented [34], [35].

This design contains 16 stages of processors with 28 bit fixed-point arithmetic units working on 8 bit (input dynamic range) input data. Suppose that the system clock is operating on a 4 MHz rate; it is estimated that a 16 X 16 Toeplitz system equation can be solved in about 0.6 ms (minisecond). The design is based on NMOS depletion load technology. In the

[10] The term "array processing" in signal processing commonly refers to the processing of signals from a multiple sensor array.

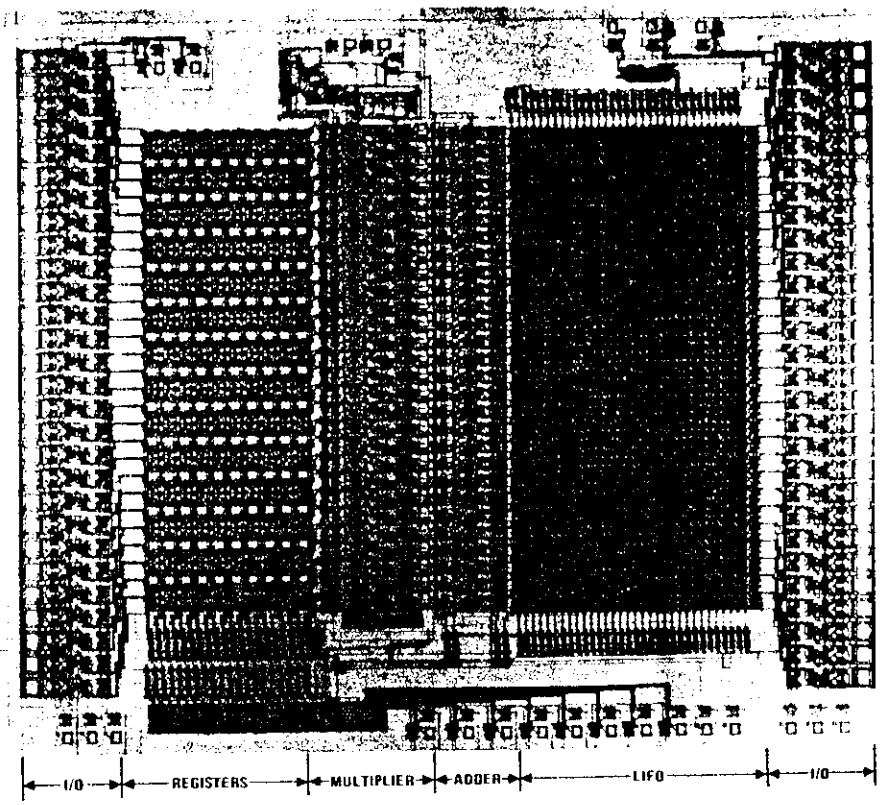

Fig. 4. Chip layout of Toeplitz systems solver. (Courtesy Hughes Research Laboratory, Malibu, CA.)

chip design, 6 μm feature size ($\lambda = 6$ μm) is used, and a single stage of processor will occupy a whole chip (230×300 mil^2). The chip layout is shown in Fig. 4. It is estimated that with the feature size reduced to 1 μm, 32 stages can be implemented on a single chip in a later implementation phase. A full report on the final implementation will appear in a future publication.

C. Extension to Multichannel Signal Processing

Quite often, one has to deal with multichannel signal processing which involves a number of input signals simultaneously. In this situation and under the stationary statistic assumption, the main computation involved is usually to solve a *block* Toeplitz system:

$$B = \begin{bmatrix} B_0 & B_{-1} & \cdots & B_{-N} \\ B_1 & B_0 & & \\ \vdots & & \ddots & \\ B_N & & & B_0 \end{bmatrix}$$

where each block element B_k is by itself a $q \times q$ matrix. *The extension of the scalar case (nonsymmetric) main algorithm to the matrix case can be carried out easily. However, the dual relations established in the scalar case fail to extend to the matrix case* due to the noncommutability nature of matrix multiplication. Consequently, we need a total of $2N$ block-matrix processors (as opposed to N) to implement the corresponding lattice computing structure for a symmetrical block Toeplitz system. To strive towards a saving of half the required processors, we propose the following further modification.

D. Normalized Levinson Algorithm

Recently, a normalized version of the Levinson algorithm was proposed by Vieara *et al.* [36]. This algorithm employs a matrix square root procedure to accomplish normalization of the reflection coefficient.[11] It turns out that the very same version may be applied to the parallel algorithm to retain the duality relations and whereby save half of the hardware. Roughly speaking, during the execution of each recursion, each entry of the u and v vector is modified by[12]

$$\overset{*}{V}{}_k^{(i)} = [V_0^{(i)}]^{-1/2} V_k^{(i)}$$

$$\overset{*}{U}{}_k^{(i)} = [U_{-i+1}^{(i)}]^{-1/2} U_k^{(i)}.$$

By substituting the above relation into the new algorithm (in multichannel formulation), a similar analysis as in Section III-A can be carried out and a *normalized* version of the new algorithm can be derived. For completeness, this normalized (high parallelism) algorithm is present in Appendix C.

VII. Conclusion

Two fundamental issues should be kept in mind in designing modern VLSI signal processors. The first is to formulate the signal processing algorithm to allow the maximal extent of parallel processing. The second is to make sure that the parallel architecture meets the (localized) communication constraint imposed by the device technology.

In this paper, we have demonstrated an integrated approach to the above issues by tackling a specific (but practically

[11] That is, for each reflection coefficient matrix $K^{(i)}$, its L_2 norm is less than or equal to unity provided that the B matrix is nonnegative definite.

[12] For a symmetric nonnegative definite matrix M, $M^{1/2}$ is defined as $M = M^{1/2}[M^{1/2}]^t$.

important) problem–solving Toeplitz systems. We have proposed a highly concurrent algorithm which enjoys $O(N)$ computing time by a vector processor array with $O(N)$ processors. This is to be compared to the $O(N^2)$ processing time of the Levinson algorithms when implemented sequentially or $O(N \log_2 N)$ when implemented in parallel. Furthermore, we have developed a pipelined lattice processor (PLP) to implement the new algorithm. The PLP architecture employs a localized communication scheme without sacrificing the overall parallel processing speed.

Appendix A
Proof of (3.13)

The duality in (3.13) originates from the symmetric row operation in the algorithm (3.12). Suppose that a matrix $\hat{T} \triangleq T^t$ is introduced. Due to Toeplitz structure, we know that $\hat{T}$ is also a Toeplitz matrix with $\hat{t}_k = t_{-k}$ for $-N \leq k \leq N$. Obviously, the algorithm (3.12) can be applied on $\hat{T}$ as well. Thus,[13]

$$\hat{T} = \hat{L}\hat{D}^{-1}\hat{U}. \tag{A.1}$$

Taking the transpose on both sides of (A.1) and using the uniqueness property of triangular factorization, it is easy to show that

$$\hat{L} = U^t, \quad \hat{D} = D, \quad \text{and} \quad \hat{U} = L^t. \tag{A.2}$$

On the other hand, by carrying out the algorithm (3.12), we find that

$$1)\ \hat{Ke}^{(i)} = \hat{Kr}^{(i)}, \qquad \hat{Kr}^{(i)} = \hat{Ke}^{(i)} \tag{A.3}$$

$$2)\ \hat{v}_k^{(i)} = u_{-k-i+1}^{(i)}, \qquad \hat{u}_k^{(i)} = v_{-k-i+1}^{(i)}. \tag{A.4}$$

Equations (A.3) and (A.4) are proved by induction as follows.

For $i = 1$, (A.4) is true from (3.12a). Then, from (3.12b) and (3.12c),

$$\hat{Ke}^{(2)} = -u_1^{(1)}/v_0^{(1)} = -t_{-1}/t_0 = Kr^{(2)} \tag{A.5a}$$

$$\hat{Kr}^{(2)} = -u_{-1}^{(1)}/v_0^{(1)} = -t_1/t_0 = Ke^{(2)}. \tag{A.5b}$$

Suppose that for $i = I$, (A.3) and (A.4) are valid; let $i = I + 1$:

$$\hat{Ke}^{(I+1)} = -\hat{u}_1^{(I)}/\hat{v}_0^{(I)} = -v_{-I}^{(I)}/u_{-I+1}^{(I)} = Kr^{(I+1)} \tag{A.6a}$$

$$\hat{Kr}^{(I+1)} = -\hat{u}_{-I}^{(I)}/\hat{v}_{-I+1}^{(I)} = -v_1^{(I)}/u_0^{(I)} = Ke^{(I+1)}. \tag{A.6b}$$

Moreover,

$$\begin{aligned}\hat{v}_k^{(I+1)} &= \hat{v}_k^{(I)} + \hat{Kr}^{(I+1)}\hat{u}_{k+1}^{(I)} \\ &= u_{-k-I}^{(I)} + Ke^{(I+1)} v_{-k-I+1}^{(I)} = u_{-k-(I+1)+1}^{(I+1)}\end{aligned} \tag{A.7a}$$

$$\begin{aligned}\hat{u}_k^{(I+1)} &= \hat{u}_k^{(I)} + \hat{Ke}^{(I+1)}\hat{v}_{k+1}^{(I)} \\ &= v_{-k-I}^{(I)} + Kr^{(I+1)} u_{-k-I+1}^{(I)} = v_{-k-(I+1)+1}^{(I+1)}.\end{aligned} \tag{A.7b}$$

By mathematical induction, (A.3) and (A.4) are proved.

From (A.1) and (A.4), it is clear that

$$\hat{U}_i = [0, \cdots, 0, v_0^{(i)}, \cdots, v_{N-i}^{(i)}]. \tag{A.8}$$

Using (A.8) and the relation $\hat{U} = L^t$ in (A.2), (3.13) follows.

[13] In Appendix A, all the symbols with "^" will be regarded as those associated with the $\hat{T}$ matrix.

Appendix B
Program for Pipelined Lattice Algorithm

(In the program presented below, the program constructs are close to those of Pascal language, while the arithmetic operations are similar to those of Assembly language. Note that the statement after the "!" sign is considered as a comment.)

Array Size: $2 \times N$ processing elements.

Computation: Pipelined lattice algorithm.

Initial: First row of the Toeplitz matrix T in the A register of the first row PE's, and the B register of the second row PE's.

Output: Rows of the U matrix ($T = U^t D^{-1} U$) are output from the second row PE's.

```
BEGIN
  SET COUNT 1;
  REPEAT
    WHILE WAVEFRONT IN ARRAY DO
      BEGIN
        CASE KIND =
          (1,*) : FLOW A, LEFT;
          (2,1),(2,*) : FLOW B, UP;
        ENDCASE;
      END;
    DECREMENT COUNT;
  UNTIL TERMINATED;
  SET COUNT N;
  REPEAT
    WHILE WAVEFRONT IN ARRAY DO
      BEGIN
        CASE KIND =
          (1,1),(1,*) : BEGIN
                          FETCH A, RIGHT;
                          FLOW A, DOWN;
                          FETCH B, DOWN;
                        END;
          (1,1) : BEGIN
                    DIV A,B,C; ! C=A/B;
                    FLOW C, DOWN;
                  END;
          (1,*) : BEGIN
                    FETCH C, LEFT;
                    ! A <= A - B X C;
                    MULT B,C,R;
                    SUB A,R,A;
                    FLOW A, LEFT;
                  END;
          (2,1) : FETCH C, UP;
          (2,*) : FETCH C, LEFT;
          (2,1),(2,*) : BEGIN
                          FETCH A, UP;
                          ! B <= B - A X C;
                          MULT A,C,R;
                          SUB B,R,B;
                          FLOW B, UP;
                          FLOW B, DOWN;  ! OUTPUT;
                        END;
        ENDCASE;
        FLOW C, RIGHT;
```

END;
DECREMENT COUNT;
UNTIL TERMINATED;
ENDPROGRAM.

Appendix C
Normalized Algorithm

INITIAL CONDITIONS:

$$\overset{*}{V}{}_k^{(1)} = \overset{*}{U}{}_k^{(1)} = B_0^{-1/2} B_k \qquad (-N \leq k \leq 0)$$

FOR $i = 1$ UNTIL N DO

BEGIN

$$K^{(i+1)} = -\overset{*}{U}{}_1^{(i)} [\overset{*}{V}{}_0^{(i)}]^{-1}$$

IN PARALLEL DO

BEGIN

$$P_{i+1} = I - K^{(i+1)} [K^{(i+1)}]^t$$

$$Q_{i+1} = I - [K^{(i+1)}]^t K^{(i+1)}$$

END IN PARALLEL DO;

IN PARALLEL FOR $-N \leq K \leq -1$ DO BEGIN

$$\overset{*}{V}{}_k^{(i+1)} = P_{i+1}^{-1/2} [\overset{*}{V}{}_k^{(i)} + K^{(i+1)} \overset{*}{U}{}_{k+1}^{(i)}]$$

$$\overset{*}{V}{}_k^{(i+1)} = P_{i+1}^{-1/2} [\overset{*}{V}{}_k^{(i)} + K^{(i+1)} \overset{*}{U}{}_{k+1}^{(i)}]$$

END IN PARALLEL DO;

OUTPUT;

END FOR LOOP;

END NORMALIZED PARALLEL LEVINSON ALGORITHM.

Acknowledgment

The authors wish to thank Prof. T. Kailath at Stanford University, Stanford, CA, for many very valuable discussions, and the reviewers for bringing [26] and [28] to our attention and for many helpful comments.

References

[1] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.
[2] L. S. Haynes, R. L. Lau, D. P. Siewiorek, and D. W. Mizell, "A survey of highly parallel computing," *IEEE Computer*, vol. 15, pp. 9–26, Jan. 1982.
[3] D. Heller, "A survey of parallel algorithms in numerical linear algebra," *SIAM Rev.*, vol. 20, pp. 740-777, Oct. 1978.
[4] H. T. Kung, "Let's design algorithms for VLSI systems," in *Proc. Conf. VLSI*, California Inst. Technol., Pasadena, Jan. 1979, pp. 65–90.
[5] S. Y. Kung, R. J. Gal-Ezer, and K. S. Arun, "Wavefront array processor: Architecture, language and applications," in *Proc. Conf. Adv. Res. in VLSI*, Massachusetts Inst. Technol., Cambridge, Jan. 1982.
[6] S. Y. Kung, K. S. Arun, R. J. Gal-Ezer, and D. V. Bhaskar Rao, "Wavefront array processor: Language, architecture, and applications," *IEEE Trans. Comput.*, Nov. 1982.
[7] S. Y. Kung, "VLSI array processor for signal processing," presented at the M.I.T. Conf. Adv. Res. on IC, Cambridge, MA, 1980.
[8] O. Grenander and G. Szego, *Toeplitz Forms and Their Applications*. Berkeley, CA: Univ. California Press, 1958.
[9] A. V. Oppenheim, Ed., *Applications of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1978.
[10] S. S. Haykin, Ed., *Nonlinear Methods of Spectral Analysis*. New York: Springer-Verlag, 1979.
[11] S. M. Kay and S. L. Marple, Jr., "Spectrum analysis—A modern perspective," *Proc. IEEE*, vol. 69, pp. 1380-1498, Nov. 1981.
[12] N. Levinson, "The Wiener RMS (root-mean-square) error criterion in filter design and prediction," *J. Math. Phys.*, vol. 25, pp. 261-278, Jan. 1947.
[13] T. Kailath, "A view of three decades of linear filtering theory," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 145–181, Mar. 1974.
[14] A. K. Jain, "Fast inversion of banded Toeplitz matrices by circular decomposition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, pp. 121–126, Apr. 1978.
[15] M. Morf, "Doubling algorithm for Toeplitz and related equation," in *Proc. ICCASP 80*, Denver, CO, Apr. 1980, pp. 954–959.
[16] —, "Fast algorithms for multivariable systems," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1974.
[17] S. Y. Kung and Y. H. Hu, "Fast and parallel algorithms for solving Toeplitz systems," presented at the Int. Symp. Mini and Microcomputers in Control and Measurement, San Francisco, CA, May 1981.
[18] J. Durbin, "The filtering of time series models," *Rev. Int. Statist. Inst.*, vol. 28, pp. 233-244, 1960.
[19] W. F. Trench, "An algorithm for inversion of finite Toeplitz matrices," *J. SIAM*, vol. 12, pp. 515–522, 1964.
[20] S. Zohar, "The algorithm of W. F. Trench," *J. Ass. Comput. Mach.*, vol. 16, no. 4, pp. 592–601, 1969.
[21] P. Whittle, "The analysis of multiple stationary time series," *J. Royal Statist. Soc.*, ser. b, vol. 15, pp. 125–139, 1953.
[22] R. Wiggins and E. A. Robinson, "Recursive solution to the multichannel filtering problems," *J. Geophys. Rev.*, vol. 70, no. 8, pp. 1885–1891, 1965.
[23] H. Akaike, "Block Toeplitz inversion," *SIAM J. Appl. Math.*, vol. 24, pp. 234–241, Mar. 1975.
[24] I. Schur, "Uber Potenzreihen die in Innern des Einheitskreises Beschrankt Sind," *J. Reine Angewandte Mathematik*, vol. 147, 1917, pp. 205–232.
[25] P. Dewilde, A. Vieira, and T. Kailath, "On a generalized Szego-Levinson realization algorithm for optimal linear predictors based on a network synthesis approach," *IEEE Trans. Circuits Syst.*, vol. CAS-25, 1978, pp. 663–675.
[26] E. H. Bareiss, "Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices," *Numer. Math.*, vol. 13, pp. 404–424, 1969.
[27] J. Rissanen, "Algorithms for triangular decomposition of block Hankel and Toeplitz matrices with applications to factoring positive matrix polynomials," *Math. Comp.*, vol. 27, pp. 147-159, Jan. 1973.
[28] R. K. Brouwer, "Particular methods for solving particular systems of linear equations," M. Phil. thesis, Dep. Comput. Sci., Univ. Waterloo, Waterloo, Ont., Canada, July 1971.
[29] S. Y. Kung, "Matrix data flow in language for matrix operation dedicated array processors," in *Proc. ECCTD 1981*, The Hague, The Netherlands, Aug. 1981, pp. 393–398.
[30] S. Y. Kung, K. S. Arun, D. V. Bhaskar Rao, and Y. H. Hu, "A matrix data flow languate/architecture for parallel matrix operations based on computational wavefront concept," in *Proc. Carnegie-Mellon Univ. Conf. VLSI Syst. Computations*, Oct. 1981, pp. 226–234 (published by Comput. Sci. Press).
[31] S. Y. Kung *et al.*, "Highly parallel modern signal processing," Univ. Southern California, Los Angeles, USC-SRO Rep. 1, Oct. 1981.
[32] G. Cybenko, "The numerical stability of the Levinson-Durbin algorithm for Toeplitz systems of equations," *SIAM J. Sci. Statist. Comput.*, vol. 1, Sept. 1980.
[33] I. C. Gohberg and I. A. Fel'dman, *Convolution Equations and Projection Methods for Their Solutions* (transl. of Math. Monograph, vol. 41, Amer. Math. Soc., Providence, RI, 1974).
[34] J. G. Nash, S. Hansen, and G. R. Nudd, "VLSI processor array for matrix manipulation," in *VLSI System and Computations*, H. T. Kung, Ed. Baltimore, MD: Comput. Sci. Press, 1981, pp. 367–373.
[35] J. G. Nash and G. R. Nudd, "Concurrent VLSI architectures for two dimensional signal processing systems," presented at the USC workshop on VLSI and Modern Signal Processing, Los Angeles, CA, Nov. 1–3, 1982. The full paper appears in *VLSI and Modern Signal Processing*, S. Y. Kung *et al.*, Eds. Englewood Cliffs, NJ: Prentice-Hall.
[36] A. C. G. Vieara, "Matrix orthogonal polynomials, with applications to autoregressive modeling and ladder forms," Ph.D. dissertation, Stanford Univ., Stanford, CA, Dec. 1977.

Sun-Yuan Kung (M'77) received the B.S. degree in electrical engineering in 1971 from the National Taiwan University, Taipei, Taiwan, the M.S. degree in electrical engineering in 1974 from the University of Roches-

ter, Rochester, NY, and the Ph.D. degree in electrical engineering in 1977 from Stanford University, Stanford, CA.

In 1973 he held a Fellowship at the University of Rochester. In 1974 he joined the Amdahl Corporation, Sunnyvale, CA, as an Associate Engineer. From 1974 to 1977 he was a Research Assistant at Stanford University, Information Systems Laboratories. Since July 1977 he has been with the Faculty of Electrical Engineering–Systems, University of Southern California, Los Angeles, where he is presently an Associate Professor. Since 1979 he has also been a Research Consultant with Stanford University and the General Electric Company, Syracuse, NY. His research interests are in the areas of multivariable and two-dimensional system theory, digital signal processing, modern spectrum analysis, and VLSI parallel processing.

Yu Hen Hu (S'82) was born in Taipei, Taiwan, Republic of China, on August 5, 1954. He received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, in 1976. Presently, he is working towards the completion of the Ph.D. degree in the Department of Electrical Engineering–Systems, University of Southern California, Los Angeles.

Since September 1978 he has been a Teaching Assistant and Research Assistant in the Department of Electrical Engineering–Systems, University of Southern California. His current research interests include VLSI architectures and system design, spectrum estimation, and signal processing algorithms.

Mr. Hu is a member of the Society of Photo-Optical Instrumental Engineers and Eta Kappa Nu.