Abstract An algorithm based on hyperbolic rotations is presented for the solution of linear systems of equations,

$$Ax = b,$$

with symmetric positive definite coefficient matrix A. Forward elimination and backsubstitution are replaced by matrix vector multiplications, rendering the method amenable to implementation on a variety of parallel and vector machines.

The method can be simplified and formulated without square-roots if A is also Toeplitz; a systolic (VLSI) architecture implementing the resulting recurrence equations is more efficient than previously proposed pipelined Toeplitz system solvers. The hardware count becomes independent of the matrix size if its inverse is banded.

Parallel Solution of Symmetric Positive Definite Systems with Hyperbolic Rotations

Jean-Marc Delosme[1], Ilse C.F. Ipsen[2]

Research Report YALEU/DCS/RR-341
September 1985

[1] Department of Electrical Engineering, Yale University
[2] Department of Computer Science, Yale University

The work presented in this paper was supported by the Office of Naval Research under contracts N000014-82-K-0184 and N00014-84-K-0092.

Abstract An algorithm based on hyperbolic rotations is presented for the solution of linear systems of equations,

$$Ax = b,$$

with symmetric positive definite coefficient matrix A. Forward elimination and backsubstitution are replaced by matrix vector multiplications, rendering the method amenable to implementation on a variety of parallel and vector machines.

The method can be simplified and formulated without square-roots if A is also Toeplitz; a systolic (VLSI) architecture implementing the resulting recurrence equations is more efficient than previously proposed pipelined Toeplitz system solvers. The hardware count becomes independent of the matrix size if its inverse is banded.

Parallel Solution of Symmetric Positive Definite Systems with Hyperbolic Rotations

Jean-Marc Delosme[1], Ilse C.F. Ipsen[2]

Research Report YALEU/DCS/RR-341
September 1985

[1] Department of Electrical Engineering, Yale University
[2] Department of Computer Science, Yale University

The work presented in this paper was supported by the Office of Naval Research under contracts N000014-82-K-0184 and N00014-84-K-0092.

1. Introduction

This paper presents the derivation and discusses parallel implementations of an algorithm for the solution of linear systems of equations,

$$Ax = b, \tag{1.1}$$

with symmetric positive definite coefficient matrix A.

In order to solve (1.1), the Cholesky factor of A is first determined by essentially premultiplying A with appropriate hyperbolic rotations, whose product will be called Q (succinct descriptions of this step and of a possible concurrent implementation appeared earlier in [1, 17]). Next, simple matrix vector multiplications, involving Q, applied to the righthand side of (1.1) provide a novel way of solving the system. The ensuing avoidance of forward elimination and backsubstitution is a very desirable feature for implementation on a variety of parallel architectures.

Our method for the solution of (1.1), related to the one in [11], is particularly appealing since it does not impose as much sequentiality in its solution of linear systems as standard methods. The matrix A, rather than its inverse [11], is factored. Thereby the potential for parallelism in the method of [11] is exploited, yet at the same time its difficulties with pipelining of successive operations are eliminated. When A is in addition a Toeplitz matrix the algorithm in [11] reduces to the classical Levinson algorithm [16] and similarly, our algorithm simplifies to a more easily pipelinable procedure whose first, Cholesky factorization, step is essentially an old algorithm due to I. Schur, see [10].

Several highly concurrent systolic architectures for the solution of Toeplitz systems have recently been presented [3, 4, 8, 14, 18] (for an introduction to systolic architectures the reader is referred to [15]). All the proposed implementations start with the pipelined factorization of A via variants of Schur's algorithm (cf. [1, 17]). They differ, however, in the way the results of the first step are used to determine the solution vector. This paper demonstrates that substantial savings in the number of processing units and amount of memory are possible if full advantage is taken of the parametrization in terms of the intermediate quantities which are computed by the Schur algorithm. Systolic architectures with high processor utilization will be presented, for matrices A of bounded dimension and for arbitrarily sized A with banded inverse.

Since the algorithms proposed for the solution of (1.1) are based on hyperbolic rotations, one might be concerned about their numerical behavior. Stability analyses of Schur's and Durbin's algorithms, the latter being a special case of Levinson's algorithm, have been presented in [5, 6, 7]. According to A. Bultheel [5, 6], Durbin's and Schur's algorithms are stable for matrices with bounded condition number; the analysis further suggests a slight preference for Schur's method over Durbin's. G. Cybenko reaches the stronger conclusion that the Durbin [7] and Levinson [12] algorithms are essentially as stable as the Cholesky factorization algorithm. Work is in progress to refine these results and extend them to the general non-Toeplitz case.

2. The Hyperbolic Cholesky Factorization

The computation of the Cholesky decomposition,

$$A = U^T U, \qquad U \text{ is } n \times n \text{ upper triangular}, \tag{2.1}$$

of a real symmetric positive definite (spd) $n \times n$ matrix $A = (a_{ij})$ by means of hyperbolic rotations is called 'hyperbolic Cholesky factorization'. Its derivation is based on a particular decomposition of the matrix A :

$$A = \sum_{k=1}^{n} A^{(k)},$$

where $A^{(k)}$ has elements

$$a_{ij}^{(k)} = \begin{cases} a_{ij} & i = k,\ j \geq i \text{ or } j = k,\ i \geq j \\ 0 & \text{otherwise} \end{cases},$$

i.e., its nonzero elements form a 'composing stick' with vertex at the kth diagonal element. Since A is spd, a_{kk} is strictly positive and $A^{(k)}$ can be written as the difference of outer products

$$A^{(k)} = v_k^T v_k - w_k^T w_k,$$

where v_k and w_k are row vectors with elements

$$v_{kj} = \begin{cases} a_{kk}^{-1/2} a_{kj} & j \geq k \\ 0 & \text{otherwise} \end{cases}, \qquad w_{kj} = \begin{cases} v_{kj} & j \neq k \\ 0 & j = k \end{cases}. \tag{2.2}$$

That is, v_k consists of the nonzero row of $A^{(k)}$ scaled by the square-root of the diagonal element, while w_k differs from v_k only in its kth entry. Stacking the v_k and w_k, respectively, in upper triangular matrices

$$V = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}, \quad W = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix},$$

one has

$$A = V^T V - W^T W. \tag{2.3}$$

Identification of equations (2.1) and (2.3) yields

$$\begin{pmatrix} U^T & 0 \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} \begin{pmatrix} U \\ 0 \end{pmatrix} = \begin{pmatrix} V^T & W^T \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} \begin{pmatrix} V \\ W \end{pmatrix},$$

I being the $n \times n$ identity matrix.

Definition 2.1. A $2m \times 2m$ matrix Θ is called *pseudo orthogonal* if it satisfies

$$\Theta^T \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} \Theta = \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix},$$

where I is the $m \times m$ identity matrix.

It will be shown, that there exists a $2n \times 2n$ pseudo orthogonal matrix Q, obtained as a product of hyperbolic rotations acting on pairs of rows, so that

$$Q \begin{pmatrix} V \\ W \end{pmatrix} = \begin{pmatrix} U \\ 0 \end{pmatrix}. \tag{2.4}$$

Before construction of Q (in Theorem 2.1), the next two lemmata assure that the hyperbolic rotations to be applied to $(V^T \quad W^T)^T$ exist and are well defined.

Lemma 2.1. *If R and S are upper triangular $n \times n$ matrices such that $R^T R - S^T S$ is positive definite then R is invertible and*

$$|s_{kk} r_{kk}^{-1}| < 1, \qquad 1 \le k \le n.$$

Proof. Assume $R^T R - S^T S$ were positive definite and R singular. Then there would exist a non-zero vector x with $Rx = 0$ and

$$x^T(R^T R - S^T S)x = -(Sx)^T Sx \le 0,$$

which would contradict the assumption. Thus R is invertible.

Now, to establish the inequality, consider the matrix $P \equiv I - T^T T$ with $T \equiv SR^{-1}$. Since T is upper triangular,

$$t_{kk} = s_{kk} r_{kk}^{-1}, \qquad 1 \le k \le n,$$

and

$$p_{kk} = 1 - \sum_{i=1}^{n} t_{ik}^2 = 1 - \sum_{i=1}^{k-1} t_{ik}^2 - (s_{kk} r_{kk}^{-1})^2.$$

Since P is congruent to $R^T R - S^T S$, it is spd. Hence p_{kk} is strictly positive, implying

$$(s_{kk} r_{kk}^{-1})^2 < 1 - \sum_{i=1}^{k-1} t_{ik}^2 \le 1.$$

∎

Lemma 2.2. *Let R and S be upper triangular $n \times n$ matrices such that $R^T R - S^T S$ is positive definite, and let $\rho_k \equiv s_{kk} r_{kk}^{-1}$, $1 \le k \le n$. If*

$$\begin{pmatrix} \tilde{R} \\ \tilde{S} \end{pmatrix} = \hat{Q} \begin{pmatrix} R \\ S \end{pmatrix}, \qquad \text{where } \hat{Q} = \tilde{Q}^{(n)} \ldots \tilde{Q}^{(1)}$$

and

$$\tilde{q}_{ij}^{(k)} = \begin{cases} 1 & i = j \ne k \text{ or } i = j \ne n+k \\ (1-\rho_k^2)^{-1/2} & i = j = k \text{ or } i = j = n+k \\ -(1-\rho_k^2)^{-1/2}\rho_k & (i,j) = (k, n+k) \text{ or } (i,j) = (n+k, k) \\ 0 & \text{otherwise} \end{cases}.$$

Then $\hat{Q}$ is pseudo orthogonal, $\tilde{R}$ is upper triangular and $\tilde{S}$ is strictly upper triangular (upper triangular with zero diagonal).

Proof. According to Lemma 2.1, ρ_k is defined and satisfies $|\rho_k| < 1$, $1 \le k \le n$, so that $\hat{Q}$ is well defined. Since the $\tilde{Q}^{(k)}$ in the product $\hat{Q}$ are disjoint (each of them operates on a different set of rows), it is sufficient to prove that $\tilde{Q}^{(k)}$ is pseudo orthogonal and that $\tilde{r}_{kj} = 0$, $j < k$ and $\tilde{s}_{kj} = 0$, $j \le k$.

Checking that $\tilde{Q}^{(k)}$ is pseudo orthogonal reduces to checking that

$$H_k \equiv (1-\rho_k^2)^{-1/2} \begin{pmatrix} 1 & -\rho_k \\ -\rho_k & 1 \end{pmatrix}$$

is pseudo orthogonal, i.e.,

$$H_k{}^T \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} H_k = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

which is easily seen. In fact, H_k is just a hyperbolic rotation

$$H_k = \begin{pmatrix} \cosh\phi_k & \sinh\phi_k \\ \sinh\phi_k & \cosh\phi_k \end{pmatrix}, \qquad \phi_k = -\tanh^{-1}\rho_k.$$

Rows k of $\tilde{R}$, $\tilde{S}$ are related to rows k of R, S via

$$\begin{pmatrix} \tilde{r}_{kj} \\ \tilde{s}_{kj} \end{pmatrix} = H_k \begin{pmatrix} r_{kj} \\ s_{kj} \end{pmatrix}, \qquad 1 \leq j \leq n.$$

Therefore one has

$$\begin{pmatrix} \tilde{r}_{kk} \\ \tilde{s}_{kk} \end{pmatrix} = (1-\rho_k^2)^{-1/2} \begin{pmatrix} r_{kk} - \rho_k s_{kk} \\ s_{kk} - \rho_k r_{kk} \end{pmatrix} = \begin{pmatrix} \operatorname{sign}(r_{kk})(r_{kk}^2 - s_{kk}^2)^{1/2} \\ 0 \end{pmatrix}$$

as well as $\tilde{r}_{kj} = \tilde{s}_{kj} = 0$ for $j < k$, since $r_{kj} = s_{kj} = 0$ for $j < k$, thus proving that $\tilde{R}$ is upper triangular and $\tilde{S}$ is strictly upper triangular.

∎

Remarks

- Since the matrices $\tilde{Q}^{(k)}$ constitute disjoint rotations they commute and can be applied in any order. Their product $\hat{Q}$ has a very simple expression :

$$\begin{aligned} \hat{q}_{kk} &= \hat{q}_{n+k,n+k} = (1-\rho_k^2)^{-1/2}, & 1 \leq k \leq n \\ \hat{q}_{k,n+k} &= \hat{q}_{n+k,k} = -(1-\rho_k^2)^{-1/2}\rho_k, & 1 \leq k \leq n. \\ \hat{q}_{ij} &= 0 \qquad i \neq j(\operatorname{mod} n) & \end{aligned}$$

- The diagonal elements of $\tilde{R}$ have the same sign as the corresponding diagonal elements of R, thus if R has a positive diagonal, $\tilde{R}$ also has a positive diagonal.

Theorem 2.1. *The Hyperbolic Cholesky Algorithm*
Let A be a $n \times n$ spd matrix and V and W be upper triangular matrices as defined in (2.2) so that $A = V^TV - W^TW$. Set

$$\begin{pmatrix} R^{(0)} \\ S^{(0)} \end{pmatrix} = \begin{pmatrix} V \\ W \end{pmatrix}.$$

and apply the sequence of operations

$$\begin{pmatrix} R^{(l+1)} \\ S^{(l+1)} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \hat{Q}^{(l)} \begin{pmatrix} R^{(l)} \\ S^{(l)} \end{pmatrix}, \qquad l = 0 \ldots n-1,$$

where $\hat{Q}^{(l)}$ is obtained from $R^{(l)}$ and $S^{(l)}$ the same way $\hat{Q}$ is constructed from R and S in Lemma 2.2, and where P is the $n \times n$ circular permutation matrix with $p_{1,n} = 1$ and $p_{i,i-1} = 1$, $2 \leq i \leq n$. Then $R^{(n)} = U$, the Cholesky factor of A, and $S^{(n)} = 0$.

Proof. The proof proceeds by induction. Define the following property (P_l) :

$(P_l.1)$ $R^{(l)^T} R^{(l)} - S^{(l)^T} S^{(l)} = A$,

$(P_l.2)$ $R^{(l)}$ is $n \times n$ upper triangular with positive diagonal elements,

(P_l.3) $S^{(l)}$ is $n \times n$ upper triangular with l leading zero rows.

(P_0) is easily shown to hold. If the inference (P_l) *is true implies* (P_{l+1}) *is true* is correct then (P_n) will be true, i.e., $S^{(n)} = 0$ and $R^{(n)} = U$, by uniqueness of the Cholesky decomposition. It remains to be demonstrated that if $R^{(l)}$ and $S^{(l)}$ satisfy (P_l) then $\hat{Q}^{(l)}$ exists, and $R^{(l+1)}$ and $S^{(l+1)}$ satisfy (P_{l+1}).

Let

$$\begin{pmatrix} \tilde{R}^{(l)} \\ \tilde{S}^{(l)} \end{pmatrix} = \hat{Q}^{(l)} \begin{pmatrix} R^{(l)} \\ S^{(l)} \end{pmatrix}.$$

Since $R^{(l)}$ and $S^{(l)}$ are upper triangular and (P_l.1) holds with A spd, it follows from Lemma 2.2 that a pseudo orthogonal $\hat{Q}^{(l)}$ exists, that $\tilde{R}^{(l)}$ is upper triangular with positive diagonal elements (because of (P_l.2)) and that $\tilde{S}^{(l)}$ is strictly upper triangular. Now the following properties hold :

(P_{l+1}.1) Since $\hat{Q}^{(l)}$ is pseudo orthogonal,

$$\begin{pmatrix} \tilde{R}^{(l)T} & \tilde{S}^{(l)T} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} \begin{pmatrix} \tilde{R}^{(l)} \\ \tilde{S}^{(l)} \end{pmatrix} = \begin{pmatrix} R^{(l)T} & S^{(l)T} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} \begin{pmatrix} R^{(l)} \\ S^{(l)} \end{pmatrix},$$

thus

$$\tilde{R}^{(l)T}\tilde{R}^{(l)} - \tilde{S}^{(l)T}\tilde{S}^{(l)} = A.$$

From $R^{(l+1)} = \tilde{R}^{(l)}$, $S^{(l+1)} = P\tilde{S}^{(l)}$ and P orthogonal one obtains (P_{l+1}.1).

(P_{l+1}.2) Since $R^{(l+1)} = \tilde{R}^{(l)}$, $R^{(l+1)}$ is upper triangular with positive diagonal elements.

(P_{l+1}.3) As $\tilde{S}^{(l)}$ is strictly upper triangular, $S^{(l+1)}$, obtained by moving the last row of $\tilde{S}^{(l)}$ into the topmost position and moving down the other rows by one position, is upper triangular with first row identical to zero. Now, since $S^{(l)}$ has l leading zero rows, the associated $\rho_k^{(l)} = s_{kk}^{(l)}/r_{kk}^{(l)}$ are zero, $1 \leq k \leq l$, so that rows 1 to l of $\tilde{S}^{(l)}$ are identical to rows 1 to l of $S^{(l)}$, i.e., they are zero. Hence, rows 2 to $l+1$ of $S^{(l+1)}$ are zero. Therefore $S^{(l+1)}$ is upper triangular with $l+1$ leading zero rows.

∎

Remarks

- Since $S^{(0)}$ is strictly upper triangular, $\hat{Q}^{(0)}$ is the $2n \times 2n$ identity matrix.
- In step l $\hat{Q}^{(l)}$ removes the diagonal of $S^{(l)}$.
- The further the reduction of S to zero progresses, the more rotations $H_k^{(l)}$ are identities ($\rho_k^{(l)} = 0$). That is, in step l rows 1 to l and $n+1$ to $n+l$ of $\hat{Q}^{(l)}$ are equal to the corresponding rows of the identity matrix. This implies that rows 1 to l of $R^{(l)}$ are identical to rows 1 to l of U. In other words, step l determines the $(l+1)$st row of U, equal to the $(l+1)$st row of $R^{(l+1)}$.

The transformation performed by the hyperbolic Cholesky algorithm will be denoted by

$$Q = \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \hat{Q}^{(n-1)} \cdots \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \hat{Q}^{(l)} \cdots \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \hat{Q}^{(1)} \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \hat{Q}^{(0)}.$$

The matrix Q is pseudo orthogonal since it is the product of pseudo orthogonal matrices.

3. An Explicit Expression for Q

In the next section it will be shown how to avoid backsubstitution as well as forward elimination by performing matrix vector multiplications with the matrix Q. To this end, it is necessary to derive an explicit expression for Q.

Given a $n \times n$ spd matrix A, the $2n \times 2n$ pseudo orthogonal matrix Q is uniquely defined since it is constructed in a unique fashion by the hyperbolic Cholesky algorithm. Thus one could expect that there exists a closed form expression for Q in terms of A. Before being able to exhibit such an expression however, another characterization for A of the same type as (2.3) is needed. Write

$$A = \sum_{k=1}^{n} \tilde{A}^{(k)},$$

where $\tilde{A}^{(k)}$ has elements

$$\tilde{a}_{ij}^{(k)} = \begin{cases} a_{ij} & i = k,\, j \leq i \text{ or } j = k,\, i \leq j \\ 0 & \text{otherwise} \end{cases}.$$

Since A is spd, a_{kk} is strictly positive and analogously to (2.2) $\tilde{A}^{(k)}$ can be written as the difference of outer products

$$\tilde{A}^{(k)} = m_k^T m_k - n_k^T n_k,$$

where m_k and n_k are row vectors with elements

$$m_{kj} = \begin{cases} a_{kk}^{-1/2} a_{kj} & j \leq k \\ 0 & \text{otherwise} \end{cases}, \quad n_{kj} = \begin{cases} m_{kj} & j \neq k \\ 0 & j = k \end{cases}.$$

Again, m_k consists of the nonzero row of $\tilde{A}^{(k)}$ scaled by the square-root of the diagonal element, while n_k differs from m_k only in its kth entry. Stacking the m_k and n_k, respectively, in lower triangular matrices

$$M = \begin{pmatrix} m_1 \\ \vdots \\ m_n \end{pmatrix}, \quad N = \begin{pmatrix} n_1 \\ \vdots \\ n_n \end{pmatrix},$$

one has

$$A = M^T M - N^T N.$$

The Cholesky decomposition of A into $A = U^T U$, where U is a $n \times n$ upper triangular matrix with strictly positive diagonal elements may be called more specifically a *lower-upper* Cholesky decomposition (i.e., the decomposition consists of a lower times an upper triangular matrix). The upper Cholesky factor U is unique. It is easily seen (e.g., by regarding the matrix JAJ where J is the permutation matrix with ones on the antidiagonal) that A also admits to an *upper-lower* Cholesky decomposition as $A = L^T L$, L being a $n \times n$ lower triangular matrix with strictly positive diagonal elements that is also unique. The following two lemmata are needed in order to derive the expression for Q given in Theorem 3.1.

Lemma 3.1. *The matrix*

$$Q^* = \begin{pmatrix} (VU^{-1})^T & -(WU^{-1})^T \\ -(NL^{-1})^T & (ML^{-1})^T \end{pmatrix}$$

satisfies

$$Q^* \begin{pmatrix} V \\ W \end{pmatrix} = \begin{pmatrix} U \\ 0 \end{pmatrix}$$

and Q^{*T} is pseudo orthogonal.

Proof. First observe that U and L are nonsingular, since A is spd and hence nonsingular. Thus Q^* is well defined. Application of Q^* to $(V^T \quad W^T)^T$ yields

$$Q^* \begin{pmatrix} V \\ W \end{pmatrix} = \begin{pmatrix} U^{-T}(V^T V - W^T W) \\ L^{-T}(-N^T V + M^T W) \end{pmatrix}.$$

Since

$$V^T V - W^T W = A = U^T U,$$

the top $n \times n$ block is U. On the other hand, letting

$$D = diag(a_{11}^{1/2} \dots a_{nn}^{1/2}), \qquad (3.1)$$

one has

$$M^T W - N^T V = (D + N^T)W - N^T(D + W) = DW - N^T D,$$

and, since $DW = N^T D$ is the strictly upper triangular part of A,

$$M^T W - N^T V = 0.$$

Hence,

$$Q^* \begin{pmatrix} V \\ W \end{pmatrix} = \begin{pmatrix} U \\ 0 \end{pmatrix}.$$

The pseudo orthogonality of Q^{*T} follows from

$$\begin{aligned} Q^* \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} Q^{*T} &= \begin{pmatrix} U^{-T}(V^T V - W^T W)U^{-1} & U^{-T}(-V^T N + W^T M)L^{-1} \\ L^{-T}(-N^T V + M^T W)U^{-1} & L^{-T}(N^T N - M^T M)L^{-1} \end{pmatrix} \\ &= \begin{pmatrix} U^{-T} A U^{-1} & 0 \\ 0 & -L^{-T} A L^{-1} \end{pmatrix}, \end{aligned}$$

and the factorizations $A = U^T U = L^T L$.

∎

Lemma 3.2. *Let $E^{(1)}$ and $F^{(1)}$ be $n \times n$ diagonal matrices and define $E^{(l)}$ and $F^{(l)}$ recursively :*

$$\begin{pmatrix} E^{(l+1)} \\ \tilde{F}^{(l+1)} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \hat{Q}^{(l)} \begin{pmatrix} E^{(l)} \\ \tilde{F}^{(l)} \end{pmatrix}, \qquad l = 1 \dots n-1,$$

where $\tilde{F}^{(l)} = P^l F^{(l)}$. Denote by $e_m^{(l)}$ and $f_m^{(l)}$ the (co-)diagonals of $E^{(l)}$ and $F^{(l)}$, respectively, $-(n-1) \leq m \leq n-1$. That is, the mth subdiagonal has subscript $-m$, the main diagonal subscript 0 and the mth superdiagonal subscript m. Then, for $l = 1 \dots n$,

$$\begin{cases} e_m^{(l)} = 0 & m \leq -l \text{ or } m > 0 \\ f_m^{(l)} = 0 & m < 0 \text{ or } m \geq l \end{cases}.$$

Proof. The proof proceeds by induction. The above is true for $l = 1$. If it is true for some l, $1 \leq l < n-1$, show that it holds for $l+1$ and the lemma is proved. According to Lemma 2.2,

$$\hat{Q}^{(l)} = \begin{pmatrix} D^{(l)} & \Delta^{(l)} \\ \Delta^{(l)} & D^{(l)} \end{pmatrix},$$

where $D^{(l)}$ is a $n \times n$ diagonal matrix and $\Delta^{(l)}$ is a $n \times n$ diagonal matrix with its first l diagonal entries equal to zero. Thus,

$$E^{(l+1)} = D^{(l)}E^{(l)} + \Delta^{(l)}\tilde{F}^{(l)} = D^{(l)}E^{(l)} + \Delta^{(l)}P^{l}F^{(l)},$$

and

$$\begin{aligned} F^{(l+1)} &= P^{n-l-1}\tilde{F}^{(l+1)} = P^{n-l-1}(P\Delta^{(l)}E^{(l)} + PD^{(l)}\tilde{F}^{(l)}) \\ &= P^{n-l}\Delta^{(l)}E^{(l)} + P^{n-l}D^{(l)}P^{l}F^{(l)} \end{aligned}.$$

Row i of $\Delta^{(l)}P^{l}F^{(l)}$ equals zero for $i \leq l$ and equals row $i-l$ of $F^{(l)}$ for $i > l$. Therefore, $\Delta^{(l)}P^{l}F^{(l)}$ has zero diagonals for $m < -l$ or $m \geq 0$. Since $D^{(l)}E^{(l)}$ has zero diagonals for $m \leq -l$ or $m > 0$, $e_m^{(l+1)} = 0$ for $m \leq -(l+1)$ or $m > 0$. Row i of $P^{n-l}\Delta^{(l)}E^{(l)}$ equals row $i+l$ of $E^{(l)}$ for $i \leq n-l$ and equals zero for $i > n-l$. Hence, $P^{n-l}\Delta^{(l)}E^{(l)}$ has zero diagonals for $m \leq 0$ or $m > l$. On the other hand, $P^{n-l}D^{(l)}P^{l}F^{(l)}$ has the same zero pattern as $F^{(l)}$. Therefore, $f_m^{(l+1)} = 0$ for $m < 0$ or $m \geq l+1$.

∎

Theorem 3.1. *The matrix Q satisfies*

$$Q = Q^* = \begin{pmatrix} (VU^{-1})^T & -(WU^{-1})^T \\ -(NL^{-1})^T & (ML^{-1})^T \end{pmatrix}.$$

Proof. Since Q and Q^* have full rank, one can write

$$Q = GQ^*,$$

and it will be shown that G is the identity. Let

$$G = \begin{pmatrix} G_1 & G_2 \\ G_3 & G_4 \end{pmatrix}$$

be the decomposition of G into $n \times n$ blocks. Since

$$Q\begin{pmatrix} V \\ W \end{pmatrix} = \begin{pmatrix} U \\ 0 \end{pmatrix} \quad \text{and} \quad Q^*\begin{pmatrix} V \\ W \end{pmatrix} = \begin{pmatrix} U \\ 0 \end{pmatrix},$$

$$G\begin{pmatrix} U \\ 0 \end{pmatrix} = \begin{pmatrix} U \\ 0 \end{pmatrix},$$

hence $G_1 = I$ and $G_3 = 0$.

In order to determine G_2 and G_4 the pseudo orthogonality of G is proved next. Indeed, since $G = Q(Q^*)^{-1}$,

$$\begin{aligned} G^T\begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}G &= (Q^*)^{-T}Q^T\begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}Q(Q^*)^{-1} = (Q^*)^{-T}\begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}(Q^*)^{-1} \\ &= (Q^*\begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}Q^{*T})^{-1} = \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} \end{aligned}$$

where the second and fourth equality, respectively, follow from the pseudo orthogonality of Q and of Q^{*T}. Consequently,

$$G_1^TG_2 - G_3^TG_4 = 0, \quad G_2^TG_2 - G_4^TG_4 = -I,$$

and, since $G_1 = I$ and $G_3 = 0$,

$$G_2 = 0 \quad \text{and} \quad G_4^T G_4 = I.$$

Therefore,

$$G = \begin{pmatrix} I & 0 \\ 0 & G_4 \end{pmatrix}$$

with G_4 orthogonal.

Now Lemma 3.2 is employed to prove that G_4 is the identity. Let

$$\begin{pmatrix} E^{(1)} \\ F^{(1)} \end{pmatrix} = \begin{pmatrix} 0 \\ I \end{pmatrix},$$

then

$$\begin{pmatrix} E^{(n)} \\ \tilde{F}^{(n)} \end{pmatrix} = Q \begin{pmatrix} 0 \\ I \end{pmatrix}, \quad \text{with} \quad Q = \begin{pmatrix} I & 0 \\ 0 & G_4 \end{pmatrix} Q^*,$$

so that

$$\tilde{F}^{(n)} = G_4 (ML^{-1})^T.$$

Since $E^{(1)}$ and $F^{(1)}$ are diagonal matrices, Lemma 3.2 applied to the case $l = n$ shows that the subdiagonals of $F^{(n)}$ are zero, i.e., $\tilde{F}^{(n)} = F^{(n)}$ is upper triangular. Since $(ML^{-1})^T$ is upper triangular as well and M is nonsingular,

$$G_4 = \tilde{F}^{(n)} (ML^{-1})^{-T}$$

is upper triangular. But $G_4^{-1} = G_4^T$, so that G_4 must be diagonal and its diagonal entries must be ± 1.

Since the main diagonal of $P^{n-l}\Delta^{(l)}E^{(l)}$ is zero, the diagonal of $\tilde{F}^{(n)}$ is just $PD^{(n-1)} \ldots PD^{(1)}P$. Now, the diagonal entries of $D^{(l)}$ are $(1 - \rho_k^{(l)})^{-1/2} > 0$, so that the diagonal entries of $\tilde{F}^{(n)}$ are strictly positive. Since the diagonal entries of M and L^{-1} are also strictly positive, the diagonal entries of G_4 must be strictly positive. Consequently, $G_4 = I$.

∎

4. Application of Hyperbolic Rotations to the Solution of Linear Systems

The Hyperbolic Cholesky algorithm determines simultaneously the Cholesky factor U of a spd matrix A and a set of $n(n-1)/2$ parameters $\rho_k^{(l)}$, $1 \leq l \leq n-1$, $l < k \leq n$, which define the hyperbolic rotations that make up the matrix Q. For the solution of a spd system $Ax = b$, the above algorithm could be used to find U followed by forward elimination to solve the system $U^T y = b$ and by backsubstitution to solve $Ux = y$. Instead, the above algorithm can also be used to find the parameters $\rho_k^{(l)}$ followed by the application of the hyperbolic rotations to the righthand side b in a particular way, described below, to get the solution vector x.

It is already known that

$$Q \begin{pmatrix} V \\ W \end{pmatrix} = \begin{pmatrix} U \\ 0 \end{pmatrix}. \tag{4.1}$$

Now consider the product

$$Q \begin{pmatrix} N \\ M \end{pmatrix} = \begin{pmatrix} U^{-T}(V^T N - W^T M) \\ L^{-T}(-N^T N + M^T M) \end{pmatrix}.$$

Since $M^T W - N^T V = 0$ and $M^T M - N^T N = A = L^T L$,

$$Q \begin{pmatrix} N \\ M \end{pmatrix} = \begin{pmatrix} 0 \\ L \end{pmatrix}. \tag{4.2}$$

Thus, the same hyperbolic rotations which are used to compute the upper Cholesky factor U can be applied to compute the lower Cholesky factor L. Adding up equations (4.1) and (4.2) yields

$$Q \begin{pmatrix} N+V \\ M+W \end{pmatrix} = \begin{pmatrix} U \\ L \end{pmatrix}.$$

Now $N + V = M + W = D^{-1}A$, where D is specified in (3.1), so that with

$$\Delta = \begin{pmatrix} D^{-1} & 0 \\ 0 & D^{-1} \end{pmatrix},$$

$$Q\Delta \begin{pmatrix} A \\ A \end{pmatrix} = \begin{pmatrix} U \\ L \end{pmatrix}.$$

Postmultiplying both sides by $A^{-1} = U^{-1}U^{-T} = L^{-1}L^{-T}$, results in

$$Q\Delta \begin{pmatrix} I \\ I \end{pmatrix} = \begin{pmatrix} U^{-T} \\ L^{-T} \end{pmatrix}. \tag{4.3}$$

Equation (4.3) demonstrates a novel way of solving $U^T y = b$ (and simultaneously $L^T z = b$) : to the vector $(b^T \quad b^T)^T$ apply $Q\Delta$, in other words, scale it by the $a_{ii}^{-1/2}$ and then multiply it by the rotations determined in the hyperbolic Cholesky algorithm,

$$\begin{pmatrix} y \\ z \end{pmatrix} \equiv Q\Delta \begin{pmatrix} b \\ b \end{pmatrix} = \begin{pmatrix} U^{-T} b \\ L^{-T} b \end{pmatrix}. \tag{4.4}$$

Now consider the transformation Q^T. Because the matrices $\hat{Q}^{(l)}$, $1 \leq l \leq n-1$, are symmetric and P is orthogonal, premultiplication by

$$Q^T = \hat{Q}^{(0)} \begin{pmatrix} I & 0 \\ 0 & P^{-1} \end{pmatrix} \hat{Q}^{(1)} \cdots \begin{pmatrix} I & 0 \\ 0 & P^{-1} \end{pmatrix} \hat{Q}^{(l)} \cdots \begin{pmatrix} I & 0 \\ 0 & P^{-1} \end{pmatrix} \hat{Q}^{(n-1)} \begin{pmatrix} I & 0 \\ 0 & P^{-1} \end{pmatrix}$$

is readily implementable once the parameters $\rho_k^{(l)}$, i.e., the matrices $\hat{Q}^{(l)}$, have been computed. Let y and z be two column vectors of length n,

$$Q^T\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} VU^{-1} & -NL^{-1} \\ -WU^{-1} & ML^{-1} \end{pmatrix}\begin{pmatrix} y \\ z \end{pmatrix}.$$

Consequently,

$$\Delta Q^T\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} D^{-1}VU^{-1}y - D^{-1}NL^{-1}z \\ -D^{-1}WU^{-1}y + D^{-1}ML^{-1}z \end{pmatrix}$$

and

$$(I \quad I)\,\Delta Q^T\begin{pmatrix} y \\ z \end{pmatrix} = D^{-1}(V-W)U^{-1}y + D^{-1}(M-N)L^{-1}z = U^{-1}y + L^{-1}z.$$

Therefore,

$$(I \quad I)\,\Delta Q^T\begin{pmatrix} \alpha y \\ (1-\alpha)z \end{pmatrix} = A^{-1}b$$

if

$$y = U^{-T}b \quad \text{and} \quad z = L^{-T}b.$$

The algorithm for the solution of $Ax = b$ can be summed up as follows : let A_+ be the upper triangular part of A and $A_\#$ its strictly upper triangular part ($A_+ = D^2 + A_\#$). Using the hyperbolic Cholesky algorithm as specified in Theorem 2.1, express the matrix Q as a product of hyperbolic rotations such that

$$Q\Delta\begin{pmatrix} A_+ \\ A_\# \end{pmatrix} = \begin{pmatrix} U \\ 0 \end{pmatrix}. \tag{4.5}$$

Apply the same operations to $\begin{pmatrix} b \\ b \end{pmatrix}$ to obtain

$$Q\Delta\begin{pmatrix} b \\ b \end{pmatrix} = \begin{pmatrix} U^{-T}b \\ L^{-T}b \end{pmatrix}. \tag{4.6}$$

Then apply these operations essentially in reverse order to get x from

$$x = (I \quad I)\,\Delta Q^T\begin{pmatrix} \alpha U^{-T}b \\ (1-\alpha)L^{-T}b \end{pmatrix}, \tag{4.7}$$

where α is an arbitrary real number, which can be selected equal to 0 or 1 for convenience.

Remark

If $\rho_k^{(l)} = 0$ for $l \geq p$, $\hat{Q}^{(l)}$ is the identity matrix for $l \geq p$ and the number of operations performed by the algorithm becomes proportional to n^2p instead of n^3. This property of the parameters $\rho_k^{(l)}$ has a simple interpretation in terms of the original matrix A. Let $E^{(1)} = F^{(1)} = D^{-1}$ and use the definitions and notations of Lemma 3.2. Clearly,

$$\begin{pmatrix} E^{(n)} \\ \tilde{F}^{(n)} \end{pmatrix} = Q\Delta\begin{pmatrix} I \\ I \end{pmatrix} = \begin{pmatrix} U^{-T} \\ L^{-T} \end{pmatrix}.$$

Since $E^{(1)}$ and $F^{(1)}$ are diagonal, Lemma 3.2 applies and, for $l = p$,

$$\begin{cases} e_m^{(p)} = 0 & m \leq -p \text{ or } m > 0 \\ f_m^{(p)} = 0 & m < 0 \text{ or } m \geq p \end{cases}.$$

Now, for $l \geq p$, $D^{(l)} = I$ and $\Delta^{(l)} = 0$ so that $E^{(l+1)} = E^{(l)}$ and $F^{(l+1)} = F^{(l)}$. Therefore,

$$\begin{cases} e_m^{(n)} = 0 & m \leq -p \text{ or } m > 0 \\ f_m^{(n)} = 0 & m < 0 \text{ or } m \geq p \end{cases},$$

i.e., U^{-1} and L^{-1} are banded with upper and lower, respectively, bandwidth p. Hence the *inverse* of the coefficient matrix A has bandwidth p.

5. Specialization to Toeplitz Matrices - Square-Root Free Algorithm

When the spd coefficient matrix A is Toeplitz ($a_{ij} \equiv a_{j-i}$) the algorithm described in Section 4 simplifies considerably as a result of the following property.

Lemma 5.1. *With the definitions and notations of Theorem 2.1, if A is Toeplitz then the matrices $R_*^{(l)}$ and $S_*^{(l)}$, defined to be rows $l+1$ to n of $R^{(l)}$ and $S^{(l)}$, respectively, are Toeplitz for $0 \leq l \leq n-1$.*

Proof. The proof proceeds by induction. The lemma is valid for $l = 0$ since V and W are easily seen to be Toeplitz. It will now be shown that if it is true for some l, $0 \leq l < n-1$, then it holds for $l+1$. Recall that

$$\begin{pmatrix} R^{(l+1)} \\ S^{(l+1)} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \begin{pmatrix} \tilde{R}^{(l)} \\ \tilde{S}^{(l)} \end{pmatrix}, \quad \text{with} \quad \begin{pmatrix} \tilde{R}^{(l)} \\ \tilde{S}^{(l)} \end{pmatrix} = \hat{Q}^{(l)} \begin{pmatrix} R^{(l)} \\ S^{(l)} \end{pmatrix},$$

and furthermore,

$$\hat{Q}^{(l)} = \begin{pmatrix} D^{(l)} & \Delta^{(l)} \\ \Delta^{(l)} & D^{(l)} \end{pmatrix},$$

where $D^{(l)}$ and $\Delta^{(l)}$ are $n \times n$ diagonal matrices. Denote by $\tilde{R}_*^{(l)}$, $D_*^{(l)}$, etc, the matrices formed by rows $l+1$ to n of $\tilde{R}^{(l)}$, $D^{(l)}$, etc. Clearly,

$$\begin{pmatrix} \tilde{R}_*^{(l)} \\ \tilde{S}_*^{(l)} \end{pmatrix} = \begin{pmatrix} D_*^{(l)} & \Delta_*^{(l)} \\ \Delta_*^{(l)} & D_*^{(l)} \end{pmatrix} \begin{pmatrix} R^{(l)} \\ S^{(l)} \end{pmatrix}$$

and, since the first l columns of $D_*^{(l)}$ and $\Delta_*^{(l)}$ are identically zero,

$$\begin{pmatrix} \tilde{R}_*^{(l)} \\ \tilde{S}_*^{(l)} \end{pmatrix} = \begin{pmatrix} \tilde{D}^{(l)} & \tilde{\Delta}^{(l)} \\ \tilde{\Delta}^{(l)} & \tilde{D}^{(l)} \end{pmatrix} \begin{pmatrix} R_*^{(l)} \\ S_*^{(l)} \end{pmatrix},$$

where $\tilde{D}^{(l)}$ and $\tilde{\Delta}^{(l)}$ are the diagonal matrices formed by columns $l+1$ to n of $D_*^{(l)}$ and $\Delta_*^{(l)}$. By assumption, $R_*^{(l)}$ and $S_*^{(l)}$ are Toeplitz, so $r_{kk}^{(l)}$ and $s_{kk}^{(l)}$ do not depend on k for $l < k \leq n$. Therefore, $\rho_k^{(l)} = s_{kk}^{(l)}/r_{kk}^{(l)}$ is independent of k for $l < k \leq n$ and $\tilde{D}^{(l)}$ and $\tilde{\Delta}^{(l)}$ are diagonal Toeplitz matrices. The products of $\tilde{D}^{(l)}$ and $\tilde{\Delta}^{(l)}$ with the Toeplitz matrices $R_*^{(l)}$ and $S_*^{(l)}$ are Toeplitz and so are $\tilde{R}_*^{(l)}$ and $\tilde{S}_*^{(l)}$ which are sums of such products. Finally, since $R_*^{(l+1)}$ and $S_*^{(l+1)}$ are obtained by omitting the first row of $\tilde{R}_*^{(l)}$ and the last row of $\tilde{S}_*^{(l)}$, respectively, they are Toeplitz matrices, too.

∎

Because the matrices $R_*^{(l)}$ and $S_*^{(l)}$ are upper triangular Toeplitz, they are fully determined from their first row by a trivial extension. Thus, the computation of matrices $R_*^{(l+1)}$ and $S_*^{(l+1)}$, given $R_*^{(l)}$ and $S_*^{(l)}$, reduces to updating only the first row of $R_*^{(l)}$ and $S_*^{(l)}$.

The hyperbolic Cholesky algorithm thus simplifies to a Toeplitz-specific algorithm, the so-called 'Schur algorithm', with an operation count proportional to n^2 instead of n^3 and, in case A^{-1} has bandwidth p, to np instead of n^2p. The matrix vector products involving Q or Q^T that replace forward elimination and backsubstitution in the solution of $Ax = b$ require as many operations for a Toeplitz matrix as for an arbitrary coefficient matrix. However, the transformations $\hat{Q}^{(l)}$ depend on only one parameter $\rho^{(l)} \equiv \rho_k^{(l)}$, $l < k \leq n$, determined by the Schur algorithm. Thus, given a Toeplitz matrix A, multiplication by $Q\Delta$ and ΔQ^T involves storage of merely n parameters instead of $n(n+1)/2$ when forward elimination and backsubstitution are used. Moreover, in contrast to the general case, the three steps

(1) hyperbolic Cholesky factorization

(2) matrix vector multiplication with $Q\Delta$

(3) matrix vector multiplication with ΔQ^T

now require the same number of operations, and the factorization does not constitute the major part of the computation anymore. A detailed description of these three steps is given next.

Step 1 : Schur Algorithm

The relationship between the first row of $\tilde{R}_*^{(l-1)}$, $\tilde{S}_*^{(l-1)}$ and the first row of $R_*^{(l-1)}$ and $S_*^{(l-1)}$, $1 \leq l \leq n$, follows directly from the hyperbolic Cholesky algorithm :

$$\begin{pmatrix} 0 & \dots & 0 & \tilde{r}_{l,l}^{(l-1)} & \dots & \tilde{r}_{l,n}^{(l-1)} \\ 0 & \dots & 0 & \tilde{s}_{l,l}^{(l-1)} & \dots & \tilde{s}_{l,n}^{(l-1)} \end{pmatrix}$$

$$= (1 - \rho_l^{(l-1)^2})^{-1/2} \begin{pmatrix} 1 & -\rho_l^{(l-1)} \\ -\rho_l^{(l-1)} & 1 \end{pmatrix} \begin{pmatrix} 0 & \dots & 0 & r_{l,l}^{(l-1)} & \dots & r_{l,n}^{(l-1)} \\ 0 & \dots & 0 & s_{l,l}^{(l-1)} & \dots & s_{l,n}^{(l-1)} \end{pmatrix},$$

with $\rho_l^{(l-1)} = s_{l,l}^{(l-1)} / r_{l,l}^{(l-1)}$. The first row of $R_*^{(l)}$ is the second row of $\tilde{R}_*^{(l-1)}$ while the first row of $S_*^{(l)}$ is the first row of $\tilde{S}_*^{(l-1)}$. Hence, exploiting the Toeplitzness of $\tilde{R}_*^{(l-1)}$ and noting that $\tilde{s}_{l,l}^{(l-1)} = 0$,

$$\begin{pmatrix} 0 & \dots & 0 & r_{l+1,l+1}^{(l)} & \dots & r_{l+1,n}^{(l)} \\ 0 & \dots & 0 & s_{l+1,l+1}^{(l)} & \dots & s_{l+1,n}^{(l)} \end{pmatrix} = \begin{pmatrix} 0 & \dots & 0 & \tilde{r}_{l,l}^{(l-1)} & \dots & \tilde{r}_{l,n-1}^{(l-1)} \\ 0 & \dots & 0 & \tilde{s}_{l,l+1}^{(l-1)} & \dots & \tilde{s}_{l,n}^{(l-1)} \end{pmatrix}.$$

The matrix possessing as lth row the first row of $\tilde{R}_*^{(l-1)}$, $1 \leq l \leq n$, is the Cholesky factor U of A.

Having completed this derivation, one can now employ the more compact notation $r_j^{(l-1)} \equiv r_{l,j}^{(l-1)}$, $s_j^{(l-1)} \equiv s_{l,j}^{(l-1)}$, $\tilde{r}_j^{(l-1)} \equiv \tilde{r}_{l,j}^{(l-1)}$, $\tilde{s}_j^{(l-1)} \equiv \tilde{s}_{l,j}^{(l-1)}$ and $\rho^{(l-1)} \equiv \rho_l^{(l-1)}$ to summarize the algorithm.

Initialization :

$$\begin{pmatrix} r_1^{(0)} & \dots & r_n^{(0)} \\ s_1^{(0)} & \dots & s_n^{(0)} \end{pmatrix} = a_0^{-1/2} \begin{pmatrix} a_0 & a_1 & \dots & a_{n-1} \\ 0 & a_1 & \dots & a_{n-1} \end{pmatrix}$$

For $l = 1$ to n do

$$\rho^{(l-1)} = s_l^{(l-1)} / r_l^{(l-1)}$$

$$\begin{pmatrix} \tilde{r}_l^{(l-1)} & \dots & \tilde{r}_n^{(l-1)} \\ \tilde{s}_l^{(l-1)} & \dots & \tilde{s}_n^{(l-1)} \end{pmatrix} = (1 - \rho^{(l-1)^2})^{-1/2} \begin{pmatrix} 1 & -\rho^{(l-1)} \\ -\rho^{(l-1)} & 1 \end{pmatrix} \begin{pmatrix} r_l^{(l-1)} & \dots & r_n^{(l-1)} \\ s_l^{(l-1)} & \dots & s_n^{(l-1)} \end{pmatrix}$$

$$\begin{pmatrix} r_{l+1}^{(l)} & \dots & r_n^{(l)} \\ s_{l+1}^{(l)} & \dots & s_n^{(l)} \end{pmatrix} = \begin{pmatrix} \tilde{r}_l^{(l-1)} & \dots & \tilde{r}_{n-1}^{(l-1)} \\ \tilde{s}_{l+1}^{(l-1)} & \dots & \tilde{s}_n^{(l-1)} \end{pmatrix}.$$

Note, that $\rho^{(0)} = 0$ and $\tilde{s}_l^{(l-1)} = 0$, $1 \leq l \leq n$. The algorithm determines the sequence of 'Schur parameters' $\rho^{(l)}$, $1 \leq l \leq n-1$, as well as the Cholesky factor U,

$$u_{ij} = \begin{cases} 0 & j < i \\ \tilde{r}_j^{(i-1)} & j \geq i \end{cases}.$$

Step 2 : Evaluation of $y = U^{-T}b$ and $z = L^{-T}b$

Let $y^{(0)} = z^{(0)} = D^{-1}b$ and

$$\begin{pmatrix} y^{(l+1)} \\ z^{(l+1)} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & P \end{pmatrix} \hat{Q}^{(l)} \begin{pmatrix} y^{(l)} \\ z^{(l)} \end{pmatrix},$$

then, according to (4.4),

$$\begin{pmatrix} y \\ z \end{pmatrix} \equiv \begin{pmatrix} U^{-T}b \\ L^{-T}b \end{pmatrix} = \begin{pmatrix} y^{(n)} \\ z^{(n)} \end{pmatrix}.$$

Some easy manipulations and the fact that $\tilde{D}^{(l)}$ and $\tilde{\Delta}^{(l)}$ are scalar matrices lead to an algorithm for evaluating y and z.

Initialization :

$$\begin{pmatrix} y_1^{(0)} & \dots & y_n^{(0)} \\ z_1^{(0)} & \dots & z_n^{(0)} \end{pmatrix} = a_0^{-1/2} \begin{pmatrix} b_1 & \dots & b_n \\ b_1 & \dots & b_n \end{pmatrix}$$

For $l = 1$ to n do

$$\begin{pmatrix} \tilde{y}_l^{(l-1)} & \dots & \tilde{y}_n^{(l-1)} \\ \tilde{z}_l^{(l-1)} & \dots & \tilde{z}_n^{(l-1)} \end{pmatrix} = (1 - \rho^{(l-1)^2})^{-1/2} \begin{pmatrix} 1 & -\rho^{(l-1)} \\ -\rho^{(l-1)} & 1 \end{pmatrix} \begin{pmatrix} y_l^{(l-1)} & \dots & y_n^{(l-1)} \\ z_l^{(l-1)} & \dots & z_n^{(l-1)} \end{pmatrix}$$

$$\begin{pmatrix} y_{l+1}^{(l)} & \dots & y_n^{(l)} \\ z_{l+1}^{(l)} & \dots & z_n^{(l)} \end{pmatrix} = \begin{pmatrix} \tilde{y}_{l+1}^{(l-1)} & \dots & \tilde{y}_n^{(l-1)} \\ \tilde{z}_l^{(l-1)} & \dots & \tilde{z}_{n-1}^{(l-1)} \end{pmatrix}.$$

The algorithm determines $y_i = \tilde{y}_i^{(i-1)}$ and $z_i = \tilde{z}_n^{(n-i)}$, $1 \leq i \leq n$.

Step 3 : Evaluation of $x = A^{-1}b$

The solution vector x is computed from the expression

$$x = (I \quad I)\, \Delta Q^T \begin{pmatrix} \alpha y \\ (1-\alpha) z \end{pmatrix},$$

where

$$y = U^{-T}b, \quad z = L^{-T}b. \tag{5.1}$$

Thus,

$$x = D^{-1}(f^{(n)} + g^{(n)}),$$

with $f^{(0)} = \alpha y$, $g^{(0)} = (1-\alpha)z$ and

$$\begin{pmatrix} f^{(l)} \\ g^{(l)} \end{pmatrix} = \hat{Q}^{(n-l)} \begin{pmatrix} I & 0 \\ 0 & P^{-1} \end{pmatrix} \begin{pmatrix} f^{(l-1)} \\ g^{(l-1)} \end{pmatrix}.$$

Straightforward reorganization of the equations results in the following algorithm.

For $l = 1$ to n do

$$\begin{pmatrix} \tilde{f}_{n-l+1}^{(l-1)} & \dots & \tilde{f}_n^{(l-1)} \\ \tilde{g}_{n-l+1}^{(l-1)} & \dots & \tilde{g}_n^{(l-1)} \end{pmatrix} = \begin{pmatrix} \alpha y_{n-l+1} & f_{n-l+2}^{(l-1)} & \dots & f_{n-1}^{(l-1)} & f_n^{(l-1)} \\ g_{n-l+2}^{(l-1)} & g_{n-l+3}^{(l-1)} & \dots & g_n^{(l-1)} & (1-\alpha) z_l \end{pmatrix}$$

$$\begin{pmatrix} f_{n-l+1}^{(l)} & \dots & f_n^{(l)} \\ g_{n-l+1}^{(l)} & \dots & g_n^{(l)} \end{pmatrix} = (1 - \rho^{(n-l)^2})^{-1/2} \begin{pmatrix} 1 & -\rho^{(n-l)} \\ -\rho^{(n-l)} & 1 \end{pmatrix} \begin{pmatrix} \tilde{f}_{n-l+1}^{(l-1)} & \dots & \tilde{f}_n^{(l-1)} \\ \tilde{g}_{n-l+1}^{(l-1)} & \dots & \tilde{g}_n^{(l-1)} \end{pmatrix}$$

$$(\,x_1 \quad \dots \quad x_n\,) = a_0^{-1/2}\,(\,1 \quad 1\,)\begin{pmatrix} f_1^{(n)} & \dots & f_n^{(n)} \\ g_1^{(n)} & \dots & g_n^{(n)} \end{pmatrix}.$$

Now this algorithm can be modified to do without the divisions by square-roots in all of its three steps. To begin with step 1, the parameters $\rho^{(l)}$ are still correctly computed even if the divisions by $a_0^{1/2}$ and $(1-\rho^{(l-1)^2})^{1/2}$ are omitted.

Step 1

Initialization :

$$\begin{pmatrix} r_1^{(0)} & \dots & r_n^{(0)} \\ s_1^{(0)} & \dots & s_n^{(0)} \end{pmatrix} = \begin{pmatrix} a_0 & a_1 & \dots & a_{n-1} \\ 0 & a_1 & \dots & a_{n-1} \end{pmatrix}$$

For $l = 1$ to n do

$$\rho^{(l-1)} = s_l^{(l-1)}/r_l^{(l-1)}$$

$$\begin{pmatrix} \tilde{r}_l^{(l-1)} & \dots & \tilde{r}_n^{(l-1)} \\ \tilde{s}_l^{(l-1)} & \dots & \tilde{s}_n^{(l-1)} \end{pmatrix} = \begin{pmatrix} 1 & -\rho^{(l-1)} \\ -\rho^{(l-1)} & 1 \end{pmatrix}\begin{pmatrix} r_l^{(l-1)} & \dots & r_n^{(l-1)} \\ s_l^{(l-1)} & \dots & s_n^{(l-1)} \end{pmatrix}$$

$$\begin{pmatrix} r_{l+1}^{(l)} & \dots & r_n^{(l)} \\ s_{l+1}^{(l)} & \dots & s_n^{(l)} \end{pmatrix} = \begin{pmatrix} \tilde{r}_l^{(l-1)} & \dots & \tilde{r}_{n-1}^{(l-1)} \\ \tilde{s}_{l+1}^{(l-1)} & \dots & \tilde{s}_n^{(l-1)} \end{pmatrix}.$$

Observe that while the parameters $\rho^{(l)}$ are precisely the 'Schur parameters', the variables $r^{(l)}$, $s^{(l)}$, $\tilde{r}^{(l)}$ and $\tilde{s}^{(l)}$ differ from the 'normalized' ones computed above. In particular, it is easily seen that

$$\tilde{r}_i^{(i-1)} = a_0 \prod_{l=1}^{i} (1-\rho^{(l-1)^2}).$$

As in step 1, divisions by quantities under a square-root are avoided in step 2. The algorithm now computes scaled versions of the vectors $y = U^{-T}b$ and $z = L^{-T}b$.

Step 2

Initialization :

$$\begin{pmatrix} y_1^{(0)} & \dots & y_n^{(0)} \\ z_1^{(0)} & \dots & z_n^{(0)} \end{pmatrix} = \begin{pmatrix} b_1 & \dots & b_n \\ b_1 & \dots & b_n \end{pmatrix}$$

For $l = 1$ to n do

$$\begin{pmatrix} \tilde{y}_l^{(l-1)} & \dots & \tilde{y}_n^{(l-1)} \\ \tilde{z}_l^{(l-1)} & \dots & \tilde{z}_n^{(l-1)} \end{pmatrix} = \begin{pmatrix} 1 & -\rho^{(l-1)} \\ -\rho^{(l-1)} & 1 \end{pmatrix}\begin{pmatrix} y_l^{(l-1)} & \dots & y_n^{(l-1)} \\ z_l^{(l-1)} & \dots & z_n^{(l-1)} \end{pmatrix}$$

$$\begin{pmatrix} y_{l+1}^{(l)} & \dots & y_n^{(l)} \\ z_{l+1}^{(l)} & \dots & z_n^{(l)} \end{pmatrix} = \begin{pmatrix} \tilde{y}_{l+1}^{(l-1)} & \dots & \tilde{y}_n^{(l-1)} \\ \tilde{z}_l^{(l-1)} & \dots & \tilde{z}_{n-1}^{(l-1)} \end{pmatrix}.$$

The algorithm determines

$$\tilde{y}_i^{(i-1)} = (\tilde{r}_i^{(i-1)})^{1/2} y_i \quad \text{and} \quad \tilde{z}_n^{(n-i)} = (\tilde{r}_i^{(i-1)})^{1/2} z_i, \qquad 1 \le i \le n.$$

In order to compensate for the absent 'normalization' factors $(\tilde{r}_i^{(i-1)})^{1/2}$ in step 2 of the square-root free version, one could in step 3 divide by a_0 and $(1-\rho^{(n-l)^2})$ instead of their square-roots. However, this is equivalent to an even simpler procedure : merely divide by $\tilde{r}_{n-l+1}^{(n-l)}$ the unnormalized values $\tilde{y}_{n-l+1}^{(n-l)}$ and $\tilde{z}_n^{(n-l)}$, and no other divisions are required.

Step 3

For $l = 1$ to n do

$$\begin{pmatrix} \tilde{f}_{n-l+1}^{(l-1)} & \cdots & \tilde{f}_n^{(l-1)} \\ \tilde{g}_{n-l+1}^{(l-1)} & \cdots & \tilde{g}_n^{(l-1)} \end{pmatrix} = \begin{pmatrix} \alpha y_{n-l+1}/\tilde{r}_{n-l+1}^{(n-l)} & f_{n-l+2}^{(l-1)} & \cdots & f_{n-1}^{(l-1)} & f_n^{(l-1)} \\ g_{n-l+2}^{(l-1)} & g_{n-l+3}^{(l-1)} & \cdots & g_n^{(l-1)} & (1-\alpha)z_l/\tilde{r}_{n-l+1}^{(n-l)} \end{pmatrix}$$

$$\begin{pmatrix} f_{n-l+1}^{(l)} & \cdots & f_n^{(l)} \\ g_{n-l+1}^{(l)} & \cdots & g_n^{(l)} \end{pmatrix} = \begin{pmatrix} 1 & -\rho^{(n-l)} \\ -\rho^{(n-l)} & 1 \end{pmatrix} \begin{pmatrix} \tilde{f}_{n-l+1}^{(l-1)} & \cdots & \tilde{f}_n^{(l-1)} \\ \tilde{g}_{n-l+1}^{(l-1)} & \cdots & \tilde{g}_n^{(l-1)} \end{pmatrix}$$

$$(\, x_1 \quad \ldots \quad x_n \,) = (\, 1 \quad 1 \,) \begin{pmatrix} f_1^{(n)} & \cdots & f_n^{(n)} \\ g_1^{(n)} & \cdots & g_n^{(n)} \end{pmatrix} .$$

6. A Systolic Array for Dense Toeplitz Matrices

This section presents a n-processor linear systolic array which solves a linear system with Toeplitz coefficient matrix in $5n$ time steps and can pipeline different problems with a period as low as n. The square-root free algorithm of the previous section, with the parameter α set to one, can be rewritten as follows

Step 1 :

$$\begin{array}{llll} & 1 \le i \le n, & r_{i,0} \equiv a_{i-1} & \\ & 2 \le i \le n, & s_{i,0} \equiv a_{i-1} & \\ & 1 \le j \le n-1, & \rho_j = s_{j+1,j-1}/r_{j,j-1} & \\ & & j+1 \le i \le n, & r_{i,j} = r_{i-1,j-1} - \rho_j s_{i,j-1} \\ & & & s_{i,j} = -\rho_j r_{i-1,j-1} + s_{i,j-1} \end{array}$$

Step 2 :

$$\begin{array}{llll} & 1 \le i \le n, & y_{i,0} \equiv b_i & \\ & & z_{i,0} \equiv b_i & \\ & 1 \le j \le n-1, & j+1 \le i \le n, & y_{i,j} = y_{i,j-1} - \rho_j z_{i-1,j-1} \\ & & & z_{i,j} = -\rho_j y_{i,j-1} + z_{i-1,j-1} \end{array}$$

Step 3 :

$$\begin{array}{lll} 1 \le j \le n, & f_{j,n-j} \equiv y_{j,j-1}/r_{j,j-1} & \\ & g_{n+1,n-j} \equiv 0 & \\ 1 \le j \le n-1, & n-j+1 \le i \le n, & f_{i,j} = f_{i,j-1} - \rho_{n-j} g_{i+1,j-1} \\ & & g_{i,j} = -\rho_{n-j} f_{i,j-1} + g_{i+1,j-1} \\ 1 \le i \le n, & x_i = f_{i,n-1} + g_{i+1,n-1} & \end{array}$$

The variables ρ_j, $r_{i,j}$, $s_{i,j}$, $y_{i,j}$, $z_{i,j}$, $f_{i,j}$, $g_{i,j}$ and x_i correspond to the variables $\rho^{(j)}$, $\tilde{r}_i^{(j)}$, $\tilde{s}_i^{(j)}$, $\tilde{y}_i^{(j)}$, $\tilde{z}_i^{(j)}$, $\tilde{f}_i^{(j)}$, $\tilde{g}_i^{(j)}$, and x_i in Section 5.

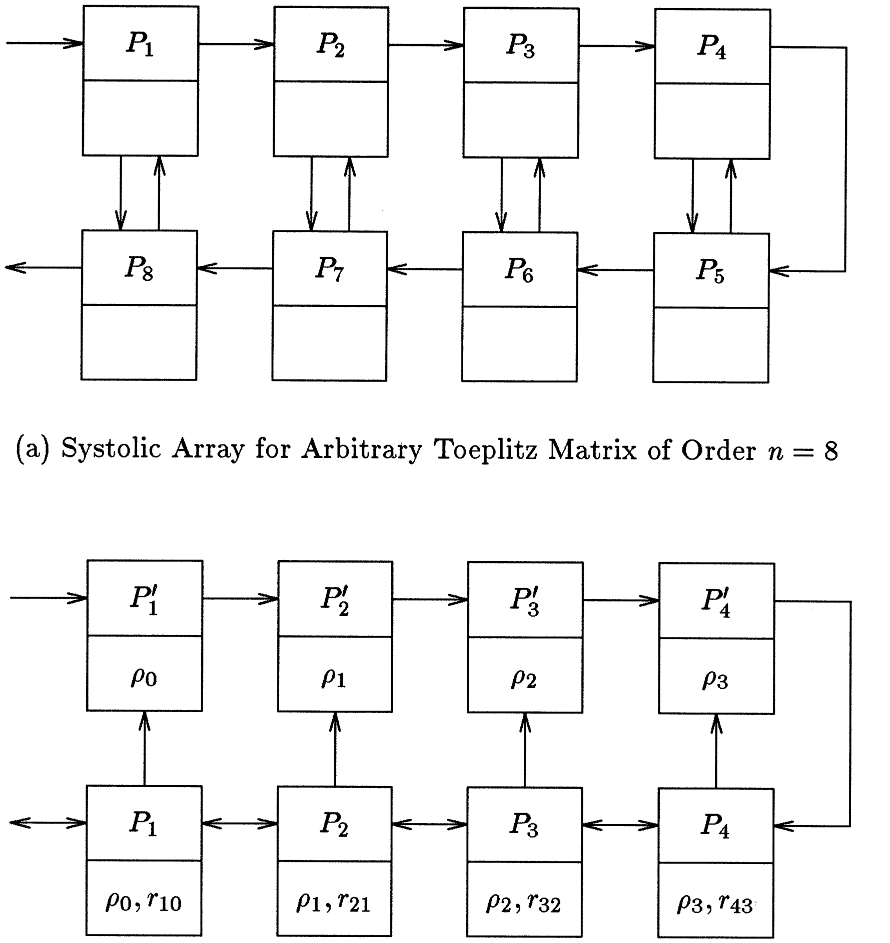
6.1. Description of the Array

The systolic array that computes the above equations consists of n identical processors, each of which simultaneously performs in one time cycle either two multiply-and-accumulate operations, or one divide and one multiply-and-accumulate operation. The processors are numbered consecutively from P_1 to P_n. This linear array is physically folded as shown in Figure 1(a). From each processor P_j, $1 \le j < n$, there is a data link towards P_{j+1}. Additionally, there are two one-bit lines, in opposite directions, between processors P_j and P_{n-j+1}.

The equations are mapped into a space-time representation by applying affine transformations to the indices of the variables. The execution trace in Figure 2 for the solution of an arbitrary Toeplitz system of order $n = 8$ illustrates this mapping. There, the *indices* of the quantities r and s in Step 1, y and z in Step 2, and f and g in Step 3 are displayed at the location and time of computation of these quantities. The horizontal axis represents the processors while the vertical axis indicates consecutive time cycles.

Step 1 : The matrix elements a_i are input to processor P_1 at the beginning of cycle $t = i + 1$. $r_{i,j}$ and $s_{i,j}$ are computed in processor P_π during cycle $t = \tau$, where τ and π are given by

$$\begin{pmatrix} \tau \\ \pi \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$



(a) Systolic Array for Arbitrary Toeplitz Matrix of Order $n = 8$

(b) Systolic Array for Toeplitz Matrix with Inverse of Bandwidth $p = 4$

Figure 1: Systolic Arrays for Toeplitz Matrices.

ρ_j is computed during the same cycle, $t = 2j + 1$, and in the same processor, P_{j+1}, as $r_{j+1,j}$. During cycle $2j + 1 \leq t \leq 2n + j + 1$, processor P_{j+1} contains ρ_j and $r_{j+1,j}$. $r_{i,j}$ is stored in processor P_{j+1} during cycle $t = i + j + 1$ and moved to processor P_{j+2} to be available for cycle $t = i + j + 2$ (alternatively, $r_{i,j}$ could be moved to P_{j+2} right away, and stored there during cycle $t = i + j + 1$). $s_{i,j}$ is 'directly' sent to processor P_{j+2} to be available for cycle $t = i + j + 1$.

Note, that the computation of ρ_j can be performed simultaneously with the evaluation of $r_{j+1,j}$ if the processors implement an algorithm based on non-restoring division [20]. In that case, ρ_j is determined bit-by-bit. During cycle $t = n + j$ a copy of ρ_j is sent, bit by bit, from processor P_{j+1} to the processor 'across', P_{n-j}, so that at $t = 2n$ P_{j+1} contains both, ρ_j and ρ_{n-j-1}, $1 \leq j \leq n - 1$.

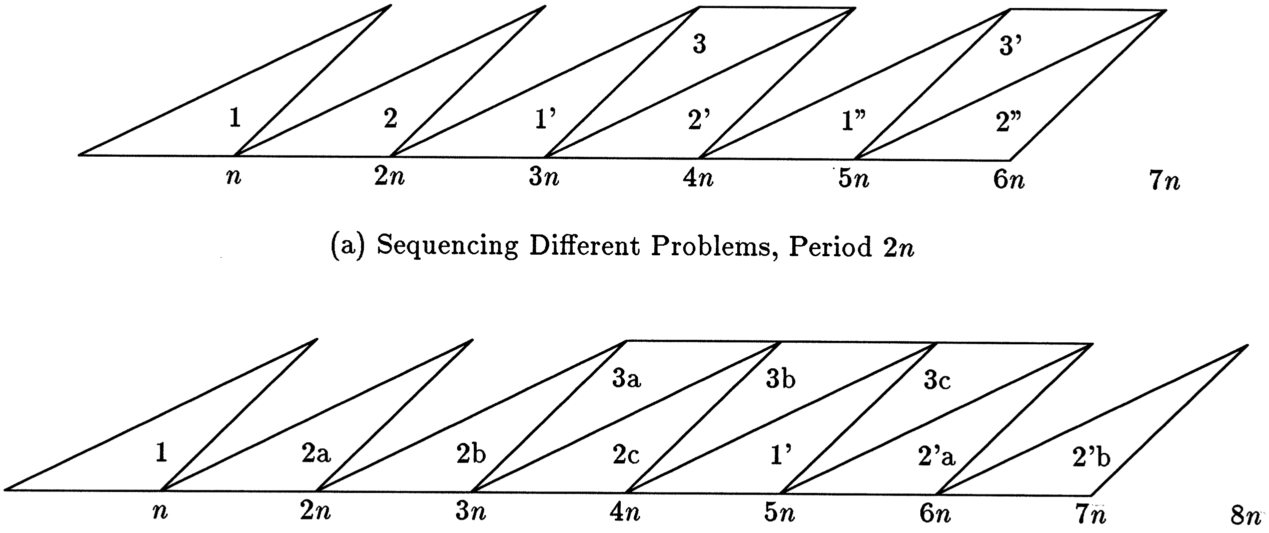
Step 2 : The elements b_i of the right-hand side vector are input to processor P_1 at the beginning of cycle $t = n + i$. $y_{i,j}$ and $z_{i,j}$ are determined in processor P_π during cycle $t = \tau$, where

$$\begin{pmatrix} \tau \\ \pi \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} n \\ 1 \end{pmatrix}.$$

$y_{j,j-1}$ is computed in processor P_j during cycle $t = n + 2j - 1$.

t	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
1	a_0							
2	a_1							
3	a_2	21						
4	a_3	31						
5	a_4	41	32					
6	a_5	51	42					
7	a_6	61	52	43				
8	a_7	71	62	53				
9	b_1	81	72	63	54			
10	b_2		82	73	64			
11	b_3	21		83	74	65		
12	b_4	31			84	75		
13	b_5	41	32			85	76	
14	b_6	51	42				86	
15	b_7	61	52	43				87
16	b_8	71	62	53				
17	17 /	81	72	63	54			
18		26 /	82	73	64			
19			35 /	83	74	65		
20				43 /	84	75		
21					53 /	85	76	
22						62 /	86	
23							71 /	87
24								80 /
25	81							
26		72						
27		82	63					
28			73	54				
29			83	64	45			
30				74	55	36		
31				84	65	46	27	
32					75	56	37	x_1
33					85	66	47	x_2
34						76	57	x_3
35						86	67	x_4
36							77	x_5
37							87	x_6
38								x_7
39								x_8

Figure 2: Execution Trace for a Toeplitz Problem of Order $n = 8$ on the Array of Figure 1(a).



(a) Sequencing Different Problems, Period $2n$

(b) Sequencing Different Problems with Several Righthand Sides

Figure 3: Processor Activity Charts.

The following operations are undertaken as preparations for Step 3. During cycle $t = 2n + j$, processor P_j performs a division to scale the $y_{j,j-1}$, i.e., to determine the initial values for Step 3, $f_{j,n-j}$ (the division is indicated through a '/' in Figure 2, and the associated indices are those for f). Again, when using the non-restoring division algorithm, the $f_{j,n-j}$ are determined in a bit-by-bit fashion. Hence, as soon as the next bit of $f_{j,n-j}$ is available, it may be transmitted from P_j to the processor 'across', P_{n-j+1}. This way, the division and the transfer of information between pairs of processors occur concurrently. Finally, at the end of cycle $t = 3n$, one finds $f_{n-j+1,j-1}$ and ρ_{n-j} in processor P_j, $1 \leq j \leq n-1$. Note that more registers are required for this step than for Steps 1 and 3; they are also needed when pipelining different problems.

Step 3 : $f_{i,j}$ and $g_{i,j}$ are determined in processor P_π during cycle $t = \tau$, where

$$\begin{pmatrix} \tau \\ \pi \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 2n-1 \\ 0 \end{pmatrix}.$$

After their computation in P_{n-1} during cycle $t = 4n + i - 3$, $f_{i,n-1}$ and $g_{i+1,n-1}$ are transferred to processor P_n where x_i is determined during the next cycle as the sum of those two quantities (remember that $g_{n+1,n-1}$ is initialized to zero).

Observe, that use of a non-restoring division algorithm makes it possible to overlap different computations, or to overlap a computation with data transfers. The decision to exchange the ρ_{j-1} and $f_{j,n-j}$ between each pair of processors was taken so as to not only maintain uni-directional, as opposed to bi-directional, data-flow for the three steps but more importantly, so as to be able to efficiently pipeline different problems on the same array. In order to provide a global perspective and more easily discuss the pipelining of problems on the array, the compact representation of the execution trace in Figure 3 is used, which depicts in space-time the *activity* pattern of the processors ('activity' here means *computation*, as opposed to mere *data transfers*). The vertical axis represents the spatial direction (processors) while the horizontal one denotes time; numbers inside the triangles indicate the corresponding steps of the algorithm. Figure 3(a) represents the activity pattern when different problems (distinguished through primes) are solved in sequence, the period is $2n + O(1)$. The activity pattern drawn in Figure 3(b) corresponds to the pipelined computation of several solution vectors associated with different right-hand side vectors for each new coefficient matrix, i.e., for the solution of $Ax_k = b_k$, given several right-hand sides b_k. Numbers $2a$ and $3a$ in Figure 3(b) refer to the application of Steps 2 and 3 to the first vector b_1, while $2b$ and $3b$ refer to the treatment of b_2. Right-hand sides associated with the same coefficient matrix are entered in a continuous fashion – without any gaps; the solution vectors are obtained at a period of $n + O(1)$.

6.2. Discussion

In order to make meaningful comparisons between previously published designs and the one proposed here, the following assumptions are made

- the coefficient matrix is symmetric positive definite and neither banded nor with a banded inverse,
- the processor and time requirements for a division or a multiply-and-accumulate are identical,
- one processing element executes one division or one multiply-and-accumulate per time step.

Given the availability of $P = O(n)$ processing elements, the *best* $O(n)$ processor array is the one that allows the solution of the *largest* number m of problems of given size n in a given time $T = O(n)$. If a problem of size n can be solved on p processors with latency l ('latency' is defined to be the time between first input and last output) then this array can solve $m = \lfloor PT/pl \rfloor$ problems. Thus one wishes to minimize the processor-latency product pl. In fact, if the array allows pipelining of problems, one can solve more than $\lfloor PT/pl \rfloor$ problems. The period then replaces the latency when evaluating m, as a criterion for measuring the quality of a systolic array. The arrays proposed in the literature do not attempt to pipeline problems and hence will be compared on the basis of their processor-latency product. The array presented here has $p = 2n$ and $l = 5n$, hence $pl = 10n^2$.

Kung and Hu [13] present three arrays (labelled A,B,C); array A is the one worked out with the most details by the authors as well as by Nash et al. [19]. Designs A and B require $O(n^2)$ storage as compared to $O(n)$ for all the other designs. For design A, $p = 3n$ and the time required to set up the upper-triangular system to be solved by backsubstitution is $2n$. Since in that design backsubstitution requires time $2n$, the latency is $l = 4n$ and $pl = 12n^2$ for array A. Array B has $p = 5n$. It requires time $2n$ to perform the Cholesky factorization of the inverse of the coefficient matrix and the solution is then found via matrix-vector products; therefore $pl > 10n^2$. Array C has $p \geq 4n$; a number of processors in excess of $4n$ may be used in order to perform faster the four convolutions called for by this algorithm. Since it requires time $2n$ to set up the vectors involved in the convolutions, this array has $pl \geq 8n^2 + q$ where q is the number of operations needed to perform the convolutions. It may be possible to obtain a processor-latency product that is less than $10n^2$ for n sufficiently large, but the gain will be marginal and the design complex when compared to the other ones. Brent and Luk [4] implement an algorithm that consists essentially of our first and second steps followed by a backsubstitution step where the elements of the triangular matrix U are generated on-line from the parameters. For this array, $p = 5\lceil n/2 \rceil$ and $l = 4n$, hence $pl \geq 10n^2$. The authors neglect the extra circuitry and time needed to load in the data and unload the results from each processor in the array.

Using the square-root free algorithm of Section 5 but allowing fast loading and unloading of every processor in the array like in [4], the latency of our n-processor systolic array can be reduced to $3n$ [9], hence $pl \geq 6n^2$. More importantly, our array with latency $5n$ allows pipelining of problems with a period of only $2n$ at a very small additional expense in terms of processor storage and control complexity. Then the relevant processor-period product is only $4n^2$.

7. A Systolic Array for Toeplitz Matrices with Banded Inverse

Toeplitz matrices with banded inverses occur, for example, in maximum entropy spectral estimation and filtering of stationary time series with auto-regressive models [2]. This section presents a $2p$-processor linear systolic array which solves a linear system involving a Toeplitz matrix whose inverse has bandwidth p in $n + 3p$ time steps and can pipeline different problems with a period as low as n. The square-root free algorithm, exploiting the bandedness of the inverse, takes the following form (again, α is set to one)

Step 1 :

$$
\begin{array}{lll}
1 \le i \le p, & r_{i,0} \equiv a_{i-1} & \\
2 \le i \le p, & s_{i,0} \equiv a_{i-1} & \\
1 \le j \le p-1, & \rho_j = s_{j+1,j-1}/r_{j,j-1} & \\
 & j+1 \le i \le n, & r_{i,j} = r_{i-1,j-1} - \rho_j s_{i,j-1} \\
 & & s_{i,j} = -\rho_j r_{i-1,j-1} + s_{i,j-1}
\end{array}
$$

Step 2 :

$$
\begin{array}{lll}
1 \le i \le n, & y_{i,0} \equiv b_i & \\
 & z_{i,0} \equiv b_i & \\
1 \le j \le p-1, & j+1 \le i \le n, & y_{i,j} = y_{i,j-1} - \rho_j z_{i-1,j-1} \\
 & & z_{i,j} = -\rho_j y_{i,j-1} + z_{i-1,j-1}
\end{array}
$$

Step 3 :

$$
\begin{array}{lll}
1 \le j \le p, & f_{j,n-j} \equiv y_{j,j-1}/r_{j,j-1} & \\
 & g_{n+1,n-j} \equiv 0 & \\
p+1 \le j \le n, & f_{j,n-p} \equiv y_{j,p-1}/r_{p,p-1} & \\
 & g_{j,n-p} \equiv 0 & \\
n-p+1 \le j \le n-1, & n-j+1 \le i \le n, & f_{i,j} = f_{i,j-1} - \rho_{n-j} g_{i+1,j-1} \\
 & & g_{i,j} = -\rho_{n-j} f_{i,j-1} + g_{i+1,j-1} \\
1 \le i \le n, & x_i = f_{i,n-1} + g_{i+1,n-1} &
\end{array}
$$

7.1. Description of the Array

The systolic array that computes the above equations consists of two linearly connected arrays of p processors each, see Figure 1(b). As in the non-banded case each processor can, in one time cycle, perform either two multiply-and-accumulate operations, or one divide and one multiply-and-accumulate operation. Within each linear array, the processors are numbered consecutively from 1 to p. In the top array, there is a data link from each processor P'_j, $1 \le j \le p$, towards P'_{j+1}, while the bottom array contains bi-directional data links between adjacent processors P_j and P_{j+1}. Additionally, there is a one-bit line from a processor P_j in the bottom array to the corresponding processor P'_j in the top array, as well as a data link from the last processor P'_p in the top array to the last processor in the bottom array.

As in Section 6, the equations are mapped into a space-time representation by applying affine transformations to the indices of the variables. The execution trace in Figure 4 illustrates this mapping for the solution of a linear system whose Toeplitz matrix, of order $n = 28$, has an inverse of bandwidth $p = 4$. The top array computes Step 2 of the algorithm while the bottom array computes Steps 1 and 3. Hence, the *indices* of the quantities y and z in Step 2 are displayed in the

	BOTTOM ARRAY				TOP ARRAY			
t	P_1	P_2	P_3	P_4	P'_1	P'_2	P'_3	P'_4
1	a_0				b_1			
2	a_1				b_2			
3	a_2	2,1			b_3	2,1		
4	a_3	3,1			b_4	3,1		
5		4,1	3,2		b_5	4,1	3,2	
6			4,2		b_6	5,1	4,2	
7				4,3	b_7	6,1	5,2	4,3
8				4,24 /	b_8	7,1	6,2	5,3
9			3,25 /	4,25 /	b_9	8,1	7,2	6,3
10		2,26 /	3,26	5,25 /	b_{10}	9,1	8,2	7,3
11	1,27 /	2,27	4,26	6,25 /	b_{11}	10,1	9,2	8,3
12	x_1	3,27	5,26	7,25 /	b_{12}	11,1	10,2	9,3
13	x_2	4,27	6,26	8,25 /	b_{13}	12,1	11,2	10,3
14	x_3	5,27	7,26	9,25 /	b_{14}	13,1	12,2	11,3
15	x_4	6,27	8,26	10,25 /	b_{15}	14,1	13,2	12,3
16	x_5	7,27	9,26	11,25 /	b_{16}	15,1	14,2	13,3
17	x_6	8,27	10,26	12,25 /	b_{17}	16,1	15,2	14,3
18	x_7	9,27	11,26	13,25 /	b_{18}	17,1	16,2	15,3
19	x_8	10,27	12,26	14,25 /	b_{19}	18,1	17,2	16,3
20	x_9	11,27	13,26	15,25 /	b_{20}	19,1	18,2	17,3
21	x_{10}	12,27	14,26	16,25 /	b_{21}	20,1	19,2	18,3
22	x_{11}	13,27	15,26	17,25 /	b_{22}	21,1	20,2	19,3
23	x_{12}	14,27	16,26	18,25 /	b_{23}	22,1	21,2	20,3
24	x_{13}	15,27	17,26	19,25 /	b_{24}	23,1	22,2	21,3
25	x_{14}	16,27	18,26	20,25 /	b_{25}	24,1	23,2	22,3
26	x_{15}	17,27	19,26	21,25 /	b_{26}	25,1	24,2	23,3
27	x_{16}	18,27	20,26	22,25 /	b_{27}	26,1	25,2	24,3
28	x_{17}	19,27	21,26	23,25 /	b_{28}	27,1	26,2	25,3
29	x_{18}	20,27	22,26	24,25 /		28,1	27,2	26,3
30	x_{19}	21,27	23,26	25,25 /			28,2	27,3
31	x_{20}	22,27	24,26	26,25 /				28,3
32	x_{21}	23,27	25,26	27,25 /				
33	x_{22}	24,27	26,26	28,25 /				
34	x_{23}	25,27	27,26					
35	x_{24}	26,27	28,26					
36	x_{25}	27,27						
37	x_{26}	28,27						
38	x_{27}							
39	x_{28}							

Figure 4: Execution Trace for a Toeplitz Problem of Order $n = 28$ and Inverse of Bandwidth $p = 4$ on the Array of Figure 1(b).

top array, and the indices of r and s in Step 1, and f and g in Step 3 are displayed in the bottom array.

Step 1 : The matrix elements a_i, $0 \leq i \leq p-1$, are input to processor P_1 of the bottom array at the beginning of cycle $t = i+1$. $r_{i,j}$ and $s_{i,j}$ are computed in processor P_π of the bottom array during cycle $t = \tau$, where τ and π are given by

$$\begin{pmatrix} \tau \\ \pi \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Using non-restoring division, the computation of ρ_j and the evaluation of $r_{j+1,j}$ are performed simultaneously during cycle $t = 2j+1$ in processor P_{j+1}. Moreover, since ρ_j is determined bit by bit, each bit is sent as soon as determined from processor P_{j+1} to processor P'_{j+1} in the top array, so computations using ρ_j can be performed in P'_{j+1} already during cycle $t = 2j+1$.

$r_{i,j}$ is stored in processor P_{j+1} during cycle $t = i+j+1$ and moved to processor P_{j+2} to be available for cycle $t = i+j+2$, while $s_{i,j}$ is immediately sent to processor P_{j+2} to be available for cycle $t = i+j+1$. ρ_j and $r_{j+1,j}$, $1 \leq j < p$, are stored in processor P_{j+1} to be used in Step 3.

Step 2 : The elements b_i of the right-hand side vector are input to processor P'_1 in the top array at the beginning of cycle $t = i+1$. $y_{i,j}$ and $z_{i,j}$ are determined in processor P'_π during cycle $t = \tau$, where

$$\begin{pmatrix} \tau \\ \pi \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

In particular, $y_{j+1,j}$, $0 \leq j \leq p-1$, is computed in processor P'_{j+1} during cycle $t = 2j+1$, and travels to P_{j+1} via $P'_{j+2}, \ldots, P'_p, P_p, \ldots, P_{j+2}$, using one time cycle to proceed through each processor. $y_{i,p-1}$, $p < i \leq n$, is computed in processor P'_p during cycle $t = i+p-1$ and sent directly down to P_p.

Step 3 : During cycle $t = 3p-j$, processor P_j of the bottom array performs a division to scale $y_{j,j-1}$, i.e., to determine the initial value for Step 3, $f_{j,n-j}$, $1 \leq j \leq p$ (the division is indicated through a '/' in Figure 4, and the associated indices are those for f). Using the non-restoring division algorithm, $f_{j,n-j}$ is determined in a bit-by-bit fashion. Thus, during the same cycle and in the same processor one manages to compute not only $f_{j,n-j}$, hence $f_{j,n-j+1} = f_{j,n-j}$ (since $g_{j+1,n-j} = 0$), but also $g_{j,n-j+1} = -\rho_{j-1} f_{j,n-j}$. Similarly, during cycle $t = p+j$, $p < j \leq n$, the initial value $f_{j,n-p}$, and the pair $(f_{j,n-p+1}, g_{j,n-p+1})$, are computed simultaneously in processor P_p.

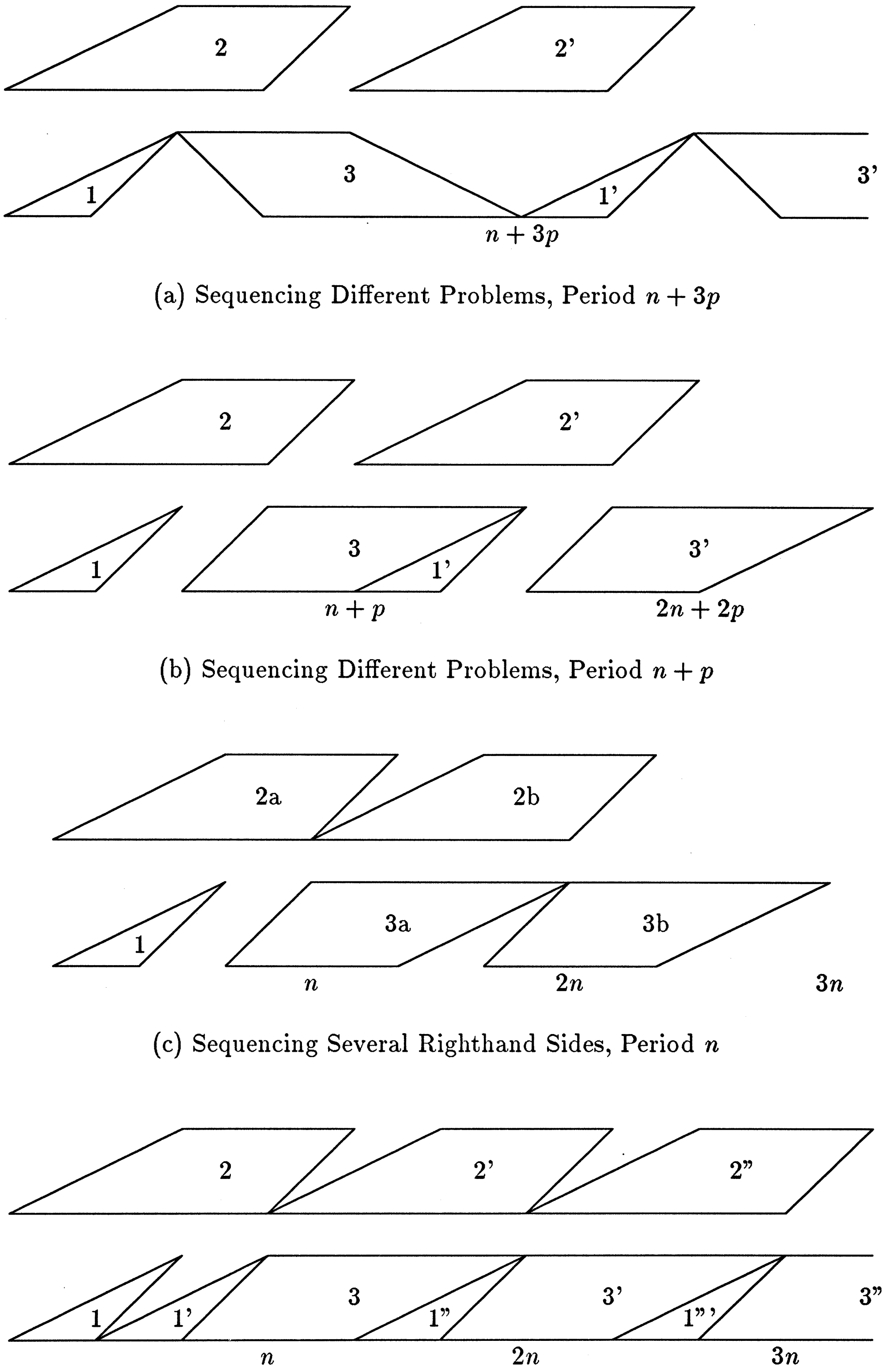
$f_{i,j}$ and $g_{i,j}$ are determined in processor P_π of the bottom array during cycle $t = \tau$, where

$$\begin{pmatrix} \tau \\ \pi \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -2n+3p-1 \\ n+1 \end{pmatrix}.$$

x_i is determined in processor P_1 of the bottom array during cycle $t = 3p+i-1$ as the sum of $f_{i,n-1}$ and $g_{i+1,n-1}$.

7.2. Discussion

When the algorithm is consecutively applied to two different problems, see Figure 5(a), computation of the second problem (primed quantities) cannot be started before the first problem is completely solved, resulting in a period identical to the latency, that is, $n+3p$. This inefficiency is due to opposite directions of data flow in the bottom array during Steps 1 and 3. Reduction of the period to $n+p$ can be accomplished by following the approach of Section 6 and reversing the data flow of Step 3, so that data flow has the same direction for all three steps. Bi-directional data links are now avoided but extra control and storage elements per processor are required. For instance,



(a) Sequencing Different Problems, Period $n + 3p$

(b) Sequencing Different Problems, Period $n + p$

(c) Sequencing Several Righthand Sides, Period n

(d) Sequencing Different Problems, Period n

Figure 5: Processor Activity Charts.

ρ_{j-1} and $r_{j,j-1}$ have to move from processor P_j to processor P_{p-j+1} between Steps 1 and 3, and beside retaining ρ_{j-1} and $r_{j,j-1}$, processor P_j also has to accommodate ρ_{p-j} and $r_{p-j+1,p-j}$. The according activity pattern is depicted in Figure 5(b). Moreover, the slightly modified architecture, with uni-directional data flow, permits the reduction to n of the period for the solution of $Ax_k = b_k$ for several right-hand sides b_k.

The last activity chart suggests an even more efficient way of solving different problems in sequence, cf. Figure 5(d) (to be compared to Figure 5(b)). The reduced period is now essentially equal to the problem size, n, instead of $n + 3p$ or $n + p$ as in Figures 5(a) and (b). Even though control and storage increase again, the number of memory locations per processor is still small and independent of n and p.

Note that in the inverse banded case, as opposed to the full dense case in Section 6, it is possible to start Step 3 before the end of Step 2 : Step 3 can commence roughly $2p$ steps into computation of Step 2. Hence, for $n \gg p$, Steps 2 and 3 take place practically in parallel. This property is specific to the inverse banded case; if instead the matrix A had bandwidth p, a direct method based on a Cholesky factorization of A would not allow overlapping of forward elimination and backsubstitution. Observe that the algorithm of Section 5 exploits the banded structure of A and solves the system in time $O(n)$ on $O(p)$ processors. Moreover, this algorithm only requires storage of the n parameters ρ_j rather than storage of the Cholesky factor, which amounts to np.

Acknowledgement

We are grateful to Professor Martin Morf for having suggested the approach presented in this paper.

References

[1] Ahmed, H.M., Delosme, J.-M. and Morf, M., *Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing,* IEEE Computer, 15 (1982), pp. 65–82.

[2] Barrett, W. W., *Toeplitz Matrices with Banded Inverses,* Linear Algebra Appl., 57 (1984), pp. 131–45.

[3] Brent, R.P., Kung, H.T. and Luk, F.T., Some Linear-Time Algorithms for Systolic Arrays, *Proc. IFIP 9th World Computer Congress,* North Holland, Amsterdam, 1983, pp. 865–76.

[4] Brent, R.P. and Luk, F.T., *A Systolic Array for the Linear-Time Solution of Toeplitz Systems of Equations,* J. VLSI and Computer Systems, 1 (1983), pp. 1–22.

[5] Bultheel, A., *Toward an Error Analysis of Fast Toeplitz Factorisation,* Technical Report TW 44, Applied Mathematic and Programming Division, Katholieke Universiteit Leuven, Belgium, 1979.

[6] ————, Error Analysis of Incoming and Outgoing Schemes for the Trigonometric Moment Problem, *Proc. Conf. Padé Approximation and its Applications, Lecture Notes in Mathematics, No. 888,* Springer Verlag, 1980, pp. 100–9.

[7] Cybenko, G., *The Numerical Stability of the Levinson-Durbin Algorithm for Toeplitz Systems of Equations,* SIAM J. Sci. Stat. Comp., 1 (1980), pp. 303–19.

[8] Delosme, J.-M., *Algorithms for Finite Shift-Rank Processes,* Ph.D. Thesis, Dept of Electrical Engineering, Stanford University, 1982.

[9] Delosme, J.-M. and Ipsen, I.C.F., *Efficient Systolic Arrays for the Solution of Toeplitz Systems : An Illustration of a Methodology for the Construction of Systolic Architectures in VLSI,* Research Report 370, Dept. of Computer Science, Yale University, 1985.

[10] Delsarte, P., Genin, Y. and Kamp, Y., *Schur Parametrization of Positive Definite Block-Toeplitz Systems,* SIAM J. Appl. Math., 36 (1979), pp. 34–46.

[11] ————, *A Method of Matrix Inverse Triangular Decomposition, Based on Contiguous Principal Submatrices,* J. Lin. Alg. Appl., 31 (1980), pp. 199–212.

[12] Golub, G.H. and van Loan, C.F., *Matrix Computations,* The Johns Hopkins Press, Baltimore, MD, 1983.

[13] Kung, S.-Y. and Hu, Y.H., Fast and Parallel Algorithms for Solving Toeplitz Systems, *Proc. Int. Symposium on Mini and Micro Computers in Control and Measurement,* 1981, pp. 163–8.

[14] ————, *A Highly Concurrent Algorithm and Pipelined Architecture for Solving Toeplitz Systems,* IEEE Trans. Acoustics, Speech, and Signal Processing, ASSP-21 (1983), pp. 66–76.

[15] Kung, H.T. and Leiserson, C.E., Systolic Arrays (for VLSI), *Sparse Matrix Proceedings,* SIAM, Philadelphia, PA, 1978, pp. 256–82.

[16] Levinson, N., *The Wiener RMS (Root-Mean-Square) Error Criterion in Filter Design and Prediction,* J. Math. Phys., 25 (1947), pp. 261–78.

[17] Morf, M. and Delosme, J.-M., Matrix Decompositions and Inversions via Elementary Signature-Orthogonal Transformations, *Proc. Int. Symposium on Mini and Micro Computers in Control and Measurement,* 1981.

[18] Nash, J.G., Hansen, S. and Nudd, G.R., VLSI Processor Array for Matrix Manipulation, *CMU Conference on VLSI Systems and Computations,* Computer Science Press, 1981, pp. 367–73.

[19] Nash, J.G., Nudd, G.R. and Hansen, S., Concurrent VLSI Architectures for Toeplitz Linear Systems Solution, *Government Microcircuit Applications Conference,* 1982.

[20] Waser, S. and Flynn, M.J., *Introduction to Arithmetic for Digital Systems Designers,* Holt, Rinehart and Winston, New York, 1982.