Parallel Solution of Certain Toeplitz Least-Squares Problems

A. Bojańczyk and R. P. Brent
Centre for Mathematical Analysis
The Australian National University
GPO Box 4
Canberra ACT 2601, Australia

Submitted by G. W. Stewart

ABSTRACT

We describe a systolic algorithm for solving a Toeplitz least-squares problem of special form. Such problems arise, for example, when Volterra convolution equations of the first kind are solved by regularization. The systolic algorithm is based on a sequential algorithm of Eldén, but we show how the storage requirements of Eldén's algorithm can be reduced from $O(n^2)$ to $O(n)$. The sequential algorithm takes time $O(n^2)$; the systolic algorithm takes time $O(n)$ using a linear systolic array of $O(n)$ cells. We also show how large problems may be decomposed and solved on a small systolic array.

1. INTRODUCTION

In this paper we discuss a systolic algorithm for solving the regularized linear least-squares problem

$$\min_f \{ \|Kf - g\|^2 + \mu^2 \|Lf\|^2 \} \tag{1.1}$$

where both K and L are upper triangular $n \times n$ Toeplitz matrices. Such problems arise when a Volterra convolution equation of the first kind,

$$\int_0^t K(t-s) f(s)\, ds = g(t),$$

is solved by the regularization technique with a differential operator as stabilizing functional.

Our systolic algorithm is based on a sequential algorithm of Eldén [2] which computes a QR decomposition by plane rotations. We present an

52 Vanderbilt Ave., New York, NY 10017 0024-3795/86/$3.50

implementation of Eldén's algorithm that requires $O(n)$ time on a one-dimensional systolic array of n processors. The storage requirements are $O(n)$ instead of $O(n^2)$ as in Eldén's original approach. This storage saving is possible at the expense of some extra computation. A similar observation was made by Brent and Luk [1] and exploited in a systolic algorithm for the solution of Toeplitz systems of equations.

The paper is organized as follows. In Section 2 we briefly outline Eldén's algorithm. The systolic implementation is developed in Section 3. In Section 4 we consider the case when the size of the matrix exceeds the number of processors available.

2. ELDÉN'S ALGORITHM AND ITS MODIFICATION

The minimization problem (1.1) can be written in an equivalent standard form as

$$\min_f \left\| \begin{bmatrix} K \\ \mu L \end{bmatrix} f - \begin{bmatrix} g \\ 0 \end{bmatrix} \right\|. \tag{2.1}$$

The solution of (2.1) is obtained in two steps. In the first step we transform the matrix

$$K_c = \begin{bmatrix} K \\ \mu L \end{bmatrix}$$

into upper triangular form $\begin{bmatrix} K_\mu \\ 0 \end{bmatrix}$ using a sequence of plane rotations; the right-hand-side vector $\begin{bmatrix} g \\ 0 \end{bmatrix}$ is transformed by the same sequence of plane rotations into the vector $\begin{bmatrix} \bar{g} \\ h \end{bmatrix}$. In the second step the solution vector f is computed by solving the upper triangular system

$$K_\mu f = \bar{g}. \tag{2.2}$$

Because of the Toeplitz structure of both triangular matrices K and μL, the triangularization of the matrix K_c can be done efficiently by annihilating successive diagonals rather than columns of the matrix μL. This is the basic idea of Eldén's algorithm.

Illustrating by an example, consider the augmented matrix

$$\begin{bmatrix} K & g \\ \mu L & 0 \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & \cdot & \cdot & \cdot & \cdot & k_n & g_1 \\ & k_1 & k_2 & & & & \cdot & g_2 \\ & & \cdot & \cdot & & & \cdot & \cdot \\ & & & \cdot & \cdot & & \cdot & \cdot \\ & & & & \cdot & \cdot & \cdot & \cdot \\ & & & & & \cdot & k_2 & \cdot \\ & & & & & & k_1 & g_n \\ l_1 & l_2 & \cdot & \cdot & \cdot & \cdot & l_n & 0 \\ & l_1 & l_2 & & & & \cdot & 0 \\ & & \cdot & \cdot & & & \cdot & \cdot \\ & & & \cdot & \cdot & & \cdot & \cdot \\ & & & & \cdot & \cdot & \cdot & \cdot \\ & & & & & \cdot & l_2 & \cdot \\ & & & & & & l_1 & 0 \end{bmatrix}. \tag{2.3}$$

First we zero all elements on the main diagonal of L by rotating in the planes $(1, n+1),\ldots,(n,2n)$. Because the matrices K and L have constant elements along diagonals, each row of the matrix K (L) is a "shortened" copy of the first row of the matrix K (L). Thus it is sufficient to rotate only in the plane $(1, n+1)$ and change all other elements so the updated matrices $K^{(1)}$ and $L^{(1)}$ have Toeplitz structure. In addition, we have to update the right-hand-side vector, this time rotating in all the planes $(1, n+1),\ldots,(n,2n)$. Note that the lower part of the right-hand-side vector is filled in. The result of the transformation is

$$\begin{bmatrix} k_1^{(1)} & k_2^{(1)} & \cdot & \cdot & \cdot & \cdot & k_n^{(1)} & g_1^{(1)} \\ & k_1^{(1)} & k_2^{(1)} & & & & \cdot & g_2^{(1)} \\ & & \cdot & \cdot & & & \cdot & \cdot \\ & & & \cdot & \cdot & & \cdot & \cdot \\ & & & & \cdot & \cdot & \cdot & \cdot \\ & & & & & \cdot & k_2^{(1)} & \cdot \\ & & & & & & k_1^{(1)} & g_n^{(1)} \\ 0 & l_2^{(1)} & \cdot & \cdot & \cdot & \cdot & l_n^{(1)} & h_1^{(1)} \\ & 0 & l_2^{(1)} & & & & \cdot & \cdot \\ & & \cdot & \cdot & & & \cdot & \cdot \\ & & & \cdot & \cdot & & \cdot & \cdot \\ & & & & \cdot & \cdot & \cdot & \cdot \\ & & & & & \cdot & l_2^{(1)} & \cdot \\ & & & & & & 0 & h_n^{(1)} \end{bmatrix}.$$

We continue this process, recursively annihilating successive superdiagonals of the transformed matrix L. All steps are virtually the same as the first except that we operate on smaller and smaller matrices. Typically, after the ith step the matrix has the following form:

$$\begin{bmatrix} k_1^{(1)} & k_2^{(1)} & \cdots & \cdots & \cdots & k_n^{(1)} & g_1^{(1)} \\ & k_1^{(2)} & k_2^{(2)} & \cdots & \cdots & k_{n-1}^{(2)} & g_2^{(2)} \\ & & \ddots & & & \vdots & \vdots \\ & & & k_1^{(i)} & \cdots & k_{n-i+1}^{(i)} & g_i^{(i)} \\ & & & & \ddots & \vdots & \vdots \\ & & & & & k_1^{(i)} & g_n^{(i)} \\ 0 & \cdots & 0 & l_{i+1}^{(i)} & \cdots & l_n^{(i)} & h_1^{(i)} \\ & & & & \ddots & \vdots & \vdots \\ & & & & & l_{i+1}^{(i)} & h_{n-i}^{(i)} \\ & & & & & 0 & h_{n-i+1}^{(i)} \\ & & & & & \vdots & \vdots \\ & & & & & 0 & h_n^{(1)} \end{bmatrix}. \tag{2.4}$$

Rows $i+1$ to n of the top triangular matrix and the whole bottom triangular matrix have Toeplitz structure. Thus we can proceed as in the first step, calculating only one pair of cosine and sine, and rotating in one plane, namely the plane $(i+1, n+1)$, except for the transformation of the last column of the augmented matrix. After n steps the minimization problem (2.1) is transformed to the equivalent problem

$$\min_f \left\| \begin{bmatrix} K_\mu \\ 0 \end{bmatrix} f - \begin{bmatrix} \bar{g} \\ h \end{bmatrix} \right\|,$$

whose solution is obtained by applying the back-substitution process to the equation (2.2).

Because K_μ is not Toeplitz, it would seem that $n(n+1)/2$ elements have to be stored. However, during the back-substitution process, in order to determine successive components of the solution vector only one row of the matrix K_μ is needed at a time. But row i can be recovered from its last element and row $i+1$, by applying an inverse rotation in the plane $(i, n+1)$. Thus, by regenerating rows we are able to reduce storage to $O(n)$ at the expense of some extra computation.

The process of recovering rows and back substitution can be illustrated by considering its ith step. After $i-1$ steps have been completed we have the following picture:

$$
\begin{bmatrix}
 & & & & & & & & & k_n^{(1)} \\
 & & & & & & & & & \vdots \\
 & & & & & & & & & k_{i+1}^{(n-i)} \\
(n-i+1)\rightarrow & k_1^{(n-i+1)} & k_2^{(n-i+1)} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & k_i^{(n-i+1)} \\
 & & k_1^{(n-i+2)} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & k_{i-1}^{(n-i+2)} \\
 & & & \cdot & & & & & & \cdot \\
 & & & & \cdot & & & & & \cdot \\
 & & & & & \cdot & & & & \cdot \\
 & & & & & & & & & k_1^{(n)} \\
(n+1)\rightarrow & 0 & \cdots & 0 & l_{n-i+2}^{(n-i+1)} & l_{n-i+3}^{(n-i+1)} & \cdot & \cdot & \cdot & l_n^{(n-i+1)} \\
 & & & & & \cdot & & & & \cdot \\
 & & & & & & \cdot & & & \cdot \\
 & & & & & & & \cdot & & \cdot \\
 & & & & & & & & \cdot & \cdot \\
 & & & & & & & & & l_{n-i+2}^{(n-i+1)} \\
 & & & & & & & & & 0 \\
 & & & & & & & & & \vdots \\
 & & & & & & & & & 0
\end{bmatrix}
$$

$$
\times \begin{bmatrix} \vdots \\ f_{n-i+1} \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} g_1^{(1)} \\ \vdots \\ g_{n-i}^{(n-i)} \\ g_{n-i+1}^{(n-i+1)} \end{bmatrix},
$$

$$
\boxed{(c_1, s_1), \ldots, (c_{n-i+1}, s_{n-i+1})}, \tag{2.5}
$$

where dots represent coefficients and unknowns to be computed, and only

elements within boxes, including rotation parameters $(c_1, s_1), \ldots, (c_{n-i+1}, s_{n-i+1})$ are actually stored.

In the ith step, by applying to rows $n-i+1$ and $n+1$ the inverse transformation to that which zeroed the $(n-i+1)$th diagonal of the matrix L in the triangularization step, we get the $(n-i)$th row of K_μ, i.e. the coefficients $k_1^{(n-i)}, \ldots, k_i^{(n-i)}$. This allows us to compute the next component f_{n-i} of the solution vector f via back substitution. Now we store the following quantities:

$(n-i)$th row of K_μ, i.e.

$$k_1^{(n-i)}, \ldots, k_{i+1}^{(n-i)},$$

last column of L, i.e.

$$k_n^{(1)}, \ldots, k_{i+2}^{n-i-1},$$

first row of L, i.e.

$$l_{n-i+1}^{(n-i)}, \ldots, l_n^{(n-i)},$$

solution vector, i.e.

$$f_{n-i}, f_{n-i+1}, \ldots, f_n,$$

rhs vector, i.e.

$$g_1^{(1)}, \ldots, g_{n-i}^{(n-i)},$$

rotation parameters, i.e.

$$(c_1, s_1), \ldots, (c_{n-i}, s_{n-i}).$$

These correspond to the quantities within boxes in (2.5) with i replaced by $i+1$.

The complete algorithm can be represented as follows.

ALGORITHM BE.

Data:

$$k = [k[1],\ldots,k[u]],$$

$$l = [l[1],\ldots,l[n]],$$

$$g = [g[1],\ldots,g[n]],$$

$$h = [0,\ldots,0],$$

$$\begin{pmatrix} c \\ s \end{pmatrix} = \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix},\ldots,\begin{pmatrix} 0 \\ 0 \end{pmatrix}\right],$$

$$f = [0,\ldots,0].$$

Phase 1 (triangularization):

for $i = 1,2,\ldots,n$ do

(a) calculate $\begin{pmatrix} c[i] \\ s[i] \end{pmatrix}$ so that $s[i]k[1] = c[i]l[i]$;

(b) $$\begin{bmatrix} k[1], k[2],\ldots, k[n-i+1] \\ 0, l[i+1],\ldots, l[n] \end{bmatrix}$$
$$:= \begin{bmatrix} c[i] & s[i] \\ -s[i] & c[i] \end{bmatrix} \begin{bmatrix} k[1],\ldots, k[n-i+1] \\ l[i],\ldots, l[n] \end{bmatrix};$$

(c) $$\begin{bmatrix} g[i],\ldots, g[n] \\ h[1],\ldots, h[n-i+1] \end{bmatrix}$$
$$:= \begin{bmatrix} c[i] & s[i] \\ -s[i] & c[i] \end{bmatrix} \begin{bmatrix} g[i],\ldots, g[n] \\ h[1],\ldots, h[n-i+1] \end{bmatrix}.$$

Phase 2 (back substitution):

for $i = n, n-1,\ldots,1$ do

(d) $$\begin{bmatrix} k[1],\ldots, k[n-i+1] \\ l[i],\ldots, l[n] \end{bmatrix}$$
$$:= \begin{bmatrix} c[i] & -s[i] \\ s[i] & c[i] \end{bmatrix} \begin{bmatrix} k[1], k[2],\ldots, k[n-i+1] \\ 0, l[i+1],\ldots, l[n] \end{bmatrix};$$

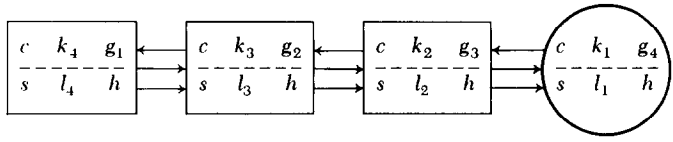
(e) $$f[i] := \left(g[i] - \sum_{j=1}^{n-i} k[j+1]f[i+j]\right) \Big/ k[1].$$

3. SYSTOLIC IMPLEMENTATION

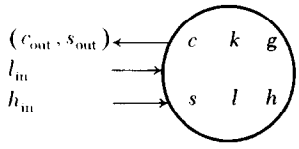
In this section we describe a systolic version of Algorithm BE. First we consider the case when the problem "fits" the systolic array, i.e., the dimension n of the problem is equal to the number N of processing cells. The case $n > N$ is considered in Section 4.

The complexity of systolic algorithms is measured in units of time. A time unit is the maximal time that is necessary for a processor (often called a cell) to perform its set of operations together with transferring data to and from neighboring processors.

The cost of data transfer may be greater than the cost of arithmetic operations in the typical case when the number of arithmetic operations performed during one unit of time is moderate. Thus we shall mainly focus on data communication. We assume that we have at our disposal a synchronous



Initially all s, c and h registers are empty.

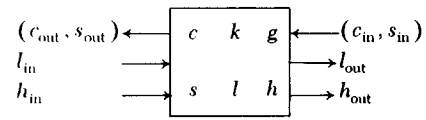


If new data on input lines then
begin

$$\begin{pmatrix} l \\ h \end{pmatrix} \leftarrow \begin{pmatrix} l_{\text{in}} \\ h_{\text{in}} \end{pmatrix}; \text{ compute } \begin{pmatrix} c \\ s \end{pmatrix} \text{ such that } cl = sk;$$

$$\begin{pmatrix} k & g \\ l & h \end{pmatrix} \leftarrow \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} k & g \\ l & h \end{pmatrix}; \begin{pmatrix} c_{\text{out}} \\ s_{\text{out}} \end{pmatrix} \leftarrow \begin{pmatrix} c \\ s \end{pmatrix}$$

end.



If new data on input lines then
begin

$$\begin{pmatrix} c & l \\ s & h \end{pmatrix} \leftarrow \begin{pmatrix} c_{\text{in}} & l_{\text{in}} \\ c_{\text{in}} & h_{\text{in}} \end{pmatrix};$$

$$\begin{pmatrix} k & g \\ l & h \end{pmatrix} \leftarrow \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} k & g \\ l & h \end{pmatrix};$$

$$\begin{pmatrix} c_{\text{out}} & l_{\text{out}} \\ s_{\text{out}} & h_{\text{out}} \end{pmatrix} \leftarrow \begin{pmatrix} c & l \\ s & h \end{pmatrix}$$

end.

FIG. 1. Phase 1 cell definitions.

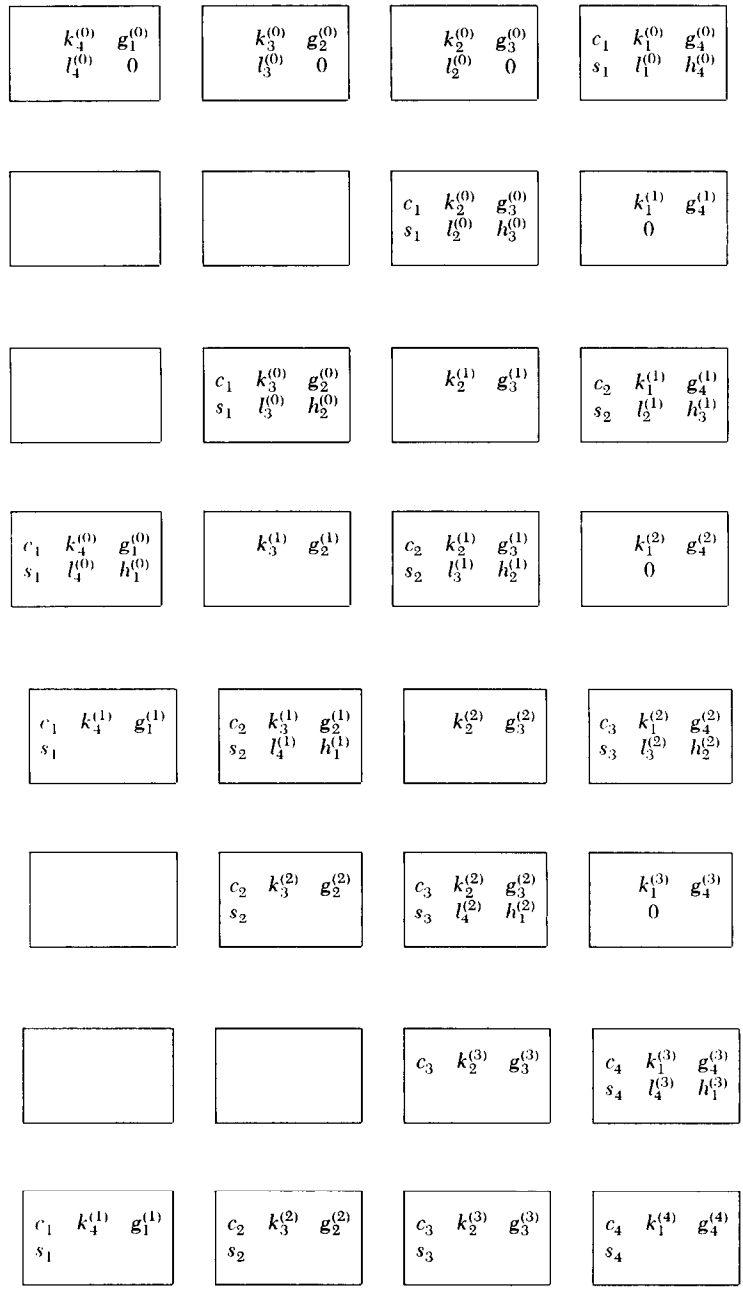
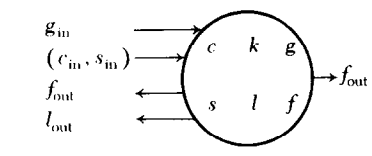


FIG. 2. Computation of phase 1.

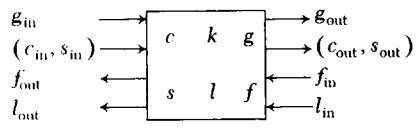
system of n linearly connected microprogrammable cells. This allows us to change the set of operations performed by each cell when necessary. Our approach is slightly different from the usual approach where truly special-purpose devices with a fixed set of operations are considered.

There are two types of operations being performed in phase 1: determination of the rotation parameters c and s, and application of the rotations. These two operations can be separated in the sense that only one cell, the boundary cell, will generate a plane rotation while the other (internal) cells apply and propagate a rotation. The array which implements phase 1 and its cell definitions are illustrated in Figure 1 for the case $n = 4$. Here, the rightmost cell is the boundary cell; the other cells are internal cells.

Prior to execution of Phase 1, the vectors $k = (k_1^{(0)}, k_2^{(0)}, \ldots, k_n^{(0)})$, $l = (l_1^{(0)}, \ldots, l_n^{(0)})$, and $g = (g_1^{(0)}, \ldots, g_n^{(0)})$ are loaded into the cells, one component per cell. The components of the vector k stay in the same cells throughout the computation. The vectors h and l move from cell to cell from left to right, the latter being successively annihilated in the boundary cell. The rotation parameters, which are generated in the rightmost cell, move in the opposite direction, meeting consecutive components of the transformed vectors k and



If new data on input lines then
begin
$f \leftarrow g/k$; $\begin{pmatrix} k \\ l \end{pmatrix} \leftarrow \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} k \\ 0 \end{pmatrix}$;
$\begin{pmatrix} g \\ c \\ s \end{pmatrix} \leftarrow \begin{pmatrix} g_{in} \\ c_{in} \\ s_{in} \end{pmatrix}$; $\begin{pmatrix} f_{out} \\ l_{out} \end{pmatrix} \leftarrow \begin{pmatrix} f \\ l \end{pmatrix}$
end.



If new data on input lines then
begin
$\begin{pmatrix} f \\ l \end{pmatrix} \leftarrow \begin{pmatrix} f_{in} \\ l_{in} \end{pmatrix}$;
$g \leftarrow g - kf$
$\begin{pmatrix} k \\ l \end{pmatrix} \leftarrow \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} k \\ l \end{pmatrix}$;
$\begin{pmatrix} g_{out} \\ c_{out} \\ s_{out} \\ f_{out} \\ l_{out} \end{pmatrix} \leftarrow \begin{pmatrix} g \\ c \\ s \\ f \\ l \end{pmatrix}$; $\begin{pmatrix} g \\ c \\ s \end{pmatrix} \leftarrow \begin{pmatrix} g_{in} \\ c_{in} \\ s_{in} \end{pmatrix}$
end.

FIG. 3. Phase 2 cell definitions.

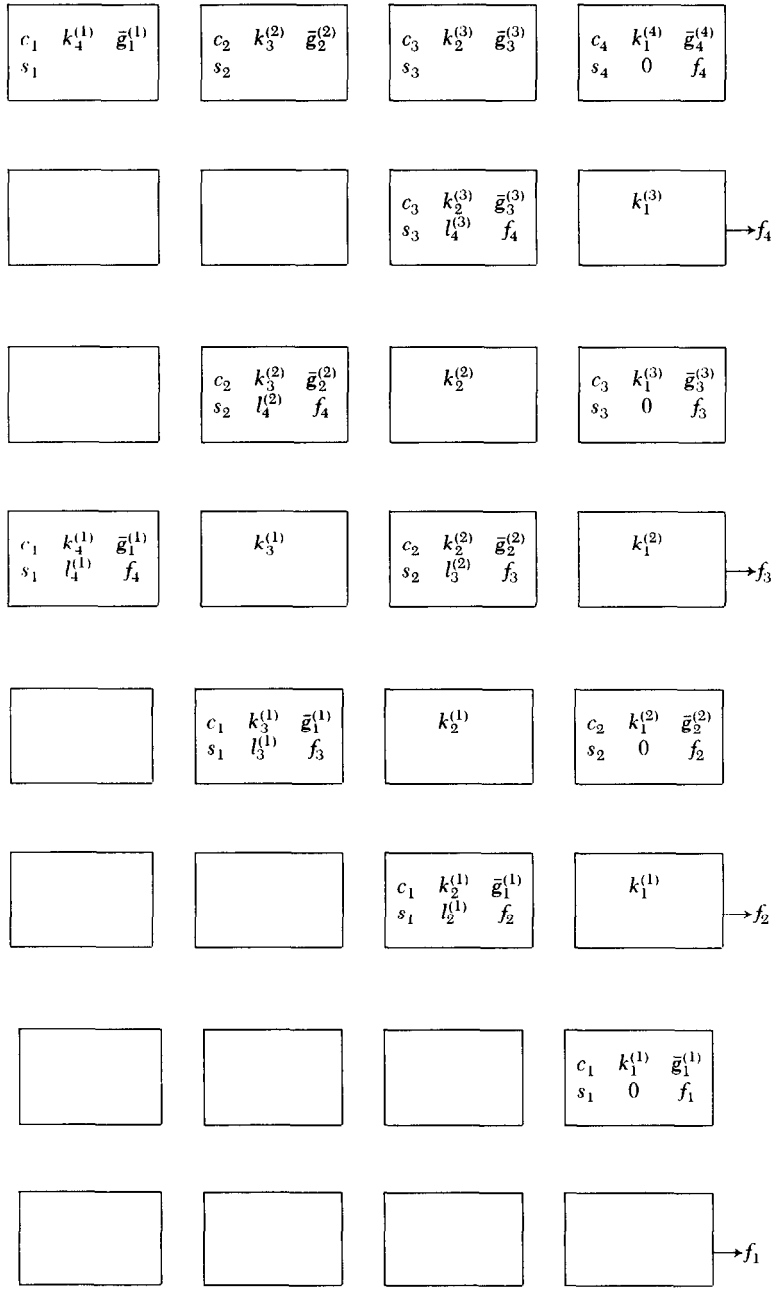


FIG. 4. Computation of phase 2.

l in the appropriate order. The computation is illustrated in Figure 2. Successive rows in the figure illustrate the state of the systolic array at successive units of time.

The way in which the transformed coefficients are stored in the systolic array after phase 1 is completed allows us to start phase 2 without any I/O delays. In phase 2 the cells regenerate rows of the matrix K_μ and simultaneously accumulate the inner products for back substitution. The cell definitions are given in Figure 3. Rotation parameters and partial inner-product terms move from left to right, the l_i move in the opposite direction, and the k_i stay in their cells. To ensure that each f_i is able to accumulate the appropriate terms, adjacent cells are active on alternate cycles. Hence, the output is collected from the rightmost cell every second cycle. The computation is illustrated in Figure 4 for $n = 4$.

Notice that in both phases only about one half of the cells are doing useful work at any time. Even-numbered cells operate at even time units, and odd-numbered cells operate at odd time units. This guarantees that computations in neighboring cells are separated and prevents conflicts or races.

4. SIMULATING THE FULL ARRAY

In this section we show how a fixed-size linear array of N cells, supported by an appropriate memory system for partial results, can generate a solution to (2.1) for an arbitrary $n > N$. Both phases of Algorithm BE have to be refined, as now our systolic array can effectively work on only a part of the matrices K and L.

In phase 1, the triangularization phase, we repeat an *annihilation step* followed by an *update step* a number of times until the whole matrix is processed.

The annihilation step is exactly the same as in phase 1 for the case when $n = N$. On completion of the annihilation step the first N diagonals of L are zeroed. Figure 5 shows the structure of the transformed matrix $\begin{bmatrix} K \\ \mu L \end{bmatrix}$ and the transformed right-hand-side vector $\begin{bmatrix} g \\ h \end{bmatrix}$ for $n = 9$ and $N = 4$. Notice that $n - N$ superdiagonals of the matrix L are unchanged and that the elements of the transformed matrix K are determined by the elements of its upper left $N \times N$ triangular submatrix.

The update step restores the Toeplitz structure of the last $n - N$ rows of the matrix K and updates nonzero elements of the matrix L accordingly. This is done as follows.

$$\begin{bmatrix}
k_1^{(1)} & k_2^{(1)} & k_3^{(1)} & k_4^{(1)} & k_5 & k_6 & k_7 & k_8 & k_9 & g_1^{(0)} \\
 & k_1^{(2)} & k_2^{(2)} & k_3^{(2)} & \cdot & \cdot & \cdot & \cdot & k_8 & g_2^{(0)} \\
 & & k_1^{(3)} & k_2^{(3)} & \cdot & \cdot & \cdot & \cdot & k_7 & g_3^{(0)} \\
 & & & k_1^{(4)} & \cdot & \cdot & \cdot & \cdot & k_6 & g_4^{(0)} \\
 & & & & k_1^{(4)} & \cdot & \cdot & \cdot & k_5 & g_5^{(0)} \\
 & & & & & \cdot & \cdot & \cdot & k_4^{(1)} & g_6^{(1)} \\
 & & & & & & \cdot & \cdot & k_3^{(2)} & g_7^{(2)} \\
 & & & & & & & \cdot & k_2^{(3)} & g_8^{(3)} \\
 & & & & & & & & k_1^{(4)} & g_9^{(4)} \\
 & & & & l_5 & l_6 & l_7 & l_8 & l_9 & 0 \\
 & & & & & \cdot & \cdot & \cdot & l_8 & 0 \\
 & & & & & & \cdot & \cdot & l_7 & 0 \\
 & & & & & & & \cdot & l_6 & 0 \\
 & & & & & & & & l_5 & 0 \\
 & & & & & & & & & h_6^{(4)} \\
 & & & & & & & & & h_7^{(3)} \\
 & & & & & & & & & h_8^{(2)} \\
 & & & & & & & & & h_9^{(1)}
\end{bmatrix}$$

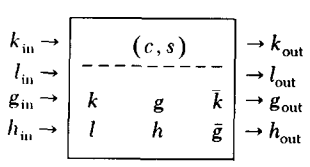
FIG. 5. Structure of the transformed augmented matrix.

First, we complete the rotation in plane $(1, n+1)$ by rotating the remaining (not yet transformed) parts of the vectors k and l, i.e., the vectors $(k_{N+1}^{(0)}, \ldots, k_n^{(0)})$ and $(l_{N+1}^{(0)}, \ldots, l_n^{(0)})$. Then, rotating in plane $(2, n+1)$, we obtain the correct values for the second row of K_μ. As the last transformation, we rotate in plane $(N, n+1)$ the vectors $(k_{N+1}^{(N-1)}, \ldots, k_n^{(N-1)})$ and $(l_{N+1}^{(N-1)}, \ldots, l_n^{(N-1)})$.

Now we describe the systolic implementation of the update step. Before executing the update step, the systolic array has data stored as in the last row in Figure 2 with k_i's shifted to the right by one cell. New data, i.e., remaining parts of unprocessed vectors k and l, are fed into the systolic array via its leftmost cell. During the computation, rotation parameters stay in fixed cells, and both k and l data streams move from left to right, but the l_i move twice as fast as the k_i. All cells perform the same set of operations, namely a single rotation (see the definition in Figure 6).

The computation is illustrated in Figure 7 for $n = 9$ and $N = 4$.

After N units of time the transformed k_i and l_i begin to emerge at the other end of the systolic array. When all components of k and l are



$$\begin{pmatrix} k_{out} \\ g_{out} \end{pmatrix} \leftarrow \begin{pmatrix} \bar{k} \\ \bar{g} \end{pmatrix};$$

$$\begin{pmatrix} \bar{k} \\ \bar{g} \end{pmatrix} \leftarrow \begin{pmatrix} k \\ g \end{pmatrix};$$

$$\begin{pmatrix} k & g \\ l_{out} & h_{out} \end{pmatrix} \leftarrow \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} k_{in} & g_{in} \\ l_{in} & h_{in} \end{pmatrix}$$

FIG. 6. Cell definition for update step (phase 1).

processed, the first $2n - N$ rows of the augmented matrix have the following form:

$$\begin{bmatrix}
k_1^{(1)} & k_2^{(1)} & k_3^{(1)} & k_4^{(1)} & k_5^{(1)} & k_6^{(1)} & k_7^{(1)} & k_8^{(1)} & k_9^{(1)} & g_1^{(1)} \\
 & k_1^{(2)} & k_2^{(2)} & \cdot & \cdot & \cdot & \cdot & \cdot & k_8^{(2)} & g_2^{(2)} \\
 & & k_1^{(3)} & k_2^{(3)} & \cdot & \cdot & \cdot & \cdot & k_7^{(3)} & g_3^{(3)} \\
 & & & k_1^{(4)} & k_2^{(4)} & \cdot & \cdot & \cdot & k_6^{(4)} & g_4^{(4)} \\
 & & & & k_1^{(4)} & \cdot & \cdot & \cdot & k_5^{(4)} & g_5^{(4)} \\
 & & & & & \cdot & & & \cdot & \cdot \\
 & & & & & & \cdot & & \cdot & \cdot \\
 & & & & & & & \cdot & \cdot & \cdot \\
 & & & & & & & & k_1^{(4)} & g_9^{(4)} \\
 & & & & l_5^{(4)} & \cdot & \cdot & \cdot & l_9^{(4)} & h_1^{(4)} \\
 & & & & & \cdot & & & \cdot & \cdot \\
 & & & & & & \cdot & & \cdot & \cdot \\
 & & & & & & & \cdot & \cdot & \cdot \\
 & & & & & & & & l_5^{(4)} & h_5^{(4)}
\end{bmatrix}.$$

The first N components of the last column of K are saved in auxiliary storage for the back substitution phase. Rows $N+1$ and $n+1$ serve as data for a new annihilation-update cycle. The annihilation-update cycle is applied $\left\lceil \frac{n}{N} \right\rceil$ times to smaller and smaller matrices until the triangularization phase is completed.

Phase 2, the back-substitution phase, is also a sequence of pairs of steps, this time the *solution* and the *update* steps. In the solution step we recover N components of the solution vector. The update step restores the structure of the problem for the next solution step. Each update step decreases the size of the problem by N, i.e., n becomes $n - N$.

The solution step operates on the $N \times N$ bottom-right triangular submatrix of a current copy of the matrix K_μ. From these data, using the systolic

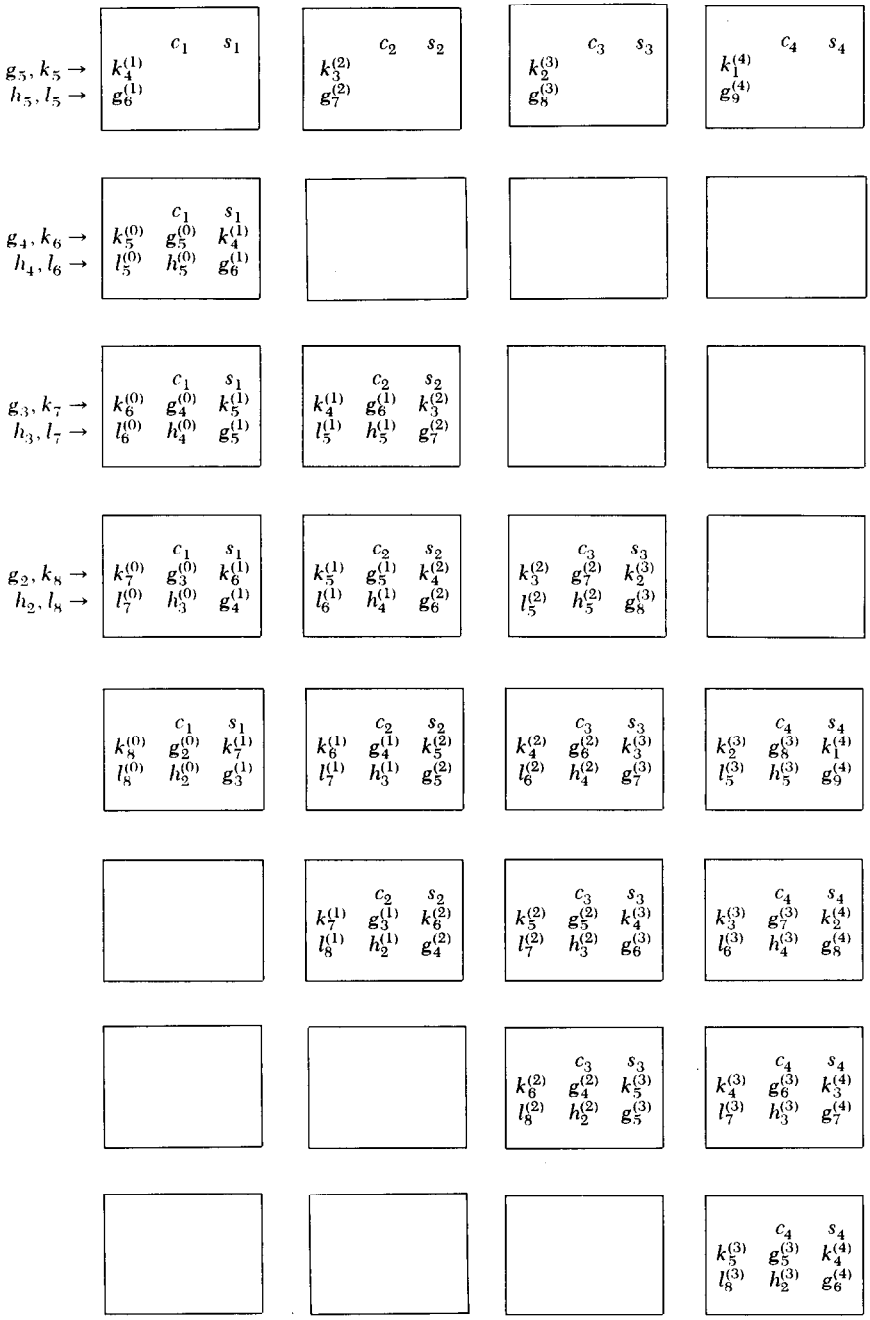


FIG. 7. The update step (phase 1).

array described in Section 3, we can compute the last N components of the solution vector. Additionally, the solution step produces the $(n-N)$th row of the matrix K_μ and the first row of the current copy of L. This is illustrated in Figure 8, where $n=9$, $N=4$.

The update step implicitly regenerates the last N components of rows $n-N,\ldots,1$ of the matrix K_μ, and (making use of N already computed components of the solution vector) performs partial back substitution on the available segments of rows $n-N$ to 1. This computation gives us a new right-hand-side vector g and, at the same time, the $(n-N)$th column of the matrix K_μ. The new g and the $(n-N)$th column of K_μ serve as data for a new solution-update step in which n is replaced by $n-N$.

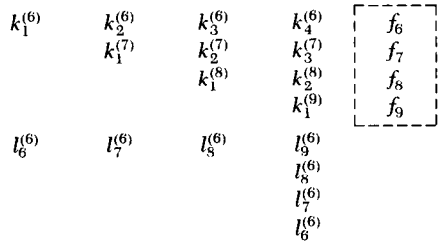


FIG. 8. Effect of the first solution step.

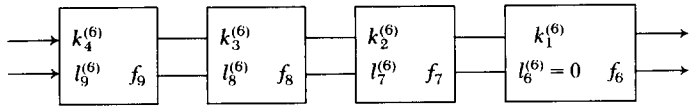


FIG. 9. Data before update step (phase 2).

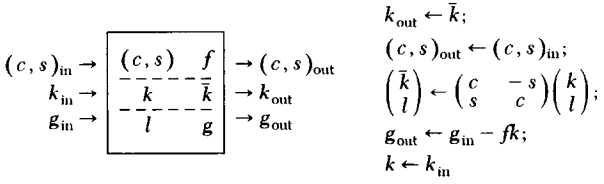


FIG. 10. Cell definition for update step (phase 2).

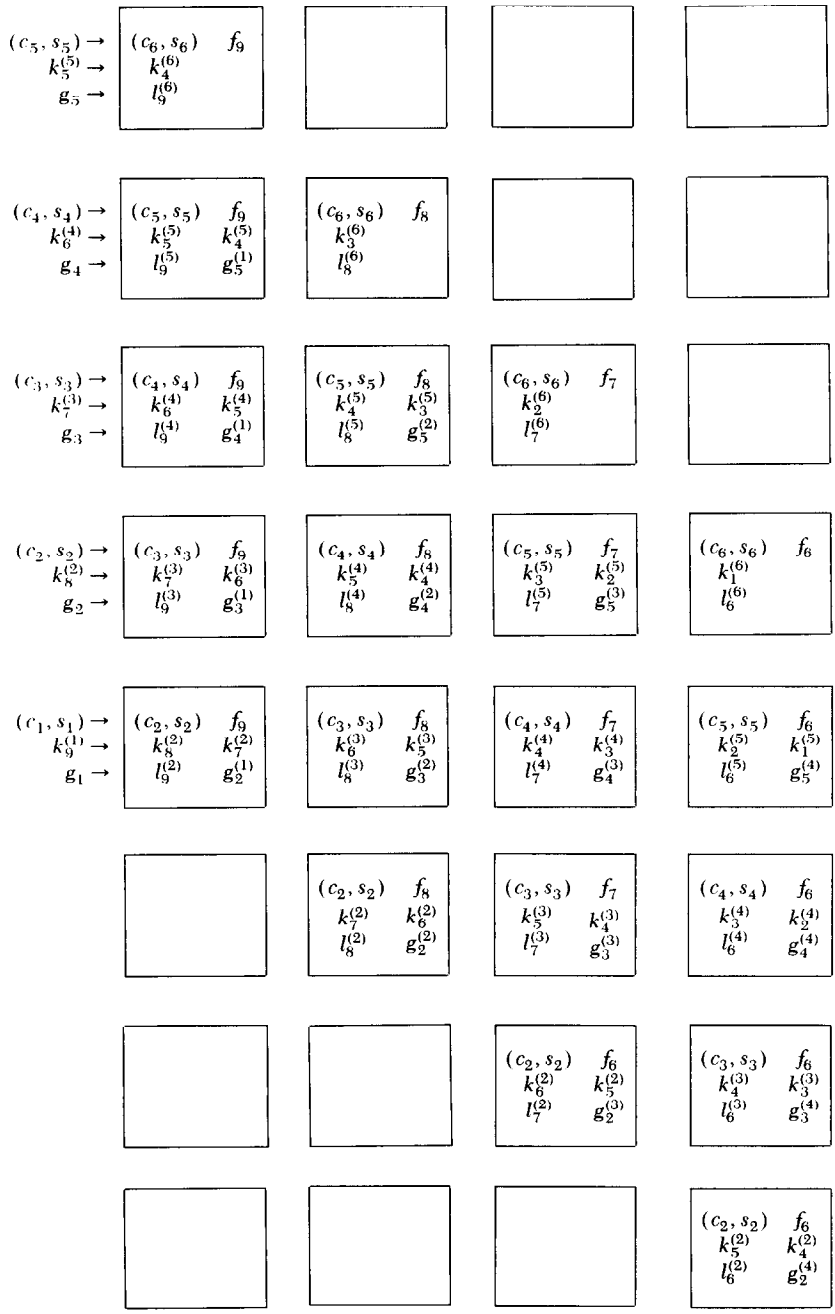


FIG. 11. The update step (phase 2).

Before the update step starts, data are stored in the systolic array as shown in Figure 9. This storage scheme corresponds to the one left after the solution step is finished. The new data—i.e., the rotation parameters, the nth column of the matrix K_μ, and the components of the right-hand-side vector g—are input through the leftmost cell and move to the right. The rotation parameters and the g_i move at normal speed, but the k_i stay in each cell they pass for one extra time unit, thus taking twice as long to move through the systolic array. The components of f and l stay in fixed cells throughout the computation. The function of each cell is identical and is defined in Figure 10.

Figure 11 illustrates part of the process for the case $n = 9$, $N = 4$.

We thank Dr. M. A. Saunders for introducing us to the algorithm of Eldén [2].

REFERENCES

1 R. P. Brent and F. T. Luk, A systolic array for the linear-time solution of Toeplitz systems of equations, *J. VLSI and Comput. Systems* 1(1):1–22 (1983).

2 L. Eldén, An efficient algorithm for the regularization of ill-conditioned least squares problems with triangular Toeplitz matrix, *SIAM J. Sci. Statist. Comput.* 5(1):229–236 (Mar. 1984).

Received 28 August 1984; revised 28 January 1985