A Systolic Array for the Linear-Time Solution of Toeplitz Systems of Equations

RICHARD P. BRENT and FRANKLIN T. LUK

A Systolic Array for the Linear-Time Solution of Toeplitz Systems of Equations*

RICHARD P. BRENT[†] and FRANKLIN T. LUK[‡]

***Abstract*—The solution of an $(n+1) \times (n+1)$ Toeplitz system of linear equations on a one-dimensional systolic architecture is studied. Our implementation of an algorithm of Bareiss is shown to require only $0(n)$ time and $0(n)$ storage, i.e. constant storage per systolic processor.**

Key words and phrases: **Systolic arrays, Toeplitz matrices, linear equations, Bareiss algorithm, VLSI.**

1. INTRODUCTION

Toeplitz systems of linear equations arise in many scientific and engineering applications, for some of which (e.g., signal processing) real-time solution is essential (cf. [25]). Recently, Bitmead and Anderson [3], and Brent, Gustavson, and Yun [4] proposed procedures which, when applied to order-$(n+1)$ systems, require only $0(n \log^2 n)$ time and $0(n)$ space. The well known algorithms of Levinson [19], Durbin [10], Trench [28], Zohar [29], and Bareiss [2], all require time $0(n^2)$ and space $0(n)$ or $0(n^2)$. These algorithms are based on recursions due to Schur [23] and Szegö [27]. See also [6, 8, 9, 11, 12, 13, 14, 18, 20]. Unfortunately, the $0(n \log^2 n)$-time algorithms are quite complicated and may be slower than the $0(n^2)$-time methods if n is not sufficiently large (cf. Sexton [24]).

In this paper we study the efficient solution of Toeplitz systems on a systolic array of $0(n)$ processors. We present an implementation of the Bareiss algorithm that requires $0(n)$ time. Related work has recently been done independently by S.Y. Kung and Hu [16] (see also [17, 22]) and Ahmed, Delosme, and

*Supported in part by U.S. Army Research Office grant DAAG 29-79-0124 and in part by the Centre for Mathematical Analysis and the Mathematical Sciences Research Centre at the Australian National University.

[†]Department of Computer Science, Australian National University, Canberra, ACT 2600, Australia.

[‡]Department of Computer Science, Cornell University, Ithaca, New York.

Morf [1]. However, our results are more desirable in two respects: we do not assume that the Toeplitz matrix is symmetric (although we can take advantage of this), and our storage requirements are $0(n)$ instead of $0(n^2)$. This saving in storage is significant because large Toeplitz systems often arise in filtering and signal processing applications [10, 19, 25, 26]: values of n greater than 1000 are common. Kung and Hu use a slightly different procedure, the so-called symmetric Bareiss algorithm (see [2]). In Section 7 we show how it is also possible to implement the symmetric Bareiss method using only $0(n)$ storage, except in a certain degenerate case, but the unsymmetric Bareiss method may still be preferred for reasons of numerical stability. Ahmed, Delosme, and Morf [1] use the HC algorithm [21] which requires the matrix to be positive definite.

2. THE BAREISS ALGORITHM

Let us describe the Bareiss algorithm [2] for the solution of

$$\mathbf{T}\mathbf{x} = \mathbf{b}, \tag{2.1}$$

where T is a Toeplitz matrix of order $(n+1)$ and **b** a column vector:

$$\mathrm{T} = \begin{pmatrix} t_0 & t_1 & t_2 & \cdots t_n \\ t_{-1} & t_0 & t_1 & \cdots t_{n-1} \\ t_{-2} & t_{-1} & t_0 & \cdots t_{n-2} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ t_{-n} & t_{-n+1} & t_{-n+2} & \cdots t_0 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_n \end{pmatrix} \tag{2.2}$$

For convenience we let the row and column indices run from 0 to n in this paper. The idea of Bareiss is to transform (2.1) successively into

$$\mathrm{T}^{(-1)}\mathbf{x} = \mathbf{b}^{(-1)}, \mathrm{T}^{(1)}\mathbf{x} = \mathbf{b}^{(1)}, \mathrm{T}^{(-2)}\mathbf{x} = \mathbf{b}^{(-2)}, \mathrm{T}^{(2)}\mathbf{x} = \mathbf{b}^{(2)}, \quad \ldots, \quad \mathrm{T}^{(-n)}\mathbf{x} = \mathbf{b}^{(-n)}, \mathrm{T}^{(n)}\mathbf{x} = \mathbf{b}^{(n)}, \tag{2.3}$$

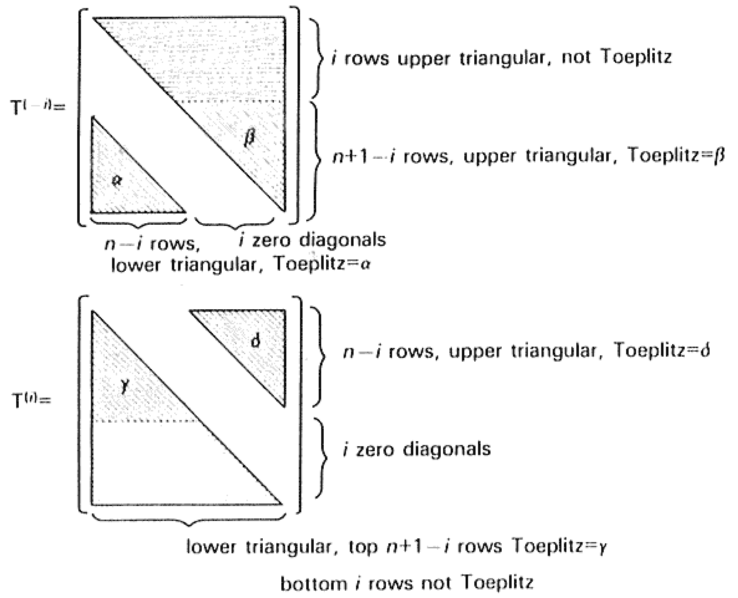
so that the final matrices $\mathrm{T}^{(-n)}$ and $\mathrm{T}^{(n)}$ are upper and lower triangular, respectively. We introduce the shift matrices

$$\mathrm{Z}_{-i} = (z_{jk}^{(-i)}) = (\delta_{j-k-i}) \quad \text{and} \quad \mathrm{Z}_i = (z_{jk}^{(i)}) = (\delta_{j-k+i}), \tag{2.4}$$

where δ is the Kronecker delta, and let $\mathrm{T}^{(0)} = \mathrm{T}$, $\mathbf{b}^{(0)} = \mathbf{b}$. The following recurrences then define the transformations on T and **b**:

$$\begin{aligned} \mathrm{T}^{(-i)} &= \mathrm{T}^{(-i+1)} - m_{-i}\,\mathrm{Z}_{-i}\,\mathrm{T}^{(i-1)}, \quad \text{with} \quad m_{-i} = \frac{t_{i,0}^{(1-i)}}{t_0}, \\ \mathbf{b}^{(-i)} &= \mathbf{b}^{(-i+1)} - m_{-i}\,\mathrm{Z}_{-i}\,\mathbf{b}^{(i-1)}, \\ \mathrm{T}^{(i)} &= \mathrm{T}^{(i-1)} - m_i\,\mathrm{Z}_i\,\mathrm{T}^{(-i)}, \quad \text{with} \quad m_i = \frac{t_{0,i}^{(i-1)}}{t_{n,n}^{(-i)}}, \\ \mathbf{b}^{(i)} &= \mathbf{b}^{(i-1)} - m_i\,\mathrm{Z}_i\,\mathbf{b}^{(-i)}. \end{aligned} \tag{2.5}$$

In (2.5) the effect of premultiplication by Z_{-i} is to downshift the matrix $\mathrm{T}^{(i-1)}$ by i rows and to replace its first i rows by zeros. Similarly, Z_i upshifts $\mathrm{T}^{(-i)}$ by i rows and replaces its last i rows by zeros. Suppose that the matrix $\mathrm{T}^{(-i+1)}$ (resp. $\mathrm{T}^{(i-1)}$) has $(i-1)$ null subdiagonals (resp. superdiagonals). The operations described by (2.5) will annihilate the i-th subdiagonal (resp. superdiagonal) without disturbing those already null elements. It follows that the matrix $\mathrm{T}^{(-n)}$ (resp. $\mathrm{T}^{(n)}$) will be upper (resp. lower) triangular.



T(−i)=
i rows upper triangular, not Toeplitz
β
n+1−i rows, upper triangular, Toeplitz=β
α
n−i rows, lower triangular, Toeplitz=α
i zero diagonals
T(i)=
δ
n−i rows, upper triangular, Toeplitz=δ
γ
i zero diagonals
lower triangular, top n+1−i rows Toeplitz=γ
bottom i rows not Toeplitz

Figure 1. Structure of $\mathrm{T}^{(-i)}$ and $\mathrm{T}^{(i)}$ in the Bareiss algorithm.

The Bareiss method will not fail as long as all the leading principal submatrices of the given matrix T are nonsingular. An LU-factorization, with L unit lower triangular, of T is then given by (c.f. Sweet [26])

$$\mathrm{T} = \mathrm{LU},$$

where

$$\mathrm{L} = \frac{1}{t_0}(\mathrm{T}^{(n)})^{\mathrm{T}2}, \qquad (2.6)$$

$$\mathrm{U} = \mathrm{T}^{(-n)},$$

and the superscript "T2" denotes matrix transpose about the main antidiagonal. The Bareiss method, thus, has the same numerical properties as Gaussian elimination without pivoting. In many applications T is either diagonally dominant or positive definite, so the lack of pivoting may not be too severe a handicap.

Sweet [26] shows that the Bareiss and the Trench-Zohar algorithms are closely related, but the Bareiss algorithm appears to be more amenable to parallel computation. For further comments on the numerical properties of these algorithms, see Sweet [26] and Cybenko [7]. In Section 7 we shall briefly discuss a symmetric variant of the Bareiss alogrithm that can be more efficient when T is symmetric.

3. A SYSTOLIC ARRAY FOR FACTORIZING T

We present here a one-dimensional systolic array for computing the two triangular matrices $\mathrm{T}^{(-n)}$ and $\mathrm{T}^{(n)}$. The array consists of $2n-1$ processors P_{-n+1}, $\mathrm{P}_{-n+2}, \ldots, \mathrm{P}_{-1}, \mathrm{P}_0, \mathrm{P}_1, \ldots, \mathrm{P}_{n-1}$, arranged from left to right. All processors are identical except for the middle one P_0. For simplicity we first assume that P_0 can broadcast a scalar quantity to all other processors in constant time. This assumption will soon be dropped. The processor P_0 has four registers U_{0-}, U_{0+}, D_{0-}, and D_{0+}, and each remaining processor $\mathrm{P}_k (k \neq 0)$ has two registers U_k and D_k. We denote the content of register R by [R]. Each processor has two output lines outu_k and outd_k, and two input lines inu_k and ind_k. The output line outu_k is connected to the input line inu_{k-1}, for $k=1, 2, \ldots, n-1$. The output line outd_{-k} is connected to the input line ind_{-k+1}, for $k=1,2,\ldots,n-1$.

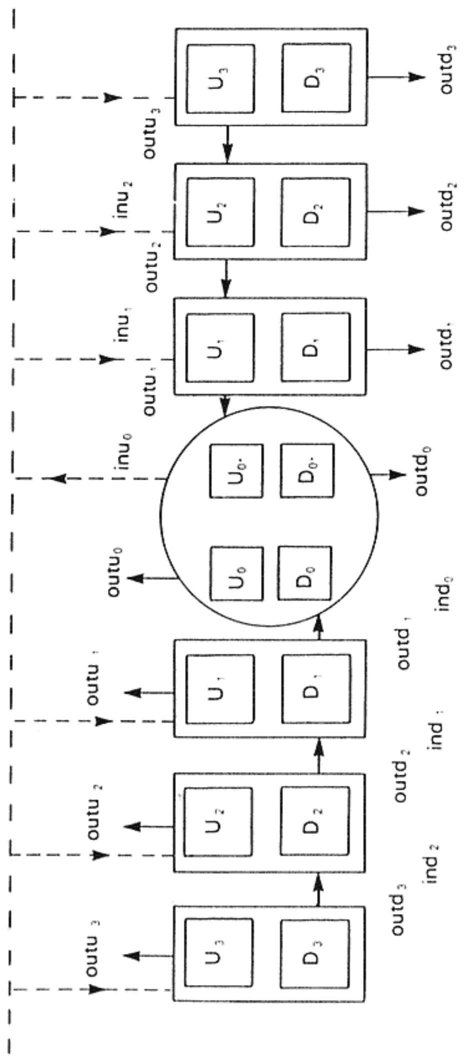


Figure 2. The processor array for $n=4$.

Before iterating, we feed data into the array so that

$$\begin{aligned} [\mathrm{U}_{0-}] &= t_0, & [\mathrm{U}_{0+}] &= t_1, \\ [\mathrm{D}_{0-}] &= t_{-1}, & [\mathrm{D}_{0+}] &= t_0, \end{aligned} \tag{3.1}$$

and

$$\begin{aligned} [\mathrm{U}_{-k}] &= t_{-k}, & [\mathrm{U}_k] &= t_{k+1}, \\ [\mathrm{D}_{-k}] &= t_{-k-1}, & [\mathrm{D}_k] &= t_k, \end{aligned} \tag{3.2}$$

for $k=1, 2, \ldots, n-1$. We now consider one iteration, say the i-th one. Each iteration consists of three steps. At step one the processor P_0 computes the multiplier

$$m_{-i} = \frac{[\mathrm{D}_{0-}]}{[\mathrm{U}_{0-}]}. \tag{3.3}$$

This value is broadcast to all processors. Processor P_k, all k, now computes

$$\text{(i)} \equiv [\mathrm{D}_k] - m_{-i}[\mathrm{U}_k]. \tag{3.4}$$

The quantity $[D_k]$, for $k \geq 0$, is sent out on the output line outd_k, and the result (i) is stored in register D_k. At step two, the processor P_0 computes the multiplier

$$m_i = \frac{[\mathrm{U}_{0+}]}{[\mathrm{D}_{0+}]} \tag{3.5}$$

and broadcasts this number. The processor P_k, for all k, then computes

$$\text{(ii)} \equiv [\mathrm{U}_k] - m_i[\mathrm{D}_k]. \tag{3.6}$$

The quantity $[\mathrm{U}_k]$, for $k \leq 0$, is output on line out_k, and the register U_k receives the value of (ii). At the third step, we do data transfer. The content of register D_k, for $k \leq -1$, is sent to processor P_{k+1} on line outd_k. The incoming data for P_{k+1} is stored in register D_{k+1}. The content of register D_{0-} is lost. At the same time, the content of register U_k, for $k \geq 1$, is sent to processor P_{k-1}, on line outu_k. The incoming data for P_{k-1} is stored in register U_{k-1}, and the content of register U_{0+} is lost. We now disable processors P_{-n+i} and P_{n-i}. This completes one iteration of the Bareiss algorithm.

Since we disable the two end processors after each iteration, our method must terminate after n iterations. Let us denote the output on line outd_k at the i-th iteration by $d_k^{(i)}$ and the corresponding output on outu_k by $u_k^{(i)}$. The two desired matrices $\mathrm{T}^{(-n)}$ and $\mathrm{T}^{(n)}$ are given by

$$\mathrm{T}^{(-n)} = \begin{pmatrix} t_0 & t_1 & t_2 & t_3 & \cdots & t_n \\ & d_0^{(1)} & d_1^{(1)} & d_2^{(1)} & \cdots & d_{n-1}^{(1)} \\ & & d_0^{(2)} & d_1^{(2)} & \cdots & d_{n-2}^{(2)} \\ & & & d_0^{(3)} & \cdots & d_{n-3}^{(3)} \\ & & & & \ddots & \vdots \\ 0 & & & & & d_0^{(n)} \end{pmatrix}, \text{ and} \tag{3.7}$$

$$\mathrm{T}^{(n)} = \begin{pmatrix} u_0^{(n)} & & & & & \\ u_{-1}^{(n-1)} & u_0^{(n-1)} & & & 0 & \\ u_{-2}^{(n-2)} & u_{-1}^{(n-2)} & u_0^{(n-2)} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ u_{-n+1}^{(1)} & u_{-n+2}^{(1)} & u_{-n+3}^{(1)} & \cdots & u_0^{(1)} & \\ t_{-n} & t_{-n+1} & t_{-n+2} & \cdots & t_1 & t_0 \end{pmatrix}. \tag{3.8}$$

By transforming the right-hand vector $\mathbf{b}$ simultaneously (see Section 4), we obtain an upper triangular system $\mathrm{T}^{(-n)}\mathbf{x} = \mathbf{b}^{(-n)}$ to solve. So $\mathrm{T}^{(n)}$ can be discarded. Since $\mathrm{T}^{(-n)}$ is not Toeplitz, the Bareiss algorithm appears to require $0(n^2)$ storage. However, at the expense of some extra computation, we can avoid using more than $\mathrm{O}(n)$ storage (the details are presented in Sections 5 and 6). This important point is not observed in [1, 2, 16].

We show now that broadcasting is not needed if each processor P_k (resp. P_{-k}), $k>0$, can pass a scalar quantity to its outer neighbor P_{k+1} (resp. P_{-k-1}) and if P_0 can pass a number to both P_1 and P_{-1}. Let us describe how the i-th iteration (say) proceeds. The middle processor P_0 reads the inputs on lines inu_0 and ind_0, and stores the numbers in registers U_{0+} and D_{0-}, respectively. The computing starts with P_0 calculating the multiplier m_{-i} and passes the value to its two neighbors P_{-1} and P_1. The processor P_0 now performs the second step of the iteration by computing the multiplier m_i and again passes its value to processors P_{-1} and P_1. The processor P_1 (resp. P_{-1}), on receiving m_{-i}, will do the computation (3.4) and pass the multiplier to the neighbor P_2 (resp. P_{-2}). Then P_1 (resp. P_{-1}) will receive the multiplier m_i. The operations described in (3.6) are now performed and the value of m_i will be passed to processor P_2 (resp. P_{-2}). The processor P_1 (resp. P_{-1}) completes its share of the iteration by sending the content of register U_1 (resp. D_{-1}) leftward (resp. rightward) to processor P_0. The processors P_2 and P_{-2}, on receiving the multipliers m_{-i} and then m_i, will perform the necessary computations, the passing on of the multipliers and finally the shifting of the necessary information toward the middle processor. This process expands outward until the two end processors P_{n-i} and P_{-n+i} have done their tasks, ending the iteration. An important observation here is that the $(i+1)$-st iteration can start as soon as processor P_0 receives the data from its two neighbors. Our technique for avoiding broadcast is quite common [5, 15]: processors are active only on alternate time steps (P_0, $P_{\pm 2}$ ... at time $\tau = 1, 3, \ldots$ and $P_{\pm 1}$, $P_{\pm 3}, \ldots$ at time $\tau = 2, 4, \ldots$), and the operations of processors $P_{\pm k}$ are delayed by k time steps relative to the operation of processor P_0.

Suppose that the given matrix T is banded with half-bandwidth w. We can, of course, disregard this special structure and still use $2n-1$ processors. But as processors $P_{\pm(w-1)}, P_{\pm w}, \ldots, P_{\pm(n-1)}$ work only with null data, we may perform the elimination using only $2w-3$ processors if the input lines inu_{w-2} and ind_{-w+2} always carry the number zero. We will disable the processors P_{n-i} and P_{-n+i} at the end of the i-th iteration, for $i = n-w+2, n-w+3, \ldots, n-1$.

Example 1. *(Bareiss [2, p. 415]).*

$$T = 120 \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 2 & 3 \\ 4 & 3 & 2 & 1 & 2 \\ 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

4.3	shift ←				120				
4.2	$m_4 = \frac{24}{288} = \frac{1}{12}$				120	0			
3.3	shift ←			144	120	24			
3.2	$m_3 = \frac{30}{300} = \frac{1}{10}$			144	120	0	24		
2.3	shift ←		180	150	120	30	60		
2.2	$m_2 = \frac{40}{320} = \frac{1}{8}$		180	150	120	0	30	60	

1.3	shift ←	240	200	160	120	40	80	120	
1.2	$m_1 = \frac{240}{-360} = -\frac{2}{3}$	240	200	160	120	0	40	80	120
		480	360	240	120	240	360	480	600
		600	480	360	240	120	240	360	480
1.1	$m_{-1} = \frac{240}{120} = 2$	−360	−240	−120	0	−360	−480	−600	−720
1.3	shift →		−360	−240	−120	−360	−480	−600	−720
2.1	$m_{-2} = \frac{-120}{120} = -1$		−160	−80	0	−320	−400	−480	
2.3	shift →			−160	−80	−320	−400	−480	
3.1	$m_{-3} = \frac{-80}{120} = -\frac{2}{3}$			−60	0	−300	−360		
3.3	shift →				−60	−300	−360		
4.1	$m_{-4} = \frac{-60}{120} = -\frac{1}{2}$				0	−288			
4.3	shift →					−288			
		P_{-3}	P_{-2}	P_{-1}		P_0	P_1	P_2	P_3

Hence

$$T^{(-4)} = \begin{pmatrix} 120 & 240 & 360 & 480 & 600 \\ & -360 & -480 & -600 & -720 \\ & & -320 & -400 & -480 \\ & & & -300 & -360 \\ & 0 & & & -288 \end{pmatrix}$$

and

$$T^{(4)} = \begin{pmatrix} 120 & & & & \\ 144 & 120 & & 0 & \\ 180 & 150 & 120 & & \\ 240 & 200 & 160 & 120 & \\ 600 & 480 & 360 & 240 & 120 \end{pmatrix}.$$

Note that $T = LU$, where $U = T^{(-4)}$ and

$$L = \frac{1}{120}(T^{(4)})^{T2} = \begin{pmatrix} 1 & & & & \\ 2 & 1 & & & 0 \\ 3 & \frac{4}{3} & 1 & & \\ 4 & \frac{5}{3} & \frac{5}{4} & 1 & \\ 5 & 2 & \frac{3}{2} & \frac{6}{5} & 1 \end{pmatrix}$$

4. MODIFYING THE RIGHT-HAND VECTOR

We want to find the vector $\mathbf{b}^{(-n)}$ that satisfies

$$T^{(-n)}\mathbf{x} = \mathbf{b}^{(-n)}.$$

We determine this vector from **b** using the recurrences (2.5). We, thus, need only the multipliers generated in the factorization phase and not the lower triangular matrix $T^{(n)}$.

The structure and the operations of the systolic array are very similar to the right-half of the systolic architecture of the previous section. The array here consists of the n processors $Q_0, Q_1, \ldots, Q_{n-1}$. All the processors are identical.

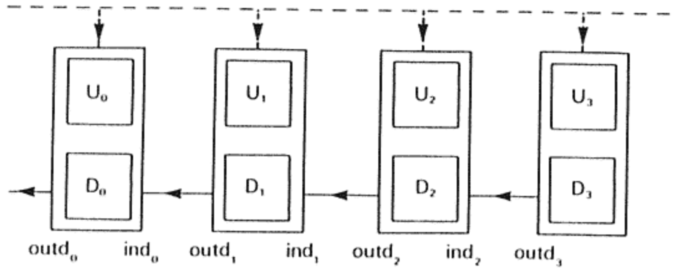


Figure 3. Systolic Array for Finding $\mathbf{b}^{(-n)}$ $(n=4)$.

Each Q_k has two registers U_k and D_k, one output line outd$_k$ and one input line ind$_k$. The lines ind$_k$ and outd$_{k+1}$ are connected (for $k=0,1,\ldots, n-2$). We also assume temporarily that all processors can receive the broadcast of a scalar quantity.

Before iteration starts, data are fed into the processors so that, for $k=0, 1, \ldots, n-1$,

$$\begin{aligned}[U_k] &= b_k, \\ [D_k] &= b_{k+1},\end{aligned} \tag{4.1}$$

where $\mathbf{b} = (b_0, b_1, \ldots, b_n)^T$. Let us consider the i-th iteration, where $i \geqslant 1$. An iteration consists of three steps, same as in the previous section. At the first step, the multiplier m_{-i} arrives at processor Q_k, which then computes

$$(iii) \equiv [D_k] - m_{-i}[U_k], \tag{4.2}$$

and stores (iii) in register D_k. The second step begins when Q_k receives the multiplier m_i. It computes

$$(iv) \equiv [U_k] - m_i[D_k], \tag{4.3}$$

and stores the result in U_k. The third step is now initiated. It involves a transfer of the content of register D_k from processor Q_k to processor Q_{k-1}, for $k \geqslant 1$. The content of D_1 is sent out on outd$_0$ and register D_k, for $k \geqslant 0$, receives the content of register D_{k+1}. The completion of the data transfer ends the i-th iteration and processor Q_{n-i} is disabled. If we denote the output on line outd$_0$ after the i-th iteration by $d_0^{(k)}$, then the vector $\mathbf{b}^{(-n)}$ is given by

$$\mathbf{b}^{(-n)} = \begin{pmatrix} b_0 \\ d_0^{(1)} \\ d_0^{(2)} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ d_0^{(n)} \end{pmatrix} \tag{4.4}$$

Since the algorithms in this and the previous sections are very similar, we can argue using the same reasonings as before that broadcasting is unnecessary if each processor Q_k can pass the multiplier to its right neighbor Q_{k+1} and if the operation of processor Q_k is delayed by k time steps relative to the operation of processor Q_0.

Example 2. *(Bareiss [2, p. 415])*

$$\mathbf{b} = 120 \begin{pmatrix} 30 \\ 22 \\ 18 \\ 20 \\ 30 \end{pmatrix}$$

4.2	$m_4 = -\frac{1}{12}$		120			
3.2	$m_3 = -\frac{1}{10}$		120	384		
2.2	$m_2 = -\frac{1}{8}$		240	390	840	
1.2	$m_1 = -\frac{2}{3}$		560	560	880	1600
			3600	2640	2160	2400
			2640	2160	2400	3600
1.1	$m_{-1} = 2$		−4560	−3120	−1920	−1200
1.3	shift ←	−4560	−3120	−1920	−1200	
2.1	$m_{-2} = -1$		−2560	−1360	−320	
2.3	shift ←	−2560	−1360	−320		
3.1	$m_{-3} = -\frac{2}{3}$		−1200	−60		
3.3	shift ←	−1200	−60			
4.1	$m_{-4} = -\frac{1}{2}$		0			
4,3	shift ←	0				
			Q_0	Q_1	Q_2	Q_3

$$\mathbf{b}^{(-4)} = \begin{pmatrix} 3600 \\ -4560 \\ -2560 \\ -1200 \\ 0 \end{pmatrix}$$

5. REGENERATING $T^{(-n)}$ USING $0(n)$ STORAGE

We consider the regeneration of the upper triangular matrix $T^{(-n)}$ using only its last column and the $2n$ multipliers $m_{\pm i}$. Our key idea is to run the elimination algorithm in Section 3 backwards:

$$\begin{aligned} T^{(i-1)} &= T^{(i)} + m_i Z_i T^{(-i)}, \\ T^{(-i+1)} &= T^{(-i)} + m_{-i} Z_{-i} T^{(i-1)}, \end{aligned} \tag{5.1}$$

for $i = n, n-1, \ldots, 1$. (Observe that rows 0 to i for $T^{(-n)}$ are equal to rows 0 to i of $T^{(-i)}$.) So our systolic array consists of n identical processors $B_0, B_1, \ldots, B_{n-1}$. Each processor B_k has two registers U_k and D_k. Initially,

$$\begin{aligned} [U_k] &= 0, \quad \text{and} \\ [D_k] &= t^{(-n)}_{n-k,n}, \end{aligned} \tag{5.2}$$

for $k=0, 1, \ldots, n-1$. We again assume for a moment that there is a broadcasting mechanism. Each processor B_k has two output lines outu_k and outd_k and one input line inu_k. The lines outu_k and inu_{k+1} are connected, for $k \geq 0$.

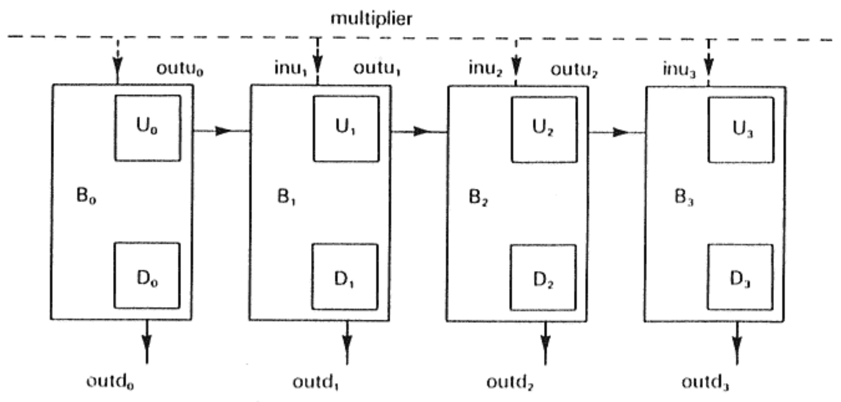


Figure 4. A systolic array for generating $T^{(-n)}$ $(n=4)$.

Only one processor, B_0, is active for the initial iteration. At the end of the i-th iteration, $i \geq 1$, processor B_i is activated for subsequent computations so that the $(i+1)$ processors $B_0, B_1, \ldots, B_i$ are active during the $(i+1)$-st iteration. Each iteration consists of three steps. Let us describe the i-th iteration. At the first step, the multiplier m_{n+1-i} is broadcast to all the processors and the following computation is done:

$$(v) \equiv [U_k] + m_{n+1-i}[D_k], \tag{5.3}$$

for $k = 0, 1, \ldots, i-1$. The result (v) is stored in register U_k. The second step starts when the multiplier m_{-n-1+i} is broadcast to all processors. Processor B_k $(0 \leq k \leq i-1)$ then computes

$$(vi) \equiv [D_k] + m_{-n-1+i}[U_k], \tag{5.4}$$

outputs the result (vi) on outd$_k$ and also stores the number in register D_k. The third step is but a shifting of the content of register U_k to register U_{k+1}, for $k = 0, 1, \ldots, i-1$. Register U_0 will contain the number zero. The complete procedure stops after n iterations.

If we denote the output on line outd$_k$ at the i-th iteration by $d_k^{(i)}$, the desired matrix $T^{(-n)}$ is given by

$$T^{(-n)} = \begin{pmatrix} d_0^{(n)} & d_1^{(n)} & \cdots & d_{n-1}^{(n)} & t_{0,n}^{(-n)} \\ & d_0^{(n-1)} & \cdots & d_{n-2}^{(n-1)} & t_{1,n}^{(-n)} \\ & & \cdot & \cdot & \cdot \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot \\ & & d_0^{(2)} & d_1^{(2)} & t_{n-2,n}^{(-n)} \\ 0 & & & d_0^{(1)} & t_{n-1,n}^{(-n)} \\ & & & & t_{n,n}^{(-n)} \end{pmatrix} \tag{5.5}$$

As before, we can argue that broadcasting is unnecessary as long as each processor can pass a scalar quantity to its right neighbor.

Since our primary concern is the solution of $T^{(-n)}\mathbf{x} = \mathbf{b}^{(-n)}$ on a linear systolic array, it is interesting to note that we have regenerated the elements of $T^{(-n)}$ in the exact order as required by the Kung-Leiserson algorithm for back substitution [15]. The details are given in the next section.

Example 3. *(See Example 1).*

4.3 shift →	0	240.	360	480
4.1 $m_1 = -\frac{2}{3}$	240	360	480	600
3.3 shift →	0	40	80	120
3.1 $m_2 = -\frac{1}{8}$	40	80	120	
2.3 shift →	0	30	60	
2.1 $m_3 = -\frac{1}{10}$	30	60		
1.3 shift →	0	24		
1.1 $m_4 = -\frac{1}{12}$	24			
	0	0	0	0
	−288	−360	−480	−720
1.2 $m_{-4} = -\frac{1}{2}$	−300			
2.2 $m_{-3} = -\frac{2}{3}$	−320	−400		
3.2 $m_{-2} = -1$	−360	−480	−600	
4.2 $m_{-1} = 2$	120	240	360	480
	B_0	B_1	B_2	B_3

We get

$$T^{(-4)} = \begin{pmatrix} 120 & 240 & 360 & 480 & 600 \\ & -360 & -480 & -600 & -720 \\ & & -320 & -400 & -480 \\ & 0 & & -300 & -360 \\ & & & & -288 \end{pmatrix}$$

6. A COMPLETE ARCHITECTURE

We can construct one systolic array that solves the given equations $\mathbf{Tx} = \mathbf{b}$. Because of the similarities in their operations, processors $P_{\pm k}$ and Q_k ($k \geq 0$) are combined into one super-processor S_k ($k \geq 0$). We then program S_k ($k \geq 0$) to do the regeneration of $T^{(-n)}$ and the solution of $T^{(-n)}\mathbf{x} = \mathbf{b}^{(-n)}$.

Let us describe our linear array of $n+1$ super-processors $S_0, S_1, \ldots, S_n$. (The last processor S_n is needed for the back substitution.) In the Bareiss algorithm four triangular Toeplitz matrices

$$\alpha=\begin{bmatrix}\alpha_0 & & 0\\ \alpha_1 & \ddots & \\ \ddots & \alpha_1 & \alpha_0\end{bmatrix},\quad \beta=\begin{bmatrix}\beta_0 & \beta_1 & \ddots\\ & \ddots & \beta_1\\ 0 & & \beta_0\end{bmatrix},\quad \gamma=\begin{bmatrix}\gamma_0 & & 0\\ \gamma_1 & \ddots & \\ \ddots & \gamma_1 & \gamma_0\end{bmatrix},\ \text{and}\ \delta=\begin{bmatrix}\delta_0 & \delta_1 & \ddots\\ & \ddots & \delta_1\\ 0 & & \delta_0\end{bmatrix}$$

are updated (see Figure 1). Now each processor S_k has registers to store α_k, β_k, γ_k, and δ_k. (When describing processor S_k we shall omit the subscripts and simply refer to registers α, β, γ, and δ.) Processor S_k requires four additional registers: λ_k for a multiplier m_{-j}, μ_k for a multiplier m_{+j}, and ξ_k and η_k which are associated with the right-hand side vector $\mathbf{b}$ and the solution $\mathbf{x}$.

Data flows in both directions between adjacent processors, as shown in Figure 5. Hence, each processor needs five input and five output data paths denoted by inL1, inL2, inR1, inR2, inR3, outL1, outL2, outL3, outR1, and outR2 (see Figure 6).

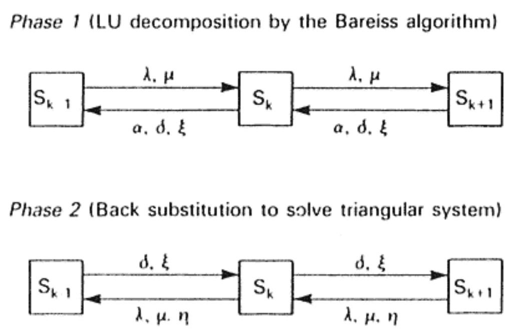

Phase 1 (LU decomposition by the Bareiss algorithm)
S_{k-1}
S_k
S_{k+1}
λ, μ
α, δ, ξ
Phase 2 (Back substitution to solve triangular system)
δ, ξ
λ, μ, η

Figure 5. Data flow for systolic Toeplitz solver.

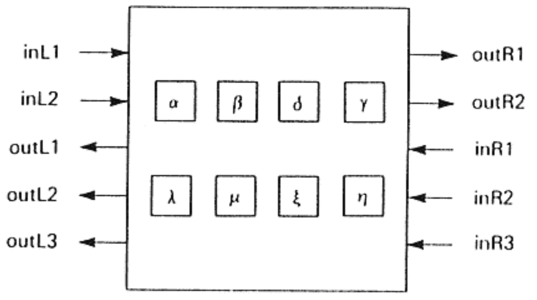


Figure 6. Systolic processor for Toeplitz systems.

Initialization is as follows: $\alpha_k := t_{-(k+1)}$; $\beta_k := t_k$; $\gamma_k := t_{-k}$; $\delta_k := t_{k+1}$; $\lambda_k := 0$; $\mu_k := 0$; $\xi_k := b_{n-k-1}$; $\eta_k := b_{n-k}$; all for $0 \leq k \leq n$ (we assume that $t_{-(n+1)} = t_{n+1} = b_{-1} = 0$ to cover end-conditions). Clearly this can be done in time $O(n)$ if T and $\mathbf{b}$ are available at either end of the systolic array.

We present the program executed by processor S_k ($0 \leq k \leq n$) at time step τ $1 \leq \tau \leq 4n$) in Figure 7. The final solution $\mathbf{x}$ is given by $x_k = \xi_k$, where ξ_k is stored in register ξ of processor S_k after step $4n$. What follows are some observations concerning the program:

1. Processor S_k is active only if $k < \tau < 2n-k$ (Phase 1) or $2n+k \leq \tau \leq 4n-k$ (Phase 2). It is assumed that S_k knows its index k and the current value of τ (though this could be avoided by the use of 1-bit systolic control paths).
2. Pairs of adjacent processors could be combined, since only one processor of each pair is active at each time step. This would increase the mean processor utilization from 25% to 50% (see observation 1 above).
3. Processor S_0 performs floating-point divisions, other processors perform only additions and multiplications. A time step has to be long enough for six floating-point additions and multiplications, plus data transfers, during Phase 1 (less during Phase 2).
4. The Bareiss algorithm requires $4 \cdot 5n^2$ multiplications as given, but a simple modification (transmitting $1+\lambda\,\mu$) will give time $4n$ if we have $\lceil \frac{n}{2} \rceil$ processors (see observation 2), each with six multiply-add units. The corresponding figures for the symmetric Bareiss algorithm for symmetric matrices (see Section 7) are $4n^2$, $4n$, and 5.
5. An alternative for Phase 2 is the use of the Gohberg-Semencul formula [11]. But the formula is more expensive in terms of both operations and time, and it also fails to take advantage of the possible band structure of the matrix.

6. Processor S_k typically reads its input lines <u>inL1</u>, . . . , <u>inR3</u>, does some floating-point computations, and writes to its output lines <u>outL1</u>, . . . , <u>outR2</u>. Hence, pairs of input and output lines could be combined into single bidirectional lines (e.g. <u>inL1</u> and <u>outL1</u> could be combined).

```
{program for processor $k$ at time step $\tau$, $0 \le k \le n$, $1 \le \tau \le 4n$}
if odd $(\tau+k)$ and $(\tau>k)$ and $(\tau<2n-k)$ then {Phase 1-LU factorization}
  begin
  if $\tau>k+1$ then {accept inputs from processor $k+1$}
    begin $\alpha$ := inR1; $\delta$ := inR2; $\xi$ := inR3 end;
  if $k = 0$ then {compute multiplier} $\lambda := \alpha / \gamma$ else
    begin {accept multipliers from processor $k-1$}
    $\lambda$ := inL1; $\mu$ := inL2;
    $\alpha := \alpha - \lambda*\gamma$
    end;
  $\beta := \beta - \lambda*\delta$; $\eta := \eta - \lambda*\xi$
  if $k = 0$ then {compute multiplier} $\mu := \delta / \beta$ else
    begin
    $\gamma := \gamma - \mu*\alpha$;
    $\delta := \delta - \mu*\beta$;
    $\xi := \xi - \mu*\eta$
    end;
  outL1 := $\alpha$; outL2 := $\delta$; outL3 := $\xi$; {ignore outL1-3 if $k = 0$}
  outR1 := $\lambda$; outR2 := $\mu$                  {ignore outR1-2 if $k = n$}
  end
else if even $(\tau+k)$ and $(\tau \ge 2n+k)$ and $(\tau \le 4n-k)$ then {Phase 2-back substitution}
  begin
  if $\tau>2n+k$ then begin $\lambda$ := inR1; $\mu$ := inR2; $\eta$ := inR3 end;
  if $k = 0$ then begin $\xi := \eta/\beta$; $\delta := \mu*\beta$ end else
    begin
    $\xi$ := inL1; $\delta$ := inL2;
    $\eta := \eta - \beta*\xi$; $\delta := \delta + \mu*\beta$
    end;
  $\beta := \beta + \lambda*\delta$;
  outL1 := $\lambda$; outL2 := $\mu$; outL3 := $\eta$; {ignore if $k = 0$}
  outR1 := $\xi$; outR2 := $\delta$                 {ignore if $k$ = n}
  end.
```

Figure 7. Systolic processor for Toeplitz equation solver.

7. SYMMETRIC TOEPLITZ MATRICES

For a symmetric matrix T we may use a symmetric version of the Bareiss algorithm [2]. This symmetric algorithm is the same in its LU-factorization phase as the procedure proposed by S. Y. Kung and Hu ([16], [17]). The symmetric Bareiss algorithm is same as (2.5) except that

$$
\begin{aligned}
m_{-i} &= \frac{t_{i,0}^{(1-i)}}{t_{0,0}^{(i-1)}}, \\
m_i &= \frac{t_{0,i}^{(i-1)}}{t_{n,n}^{(1-i)}}, \qquad (7.1) \\
T^{(i)} &= T^{(i-1)} - m_i Z_i T^{(1-i)}.
\end{aligned}
$$

By symmetry, we get $\alpha=\delta$, $\beta=\gamma$ and $\lambda=\mu$ (cf. [2] and [26]), and so we save some storage and communication cost. The penalty is that Phase 2 of the algorithm becomes trickier and involves division by $(1-\lambda^2)$, which may be undesirable from a numerical point of view.

With the initialization:

$$
\begin{aligned}
\alpha_k &= \begin{cases} t_{k+1} & \text{if } 0 \le k < n, \\ 0 & \text{if } k = n \end{cases} \\
\beta_k &= t_k \qquad 0 \le k \le n, \\
\lambda_k &= 0 \qquad \text{''} \\
\mu_k &= 0 \qquad \text{''} \\
\xi_k &= \begin{cases} b_{n-k-1} & \text{if } 0 \le k \le n\ , \\ 0 & \text{if } k = n \end{cases} \\
\eta_k &= b_{n-k} \qquad 0 \le k < n\ ,
\end{aligned}
$$

We present a program for processor S_k in Figure 8.

```
{program for processor $k$ at time step $\tau$, $0 \le k \le n$, $1 \le \tau \le 4n$}
if odd $(\tau+k)$ and $(\tau>k)$ and $(\tau<2n-k)$ then {Phase 1 - LU factorization}
  begin
  if $\tau>k+1$ then {accept inputs from processor $k+1$}
      begin
      $\alpha$ := inR1. $\xi$ := in R2
      end;
  if $k = 0$ then {compute multiplier}
      begin
      $\lambda$ := $\alpha / \beta$; $\beta$ := $\beta-\lambda*\alpha$; $\eta$ := $\eta-\lambda*\xi$
      end
  else
      begin {accept multiplier from processor $k-1$}
      $\lambda$ := inL1;
      {$\Pi$ is temporarily local to processor $k$}
      $\Pi$ := $\alpha$; $\alpha$ := $\alpha-\lambda*\beta$;
      $\beta$ := $\beta-\lambda*\Pi$; {i.e. $\beta := \beta(1-\lambda^2)-\lambda\alpha$}
      $\Pi$ := $\eta$; $\eta$ := $\eta-\lambda*\xi$;
      $\xi$ := $\xi-\lambda*\Pi$
      end;
  outL1 := $\alpha$; outL2 := $\xi$; {ignore if $k$=0}
  outR1 := $\lambda$                 {ignore if $k = n$}
  end
else if even $(\tau+k)$ and $(\tau \ge 2n+k)$ and $(\tau \le 4n-k)$ then {Phase 2 - back substitution}
  begin
  if $\tau>2n+k$ then
      begin
      $\lambda$ := inR1; $\eta$ := inR2
      end;
  if $k$ =0 then
      begin
      $\xi$ := $\eta / \beta$; $\alpha$ := 0
      end
  else
      begin
      $\xi$ := inL1; $\alpha$ := inL2; $\eta$ := $\eta-\beta*\xi$
      end;
  $\alpha$ := $(\alpha+\lambda*\beta)/((1-\lambda)*(1+\lambda))$;
  $\beta$ := $\beta+\lambda*\alpha$;
  outL1 := $\lambda$; outL2 := $\eta$;
  outR1 := $\xi$; outR2 := $\alpha$
  end.
```

Figure 8. Systolic processor for symmetric Toeplitz equation solver.

8. ACKNOWLEDGEMENT

The authors would like to thank C. Pottle for pointing out reference [16].

9. REFERENCES

[1] H. A. Ahmed, J-M. Delosme, and M. Morf, Highly concurrent computing structures for matrix arithmetic and signal processing, *IEEE Computer 15* (1982), 65−82.

[2] E. H. Bareiss, Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices, *Numer. Math. 13* (1969), 404−424.

[3] R. R. Bitmead and B. D. O. Anderson, Asymptotically fast solution of Toeplitz and related systems of linear equations, *Lin. Alg. Applics. 34* (1980), 103−116.

[4] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun, Fast solution of Toeplitz systems of equations and computation of Pade approximants, *J. Algorithms 1* (1980), 259−295.

[5] R. P. Brent and F. T. Luk, A systolic architecture for almost linear-time solution of the symmetric eigenvalue problem, Technical Report TR-CS-82-10, Department of Computer Science, Australian National University (1982).

[6] A. Buckley, On the solution of certain skew symmetric linear systems, *SIAM J. Numer. Anal. 14* (1977), 566−570.

[7] G. Cybenko, The numerical stability of the Levinson-Durbin algorithm for Toeplitz systems of equations, *SIAM J. Sci. Statist. Comput. 1* (1980), 303−320.

[8] P. Delsarte, Y. Genin, and Y. Kamp, Schur parameterization of positive definite block-Toeplitz systems, *SIAM J. Appl. Math 36* (1979), 34−36.

[9] P. Dewilde, A. Vieira, and T. Kailath, On a generalized Szegö-Levinson realization algorithm for optimal linear predictors based on a network synthesis approach, *IEEE Trans. Circuits and Systems CAS-25* (1978), 663−675.

[10] J. Durbin, The fitting of time-series models, *Rev. Inst. Internat. Statist. 28* (1960), 233−244.

[11] I. C. Gohberg and A. A. Semencul, On the inversion of finite Teoplitz matrices and their continuous analogs, *Mat. Issled 2* (1972), 201−233. (In Russian)

[12] J. Grcar and A. Sameh, On certain parallel linear system solvers, *SIAM J. Sci. Statist. Comput. 2* (1981), 238−256.

[13] J. H. Justice, The Szegö recursion relation and inverses of positive definite Toeplitz matrices, *SIAM J. Math. Anal. 5* (1974), 503−508.

[14] T. Kailath, A. Vieira, and M. Morf, Inverses of Toeplitz operators, innovations, and orthogonal polynomials, *SIAM Review 20* (1978), 106−119.

[15] H. T. Kung and C. E. Leiserson, Algorithms for VLSI processor arrays, in *Introduction to VLSI Systems* (by C. Mead and L. Conway), Addison-Wesley, Reading, Massachusetts (1980), 271−292.

[16] S. Y. Kung and Y. H. Hu, Fast and parallel algorithms for solving Toeplitz systems, *Proc. Internat. Symp. on Mini- and Micro-computers in Control and Measurement*, San Francisco, California (May 1981).

[17] S. Y. Kung, Impact of VLSI on modern signal processing, *Proc. USC Workshop on VLSI and Modern Signal Processing*, Los Angeles, California (November 1982), 123−132.

[18] H. Lev-Ari and T. Kailath, Schur and Levinson algorithms for nonstationary processes, *Proc. IEEE Internat. Conf. on Acoustics, Speech, and Signal Processing*, New Jersey (1981), 860−864.

[19] N. Levinson, The Wiener RMS (root mean square) error criterion in filter design and prediction, *J. Math. Phys. 25* (1947), 261−278.

[20] M. Morf, Doubling algorithms for Toeplitz and related equations, *Proc. IEEE Internat. Conf. on Acoustics, Speech, and Signal Processing*, Denver, Colorado (April 1980), 954−959.

[21] M. Morf and J-M. Delosme, Matrix decompositions and inversions via elementary signature-orthogonal transformations, *Proc. Internat. Symp. on Mini- and Micro-computers in Control and Measurement*, San Francisco, California (May 1981).

[22] J. G. Nash and G. R. Nudd, Concurrent VLSI architectures for two-dimensional signal processing systems, *Proc. USC Workshop on VLSI and Modern Signal Processing*, Los Angeles, California (November 1982), 166–173.

[23] J. Schur, Über Potenzreihen, die im Innern des Einheitskreises beschränkt sind, *J. für die Reine und Angewandte Mathematik 147* (1917), 205–232.

[24] H. Sexton, An analysis of an algorithm of Bitmead and Anderson for the inversion of Toeplitz systems, manuscript.

[25] J. M. Speiser and H. J. Whitehouse, Architectures for real-time matrix operations, *Proc. Government Microcircuit Applications Conf.*, Houston, Texas (1980).

[26] D. R. Sweet, Numerical methods for Toeplitz matrices, Ph.D. Thesis, Department of Computer Science, University of Adelaide, May 1982.

[27] G. Szegö, *Orthogonal Polynomials*, third edition, AMS Colloquium Pub. vol. 23, AMS, Providence, Rhode Island (1967).

[28] W. F. Trench, An algorithm for the inversion of finite Toeplitz matrices, *J. Soc. Indust. Appl. Math 12* (1964), 515–522.

[29] S. Zohar, Toeplitz matrix inversion: the algorithm of W. F. Trench, *J. Assoc. Comput. Mach. 16* (1969), 592–601.