JOURNAL OF COMPUTATIONAL PHYSICS 47, 258–280 (1982)

The Solution of Linear Complementarity Problems on an Array Processor

C. W. CRYER*†

Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge CB3 9EW, England

P. M. FLANDERS, D. J. HUNT, AND S. F. REDDAWAY

Research and Advanced Development Centre, International Computers Limited, Stevenage, Hertfordshire SG1 2BD, England

AND

J. STANSBURY*

Computer Sciences Department, University of Wisconsin–Madison, Madison, Wisconsin 53706

Received January 29, 1980; revised April 13, 1982

The Distributed Array Processor (DAP) manufactured by International Computers Limited is an array of 1-bit 200-nanosecond processors. The Pilot DAP on which the present work was done is a 32×32 array; the commercially available machine is a 64×64 array. We show how the projected SOR algorithm for the linear complementarity problem $Aw \geqslant b$, $w \geqslant 0$, $w^{\mathrm{T}}(Aw - b) = 0$, can be adapted for use on the DAP when A is the *finite-difference* matrix corresponding to the difference approximation to the Laplace operator. Application is made to two linear complementarity problems arising, respectively, from two- and three-dimensional porous flow free boundary problems.

1. INTRODUCTION

An LCP (*linear complementarity problem*) is a problem of the form: Find an n-vector $w = (w_i)$ satisfying

$$Aw \geqslant b, \tag{1.1a}$$

$$w \geqslant 0, \tag{1.1b}$$

$$w^{\mathrm{T}}(Aw - b) = 0, \tag{1.1c}$$

where $b = (b_i)$ is a known real n-vector and $A = (a_{ij})$ is a known real $n \times n$ matrix.

* Sponsored by the National Science Foundation under Grant No. MCS77-26732.
† Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

0021-9991/82/080258-23$02.00/0

Linear complementarity problems arise in many contexts (Balinski and Cottle [2]). In particular, there is a connection between linear complementarity problems and variational inequalities (Cottle, Giannessi, and Lions [5], Cryer and Dempster [9]).

Many problems in continuum mechanics can be reformulated as variational inequalities (Duvaut and Lions [10], Kinderlehrer and Stampacchia [20]), which, when discretized, reduce to linear complementarity problems of the form (1.1) with special features.

(1) A is large matrix, perhaps of order 25,000.

(2) A is a *finite-difference* or *finite-element* matrix; in particular, A is sparse with a great deal of structure. (1.2)

(3) A large percentage of the elements of the solution w are nonzero.

Because of these special features, the standard methods of solving linear complementarity problems are not very efficient, and methods of solution have been developed which take advantage of the structure of A: projected SOR (Cryer [7], Glowinski [14]); modified block SOR (Cottle, Golub, and Sacher [6]); multigrid projection (Brandt and Cryer [3]); and generalizations of projected SOR (Mangasarian [21]). Cryer [8] briefly surveys much of this work.

In the present paper we consider the use of the parallel computer DAP to solve linear complementarity problems with the features (1.2). The DAP (Distributed Array Processor, manufactured by International Computers Limited), which is an SIMD array of typically 64×64 processors, is described in Section 2. In Section 3 we describe the implementation on the DAP of projected SOR to solve a linear complementarity problem derived from a two-dimensional porous flow free boundary problem, and in Section 4 we extend this work and solve a linear complementarity problem derived from a three-dimensional porous flow free boundary problem. In Section 5 we comment on possible future developments, and the overall conclusions are in Section 6.

2. The Pilot DAP (Distributed Array Processor)

The present work was carried out on the Pilot 32×32 DAP at Stevenage, England, and we shall describe this machine first. A 64×64 version is available, and the minor differences between the two machines are indicated at the end of this section.

DAP Hardware

The essential features of the Pilot DAP hardware are as follows (Flanders *et al.* [12], Reddaway [23]):

(1) A 32×32 array of identical processing elements (PEs) with a cycle time of 200 nanoseconds.

(2) Each PE has a one-bit adder, 2K bits of storage, and three one-bit registers (a general purpose register for accessing data and performing arithmetic; a carry register; and an activity control register).

(3) Each PE is connected to its four neighboring PEs (North, South, East, and West). In a given cycle all PEs access their neighbor in the same direction (determined by the program). In addition, the PEs are linked by row and column highways which connect together all the PEs in each row and column.

(4) There is a master control unit (MCU) which broadcasts instructions to all the PEs. All PEs can perform the same instruction simultaneously, but certain instructions are only effective if the activity control register is *true*.

DAP Software

A program to run on a DAP system normally comprises a standard FORTRAN program and a number of subroutines and functions written in an array processing extension of FORTRAN known as DAP-FORTRAN (Flanders [11], Gostick [15], ICL [19]). The standard FORTRAN is executed by the host computer and provides mainly input-output and overall control. The DAP-FORTRAN is executed by the DAP and provides high speed computation. Data is shared between them using common blocks held in DAP store. Some features of DAP-FORTRAN are described below.

In addition to the data types of FORTRAN, DAP-FORTRAN has two new data types: *vector* and *matrix*. With a 32×32 DAP, a vector has 32 components and a matrix has 32×32 components; the components can be real, integer, or logical.

For example, the data statements

$$\begin{aligned} &\text{REAL } U(\ ), V(\ ,\ ), W(\ ,5), X(\ ,\ ,3) \\ &\text{INTEGER } A(\ ,1), B(\ ), C(\ ,\ ,4) \\ &\text{LOGICAL } FLAGS(\ ,2), MASK(\ ,\ ) \end{aligned} \tag{2.1}$$

declare U (a real vector), V (a real matrix), W (an array of five real vectors), X (an array of three real matrices), A (an array of one integer vector), B (an integer vector), C (an array of four integer matrices), $FLAGS$ (an array of two logical vectors), and $MASK$ (a logical matrix).

Expressions in DAP-FORTRAN consist of scalars, vectors, and matrices with the usual unary and binary operations. Operations on vectors and matrices are performed in parallel using all 32×32 PEs.

Operations between a scalar and a vector or a matrix cause implicit expansion of the scalar to the necessary dimensions. For example, if M is a matrix of size 32×32 and S is a scalar, then $M = M + S$ causes S to be implicitly expanded to size 32×32 with each element being equal to S; then the corresponding elements of "matrix" S and matrix M are added in parallel and assigned to M in parallel.

Arrays of vectors and matrices may be used to construct more complex structures . To process a vector or matrix array requires performing calculations on the individual vectors or matrices in the array.

Selection and updating of parts of vectors and matrices can be performed using the powerful indexing capabilities of DAP-FORTRAN. Matrix sections can be specified by omitting subscripts along which all elements are to be taken. Using this, whole rows or columns can be selected from matrices. For example, $M(I,)$ specifies the Ith row of matrix M.

Shift indexing is a very useful feature of DAP-FORTRAN. For example, in a simple solution of Laplace's equation on a 32×32 grid we wish to replace each element with the average of its four neighbors. This could be coded in FORTRAN as:

```
DO 10 I = 2, 31
DO 10 J = 2, 31
Y(I, J) = (X(I + 1, J) + X(I - 1, J) + X(I, J + 1) + X(I, J - 1))/4.0
10 CONTINUE
```

Further code would be needed to handle elements on the edges of the matrix.

The DAP-FORTRAN code is much simpler

$$X = (X(+,) + X(-,) + X(, +) + X(, -))/4.0. \tag{2.1}$$

The term $X(+,)$ uses shift indexing. In particular, $X(+,)$ specifies a matrix where the (I, J) element is the $(I + 1, J)$ element of X, for $1 \leqslant I \leqslant 32$ and $1 \leqslant J \leqslant 32$. Thus, $X(+,)$ contains all the *south* neighbors of X. Edge values (corresponding to subscripts 0 or 33) are defined to be zero. As an alternative, cyclic geometry may be specified by using a GEOMETRY statement.

Longer shifts can be performed by explicit system functions; for example, $\mathrm{SHS}(X, I)$ shifts the matrix X to the south I positions. Note that since all the updating is performed simultaneously, it is not necessary to write the results to another matrix.

Logical matrices and vectors can be used to select elements from an array. For example, if we wished to update only certain elements of X in statement (2.1), we could set the corresponding elements of LM, a logical matrix, to true and all other elements of LM to false. That is, if $X(I, J)$ is to contain the average of its four neighbors, then $LM(I, J)$ is set to true. Otherwise, $LM(I, J)$ is false. Then the following statement performs the required task:

$$X(LM) = (X(+,) + X(-,) + X(, +) + X(, -))/4.0.$$

DAP-FORTRAN has a number of useful system functions whose arguments and results may be scalars, vectors, or matrices. The ALTC, ALTR, MERGE, MAX, and ABS functions will be briefly described since these are used in the programs in this paper.

The functions ALTC and ALTR return logical matrices. If C is the argument to ALTC, then the first C columns of the result matrix are set to false, the next C columns to true, the next C columns to false, etc. ALTR performs similarly for rows.

The function MAX (now named MAXV) returns a scalar equal to the largest number in its vector or matrix argument. The function ABS returns a vector or matrix containing the absolute value of every element in its argument.

The function MERGE takes three arguments and returns a matrix. The first two arguments are matrices (or implicitly expanded scalars) and the third argument is a logical matrix. If the (I, J) element of the logical matrix is true then the (I, J) element of the result matrix is set equal to the (I, J) element of the first matrix; otherwise, it is set equal to the (I, J) element of the second matrix.

Examples of DAP-FORTRAN programs are given in Sections 3 and 4.

DAP Arithmetic

When a DAP-FORTRAN program is executed by the DAP, expressions involving only scalars are executed sequentially, but operations on vectors and matrices are performed in parallel by the PEs.

The DAP memory can be visualized as a cuboid, with 2 K horizontal planes, each plane being a 32×32 square of bits. The 32×32 array of PEs lies on top of the cube, and each column of 2 K bits belongs to the PE above it.

Two storage modes are used in DAP-FORTRAN, vertical and horizontal. Scalars and vectors are stored in horizontal mode while matrices are held in vertical mode.

In vertical mode, each number is held entirely within the store of one PE with successive bits in successive store locations. Thus, for an integer matrix, the sign bit of every element in the matrix would be held in the same store address of each PE.

In horizontal mode, a number is spread along a row of PEs. Thus, a scalar occupies one row while a vector occupies 32 rows. DAP instructions are also stored in this format.

All arithmetic is carried out using subroutines. Some operation times for 32 bit numbers are given in Table I.

TABLE I

Average DAP-FORTRAN Arithmetic Times for the Pilot DAP

Operation	Matrix	Vector	Scalar
Floating point addition	140–180 μs	54 μs	27 μs
Floating point multiplication	315 μs	50 μs	34 μs
Floating point multiplication by a scalar	60–200 μs	40 μs	–
One shift of a real matrix, e.g., $X(+,)$	15 μs	2 μs	–
Move a floating point matrix	15 μs	2 μs	2 μs
Logical AND	2 μs	2 μs	2 μs
Logical mask	1 μs	2 μs	–

Note. Times are slightly different on production DAPs.

It will be noted that vector arithmetic is faster than matrix arithmetic. This is because a row of PEs is available for each vector component, while only one PE is available for each matrix component.

Some of the quoted computation times are data dependent. In particular, matrix multiplication by a scalar typically varies from 170 μs to 200 μs depending upon the distribution of zeros in the binary representation of the constant; for special scalars such as 0.5 or 3 the multiplication time can be as low as 60 μs.

Host-DAP Interface

The sequence of operations for compiling and running DAP programs is as follows:

(a) The host computer compiles the host FORTRAN program and the DAP-FORTRAN subroutines into host and DAP machine codes respectively.

(b) DAP machine code, incorporating all necessary low level subroutines, is loaded into DAP memory in horizontal mode where it occupies a few bits of each PE's memory. Host machine code is loaded into the host memory.

(c) Execution begins in the host and control is transferred to the DAP as required by subroutine calls. On completion of DAP processing, the host resumes execution at the point following the call.

Detailed information on the Pilot DAP relevant to understanding the programs in this paper is given in the Appendix.

The Production DAP

The current production DAP is generally similar to the Pilot but differs as follows:

(a) there are 4096 PEs arranged in a 64×64 array;

(b) each PE has 4 K bits of memory;

(c) arithmetic operations differ somewhat in timing but are overall a little faster;

(d) coupling between host and DAP is more direct so the interface is simpler than indicated in the Appendix.

3. Numerical Solution of a Two-Dimensional Free Boundary Problem

The flow of water through a porous dam is a well-known model problem. Water seeps from a reservoir of height H through a rectangular dam of width L to a reservoir of height h. Part of the dam is saturated and the remainder of the dam is dry. The wet and dry regions are separated by an unknown free boundary Γ which must be found as part of the solution (Fig. 3.1).

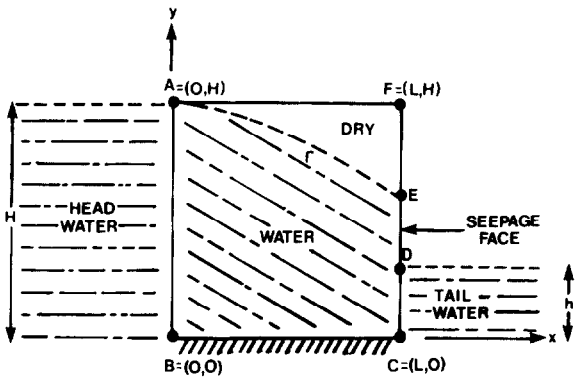

FIG. 3.1. Flow through a porous rectangular dam R.

As shown by Baiocchi [1], the problem can be formulated as follows: Find u on the rectangle $R = ABCF$ such that

$$-\nabla^2 u \geqslant -1, \quad \text{on } R, \tag{3.1a}$$

$$u \geqslant 0 \quad \text{on } R, \tag{3.1b}$$

$$u(-\nabla^2 u + 1) = 0 \quad \text{on } R; \tag{3.1c}$$

and

$$\begin{aligned} u = g &= (H - y)^2/2, && \text{on } AB, \\ &= (h - y)^2/2, && \text{on } CD, \\ &= [H^2(L - x) + h^2 x]/2L, && \text{on } BC, \\ &= 0, && \text{on } DFA. \end{aligned} \tag{3.2}$$

The wet region of the dam consists of the points where $u > 0$ and the dry region consists of the points where $u = 0$.

When the problem (3.1), (3.2) is approximated using the classical five point difference approximation for the Laplace operator, one obtains an LCP of the form (1.1), where the matrix A and right hand side b are the same as those that would be obtained if the Dirichlet problem

$$\begin{aligned} -\nabla^2 u &= -1 \quad \text{on } R, \\ u &= g \quad \text{on } \partial R \end{aligned} \tag{3.3}$$

were approximated by the finite difference equation $Aw = b$. More precisely, let an $M \times N$ grid with gridlength Δx be superimposed upon R, and let the values of u and g at the point $([j-1]\Delta x, [i-1]\Delta x)$ be denoted by u_{ij} and g_{ij}, respectively, for $1 \leqslant i \leqslant M$ and $1 \leqslant j \leqslant N$. Then (1.1) takes the form

$$4w_{ij} - w_{i+1,j} - w_{i-1,j} - w_{i,j+1} - w_{i,j-1} \geqslant -(\Delta x)^2 \qquad \text{for} \quad 1 < i < M, \quad 1 < j < N, \tag{3.4a}$$

$$w_{ij} \geqslant 0 \qquad \text{for} \quad 1 < i < M, \quad 1 < j < N, \tag{3.4b}$$

$$w_{ij}(4w_{ij} - w_{i+1,j} - w_{i-1,j} - w_{i,j+1} - w_{i,j-1} + (\Delta x)^2) = 0 \qquad \text{for} \quad 1 < i < M, \quad 1 < j < N, \tag{3.4c}$$

$$w_{ij} = g_{ij} \qquad \text{for} \quad ((j-1)\Delta x, (i-1)\Delta x) \in \partial R. \tag{3.4d}$$

We discuss below two iterative methods for solving (3.4), the projected Jacobi method and the projected SOR method. The projected Jacobi method is much slower than the projected SOR method, but is trivial to implement on the DAP and serves as a useful introduction to DAP programming.

TABLE II
The DAP Subroutine JACOBI

	SUBROUTINE JACOBI	
	LOGICAL *MASK*(,), *WSIGN*(,)	Declare logical 32×32 matrices, *MASK* and *WSIGN*
	REAL *W*(,), *Z*(,)	Declare real floating point 32×32 matrices *W* and *Z*
	REAL *INDEX*( )	Declare a real floating point 32-vector *INDEX*.
	EQUIVALENCE(*W*, *WSIGN*)	Declare the logical matrix *WSIGN* equivalent to the first bit, the sign bit, of the matrix *W*.
	HEIGHT = 31.0	
	WIDTH = 31.0	
	DO 10 *I* = 1, 32	Initialize *INDEX* vector.
	INDEX(*I*) = (32 − *I*)/31.0	
10	**CONTINUE**	
	W = 0	Clear matrix *W*
	TEMP = *HEIGHT* * *HEIGHT* * .5	
	W(1,) = *TEMP* * *INDEX*	Set values of the matrix *W* equal to *g* on bottom (*BC*).
	W(, 1) = *TEMP* * *INDEX* * *INDEX*	Set values of the matrix *W* equal to *g* on left (*AB*).
	MASK = .TRUE.	
	MASK(1,) = .FALSE.	
	MASK(32,) = .FALSE.	Set the matrix *MASK* to be true at interior points and false at boundary points.
	MASK(, 1) = .FALSE.	
	MASK(, 32) = .FALSE.	
	DO 50 *I* = 1, 100	Start of main loop
1	*Z* = *W*(+,) + *W*(−,) + *W*(, +) + *W*(, −) − 1.0	Sum neighbors and store in *Z* matrix.
2	*W*(*MASK*) = .25 * *Z*	Transfer average to *W* at interior points.
3	*W*(*MASK* .AND. *WSIGN*) = 0.0	Project by setting *W* = 0 at points where *MASK* is true and the sign of *W* is negative.
50	**CONTINUE**	
	END	

TABLE III

Statement	Operations	Time (μs)
$Z = W(+,) + W(-,) + W(,+)$ + $W(,-) - 1.0$	4 floating point matrix additions/subtractions	640
	4 index shifts	60
	1 scalar-matrix assignment	15
$W(MASK) = .25 * Z$	1 floating point matrix multiplication by a special constant	70
	1 logical mask	1
$W(MASK$ .AND. $WSIGN) = 0.0$	1 logical AND	2
	1 logical mask	1
	1 scalar-matrix assignment	15
DO 50 $I = 1, 100$		7
		811

The Projected Jacobi Method

Let $w^{(0)} = (w_{ij}^{(0)})$ be an initial guess for the solution $w = (w_{ij})$ of (3.4). One generates a sequence of approximations $w^{(k)} = (w_{ij}^{(k)})$, $k = 1, 2,...,$

$$z_{ij}^{(k)} = w_{i-1,j}^{(k)} + w_{i+1,j}^{(k)} + w_{i,j-1}^{(k)} + w_{i,j+1}^{(k)} - (\Delta x)^2, \tag{3.5a}$$

$$w_{ij}^{(k+1/2)} = \tfrac{1}{4} z_{ij}^{(k)}, \tag{3.5b}$$

$$w_{ij}^{(k+1)} = \max(0, w_{ij}^{(k+1/2)}), \qquad \text{for} \quad 1 < i < M \text{ and } 1 < j < N; \tag{3.5c}$$

$$w_{ij}^{(k+1)} = g_{ij}, \qquad \text{for} \quad ((j-1)\Delta x, (i-1)\Delta x) \in \partial R. \tag{3.5d}$$

It is known that the projected Jacobi method will converge (Mangasarian [21]).

If $M \leqslant 32$ and $N \leqslant 32$, then the gridpoints can be regarded as a subset of a 32×32 array, and one PE can be associated with each gridpoint. Defining $w^{(k)}$, $w^{(k+1)}$, and $z^{(k)}$ as real DAP-FORTRAN matrices, the computation (3.5) is trivial to implement on the DAP.

In Table II we list a DAP subroutine JACOBI which solves the dam problem for the case $h = 0$, $H = 31$, $L = 31$, $M = N = 32$, and $\Delta x = 1$. This subroutine could be called by a host program, which could then print the answers in the matrix W.

Using the operation times given in Table I, we can readily estimate the time required per iteration in the main loop of the JACOBI subroutine (see Table III).

From Table III we see that one projected Jacobi iteration over the whole 32×32 grid requires 811 μs.

The Projected SOR Method

Let $w^{(0)} = (w_{ij}^{(0)})$ be an initial guess for the solution $w = (w_{ij})$ of (3.4). In the usual implementation of projected SOR, one generates a sequence of approximations $w^{(k)} = w_{ij}^{(k)}$ as follows:

$$z_{ij}^{(k)} = w_{i-1,j}^{(k+1)} + w_{i+1,j}^{(k)} + w_{i,j-1}^{(k+1)} + w_{i,j+1}^{(k)} - (\Delta x)^2, \tag{3.6a}$$

$$w_{ij}^{(k+1/2)} = w_{ij}^{(k)} + \omega(z_{ij}^{(k)} - 4w_{ij}^{(k)})/4$$

$$= (\omega/4)\, z_{ij}^{(k)} + (1-\omega)\, w_{ij}^{(k)}, \tag{3.6b}$$

$$w_{ij}^{(k+1)} = \max\{0, w_{ij}^{(k+1/2)}\}, \qquad \text{for} \quad 1 < i < M \text{ and } 1 < j < N, \tag{3.6c}$$

where ω is a constant, the over-relaxation parameter.

It is known that the iteration (3.6) converges for all initial guesses $w^{(0)}$ iff $0 < \omega < 2$ (Cryer [7], Glowinski [14]). The implementation (3.6) is not suitable for parallel computation because the new values $w^{(k+1)}$ cannot be computed simultaneously; $w_{i-1,j}^{(k+1)}$ and $w_{i,j-1}^{(k+1)}$ must be known before $w_{ij}^{(k+1)}$ can be computed.

There is, however, a simple but ingenious way of making SOR suitable for parallel computation. In the implementation (3.6), we order the gridpoints by rows and columns (Fig. 3.2a). Instead, let us visualize the gridpoints as forming a red-black chess board and number first the red points and then the black points (Fig. 3.2b).

Applying projected SOR to the points numbered as in Fig. 3.2b we find that each projected SOR iteration can be broken down into two stages: in the red (first) stage projected SOR is applied to the red points; and in the black (second) stage projected SOR is applied to the black points.

Red stage.

$$z_{ij}^{(k,\text{red})} = w_{i+1,j}^{(k,\text{black})} + w_{i-1,j}^{(k,\text{black})} + w_{i,j+1}^{(k,\text{black})} + w_{i,j-1}^{(k,\text{black})} - (\Delta x)^2, \tag{3.7a}$$

$$w_{i,j}^{(k+1/2,\text{red})} = (\omega/4)\, z_{ij}^{(k,\text{red})} + (1-\omega)\, w_{ij}^{(k,\text{red})}, \tag{3.7b}$$

$$w_{i,j}^{(k+1,\text{red})} = \max\{0, w_{i,j}^{(k+1/2,\text{red})}\}. \tag{3.7c}$$

Black stage.

$$z_{ij}^{(k,\text{black})} = w_{i+1,j}^{(k+1,\text{red})} + w_{i-1,j}^{(k+1,\text{red})} + w_{i,j+1}^{(k+1,\text{red})} + w_{i,j-1}^{(k+1,\text{red})} - (\Delta x)^2, \tag{3.8a}$$

$$w_{ij}^{(k+1/2,\text{black})} = (\omega/4)\, z_{ij}^{(k,\text{black})} + (1-\omega)\, w_{ij}^{(k,\text{black})}, \tag{3.8b}$$

$$w_{ij}^{(k+1,\text{black})} = \max\{0, w_{ij}^{(k+1/2,\text{black})}\}. \tag{3.8c}$$

Each stage can be carried out in parallel, with the red (black) processors working and the black (red) processors idle.

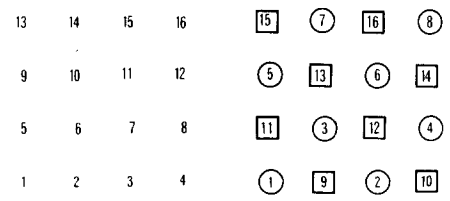

FIG. 3.2. Ordering of gridpoints (for a 4×4 grid) (a) usual, (b) red (○) and black (□).

This idea of using the red-black ordering for parallel processors has appeared several times in the literature (Heller [16]). Its use on DAP was first suggested by Hunt [17]. (In Europe, white-black chessboards are more usual than red-black ones).

In Table IV we list a DAP-FORTRAN subroutine PROJSOR for implementing the heart of the algorithm (3.7), (3.8). The subroutine is provided with several input parameters with obvious meanings. In addition, two logical matrices are provided as input: the logical matrix *MASKMASK* is true at gridpoints in the interior of the dam, and false elsewhere; the logical matrix *MASK* is true at black gridpoints and false at red gridpoints. Finally, the values of the real matrix W at the boundary points ∂R must be computed using (3.4d) before PROJSOR is called.

The computation time for one pass through the main loop of the subroutine

TABLE IV

The DAP Subroutine PROJSOR

```
      COMMON/RMAT/W( , )
      COMMON/RSCA/MAX DIFF, OMEGA, EPSILON, DAM WIDTH, DAM HEIGHT
      COMMON/ISCA/NUMB ITERATIONS, NUMB ROWS, NUMB COLS
      COMMON/SUBLMAT/MASK( , ), MASK MASK( , )
      REAL W, MAX DIFF, OMEGA, EPSILON, DAM WIDTH, DAM HEIGHT
      LOGICAL MASK, MASK MASK
      INTEGER NUMB ITERATIONS, NUMB ROWS, NUMB COLS
      REAL Z( , ), GRID2, W( , ), SAVEW( , )          Local variables.
      REAL ALPHA, BETA
      INTEGER NUMB TIMES
      LOGICAL DONE, WSIGN( , )
      EQUIVALENCE (WSIGN, W)
      W(WSIGN) = 0.0                                  Ensure that W is nonnegative everywhere.
      ALPHA = OMEGA * .25                             Calculate the constants that are needed
      BETA = 1.0 - OMEGA                                later on.
      GRID2 = (DAM HEIGHT/NUMB ROWS) ** 2
40    SAVE W = W                                      Start main loop.
      NUMB ITERATIONS = NUMB ITERATIONS + 1           Save the old value of W.
      DO 45 NUMB TIMES = 1, 2
1     MASK(MASK MASK) = .NOT. MASK                    Reverse state of MASK.
2     Z = W + W(-, -)                                 Calculate Z on only the red (or black)
3     Z = Z(+, ) + Z( , +) - GRID2                      points as determined by the MASK.
4     W(MASK) = ALPHA * Z + BETA * W
5     W(WSIGN .AND. MASK MASK) = 0.0                  Project
45    CONTINUE
      MAX DIFF = MAX(ABS(SAVEW - W))                  Find maximum difference between old
                                                        and new.
      DONE = (MAX DIFF .LE. EPSILON)                  Check if desired accuracy is attained.
      IF (.NOT. DONE) GO TO 40
      RETURN
```

TABLE V

Estimated Computation Time for the Inner Loop of PROJSOR

Statement	Operations	Time (μs)
1 *MASK(MASKMASK)* = NOT *MASK*	1 logical mask	1
	1 logical store	1
2 $Z = W + W(-,-)$	1 index shift two places	21
	1 floating point matrix addition	160
3 $Z = Z(+,\ ) + Z(\ ,+) - GRID2$	2 index shifts	30
	1 floating point matrix addition	160
	1 floating point matrix subtraction	160
	1 scalar-matrix assignment	15
4 $W(MASK) = ALPHA * Z + BETA * W$	1 floating point matrix addition	160
	2 floating point matrix multiplications by a constant	400
	1 logical mask	1
5 *W(WSIGN* .AND. *MASK MASK)* = 0.0	1 logical AND	2
	1 logical mask	1
	1 scalar-matrix assignment	15
DO 45 *NUMB TIMES* = 1, 2		7
		1135

PROJSOR is estimated in Table V, from which it follows that each PROJSOR iteration, which requires two passes through the loop, takes about $2 \times 1135\ \mu s = 2.27$ ms. To check this estimate, the average execution time per iteration in the subroutine PROJSOR was obtained by measuring (on a real external physical clock) the time required for a large number of iterations for the dam problem with $H = 24$, $h = 0$, $L = 16$, and $\Delta x = 1$. (This particular problem was chosen because it is a test problem which has been solved by many authors). The measured time per iteration on the Pilot DAP was 2.2 ms, as compared to the estimated time of 2.27 ms.

We conclude this section with some comments.

(1) For comparison, the dam problem with $H = 24$, $h = 0$, $L = 16$, and $\Delta x = 1$ was also solved on the UNIVAC 1180 at the University of Wisconsin, using the conventional ordering of gridpoints and an optimizing compiler with single precision arithmetic (36 bits), and the time per iteration was found to be 5.29 ms. For this problem the Pilot DAP was therefore 2.4 times faster than the UNIVAC 1180.

It should be noted that for this problem only $25 \times 17 = 425$ of the 1024 DAP PEs were used. On a 31×31 region the Pilot DAP would be six times faster than the UNIVAC 1180.

(2) In general, one expects to be able to predict DAP execution times to within

about 5 %, because DAP programs have little overhead and spend almost all their time in computation.

(3) Since DAP floating point operations are relatively expensive, it is worthwhile optimizing the code. (Readers who used early computers which also had relatively slow arithmetic operations may feel nostalgic). An example of such optimization occurs in the subroutine PROJSOR (see Table IV). The computation (3.7a) could have been implemented as:

$$Z = W(+,\ ) + W(-,\ ) + W(\ ,+) + W(\ ,-) - GRID2$$

which requires three additions and one subtraction, and takes

$$\begin{array}{lll} 4(15) & +\,4(160) & +\,15 \\ \text{(shifts)} & \text{(additions)} & \text{(scalar-matrix assignment)} \end{array} \qquad = 715\,\mu\text{s}.$$

By sharing intermediate results between PEs, however, the amount of arithmetic can be reduced; the implementation in PROJSOR is

$$Z = W + W(-,-)$$
$$Z = Z(+,\ ) + Z(\ ,+) - GRID2,$$

which is estimated at only 546 μs. It should be noted that both implementations use only half the PEs for arithmetic at any one time. Larger grids or three-dimensional problems (see Section 4) can use all the PEs simultaneously.

(4) The UNIVAC 1180 was used for comparison, because this was readily available. It would be of interest to have timings on a computer such as the Cray 1.

4. Numerical solution of a Three-Dimensional Free Boundary Problem

A three-dimensional extension of the dam problem of Fig. 3.1 was introduced by Stampacchia [24] (see also France [13]). Water seeps through a porous dam in a rectangular channel of width a and height H. The walls of the dam are vertical but the thickness of the dam is variable, so that the dam occupies the region

$$\Omega_3 = \Omega_2 \times (0, H), \tag{4.1}$$

where the horizontal cross section Ω_2 is of the form

$$\Omega_2 = \{(x, y): 0 < x < a, \varphi_1(x) < y < \varphi_2(x)\}. \tag{4.2}$$

In the specific problem considered here, Ω_2 is the L-shaped region

$$\Omega_2 = (0, ED) \times (0, FE) \cup [ED, AF) \times (0, AB), \tag{4.3}$$

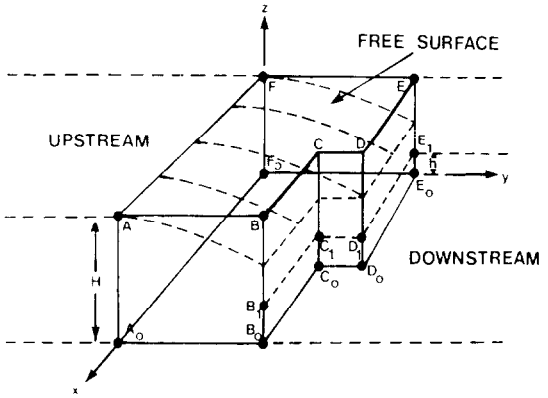

FIG. 4.1. Flow through a three-dimensional porous dam with L-shaped horizontal cross section.

where the points A, B, C, D, E, and F are as shown in Figure 4.1. The upstream water height is H and the downstream water height is h.

As shown by Stampacchia [24], the problem can be formulated as follows: Find u on the region Ω_3 such that:

$$-\nabla^2 u = -[u_{xx} + u_{yy} + u_{zz}] \geqslant -1 \quad \text{in} \quad \Omega_3, \tag{4.4a}$$

$$u \geqslant 0 \quad \text{in} \quad \Omega_3, \tag{4.4b}$$

$$u(-\nabla^2 u + 1) = 0 \quad \text{in} \quad \Omega_3; \tag{4.4c}$$

and

$$\begin{aligned} u = g &= \tfrac{1}{2}(H - z)^2, && \text{on the upstream face } AA_0F_0F, \\ &= \tfrac{1}{2}(h - z)^2, && \text{on the downstream face below water level } B_0C_0D_0E_0E_1D_1C_1B_1, \\ &= 0, && \text{on the downstream face above water level } B_1C_1D_1E_1EDCB, \\ &= 0, && \text{on the top } ABCDEF, \\ &= \alpha(x, y), && \text{on the bottom } A_0B_0C_0D_0E_0F_0; \end{aligned} \tag{4.5}$$

and

$$u_x \equiv u_n = 0, \qquad \text{on the sides } ABB_0A_0 \text{ and } EFF_0E_0. \tag{4.6}$$

Here $\alpha(x, y)$ is the solution of the two-dimensional mixed boundary value problem

$$\alpha_{xx} + \alpha_{yy} = 0, \qquad \text{on} \quad A_0B_0C_0D_0E_0F_0, \tag{4.7a}$$

$$\begin{aligned} \alpha &= \tfrac{1}{2}H^2, && \text{on} \quad A_0F_0, \\ &= \tfrac{1}{2}h^2, && \text{on} \quad B_0C_0D_0E_0, \end{aligned} \tag{4.7b}$$

$$\alpha_x \equiv \alpha_n = 0, \qquad \text{on} \quad A_0B_0 \cup E_0F_0. \tag{4.7c}$$

To solve problem (4.4)–(4.7) numericaly, we introduce a grid with $\Delta x = \Delta y = \Delta z$ and denote the approximation to $u([i-2]\Delta x, [j-1]\Delta y, [k-1]\Delta z)$ by w_{ijk}, and the approximation to $\alpha([i-2]\Delta x, [j-1]\Delta y)$ by $w_{ij} \equiv w_{ij1}$, for $2 \leqslant i \leqslant M-1$ and $1 \leqslant j \leqslant N$. As in Bruch [4], the computation proceeds in two stages.

Stage I. The two-dimensional problem (4.7) is approximated by replacing differential equation (4.7a) by the difference equations

$$4w_{ij} - w_{i+1,j} - w_{i-1,j} - w_{i,j+1} - w_{i,j-1} = 0. \tag{4.8}$$

The Dirichlet boundary conditions (4.7b) are satisfied by computing and storing the values of $w_{ij1} = \alpha_{ij}$ on A_0F_0 and $B_0C_0D_0E_0$. The Neumann conditions (4.7c) are satisfied by introducing two fictitious rows of gridpoints, adjacent to A_0B_0 and E_0F_0 respectively, and requiring that the values of w on a fictitious row should be equal to the values of w on the corresponding interior row; that is, $w_{1j} = w_{3j}$ and $w_{M,j} = w_{M-2,j}$, for $1 \leqslant j \leqslant N$.

The resulting system of equations is solved using a simple modification of the subroutine PROJSOR (see Table IV): the term $-GRID2$ is dropped from statement number 3; statement number 5 is deleted; and the statements

$$\begin{aligned} W(1,\ ) &= W(3,\ ), \\ W(M,\ ) &= W(M-2,\ ), \end{aligned} \tag{4.9}$$

are inserted between statements number 1 and 2, so as to make the values at the fictitious points equal to the corresponding interior values;

Stage II. The three-dimensional problem (4.4) is approximated by the LCP

$$\begin{aligned} 6w_{i,j,k} \geqslant\ & w_{i+1,j,k} + w_{i-1,j,k} + w_{i,j+1,k} + w_{i,j-1,k} \\ & + w_{i,j,k-1} + w_{i,j,k+1} - (\Delta x)^2, \end{aligned} \tag{4.10a}$$

$$w_{i,j,k} \geqslant 0, \tag{4.10b}$$

$$\begin{aligned} w_{i,j,k}[6w_{i,j,k} - w_{i+1,j,k} - w_{i-1,j,k} - w_{i,j+1,k} - w_{i,j-1,k} \\ - w_{i,j,k+1} - w_{i,j,k-1} + (\Delta x)^2] = 0. \end{aligned} \tag{4.10c}$$

The Dirichlet boundary conditions (4.5) are readily imposed, while the Neumann conditions (4.6) are treated by introducing fictitious sides parallel to the sides ABB_0A_0 and EFF_0E_0, and requiring that the values of w on the fictitious sides be equal to the values of w at the corresponding interior points.

To solve the LCP (4.10) we introduce a three-dimensional red-black partitioning of the gridpoints, so that each red (black) gridpoint has six black (red) neighbors. (It should be noted that the red/black ordering on any horizontal plane is the negation of the red/black orderings on the adjacent horizontal planes.) As in the two-dimensional problem treated in Section 3, each projected SOR iteration can be broken down into two stages: a red stage in which projected SOR is applied to all the

red points in the three-dimensional w array, followed by a similar black stage. In detail,

Red stage.

$$z_{ijk}^{(k,\text{red})} = w_{i+1,j,k}^{(k,\text{black})} + w_{i-1,j,k}^{(k,\text{black})} + w_{i,j+1,k}^{(k,\text{black})} + w_{i,j-1,k}^{(k,\text{black})} + w_{i,j,k+1}^{(k,\text{black})} + w_{i,j,k-1}^{(k,\text{black})} - (\Delta x)^2, \tag{4.11a}$$

$$w_{ijk}^{(k+1/2,\text{red})} = (\omega/6)\, z_{ijk}^{(k,\text{red})} + (1-\omega)\, w_{ijk}^{(k,\text{red})}, \tag{4.11b}$$

$$w_{ijk}^{(k+1,\text{red})} = \max\{0, w_{ijk}^{(k+1/2,\text{red})}\}. \tag{4.11c}$$

Black stage.

$$z_{ijk}^{(k,\text{black})} = w_{i+1,j,k}^{(k+1,\text{red})} + w_{i-1,j,k}^{(k+1,\text{red})} + w_{i,j+1,k}^{(k+1,\text{red})} + w_{i,j-1,k}^{(k+1,\text{red})} + w_{i,j,k+1}^{(k+1,\text{red})} + w_{i,j,k-1}^{(k+1,\text{red})} - (\Delta x)^2, \tag{4.12a}$$

$$w_{ijk}^{(k+1/2,\text{black})} = (\omega/6)\, z_{ijk}^{(k,\text{black})} + (1-\omega)\, w_{ijk}^{(k,\text{black})}, \tag{4.12b}$$

$$w_{ijk}^{(k+1,\text{black})} = \max\{0, w_{ijk}^{(k+1/2,\text{black})}\}. \tag{4.12c}$$

To implement the algorithm (4.11), (4.12) it was assumed that the dimensions of Ω_2 were such that the gridpoints on any horizontal cross section of the dam could be regarded as a subset of a 32×32 array. The solution w was stored as an array of matrices, the matrix $W(\ ,\ ,k)$ containing the values of w on the horizontal plane at a height $(k-1)\Delta z$. To control the parallel computation, two logical matrices were used: *MASKRB* which is true at interior red gridpoints in the current horizontal cross section and false otherwise; and *MASKMASK* which is true at interior points of Ω_2 and false otherwise.

The algorithm (4.11), (4.12) was implemented in two ways.

Implementation 1. During each red (black) stage the horizontal planes were updated in turn, and on each plane the red (black) points were updated in parallel.

The computation of $z^{(k)}$ requires five additions and one subtraction. Given an unlimited number of processors, n additions/subtractions require $\log_2 n$ steps, so that six additions/subtractions require at least three steps. By taking advantage of idle PEs, and remembering that, on the DAP, shift operations are much faster than arithmetic operations, the DAP-FORTRAN subroutine in Table VI is an efficient implementation of (4.11), (4.12) (compare Table IV). A full listing of the program is available upon request from the authors.

The subroutine in Table VI uses the functions SHS(outh) and SHN(orth) to shift W instead of the equivalent, but slower, statements (4.9).

Implementation 2. As in the three-dimensional magnetohydrodynamic code of Reddaway [22] we rearrange the values of w. Horizontal planes are considered in pairs, and the red points on each even-numbered plane are exchanged with the corresponding black points on the next odd-numbered plane. As a result, instead of

TABLE VI

First Implementation of (4.11) and (4.12)

```
C THE MAIN LOOP - PROCESS ALL THE Z PLANES
C
      SUBROUTINE MAIN LOOP
      COMMON/ISCA/TOPPLANE, M
      COMMON/ISCA/DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER TOPPLANE, M
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS
      REAL DAMEPSILON, BOTTOMEPSILIN, OMEGA, MAXDIFF
      COMMON/RSCA/DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON/RMAT/W( , , 25)
      COMMON/SUBLMAT/MASKRB( , ), MASKMASK( , )
      LOGICAL MASKRB, MASKMASK
      REAL SAVEW( , ), Z( , ), Z1( , )
      REAL MAXSOFAR, ALPHA, BETA, WIDGRID2, WIDTHGRID
      INTEGER NUMBTIMES
      LOGICAL TEMPMASK( , ), DONE, WSIGN( , )
      EQUIVALENCE(WSIGN, Z)
C WIDTH OF GRID (I.E. ONE UNIT SQUARE) IS SET TO 1.0
C
      NUMBITERS = 0
      WIDTHGRID = 1.0
      WIDGRID2 = WIDTHGRID * WIDTHGRID
C
C SAVE THE MASKRB FOR LATER RESTORATION
C
      TEMPMASK = MASKRB
C
C MAXDIFF IS THE MAXIMUM DIFFERENCE BETWEEN SAVEW
C AND W( , , K) AFTER W( , , K)
C HAS ITS RED (OR BLACK) VALUES CHANGE (FOR ALL K)
C
1     MAXDIFF = 0.0
      NUMBITERS = NUMBITERS + 1
      MASKRB = TEMPMASK
      DO 30 NUMBTIMES = 1, 2
C
C ITERATE FROM THE 2ND PLANE TO THE TOP PLANE
C
      DO 20 K = 2, TOPPLANE
      SAVEW = W( , , K)
C
C REVERSE RED/BLACK FOR SUCCESSIVE PLANES
C
      MASKRB(MASKMASK) = .NOT. MASKRB
```

Table continued

TABLE VI (*continued*)

```
C
C SUM THE SIX NEIGHBORS
      SAVEW(1, ) = SHN(SAVEW, 2)
      SAVEW(M, ) = SHS(SAVEW, 2)
      Z = SAVEW(-, -)
      Z1 = SAVEW
      Z1(MASKRB) = W( , , K + 1)
      Z(MASKRB) = W( , , K - 1)
      Z = Z + Z1
      Z1 = Z( , +)
      Z1(.NOT. MASKRB) = -WIDGRID2
      Z = Z + Z1
      Z = Z + Z(+, )
C
C STORE THE AVERAGE OF THE SIX NEIGHBORS IN W
C ONLY IN THE RED (OR BLACK) CELLS
C
      Z = ALPHA * Z + BETA * SAVEW
      Z(WSIGN) = 0.0
      W(MASKRB, K) = Z
C
C FIND THE MAXIMUM DIFFERENCE ON THIS PLANE
C
      MAXSOFAR = MAX(ABS(SAVEW - Z), MASKRB)
      IF (MAXSOFAR .GT. MAXDIFF) MAXDIFF = MAXSOFAR
20    CONTINUE
C
C REVERSE STATE OF ORIGINAL MASKRB FOR THE 2ND PASS
C THROUGH THE PLANES
C
      MASKRB(MASKMASK) = .NOT. TEMPMASK
30    CONTINUE
      DONE = (NUMBITERS .GT. DAMMAXITERS) .OR. (MAXDIFF .LE. DAMEPSILON)
      IF (.NOT. DONE) GOTO 1
      MASKRB = TEMPMASK
      RETURN
      END
C
C
```

having n planes, each containing red and black points in a checkerboard pattern, we have $n/2$ planes of red points interleaved with $n/2$ planes of black points. This makes it possible to use simultaneously all interior PEs for arithmetic.

The corresponding subroutine is given in Table VII. In the full program, to save time, the test for convergence was executed only every *TIMES* iterations, where *TIMES* is an input parameter.

The subroutine in Table VII assumes that there is an even number of planes. To

TABLE VII

Second Implementation of (4.11) and (4.12)

```
   THE MAIN LOOP - PROCESS ALL THE Z PLANES
        SUBROUTINE MAIN LOOP
        COMMON/ISCA/TOPPLANE, M
        COMMON/ISCA/DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
        INTEGER TOPPLANE, M
        INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, TIMES
        REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
        COMMON/RSCA/DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF, TIMES
        COMMON/RMAT/W( , , 25)
        COMMON/SUBLMAT/MASKRB( , ), MASKMASK( , )
        COMMON/WORK/Z, WK
        LOGICAL MASKRB, MASKMASK
        REAL SAVEW( , ), Z( , ), Z1( , ), WKP1( , ), WK( , ), MAXD( , )
        REAL MAXSOFAR, ALPHA, BETA, WIDGRID2, WIDTHGRID
        INTEGER NUMBTIMES
        LOGICAL TEMPMASK( , ), DONE, WSIGN( , ), TEST, NOTTEST
        EQUIVALENCE(WSIGN, Z), (Z, Z1), (WK, WKP1)
        ALPHA = OMEGA * 1.0/6.0
        BETA = 1.0 - OMEGA
C
C  WIDTH OF GRID (I.E. ONE UNIT SQUARE) IS SET TO 1.0
C
        NUMBITERS = 0
        WIDTHGRID = 1.0
        WIDGRID2 = WIDTHGRID * WIDTHGRID
C
C
1       TEST = TIMES .EQ. 1
        NOTTEST = .NOT. TEST
        ITIMES = TIMES
        MAXD = 0.0
C
C
C  ALTER ALL THE ODD NUMBERED PLANES:
2       KM2 = 1
        DO 20 K = 2, TOPPLANE, 2
        SAVEW = W( , , K + 1)
        WK = W( , , K)
        WK(1, ) = SHN(WK, 2)
        WK(M, ) = SHS(WK, 2)
        Z = WK + WK(-, -)
        Z = (Z(+, ) + Z( , +) + WK + MERGE(W( , , KM2), W( , , K + 2), MASKRB)
            - WIDGRID2) * ALPHA + SAVEW * BETA
        Z(WSIGN) = 0.0
        W(MASKMASK, K + 1) = Z
        IF (NOTTEST) GOTO 20
        Z1 = ABS(SAVEW - Z)
        MAXD(Z1 .GT. MAXD) = Z1                                  Table continued
```

TABLE VII (*continued*)

```
20      KM2 = K
C
C
C ALTER ALL THE EVEN NUMBERED PLANES:
        DO 21 K = 2, TOPPLANE, 2
        SAVEW = W( , , K)
        WKP1 = W( , , K + 1)
        WKP1(1, ) = SHN(WKP1, 2)
        WKP1(M, ) = SHS(WKP1, 2)
        Z = WKP1 + WKP1(–, –)
        Z = (Z(+, ) + Z( , +) + WKP1 + MERGE(W( , , K + 3), W( , , K – 1), MASKRB)
            – WIDGRID2) * ALPHA + SAVEW * BETA
        Z(WSIGN) = 0.0
        W(MASKMASK, K) = Z
        IF (NOTTEST) GOTO 21
        Z1 = ABS(SAVEW – Z)
        MAXD(Z1 .GT. MAXD) = Z1
21      CONTINUE
C
C
        ITIMES = ITIMES – 1
        IF (ITIMES .GT. 1) GOTO 2
        IF (ITIMES .EQ. 0) GOTO 3
        TEST = .TRUE.
        NOTTEST = .FALSE.
        GOTO 2
C
3       NUMBITERS = NUMBITERS + TIMES
        MAXDIFF = MAX(MAXD, MASKMASK)
        IF (MAXDIFF .LT. DAMEPSILON) GOTO 4
C
        IF (NUMBITERS .LT. DAMMAXITERS) GOTO 1
4       RETURN
        END
```

avoid additional testing, it is assumed that a copy of the top plane is stored above the top plane.

The two implementations were run on the problem with $H = 10$, $h = 0$, $AF = FE = 20$, $CD = BC = 10$, which was chosen because it had previously been solved by Bruch [4]. For comparison, the problem was also solved on the UNIVAC 1180 using single precision arithmetic and optimized FORTRAN code. The measured computation times per projected SOR iteration (including both red and black stages) were:

Implementation 1: 32 ms,
Implementation 2: 16.0–18.2 ms (dependent on frequency of convergence tests),
UNIVAC 1180: 34 ms,

so that Implementation 2 on the Pilot DAP is about two times faster than the UNIVAC 1180. The estimated time per SOR iteration (Implementation 2) was found as in Table V, and was found to lie between 15.4 and 17.4 ms, depending upon the frequency of convergence tests.

For this problem, only 383 (i.e., $21 \times 23 - 10 \times 10$) of the 1024 PEs were used.

5. Future Possibilities

(a) For purposes of comparison we have used previously published problems but they have dimensions which do not match the DAP array closely. In many practical problems, the resolution would be tailored to the DAP dimensions to achieve higher performance.

(b) The programs presented are readily extensible to larger problems on correspondingly larger DAPs such as the production 64×64; it is only necessary to change the boundaries. The time to process one plane would be unchanged.

(c) Performance on small three-dimensional problems can be improved by mapping several problem planes onto one DAP matrix.

(d) Problems with large *horizontal* dimensions can be mapped with each PE holding a small neighborhood group of points. Performance improves because each PE holds both black and red points (Hunt [18]).

(e) Very large problems cannot be held entirely within DAP store. For example, with four times as many points in a horizontal plane as there are PEs, the limit is about 26 planes with 4 K bits per PE or about 122 planes with 16 K bits per PE. With backing store, the transfers rates with N active problem planes in the DAP can be minimized by advancing each plane $(N-2)/2$ iterations per backing store fetch. Hence it should be possible to achieve a balance between input-output and processing times (Reddaway [22]).

(f) Problems of this type offer possibilities for using fixed point arithmetic (with suitable scaling) and using low precision for computing the iterative corrections. This is much faster than floating point work and performance improvements as large as a factor of ten are predicted without loss of accuracy in the final solution.

6. Conclusions

We have demonstrated that two- and three-dimensional linear complementarity problems can be solved on DAP with high performance and easy programming using a version of projected SOR. There is scope for even higher performance and for tackling a wide range of problem sizes.

APPENDIX: THE PILOT HOST-DAP INTERFACE

In the Pilot DAP system, the store of the DAP is not an integral part of the host's store as with the production DAPs. It is therefore necessary to explicitly move data between the host and DAP, and this is achieved by using standard host FORTRAN subroutines. The subroutine names begin with DAPTO or DAPFROM depending on whether they move data into or out of the DAP. The remaining letters of the name indicate the type (integer or real denoted by I or E) and rank (scalar, vector or matrix denoted by S, V, or M) of the variable transferred. Parameters of DAPTO and DAPFROM give the name of the host program variable and the location within the DAP in terms of the name of the common area and the offset from the start of this area.

Initiation of DAP processing is also less direct on the Pilot system with DAP-FORTRAN subroutines being called via the standard host FORTRAN subroutine DAPGO. A statement of the form:

CALL DAPGO('DAPSUB', *N*)

will suspend execution of the host FORTRAN and transfer control to the DAP-FORTRAN subroutine DAPSUB. Execution of the host FORTRAN is resumed after DAPSUB and any further levels of DAP-FORTRAN subroutines have been executed. The parameter *N* gives the maximum number of seconds allowed for DAP processing.

ACKNOWLEDGMENTS

C. W. Cryer and J. Stansbury gratefully acknowledge the provision by International Computers Ltd. of facilities for using the Pilot DAP at Stevenage, England.

REFERENCES

1. C. BAIOCCHI, *Ann. Mat. Pura Appl.* **92**(4) (1972), 107.
2. M. L. BALINSKI AND R. W. COTTLE, Ed., "Complementarity and Fixed Point Problems," North-Holland, Amsterdam, 1978.
3. A. BRANDT AND C. W. CRYER, "Multigrid algorithms for the solution of linear complementarity problems arising from free boundary problems," Technical Summary Report No. 2131, Mathematics Research Center, Univ. of Wisconsin–Madison, 1980.
4. J. C. BRUCH, JR., *Adv. Water Resour.* **3** (1980), 115.
5. R. W. COTTLE, F. GIANNESSI, AND J. L. LIONS, Ed., "Variational Inequalities and Complementarity Problems," Wiley, New York, 1980.
6. R. W. COTTLE, G. H. GOLUB, AND R. S. SACHER, *Appl. Math. Optim.* **4** (1978), 347.
7. C. W. CRYER, *SIAM J. Control Optim.* **9** (1971), 385.
8. C. W. CRYER, *in* "Proceedings, Seminar on Free Boundary Problems, Pavia, 1979" (E. Magenes, Ed.), Vol. 2, pp. 109–131, Instituto Nazionale di Alta Matematica Francesco Severi, Rome, 1980.
9. C. W. CRYER AND M. A. H. DEMPSTER, *SIAM J. Control Optim.* **18** (1980), 76.

10. G. DUVAUT AND J. L. LIONS, "Inequalities in Mechanics and Physics," Dunod, Paris, 1976.
12. P. M. FLANDERS, D. J. HUNT, S. F. REDDAWAY, AND D. PARKINSON, *in* "High Speed Computer and Algorithm Organization" (D. J. Kuck, Ed.), Academic Press, New York, 1977.
13. P. W. FRANCE, *J. of Hydrology* **21** (1974), 381.
14. R. GLOWINSKI, *Rendiconti di Mathematica* **14** (1971), Universita di Roma.
15. R. W. GOSTICK, ICL *Tech. J.* **1** (1979), 116.
16. D. HELLER, *SIAM Rev.* **20** (1978), 740.
17. D. J. HUNT, "Numerical Solution of Poisson's Equation on an Array Processor Using Iterative Techniques," Report No. CM21, International Computers Limited, Research and Advanced Development Centre, Stevenage, 1974.
18. D. J. HUNT, *in* "Supercomputers," Infotech International, London, 1979, 205.
19. ICL Technical Publication No. 6918, "DAP Fortran Language," 1979.
20. D. KINDERLEHRER AND G. STAMPACCHIA, "An Introduction to Variational Inequalities and their Applications," Academic Press, New York, 1980.
21. O. M. MANGASARIAN, *J. Optim. Theory Appl.* **22** (1977), 465.
22. S. F. REDDAWAY, "A 3D Magnetohydrodynamics Code (3DMHD) on DAP," Report No. CM59, International Computers Limited, Research and Advanced Development Centre, Stevenage, 1976.
23. S. F. REDDAWAY, *in* "Supercomputers," Infotech International, London, 1979, 309.
24. G. STAMPACCHIA, *Russian Math. Surveys* **29** (1974), 89.