Report Number: WUCS-87-8

1987-04-01

A Unified Systolic Array for Adaptive Beamforming

Authors: Adam W. Bojanczyk and Franklin T. Luk

We present a new algorithm and systolic array for adaptive beamforming. Our approach improves on McWhirter's pioneering work in two respects. First, our algorithm uses only orthogonal transformations and this should have better numerical properties. Second, the algorithms can be implemented on one single pxp triangular array of programmable processors that offers a throughput of one residual element per cycle.

A UNIFIED SYSTOLIC ARRAY FOR ADAPTIVE BEAMFORMING

Adam W. Bojanczyk and Franklin T. Luk*

WUCS-87-8

April 1987

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

*Cornell University

The work of the second author was supported in part by the Office of Naval Research under contract N00014-85-K-0074.

A Unified Systolic Array for Adaptive Beamforming

Adam W. Bojanczyk

Department of Computer Science
Washington University
St. Louis, Missouri 63130

Franklin T. Luk

School of Electrical Engineering
Cornell University
Ithaca, New York 14853

ABSTRACT

We present a new algorithm and systolic array for adaptive beamforming. Our approach improves on McWhirter's pioneering work in two respects. First, our algorithm uses only orthogonal transformations and thus should have better numerical properties. Second, the algorithm can be implemented on one single $p \times p$ triangular array of programmable processors that offers a throughput of one residual element per cycle.

The work of the second author was supported in part by the Office of Naval Research under contract N00014-85-K-0074.

1. Introduction

The problem that we consider in this paper may be expressed as follows. Given a $k \times p$ ($k \leq p$) matrix C of direction constraints, a k-vector f, and an $n \times p$ ($n \geq p-k$) data matrix $X(n)$, we wish to solve the constrained least squares problem:

$$\min \| X(n)w(n) \|_2 \tag{1.1}$$

subject to

$$C\, w(n) = f. \tag{1.2}$$

The symbol $\|\cdot\| = \|\cdot\|_2$ denotes the euclidean norm. For certain beamforming applications, we are not interested in the unknown weight vector $w(n)$, but rather in the last element of the residual vector $e(n)$, which is defined by

$$e(n) = [\ e_1(n),\ e_2(n),\ \cdots,\ e_n(n)\ ]^T := X(n)w(n). \tag{1.3}$$

The ith row $x(i)^T$, where $i \leq n$, of the $n \times p$ matrix $X(n)$ represents data collected at time i, and p represents the sample size [A]. On arrival of a new data row $x(n+1)^T$, we construct the corresponding new data matrix $X(n+1)$:

$$X(n+1) = \begin{bmatrix} X(n) \\ x(n+1)^T \end{bmatrix}, \tag{1.4}$$

and set out to compute the new residual element $e_{n+1}(n+1)$.

In this paper, we present a new algorithm and a new multiprocessor array for the problem. McWhirter and Shepherd [MS] have developed an algorithm that directly extracts the residual element $e_n(n)$, without using the weight vector $w(n)$. Our approach improves on their pioneering work in two respects. First, our algorithm uses only orthogonal transformations and so may have better numerical properties. Second, we propose a single $p \times p$ triangular array of programmable processors that implements the algorithm with the throughput of one residual element per cycle.

2. Background

The usual way of solving a linearly constrained least squares problem is to transform the problem into an unconstrained one. The transformation is realized via the QR decomposition of both the data matrix $X(n)$ and the constraint matrix C. In this section we briefly outline the procedure; more details can be found in [GV] and [LH].

Assume that both matrices $X(n)$ and C have full rank. Construct a $p \times p$ orthogonal transformation P that "triangularizes" the $k \times p$ matrix C:

$$C\,P = [\,0\,,\,L\,]. \tag{2.1}$$

The matrix L is $k \times k$ *reverse lower triangular*, i.e., it has the following form:

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{bmatrix}.$$

The orthogonal matrix P is chosen as a product of plane rotations $G^j_{i,i+1}$:

$$P := \prod_{j=1}^{k} \prod_{i=1}^{p-j} G^j_{i,i+1}, \tag{2.2}$$

where the rotation $G^j_{i,i+1}$ operates in the $(i,i+1)$ plane and is chosen to zero out the (j,i) element of C. Accordingly, we change variables

$$y(n) := P^T w(n), \tag{2.3}$$

and consider the effects. First, the constraint (1.2) becomes

$$[\,0\,,\,L\,]\,y(n) = f\,;$$

so we may easily determine the last k components of $y(n)$. That is, by partitioning the vector $y(n)$ into two subvectors, a $(p-k)$-vector $z(n)$ and a k-vector v:

$$y(n) := \begin{bmatrix} z(n) \\ v \end{bmatrix}, \tag{2.4}$$

we can find v by solving the "triangular" set of equations:

$$L\,v = f\,. \tag{2.5}$$

We also transform the data matrix:

$$Y(n) := X(n)P\,, \tag{2.6}$$

and compute its QR factorization (assuming $n \geq p$):

$$Y(n) = Q(n)\begin{bmatrix} R(n) \\ 0 \end{bmatrix}, \tag{2.7}$$

where $Q(n)$ is $n \times n$ orthogonal and $R(n)$ is $p \times p$ upper triangular. (If $n < p$, we get

$$Y(n) = Q(n)R(n),$$

where $R(n)$ is $n \times p$ upper trapezoidal.) So, the function (1.1) becomes

$$\| X(n)w(n) \| = \| R(n)y(n) \|.$$

Partitioning $R(n)$ in the same conformal manner:

$$R(n) = \begin{bmatrix} S(n) & T(n) \\ 0 & U(n) \end{bmatrix}, \tag{2.8}$$

where $S(n)$ is $(p-k)\times(p-k)$ upper triangular, we obtain the desired unconstrained problem

$$\min \| \begin{bmatrix} S(n) \\ 0 \end{bmatrix} z(n) + \begin{bmatrix} t(n) \\ u(n) \end{bmatrix} \|\,, \tag{2.9}$$

where

$$t(n) := T(n)\,v \quad \text{and} \quad u(n) := U(n)\,v\,.$$

One more simplification is useful for the next section. Let $Q_u(n)$ be an $n \times n$ orthogonal transformation such that

$$Q_z(n)\begin{bmatrix} t(n) \\ u(n) \end{bmatrix} = \begin{bmatrix} t(n) \\ \mu_n \\ 0 \end{bmatrix}, \tag{2.10}$$

where

$$\mu_n = \| u(n) \| .$$

Since $Q_z(n)$ does not modify $S(n)$, we get an equivalent minimization problem:

$$\min \| \begin{bmatrix} S(n) \\ 0 \end{bmatrix} z(n) + \begin{bmatrix} t(n) \\ \mu_n \\ 0 \end{bmatrix} \| , \tag{2.11}$$

and thus the minimum equals μ_n. Once $z(n)$ is found by solving

$$S(n)\, z(n) + t(n) = 0 , \tag{2.12}$$

the solution vector $w(n)$ can be obtained from (2.3). Finally, the desired residual element $e_n(n)$ can be computed as an inner product:

$$e_n(n) = x(n)^T w(n) .$$

A way to determine the residual element without computing $z(n)$ has been given by McWhirter [M]. We will use the same idea to develop our new algorithm in the next section.

3. A Recursive Algorithm

In this section, we present a new recursive algorithm for updating the residual element. Suppose that we have computed the residual element $e_n(n)$ for the $n \times p$ problem, and that we want to tackle the $(n+1) \times p$ case. The quantities P, v, $S(n)$, $t(n)$ and μ_n are known, and the function to be minimized is

$$\| X(n+1)w(n+1) \| = \| \begin{bmatrix} R(n) \\ 0 \\ b(n+1)^T \end{bmatrix} y(n+1) \| , \tag{3.1}$$

where

$$y(n+1) := P^T w(n+1) \quad \text{and} \quad b(n+1)^T := x(n+1)^T P .$$

Partitioning the two vectors into subvectors of dimensions $p-k$ and k as before, we get

$$y(n+1) := \begin{bmatrix} z(n+1) \\ v \end{bmatrix} \quad \text{and} \quad b(n+1) := \begin{bmatrix} h(n+1) \\ g(n+1) \end{bmatrix} .$$

The function (3.1) becomes

$$\| \begin{bmatrix} S(n) \\ 0 \\ 0 \\ h(n+1)^T \end{bmatrix} z(n+1) + \begin{bmatrix} t(n) \\ \mu_n \\ 0 \\ \gamma_{n+1} \end{bmatrix} \| , \tag{3.2}$$

where

$$\gamma_{n+1} = g(n+1)^T v .$$

We need to triangularize the matrix in (3.2) using a sequence of plane rotations $V_{1,n+1}$, $V_{2,n+1}$, . . . , $V_{p-k,n+1}$. The transformation $V_{i,n+1}$ is $(n+1) \times (n+1)$ and has the familiar form

$$\begin{bmatrix} c_i^V & s_i^V \\ -s_i^V & c_i^V \end{bmatrix}$$

in the $(i,n+1)$ plane. Let us define

$$V := V_{p-k,n+1}\, V_{p-k-1,n+1} \;\cdots\; V_{1,n+1}\,, \tag{3.3}$$

and

$$\begin{bmatrix} t(n+1) \\ \mu_n \\ 0 \\ \kappa_{n+1} \end{bmatrix} := V \begin{bmatrix} t(n) \\ \mu_n \\ 0 \\ \gamma_{n+1} \end{bmatrix}. \tag{3.4}$$

The $(n+1)$-dimensional constrained minimization problem

$$\min \,||\, X(n+1)w(n+1) \,||_2 \quad s.t. \quad Cw(n+1) = d$$

thereby simplifies to

$$\min \,|| \begin{bmatrix} S(n+1) \\ 0 \\ 0 \\ 0 \end{bmatrix} z(n+1) + \begin{bmatrix} t(n+1) \\ \mu_n \\ 0 \\ \kappa_{n+1} \end{bmatrix} ||\,, \tag{3.5}$$

and the minimum value is $(\mu_n^2 + \kappa_{n+1}^2)^{1/2}$.

Let us explain how we avoid computing $z(n+1)$. The residual vector $e(n+1)$ is given by

$$e(n+1) = \begin{bmatrix} Q(n)^T & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} Q_u(n)^T & 0 \\ 0^T & 1 \end{bmatrix} V^T \begin{bmatrix} 0 \\ \mu_n \\ 0 \\ \kappa_{n+1} \end{bmatrix},$$

and the desired residual element is the last component of the vector

$$V_{1,n+1}^T\; V_{2,n+1}^T \;\cdots\; V_{p-k,n+1}^T \begin{bmatrix} 0 \\ \mu_n \\ 0 \\ \kappa_{n+1} \end{bmatrix}. \tag{3.6}$$

We have chosen the special order of rotations $V_{i,n+1}$'s to generate the *neat* result that

$$e_{n+1}(n+1) = c_1^V\, c_2^V \;\cdots\; c_{p-k}^V\, \kappa_{n+1}\,. \tag{3.7}$$

Finally, the parameter κ_{n+1} is annihilated by a rotation in the $(p-k+1,n+1)$ plane. The

operation amounts to updating both μ_n and $Q_u(n)$, as

$$\mu_{n+1} := (\mu_n^2 + \gamma_{n+1}^2)^{\frac{1}{2}} \tag{3.8}$$

and

$$Q_u(n+1) := V_{p-k+1,n+1} \begin{bmatrix} Q_u(n) & 0 \\ 0^T & 1 \end{bmatrix}. \tag{3.9}$$

The quantity μ_{n+1} is calculated explicitly, but the matrix $Q_u(n+1)$ is not. This ends the description of the $(n+1)$st step of the adaptive procedure. An outline of the the algorithm is given below.

Algorithm Beamforming

Initialization

1. Triangularize the matrix C as in (2.1), with the orthogonal transformation P stored in factored form (2.2).

2. Find v by solving (2.5).

3. Determine the QR factorization (2.7) of $Y(p-k)$.

4. Calculate the vector $t(p-k) = T(p-k)v$, and set $\mu_{p-k} = 0$.

Loop

For $n = p-k,\ p-k+1,\ \cdots$ repeat

5. Compute $b(n+1) \equiv \begin{bmatrix} h(n+1) \\ g(n+1) \end{bmatrix} = x(n+1)^T P$

6. Compute $\gamma_{n+1} = g(n+1)^T v$.

7. Compute $S(n+1)$ from $S(n)$ and $h(n+1)$, and $\begin{bmatrix} t(n+1) \\ \kappa_{n+1} \end{bmatrix}$ from $\begin{bmatrix} t(n) \\ \gamma_{n+1} \end{bmatrix}$.

8. Compute $e_{n+1}(n+1)$ using (3.8).

9. Compute μ_{n+1} using (3.9). □

The algorithm described here is a combination of the standard approach detailed in §2 and the method of McWhirter for directly extracting the residual element. In [MS] McWhirter and Shepherd used an elimination technique for transforming the constrained problem to the equivalent unconstrained one. Since elimination without pivoting may cause serious numerical errors, we propose here an approach based on orthogonal transformations.

4. Systolic Implementation

For a systolic implementation of our algorithm, we propose a mesh-connected triangular array of processors. As the algorithm is heterogeneous, it is not surprising that the array is, too; for different regions of the array execute different subtasks of the algorithm. In order to attain such flexibility we postulate that the cells be microprogrammable, i.e., a program executed by a cell can be changed if required. By allowing such generality we can guarantee a very smooth flow of data. For the sake of exposition we begin by describing a possible implementation of individual steps of the algorithm. Later we will combine these separate arrays into one unified triangular array.

As the first subtask we consider Step 1, the triangularization of the matrix C, for which we propose a systolic array similar to that described in Bojanczyk et al.[BBK]. The array is homogeneous; the basic operations performed by each cell is to first determine a sine-cosine pair and then rotate. We need the last k columns of the $p \times p$ triangular BBK array. These columns, when transposed with respect to the main antidiagonal, form a $k \times k$ upper trapezoidal array as depicted in Figure 4.1 with $k = 3$ and $p = 5$. The matrix C enters the array from the top in the usual skewed order, and the ith row of the array annihilates the first $(p-i)$ elements of the ith row of C, for $i = 1, 2, \cdots, k$.

$$
\begin{array}{ccccc}
\cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & c_{35} \\
\cdot & \cdot & \cdot & c_{34} & c_{25} \\
\cdot & \cdot & c_{33} & c_{24} & c_{15} \\
\cdot & c_{32} & c_{23} & c_{14} & \cdot \\
c_{31} & c_{22} & c_{13} & \cdot & \cdot \\
c_{21} & c_{12} & \cdot & \cdot & \cdot \\
c_{11} & \cdot & \cdot & \cdot & \cdot \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\times & \times & \times & \times & \times \\
\cdot & \times & \times & \times & \times \\
\cdot & \cdot & \times & \times & \times
\end{array}
$$

Figure 4.1

The elements of L leave the array in a skewed order as depicted in Figure 4.2. The last column of L leaves, one element at a time, through the output channel of the rightmost cell in the first row of the array. The element l_{1p} appears first, followed by l_{2p} and so on. The second-to-last column of L leaves the array through the output channel of the rightmost cell of the second row of the array. However, the appearance of $l_{2,p-1}$ takes place two units of time later, because the element $l_{1,p-1}$ is known to be zero and hence is not computed explicitly.

$$
\begin{array}{cccccccccccc}
\times & \times & \times & \rightarrow & \cdot & \cdot & \cdot & l_{35} & l_{25} & l_{15} \\
\times & \times & \times & \rightarrow & \cdot & \cdot & l_{34} & l_{24} & \cdot & \cdot \\
\times & \times & \times & \rightarrow & \cdot & l_{33} & \cdot & \cdot & \cdot & \cdot
\end{array}
$$

Figure 4.2

The transformation P, in factored form, is stored in the individual cells of the array with the parameters of $G^j_{i,i+1}$ stored in cell (i,j). Now by feeding the row vector $x(n)^T$ to the array, we obtain as output the row vector $b(n)^T \equiv [h(n)^T\ ,\ g(n)^T]$, with $h(n)^T$ appearing at the lower side of the array and $g(n)^T$ at the right side of the array as shown in Figure 4.3. The trapezoidal array thus realize Steps 3 and 5 of Algorithm Beamforming.

$$
\begin{array}{ccccccccccc}
\times & \times & \times & \times & \times & \rightarrow & \cdot & \cdot & g_5 \\
\cdot & \times & \times & \times & \times & \rightarrow & \cdot & g_4 & \cdot \\
\cdot & \cdot & \times & \times & \times & \rightarrow & g_3 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \downarrow & \downarrow & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & h_2 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & h_1 & \cdot & \cdot & \cdot & \cdot & \cdot
\end{array}
$$

Figure 4.3

To determine the vector v (Step 2) we have to solve the "triangular" system of linear equations

$$Lv = d\,.$$

This can be easily accomplished on a linear *vertical* array as illustrated in Figure 4.4. The matrix L is input to the array in exactly the same *time−space* order as it is output from the upper trapezoidal array in Step 1. We have eliminated any delays between the triangularization and the substitution stages, a feat made possible by the special form of L and by the fact that we operate on the transformed variables $y(n)$ instead of the original variables $w(n)$. The algorithm that we use here is somewhat different from the well known one of Kung and Leiserson, the main difference being that while L moves in the left-to-right direction and d moves top-down, the solution vector v stays in the array. The last component v_k is stored in the first (top) cell, the second-to-last component v_{k-1} is stored in the second cell, and finally the first component v_1 is stored in the last (bottom) cell of the array.

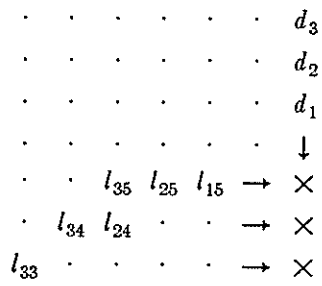

Figure 4.4

Once the linear triangular-like system is solved and the solution vector v stored in the vertical array, the vertical array is switched to execute Step 7 of the algorithm, at which point a different program is executed in the linear vertical array. Now each cell computes a partial sum of the inner product $\gamma_n = g(n)^T v$, the quantity being accumulated while it moves from top to bottom. Note that by "gluing" the upper trapezoidal array with the vertical array we can execute Steps 1 to 6 simultaneously in a pipelined fashion.

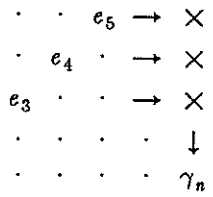

Figure 4.5

What remains to be implemented are Steps 7 to 9. They can be done on the triangular array of Gentleman and Kung as modified by McWhirter. Note that the data leaving the upper trapezoidal array and the linear vertical array are arranged exactly as required by G-K-M array.

By putting all arrays together - the upper trapezoidal array on the top, the linear vertical array to the right of it and G-K-M array at the bottom - we obtain a single upper triangular array capable of executing the algorithm in a fully pipelined fashion with *no* delays. Figure 4.6 illustrates the different regions responsible for different steps of the algorithm. Cells marked with B belong to the upper trapezoidal array and implement Steps 1 and 5. Cells marked L belong to the linear vertical array and implement Steps 2, 4 and 6. Cells marked G belong to Gentleman-Kung array and implement Steps 3, 7 and 9. Finally, cells marked M correspond to McWhirter's modification of G-K array and implement Step 8. Note that the array retains some flexibility in the sense that we can change the row dimension of the constraint matrix C dynamically without affecting the array's efficiency.

$$
\begin{array}{cccccc}
B & B & B & B & B & L \\
\cdot & B & B & B & B & L \\
\cdot & \cdot & B & B & B & L \\
\cdot & \cdot & \cdot & G & G & M \\
\cdot & \cdot & \cdot & \cdot & G & M \\
\cdot & \cdot & \cdot & \cdot & \cdot & M
\end{array}
$$

Figure 4.6

5. Final Remarks

A unified triangular systolic array has been presented for solving an important beamforming problem. We are hopeful that the theoretical array will be realized in practice, possibly as a part of the Systolic Linear Algebra Parallel Processor project at the Naval Ocean Systems Center.

References

[A] S.T. Alexander, *Adaptive Signal Processing*, Springer-Verlag, New York, NY, 1986.

[BBK] A.W. Bojanczyk, R.P. Brent and H.T. Kung, *Numerically Stable Solution of Linear Equations Using Mesh-Connected Processors*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 95-104.

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

[LH] C.L. Lawson and R.J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[M] J.G. McWhirter, *Recursive Least Squares Minimization Using a Systolic Array*, Real-Time Signal PProcesssing VI, Proceedings SPIE, vol 431, 1983.

[MS] J.G. McWhirter and T.J. Shepherd, *A Systolic Array for Constrained Least Squares Problems*, Advanced Algorithms and Architectures for Signal Processing I, J.M. Speiser, ed., Proceedings SPIE, vol. 696, 1986.