

Modified Faddeeva Algorithm for Concurrent Execution of Linear Algebraic Operations

J. GREG NASH, MEMBER, IEEE, AND SIEGFRIED HANSEN

Abstract—An algorithm is described, based on that of Faddeeva, which provides an architectural framework for systematic execution of a wide class of linear algebraic operations using a single systolic array and simple data flow. The algorithm has been modified to use numerically stable Given's rotations and is therefore suited to any matrix problem of full rank. When problem size exceeds that of the hardware array, it can be partitioned in a straightforward, numerically stable way. Numerous simulations have been performed to verify algorithm correctness.

Index Terms—Cellular processor, image processing, linear algebraic algorithms, signal processing, systolic arrays.

I. INTRODUCTION

ONE of the principal problems encountered in designing concurrent computing systems for scientific calculations is that of providing a sufficiently general range of functionality without undue addition of hardware, interconnection overhead, and software complexity. For example, although numerous proposed systolic array organizations [1], [2] can be extremely efficient for special purpose applications, it is usually difficult to justify the associated hardware and software costs considering the narrowness of the application.

An alternative approach is based on the generality of linear algebra, a class of operations that arises in a wide variety of application areas. It has been estimated that almost 75 percent of all problems in such basic sciences as economics, mechanics, engineering, physics, and business involve the solution to a linear system of equations at some point in the course of problem solution [3]. Various image and signal processing problems can also be solved using linear algebraic techniques [4]–[6]. We describe here a mesh-connected systolic/cellular processor which has been designed specifically for execution of linear algebraic operations such as matrix manipulation, linear system solution, and least squares operations. The cellular capability is desirable because there is a wide class of important operations that fall outside the domain of linear algebra, but run efficiently on cellular machines such as the Massive Parallel Processor [7]. An example is a simple element-by-element multiplication of two matrices.

The goal is not to map into a concurrent implementation all the algorithms in the immense body of knowledge associated

with linear algebra. Rather, it is to provide an architecture for fast, numerically stable solutions to the most basic, typically encountered linear system problems. Clearly, a few basic types of systolic arrays, for example those used to do matrix-matrix multiplication, triangular system solution, and matrix factorization (QR, eigenvalue, and SVD), could be used together to perform a wide variety of linear algebraic operations. Although there is an advantage in using this approach, i.e., each type of systolic array can be optimized, the overhead associated with interfacing, controlling, synchronizing, and generating the associated software for each array can be considerable. The approach suggested here is to use a single systolic array to perform all the basic functions desired. For some computations this will not result in the most efficient systolic implementation, but the enhanced generality should more than compensate for this deficiency. Our approach is based primarily on an implementation of a modification of the Faddeeva algorithm [8]. This results in a systolic architecture with a regular array of relatively simple PE's, that has a simple data flow scheme, only two different types of PE's, and straightforward control requirements. Also, by using one underlying algorithm to solve the important linear algebraic problems, there would be considerable simplification of the associated software effort, because only one parameterized program need be written.

In Section II we describe our modified Faddeeva algorithm, which is the basis for our proposed architecture, followed in Section III by a detailed description of an architectural implementation. Important issues that we address at the architectural level are those associated with the physical implementation of systolic arrays; for example, such details as how to introduce skewing delays, how to avoid multiple parallel I/O paths in and out of the array, and how to provide for alignment of data coming in and out of the array. Illustrative examples of the use of the modified Faddeeva algorithm, as in the solution of various least squares problems, are described in Section IV. The partitioning solution is discussed in Section V. Finally, we mention briefly in Section VI other non-Faddeeva-based linear algebraic algorithms that use the same systolic array.

All the algorithm mappings described in this paper, including partitioned matrix operations, have been simulated using APL. This has served to verify the correctness at a functional level of the data flow and various calculations performed in the array. The APL language is naturally suited to this task because of the ease with which linear algebraic operations can be expressed.

Manuscript received December 17, 1985; revised June 25, 1986 and November 11, 1986. This work was supported in part by NSF Grants ECS 8213358 and ECS 8016581 and the Office of Naval Research under Contract N0001 4-81-K-0191.

The authors are with Hughes Research Laboratories, Malibu, CA 90265. IEEE Log Number 8716238.

II. FADDEEVA ALGORITHM

A. Introduction

In this section we describe an algorithm of Faddeeva [9], which is the basis for a large fraction of the capabilities of our proposed processor. We will highlight its advantages and deficiencies for performing matrix computations, and then show how it can be modified to provide a wider range of capabilities with improved numerical stability.

The Faddeeva algorithm finds $d + cA^{-1}b$, given the matrix equation $Ax = b$, row vector c and scalar d . It was originally described in terms of finding $cx + d$. The problem can be codified by writing it as

$$\begin{array}{c|c} a_{11} a_{12} \cdots a_{1n} & b_1 \\ a_{21} a_{22} \cdots a_{2n} & b_2 \\ \cdots & \cdots \\ a_{n1} a_{n2} \cdots a_{nn} & b_n \\ \hline -c_1 -c_2 \cdots -c_n & d \end{array}$$

or in abbreviated form,

$$\begin{array}{c|c} A & b \\ \hline -c & d \end{array} \quad (1)$$

If a suitable linear combination of the rows above the line (from A and b) are added to the row beneath the line (e.g., $-c + WA$ and $d + Wb$, where W specifies the appropriate linear combination), so that only zeros appear in the lower left-hand quadrant, then the desired result, $cx + d$, will appear in lower right-hand quadrant. This follows because the annulment of the lower left-hand quadrant requires that

$$W = cA^{-1},$$

so that

$$d + Wb = d + cA^{-1}b.$$

Or, since $x = A^{-1}b$, we have the result

$$d + Wb = d + Cx.$$

The simplicity of the algorithm is due to the absence of a necessity to actually identify multipliers of the rows of A and the elements of b ; it is only necessary to "annul the last row." This can be done by ordinary Gaussian elimination.

One of the more important features of this algorithm, when applied to solution of linear systems, is that it avoids the usual backsubstitution or solution to the triangular linear system and obtains the values of the unknowns directly at the end of the forward course of the computation. This avoids the requirement of separate backsubstitution, or triangular solution sections, which complicate architectural implementation and can add to hardware and solution time. We note also that statistical studies we have done show that numerical accuracy is comparable to the usual LU decomposition and backsubstitution.

This result can be generalized to the case of rectangular

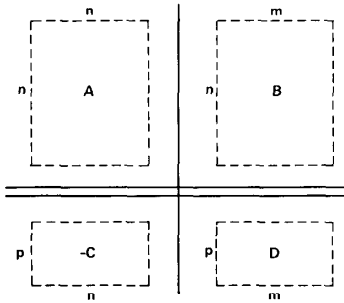


Fig. 1. Illustration of matrix positioning and sizes for Faddeeva algorithm.

$$\begin{array}{l} \begin{array}{c|c} A & I \\ \hline -I & O \end{array} \Rightarrow A^{-1} \\ \begin{array}{c|c} I & B \\ \hline -C & O \end{array} \Rightarrow CB \\ \begin{array}{c|c} I & B \\ \hline -C & D \end{array} \Rightarrow D + CB \end{array} \quad \begin{array}{l} \begin{array}{c|c} A & B \\ \hline -I & O \end{array} \Rightarrow A^{-1}B \\ \begin{array}{c|c} A & B \\ \hline -C & D \end{array} \Rightarrow CA^{-1}B + D \end{array}$$

Fig. 2. Examples of variety of matrix operations possible with Faddeeva algorithm.

matrices C , D , and B , or

$$\begin{array}{c|c} A & B \\ \hline -C & D \end{array}$$

as shown in Fig. 1. After the lower left-hand quadrant is annulled, the result $CA^{-1}B + D$ will appear in the lower right-hand quadrant.

As shown in Fig. 2, numerous matrix operations, such as multiplication, addition, and linear system solution are possible by selecting entries in the four quadrants. The Faddeeva algorithm is programmable by simply positioning the data appropriately before calculations begin.

It should be noted that a matrix of the form $D + CA^{-1}B$ is also known as the Schur complement, and appears widely in a number of applications. An exposition on the properties of the Schur complement and its uses is available [10].

B. Modified Faddeeva Algorithm

Although the Faddeeva algorithm has very desirable features, we would like to add an orthogonal factorization capability for added numerical stability. This is important because it is not easy on a nearest neighbor connected network to perform a partial pivoting operation in the Gaussian elimination procedure. Without at least partial pivoting, it is possible for a division by zero to occur even in a matrix of full rank. "Growth" in element values is another negative byproduct of this procedure.

Use of well known QR orthogonal factorization techniques (using Givens rotations in our case) avoids the difficulties cited above. In addition, a QR approach permits triangularizing matrices without changing matrix norms, which is necessary for solving over- and underdetermined systems of equations. Unfortunately, when Givens rotations are applied to the matrix

$$\begin{array}{c|c} A & B \quad m \\ \hline -C & D \quad i \\ n & p \end{array} \quad (2)$$

in the usual way (beginning in the lower left-hand corner) to annul C , where

$$Q' = \begin{bmatrix} Q'_1 & Q'_2 \\ Q'_3 & Q'_4 \end{bmatrix} \begin{matrix} n \\ p \end{matrix}$$

$m \quad i$

represents the appropriate series of rotations, the result is

$$\left[\begin{array}{c|c} R & Q'_1 B + Q'_2 D \\ \hline O & Q'_4 (CA^{-1} B + D) \end{array} \right]$$

since $Q'_3 = Q'_4 CA^{-1}$. Thus, mixing of the rows beneath the line during the rotation process causes the desired result to be modified by the matrix Q'_4 . (For the sake of illustration, in this example we are assuming A is $m \times n$, $m > n$.)

For the reasons described above, it is necessary to divide the process of annulling the lower left-hand quadrant into a two step procedure. First A is triangularized by Givens rotations (simultaneously applied to B); after this is completed, the remainder of the process can be accomplished by Gaussian elimination using the diagonal elements of R as pivot elements. In other words, after the first step (Givens rotations) we obtain

$$\left[\begin{array}{c|c} R & Q'_1 B \quad n \\ \hline O & Q'_2 B \quad m-n \end{array} \right]$$

$C \quad D$

where we have redefined $Q = [Q_1:Q_2]$. After the second step, Gaussian elimination, the final result is

$$\left[\begin{array}{c|c} R & \left[\begin{array}{c} Q'_1 B \\ Q'_2 B \end{array} \right] \begin{matrix} n \\ m-n \end{matrix} \\ \hline O & CR^{-1} Q'_1 B + D \quad i \end{array} \right]$$

$n \quad p$

where $R^{-1} Q'_1 B$ is the appropriate solution to $Ax = B$. It is important to note that as long as A is of full rank, the elements along the diagonal of R will be nonzero. Since the elimination operation uses these elements for pivoting, division by zero is not possible.

III. ARCHITECTURAL ORGANIZATION OF PROCESSING ELEMENTS

A. Introduction

In this section we describe a systolic implementation of the modified Faddeeva algorithm. (A broadcast approach is described in [11].) As detailed in Section II, the first step required is a QR factorization of the A matrix, where Q is an orthogonal matrix, and R is an upper triangular matrix. This can be done using a triangular array [12] of processing elements and passing the A matrix down through the array as shown in Fig. 3(a). As can be seen, the data are skewed in

such a way that they are not necessary to broadcast the sine and cosine values along rows of PE's. The purpose of the circular processors is to perform "rotations" on columns or vectors in such a way that zeros are introduced (corresponding to alignment of components of the vector along a major axis) for all elements a_{ij} , $i > j$. In this way the resulting matrix is triangular and is left stored as the "r" values in the PE's of Fig. 3(a).

In order to correctly process the B matrix, it is only necessary to extend the triangular matrix in the eastward direction as shown in Fig. 3(a). By passing A and B down through this array, with delays as shown, and performing the computations indicated in the circles and boxes, R and $Q'_1 B$ will be left stored in the array of PE's. (In this example, both A and B are $m \times 4$ in size, where $m \geq 4$.)

The second step in the modified Faddeeva algorithm could be accomplished as shown in Fig. 3(b). Here C and D , each of which is $i \times 4$ where i is arbitrary, are also passed down through the array of processing elements in a similar way. In this case, the set of operations performed in each PE is slightly different, as shown. The PE's indicated by the circles each zero one column of C by pivoting on the diagonal elements of R . In this case the result $CR^{-1} Q'_1 B + D$ will appear row by row coming out of the array at the bottom right. Total processing time is $O(m + i)$ and for the array shown in Fig. 3.

The triangular structure of Fig. 3 can be easily transformed into a more regular square organization shown in Fig. 4. The data flow will not be entirely uniform for this case, so that some additional control capabilities will be required. There are several advantages associated with arrangement of the PE's in this fashion, however. First, because extra circles (dashed PE's) could be made available for calculating sines and cosines, problems with varying array sizes could be accommodated. Thus, the system in Fig. 4 can be configured to accept matrices in the range $A(m \times 2)$, $B(m \times 6)$, to $A(m \times 7)$, $B(m \times 1)$. Also, with this arrangement, skewing is taken care of by the processor array itself, freeing the programmer from such considerations, and data I/O is unidirectional. This latter feature is very important from an implementation point of view, because it is difficult to interface a systolic array to a host when data enter and exit the system in parallel from more than one direction. As data leave the array, they are also effectively "realigned," ready for further processing at a later point in time. Finally, with a square array of PE's, other processing options are available. For example, data in two matrices could be processed in block format, as might be required in the simple addition of two matrices. In addition, many important algorithms can best be processed on a simple mesh-connected, cellular array.

Based on the architecture described in this section we have simulated all the algorithms listed in Table I. Clearly, this basic set could be used to solve a variety of other problems, e.g., partial differential equations, QR algorithm, and techniques using LaGrangian multipliers. Examples of problems that can be solved using just the cellular mesh capability are 1- and 2-D convolutions, sorting, median filtering, and finding histograms.

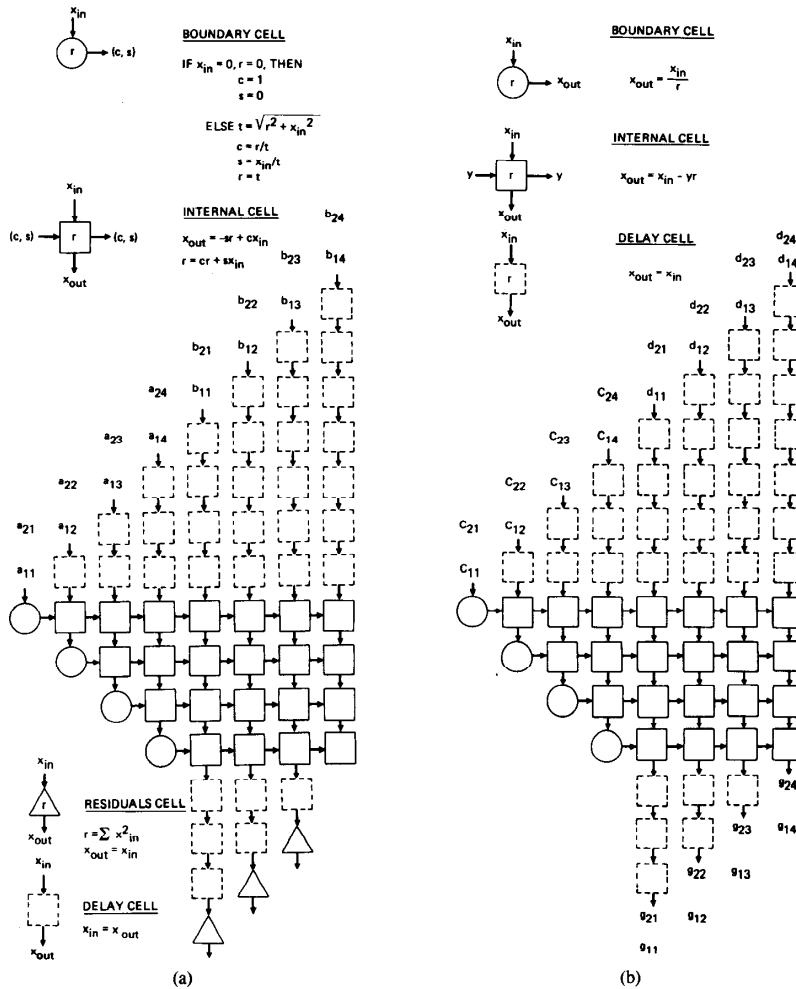


Fig. 3. (a) PE arrangement, data flow, and computations performed for modified Faddeeva algorithm during orthogonal triangularization step and (b) that for the Gaussian elimination step. The output matrix is $G = CR^{-1}Q_1B + D$. Note that the triangular processors accumulate outputs of the columns during QR factorization, the sums of which are the least square residuals.

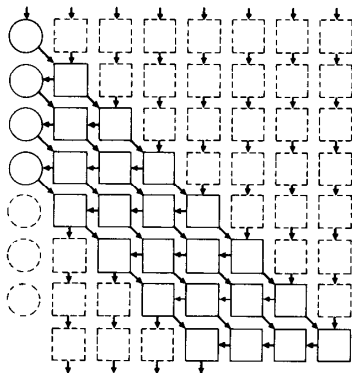


Fig. 4. Alternative square processor arrangement for set of processors of Fig. 2. Dashed circled PE's could be used for arrays larger than the example of Fig. 2.

TABLE I
 SYSTOLIC ARRAY FUNCTIONAL CAPABILITIES THAT HAVE BEEN SIMULATED

- Matrix-Vector Multiplication
- Matrix Transpose
- Vector Output Product
- Vector Inner Product
- Matrix-Matrix Addition
- Matrix-Matrix Multiplication
- Matrix-Matrix Multiplication and Addition
- Matrix Inverse
- Matrix Pseudoinverse
- Linear System Solution (Banded and Dense)
- Matrix Factorization (LU or QR)
- Least Squares Solution—Overdetermined System
- Least Squares Solution—Underdetermined System
- Generalized Least Squares

IV. LINEAR ALGEBRAIC PROCESSOR OPERATIONS

A. Introduction

In this section we give examples of how the Faddeeva-based architecture can be used to solve a variety of relatively complex linear algebra problems. We focus on least squares problems because of their relevance in signal processing. We assume throughout that the coefficient matrix A is of full rank.

Before proceeding we note that straightforward linear algebra and matrix-based problems are carried out by association of the appropriate value with the "A," "B," "C," and "D" matrices in Fig. 1. For example, operations such as an outer product of a vector $X = (x_1, x_2, x_3)$ and $Y = (y_1, y_2, y_3)$ can be performed using

$$\begin{array}{ccc|ccc}
 & & & x_1 & x_2 & x_3 \\
 I & & & x_1 & x_2 & x_3 \\
 & & & x_1 & x_2 & x_3 \\
 \hline
 -y_1 & 0 & 0 & & & \\
 0 & -y_2 & 0 & & & \\
 0 & 0 & -y_3 & & & \\
 \hline
 & & & & & 0
 \end{array} \quad (3)$$

B. Least Squares—Overdetermined

The overdetermined least squares problem corresponds to a linear system with more equations than unknowns. This type of problem is typically encountered in situations where the noise or uncertainty that is present in a system prevents an exact solution to the problem. Thus, the goal is to find the value of x that minimizes

$$\|Ax - b\|$$

where A is $m \times n$, $m > n$, x is $n \times 1$, and b is $m \times 1$. The usual procedure is to perform an orthogonal triangularization of A which, for the overdetermined case, leads to

$$Q'A = \begin{bmatrix} R \\ O \end{bmatrix} \begin{array}{l} n \\ m-n \end{array}$$

so that

$$\|Ax - b\| = \|Rx - Q_1'b\| + \|Q_2'b\| \quad (4)$$

where $Q = [Q_1:Q_2]$. The minimum value of $\|Ax - b\|$ is obtained with x as the solution to $Rx = Q_1'b$. The residual is then $\|Q_2'b\|$. These results can be found using the modified Faddeeva algorithm with the data arrangement

$$\begin{array}{c|c}
 A & b \\
 \hline
 -I & O
 \end{array} \quad (5)$$

For example, the processor arrangement shown in Fig. 3 would be suitable for computing the least squares solution, $\min \|Ax_i - b_i\|$ for $i = 1, 2, 3, 4$, where A is $m \times 4$. Note, as shown in Fig. 3, the residuals are very simply obtained by accumulation of the squares of the first $m - 4$ nonzero outputs (corresponding to the elements of $Q_2'b$) of each of the columns associated with b_i .

C. Least Squares—Underdetermined

This problem corresponds to solving $Ax = b$, where A is $m \times n$ with $m < n$, x is $n \times 1$, and b is $m \times 1$. This is equivalent to solving a problem where there are more unknowns than equations. The first step is to do a QR factorization of A , so that

$$Ax - b = [R:O]Q'x - b.$$

If we let

$$Q'x = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \begin{array}{l} m \\ n-m \end{array}$$

then y_1 can be found from the solution to the triangular system

$$Ry_1 = b$$

and y_2 is arbitrary. The usual procedure is to set $y_2 = 0$, which corresponds to taking the minimum norm solution. The underdetermined case requires that Q be applied after the solution for y_1 , so that the rotations must be accumulated during the course of the computation. This problem can be solved using the modified Faddeeva algorithm with the data entries for A , B , C , and D as follows.

$$\begin{array}{c|c}
 A' & I \\
 \hline
 -b' & O
 \end{array} \quad (6)$$

At the end of the calculation the entries will be

$$\begin{array}{c|c|c}
 m & \begin{bmatrix} R' \\ O \end{bmatrix} & \begin{bmatrix} Q' \end{bmatrix} \\
 n-m & \hline
 & O & [y_1':O] \quad Q'
 \end{array}$$

where the desired result x' is in the lower right-hand quadrant. (Of course, multiple underdetermined least squares calculations, $\min \|Ax_i - b_i\|$, for $i = 1, 2, \dots, n$, could be performed with the entries

$$\begin{array}{c|c}
 A & I \\
 \hline
 -B & O
 \end{array}$$

where $B = [b_1:b_2:\dots:b_n]'$ contains the set of right-hand side vectors. From an architectural point of view, no extra PE's are necessary when processing more than one right-hand side vector.)

D. Generalized Least Squares

In order to provide a better feel for the utility of the modified Faddeeva approach, we will illustrate how a more complex problem, such as the generalized least squares, is solved [13].

The generalized least squares problem is that of minimizing $\|w'w\|$ subject to

$$b = Ax + Bw \quad (7)$$

where A is $m \times n$ ($m > n$) and B is $m \times m$. One procedure [14] uses the orthogonal factorization of A , so that (7) results

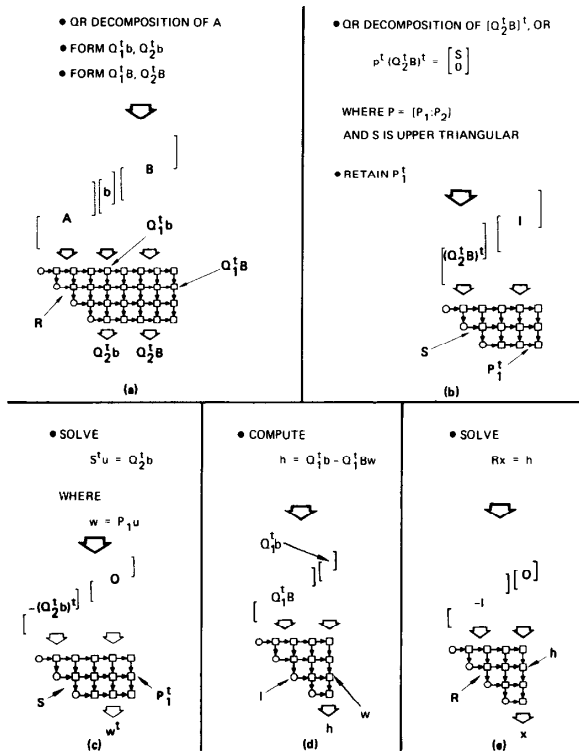


Fig. 5. Processing that occurs for solution to the generalized least squares problem. Delay cells have been omitted for clarity; arrays are included to symbolically indicate data flow.

in the two equations

$$Rx = Q_1^t b - Q_1^t B w \tag{8a}$$

and

$$0 = Q_2^t b - Q_2^t B w \tag{8b}$$

where $Q = [Q_1 : Q_2]$. These operations can be performed using only the first step in the modified Faddeeva algorithm, as illustrated in Fig. 5(a). During the factorization of A , rotations are applied to B as well as b , so that $Q_1^t b, Q_2^t b, Q_1^t B$, and $Q_2^t B$ are computed at the same time. Note that R remains in the array, while $Q_2^t b, Q_1^t B, Q_1^t b$, and $Q_2^t B$ are passed out of the array and stored in an external memory for later use. The next step involves a QR factorization of $Q_2^t B$ into an upper triangular matrix S and an orthogonal matrix P so that

$$Q_2^t B P = \begin{bmatrix} S^t & O \end{bmatrix} \quad m-n \quad n$$

with

$$P = \begin{bmatrix} P_1 & P_2 \end{bmatrix} \quad m \quad m-n \quad n$$

This operation is carried out as illustrated in Fig. 5(b), where we have introduced the identity matrix to accumulate P_1^t . Then

(8b) becomes

$$S^t P_1^t w = Q_2^t b \tag{9}$$

and the solution for x is determined from (8a). The solution for w from (9) is equivalent to solving an underdetermined system of equations, as was done in Section IV-C. There, the solution we found corresponded to the minimum norm solution. For this reason the w we calculate will be the desired minimum norm result as well, which is required for solving the generalized least squares problem. As shown in Fig. 5(c), w can be obtained very simply by passing $-(Q_2^t b)^t$ through the array as shown. In Faddeeva notation, this is equivalent to finding

$$\begin{array}{c|c} S & P_1^t \\ \hline -(Q_2^t b)^t & O \end{array} \tag{10}$$

Note that S and P_1^t are already conveniently stored in the array ready for use. Having w , we find $h = Q_1^t b - Q_1^t B w$ from (8a) using

$$\begin{array}{c|c} I & w \\ \hline Q_1^t B & Q_1^t b \end{array} \tag{11}$$

which can be done according to the flow in Fig. 5(d). Here it was assumed that the identity matrix was prestored in the triangular section. The final result x is then the solution to

$$\begin{array}{c|c} R & h \\ \hline -I & O \end{array} \tag{12}$$

where R , stored from the first step, is used as shown in Fig. 5(e).

The procedure described above for solving generalized least squares problems has been shown to be numerically stable, in that all potential problems show up in solving the triangular systems of (8a) and (9). Other important similar problems, such as the constrained least squares and weighted least squares, can be solved in a similar fashion. Note that in Fig. 5 we have only shown the data flow for the basic operations required. A few other operations such as obtaining transposes can also be performed using the same array.

V. PARTITIONED FADDEEVA ALGORITHM

Efficient partitioning of large problems is a crucial consideration in an architectural effort. Problem size is always limited by machine execution speed. Because our modified Faddeeva algorithm is the underlying foundation for our proposed processor architecture, if we can partition this algorithm, we have effectively partitioned all the linear algebraic operations shown in Table I. The Faddeeva algorithm has been the basis of other partitioning approaches [15], although several control and implementation issues still need to be resolved with these.

We consider two basic approaches to partitioning the modified Faddeeva algorithm. The first is a very straightforward multiplexing arrangement, which allows processing to be performed in a similar way to that shown in Fig. 3. The second

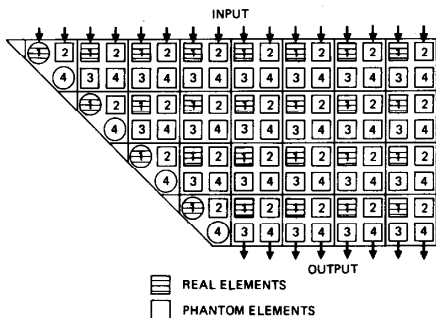


Fig. 6. Multiplexed arrangement of PE's for partitioned modified Faddeeva algorithm.

approach we discuss is more general in that it provides a scheme for decomposition of arbitrarily large matrices.

A. Multiplexed Organization

The advantage of the multiplexed arrangement is that it increases the size of the array which can be processed without requiring any fundamental change in the flow of data shown in Fig. 3. In addition, it represents a far simpler solution in terms of programming issues. The method involves building a fictitious array of the required size by adding phantom elements to each of the real elements of the prototype array. For example, Fig. 6 shows how an array, which could solve a linear system with four right-hand sides, could be configured to solve a larger problem by associating three phantom elements with each real element. This array can solve eight simultaneous equations with eight right-hand vectors. Associated with each phantom element are at least four extra registers to hold data and sin/cos values. The system speed is reduced because each PE must serially process all of the real and phantom cells.

While this approach is straightforward, the number of extra registers required grows as the square of the array size. A large RAM associated with each PE would be necessary for big arrays. Although this is possible to do, it is our goal to keep to a minimum the amount of memory distributed in the array of PE's. This not only allows more effective use of memory resources, it also results in a greater degree of integration of the PE's.

B. Partitioned Faddeeva Algorithm

Our most important concern in choosing a partitioning approach is that it be compatible with the architecture described in Section III. In other words, it should be able to use the same PE array with the same basic data flow capabilities and control. In addition, it should allow processing of matrices in a way that does not require large accumulations of intermediate results.

We begin this discussion by considering the most difficult part of the partitioning problem—that of the QR factorization of the "A" coefficient matrix and rotations of the "B" right-

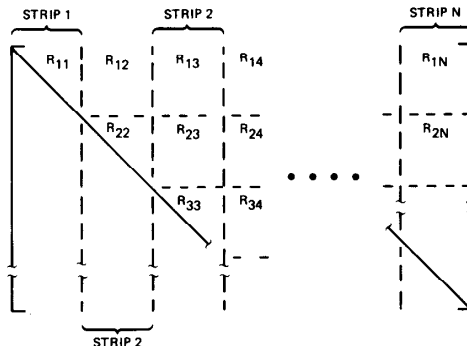


Fig. 7. Illustration of how pieces of coefficient matrix "A" and "B" are processed using Faddeeva architecture.

hand side matrix of Fig. 3. As can be seen from Fig. 3, there are two basic sets of PE groupings: the triangular sections, which contain the boundary processors, and the square sections. Our approach is to let the square array shown in Fig. 4 perform the processing associated with both sections, since it contains an embedded triangular array.

The "A" and "B" matrices can be partitioned into "strips" m elements wide, where m is the physical dimension of the hardware array (Fig. 7). One can think of the system memory feeding the PE array as containing one long strip that is a concatenated version of these strips. In the first step, STRIP 1 is fed into the triangular section in an identical manner to that shown in Fig. 3. The boundary processors are assumed to be capable of storing in local memory all the sin/cos information generated during this pass. After this step, the first result of the partitioned triangularized matrix, R_{11} , is available. This result is unloaded from the array and stored in external memory. After this operation, all the other strips in the "A" and "B" matrices must be updated or rotated with the sin/cos values produced during the processing of STRIP 1. This operation requires use of the square matrix arrangement. However, to use the square section with 100 percent efficiency in utilization of PE's, it is necessary to skew the data in the remaining strips so that sin/cos and array data are appropriately synchronized. This can be done by passing the data through the square array in a shift-register-like fashion. A simple control scheme allows us to easily skew these data. After all strips (except STRIP 1 which has already been processed) have been skewed, the strips are again passed sequentially through the square array, where data elements are rotated using the sin/cos values stored in the boundary processors. The strips pass out the bottom of the array and are stored in external memory for later use.

The next basic step is to process all of STRIP 2, except R_{12} , using the embedded triangular set of processors. Since STRIP 2 has already been skewed, it must first be deskewed by passing it again through the square section with the appropriate control. Note that after this deskewing operation is completed, calculations associated with the data R_{12} are complete. After passing all but R_{12} of the deskewed STRIP 2 through the triangular array, calculation of R_{22} is complete. This result is

then unloaded and stored and the process repeats itself with STRIP 3. The length of each successive strip decreases by an amount equal to the size of the hardware array. Because matrices are processed in strips, the only limitation to array size is the amount of system memory available for feeding the processor array.

The skewing operation does not represent a large overhead because the amount of time associated with skewing, which involves primarily shifts of data through the array, is small compared to that for the computations. In addition, the elements in the array are skewed and deskewed only once.

When the array size is not an even multiple of the hardware size, the last strip to be processed (STRIP N) simply uses only the appropriate hardware. In other words, the architecture in Fig. 4 can very easily process array sizes less than the hardware size by incorporating just the appropriate number of boundary processors.

At the end of the processing of each strip and the associated updating of the remaining elements of the array, the sin/cos values are no longer necessary and can be discarded. This eliminates the need for any large memory storage capability in the boundary processors.

The last part of the modified Faddeeva algorithm is that associated with processing "C" and "D" using Gaussian elimination. This can be accomplished in a similar manner to that described above. The "C" and "D" matrices are divided into strips in an identical way as in Fig. 7, and passed through the square and triangular sections in the same way, except that division results are stored in the boundary processors instead of sin/cos values. In addition, the square or triangular sections have to be initialized by loading the necessary blocks of partitioned triangular data values such as R_{11} , R_{12} , etc. After all the "C" elements have been annulled, the processing is completed.

VI. OTHER ALGORITHMS

As mentioned in the Introduction there are other important linear algebraic operations that can be performed on the architecture described above. Again, the goal is to map many important algorithms onto the same systolic structure. We describe two of these only briefly here since the main focus of this paper is the Faddeeva mapping.

One important matrix factorization capability, singular value decomposition, can be performed on our array based on a parallel algorithm [16] that uses a triangular architecture as well. We have simulated a variation of this algorithm on our Faddeeva-based architecture to show that they are compatible. A second class of non-Faddeeva problems that we have simulated and mapped onto the square array of Fig. 4 is that of solving large banded matrix problems, i.e., solving $Ax = b$ where A is a banded matrix [17]. For such problems, the matrix band is passed through the square array (band size must be equal to or less than the hardware array size) in systolic fashion to produce an upper triangular matrix. This is only efficient if the triangularized result passes out of the array, since the length-to-width ratio of the band is typically very large. Therefore, it is not appropriate to use the Faddeeva approach, which leaves the triangularized result in the array.

However, for compatibility our banded matrix algorithm makes use of the same square array (Fig. 4) and control structure of the modified Faddeeva algorithm. We have also shown that FFT computations can be efficiently performed in cellular fashion [18].

VII. SUMMARY

We have described here an algorithm whose utility is based on two important concepts. First, it offers a systematic means for solving a wide variety of linear algebraic problems in a very simple, numerically stable way and second, it maps directly into a simple, regular, SIMD architecture with only two basic PE's. We are now in the process of assembling and testing a system based on this architecture. It is tightly integrated with a VME bus-based 32-bit workstation and uses custom designed, high-performance 32-bit NMOS PE's to achieve good system efficiency [19].

ACKNOWLEDGMENT

The authors would like to thank Referee B for pointing out the relationship of the Faddeeva algorithm and the Schur complement.

REFERENCES

- [1] H. T. Kung, "On the implementation and use of systolic array processors," in *Proc. 1983 IEEE Int. Conf. Comput. Design*, Port Chester, NY, pp. 370-373.
- [2] J. G. Nash, "VLSI implementation of a linear systolic array," in *Proc. 1985 Int. Conf. Acoust., Speech, Signal Processing*, Tampa, FL, pp. 1392-1395.
- [3] S. J. Leon, *Linear Algebra with Applications*. New York: MacMillan, 1980.
- [4] J. Allen, "Computer architecture for digital signal processing," *Proc. IEEE*, pp. 852-873, May 1985.
- [5] J. M. Speiser and H. J. Whitehouse, "Architectures for real-time matrix operations," in *Proc. 1980 Government Microcircuit Appl. Conf.*, Houston, TX, Nov. 19 21, 1980.
- [6] H. Andrews and B. R. Hunt, *Digital Image Restoration*. Englewood Cliffs, NJ: Prentice-Hall, 1977.
- [7] J. Fisher, "The MPP at Goddard—Three years of use," in *Proc. Comput. Architecture Pattern Anal. Image Data Base Management Workshop*, Miami Beach, FL, Nov. 18-20, 1985.
- [8] J. G. Nash and S. Hansen, "Modified Faddeeva algorithm for matrix manipulation," in *Proc. 1984 SPIE Conf.*, San Diego, CA, Aug. 1984.
- [9] V. N. Faddeeva, *Computational Methods of Linear Algebra*. Dover, 1959, translated by C. D. Benster.
- [10] R. W. Cottle, "Manifestations of the Schur complement," *Linear Algebra and its Applications*, vol. 8, pp. 189-211, 1974.
- [11] J. G. Nash, S. Hansen, and G. R. Nudd, "VLSI processor arrays for matrix manipulation," in *VLSI Systems and Computations*, H. T. Kung, B. Sproull and G. Steel, Eds. Rockville, MD: Computer Science Press, 1981, pp. 367-378.
- [12] H. T. Kung, "Systolic array for orthogonal triangularization," in *Proc. SPIE*, San Diego, CA, 1981, pp. 19-26.
- [13] G. H. Golub and C. Van Loan, *Matrix Computations*. Baltimore, MD: Johns Hopkins University Press, 1983.
- [14] C. C. Paige, "Computer solution and perturbation analysis of generalized least squares problems," *Math Computat.*, vol. 33, pp. 171-183, 1979.
- [15] H. Y. H. Chuang and G. He, "A versatile systolic array for matrix computations," in *Proc. 12th Int. Conf. Comput. Architecture*, Boston, MA, June 1985, pp. 315-322.
- [16] F. Luk, "A triangular processor array for computing the singular value decomposition," Cornell Tech. Rep. TR 84-625, July 1984.
- [17] J. G. Nash, W. Przytyla, and S. Hansen, "Systolic partitioned and
- [18] K. Wojtek Przytyla, J. G. Nash, and S. Hansen, "FFT algorithm for 2-D array processors," in *Proc. SPIE*, San Diego, CA, Aug. 1987.
- [19] J. G. Nash, K. Wojtek Przytyla, and S. Hansen, "The systolic/cellular system for signal processing," *Computer*, pp. 96-97, July 1987.

banded linear algebraic computations," in *Proc. SPIE*, San Diego, CA, Aug. 1986.

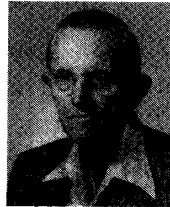
Dr. Nash is a member of the IEEE Computer Society and the Association for Computing Machinery.



J. Greg Nash (S'72-M'75) received the B.S.E. degree in engineering from Princeton University, Princeton, NJ, in 1968, and the M.S. and Ph.D. degrees in electrical engineering from University of California, Los Angeles, in 1970 and 1974.

From 1974 to 1975 he was a Postdoctoral Fellow at University of California, Los Angeles. He joined Hughes Research Labs in 1975 where he currently is in charge of an advanced computer architecture group. His current research interests are in the area of special purpose parallel processing architectures,

parallel algorithms for image and signal understanding, wafer scale integration, fault tolerance, arithmetic algorithms, and IC design.



Siegfried Hansen received the B.S.E.E. degree from the University of Washington, Seattle.

Before joining Hughes Research Laboratories in 1960, he was Director of Research at Litton Industries in Beverly Hills, having previously worked with Hughes Aircraft, Culver City, CA, and with General Electric Research Laboratories, Schenectady, NY. His research interests are in the design, production and simulations of electronic devices. He is the holder of over 100 U.S. and foreign patents.