NASA TECHNICAL MEMORANDUM

NASA TM X- 62,371

THE SOLUTION OF LARGE MULTI–DIMENSIONAL POISSON PROBLEMS

Harold S. Stone

Ames Research Center
Moffett Field, Calif. 94035
and
Digital Systems Laboratory
Departments of Electrical Engineering & Computer Science
Stanford University
Stanford, California

May 1974

SEL TR 85 NASA TMX-62,371

THE SOLUTION OF LARGE MULTI-DIMENSIONAL POISSON PROBLEMS

by

Harold S. Stone

Ames Research Center
Moffett Field, Calif. 94305

and

Digital Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California

May 1974

THE SOLUTION OF LARGE MULTI-DIMENSIONAL POISSON PROBLEMS

by Harold S. Stone

Abstract

The Buneman algorithm for solving Poisson problems can be adapted to solve large Poisson problems on computers with a rotating drum memory so that the computation is done with very little time lost due to rotational latency of the drum.

I. Introduction

Large computations that do not fit into central memory often are arranged so that the majority of the data resides on an auxiliary memory, and the computation itself is structured as a sequence of core-contained subproblems. Each subproblem is read from the auxiliary memory into central memory, hopefully while another computation proceeds, so that no time is lost waiting for the read operation. The major challenge in solving large problems is to structure the sequence of subproblems and the data storage format in auxiliary memory so as to minimize the time lost while waiting for data transfers.

In this paper we examine a method for solving large Poisson problems on a computer with a rotating drum memory, and we show that drum latency can be made very small with proper selection of parameters. Among the obvious candidate algorithms for solving Poisson problems with minimum latency are iterative schemes [Young, 1971] because these schemes access data records sequentially at uniformly spaced time intervals. In this paper we choose to ignore the iterative schemes in favor of direct methods that generally have a lower computational complexity. In particular, we examine Buneman's algorithm [Buneman, 1969] and the related algorithm known as cyclic odd-even reduction [Buzbee et al., 1970]. Both of these algorithms have interesting properties that lead to minimum latency implementations. For these algorithms the intervals between record accesses vary considerably from iteration to iteration, but the distance between records varies by an equivalent amount so that latency is held low.

Latency is not the only problem, however, in the design of such algorithms because latency can be held to very low values just be increasing buffer space. If buffer space is sufficiently large to hold the entire problem, then drum latency reduces to zero. Fortunately, the algorithms obtain very low latency with a small fixed amount of buffer storage.

In Section II of this paper we review the Buneman algorithm which is the basis of the algorithm for the solution of large problems. In Section III we discuss the storage structures for a minimum latency implementation of the Buneman algorithm for systems with drum memories. Section IV treats several peripheral matters such as practical aspects of implementation and a minimum latency implementation of the cyclic odd-even reduction algorithm. The final section contains a brief summary and some suggestions for further research.

II. The solution of core-contained problems

In this section, we present a brief review of the Buneman algorithm for solving Poisson's equation as described in Buzbee et al. [1970]. In later sections we show how this algorithm can be adapted to solve problems too large to be contained in main memory.

The problem at hand is the solution of Poisson's equation in two or three dimensions. To simplify the analysis we shall assume that the two-dimensional surface is a square of size $N \times N$, and the three-dimensional volume is a cube of size N on each side. The algorithm works best for N of the form $N = 2^m - 1$, which we assume to be the case for the remainder of the analysis. We also assume that the Poisson problem has Dirichlet boundary conditions along all boundaries. These assumptions are not necessary and can be relaxed as described by Buzbee et al. [1970] without changing our conclusions about methods for solving large problems.

Under the state boundary conditions the problem reduces to the solution of the system of equations $M \underset{\sim}{x} = \underset{\sim}{y}$ where M is block tridiagonal of the form:

$$M = \begin{bmatrix} A & I & & & & \\ I & A & I & & & \\ & I & A & I & & \\ & & \cdot & \cdot & & \\ & & \cdot & \cdot & I & \\ & & & I & A & \end{bmatrix} \qquad (1)$$

For two-dimensional problems, when the square grid is of size $N \times N$, M has dimension $N^2 \times N^2$, I is an identity matrix of size N, and A is an $N \times N$ tridiagonal matrix. For three-dimensional problems, the cubic grid has N points in each dimension, and M has dimension $N^3 \times N^3$. In this case the identity matrix I has size $N^2 \times N^2$, the A matrix is block tridiagonal of size $N^2 \times N^2$ and is the matrix for a two-dimensional system of the form just described. Thus a three-dimensional Poisson problem contains N coupled two-dimensional problems, and similarly, a two-dimensional problem contains N coupled one-dimensional problems. This obviously generalizes to higher dimensions, but dimensionality greater than three is rarely encountered in practice.

The method of solution involves computations that decouple the problems of lower dimension. Buneman's algorithm is a variation of an algorithm known as cyclic odd-even reduction. In this type of algorithm, half of the lower dimensional problems are eliminated during the first iteration, and during each successive iteration half of the remaining problems are eliminated until a single system of lower dimension remains. This is olved, and its solution is used to solve the two lower dimensional problems last eliminated. The available solutions are then substituted into four lower dimensional problems, then these into eight lower dimensional problems, etc., until all of the eliminated lower dimensional problems have been solved. The order in which the substitutions are made is the reverse of the order in which problems are eliminated. Under the stated conditions, namely that the M matrix is symmetric with all block factors constant on each diagonal, each matrix of lower dimension eliminated

by the algorithm is not a Poisson system itself, but is the product of Poisson matrices. At the k^{th} iteration, the matrices are each products of 2^k Poisson matrices, so that during back substitution, each system eliminated during the k^{th} iteration can be solved by solving a sequence of 2^k Poisson problems of the same dimension as the eliminated matrices. To make these ideas explicit the computation in brief is given below.

To solve $M \underset{\sim}{x} = \underset{\sim}{y}$ when M has the form of (1), partition $\underset{\sim}{x}$ and $\underset{\sim}{y}$ to conform to M, so that

$$\underset{\sim}{x} = \begin{bmatrix} \underset{\sim}{x}_1 \\ \underset{\sim}{x}_2 \\ \cdot \\ \cdot \\ \cdot \\ \underset{\sim}{x}_N \end{bmatrix} \qquad \underset{\sim}{y} = \begin{bmatrix} \underset{\sim}{y}_1 \\ \underset{\sim}{y}_2 \\ \cdot \\ \cdot \\ \cdot \\ \underset{\sim}{y}_N \end{bmatrix}$$

where each $\underset{\sim}{x}_i$ and $\underset{\sim}{y}_i$ is a vector with $1/N^{th}$ as many components as $\underset{\sim}{x}$ and $\underset{\sim}{y}$. For two-dimensional problems each $\underset{\sim}{x}_i$ and $\underset{\sim}{y}_i$ is a vector with N components, and for three-dimensional problems, each has N^2 components.

Next we compute a sequence of vectors $\underset{\sim}{p}^{(k)}$, $\underset{\sim}{q}^{(k)}$, $\underset{\sim}{y}^{(k)}$ and matrices $A^{(k)}$. The vectors are of dimension equal to the dimension of $\underset{\sim}{x}$ and $\underset{\sim}{y}$, and are partitioned to conform to the partitioning of $\underset{\sim}{x}$ and $\underset{\sim}{y}$. To begin the iteration, initialize the quantities as indicated below for $j = 2, 4, 6, \ldots, 2^m-2$.

$$
\begin{aligned}
A^{(1)} &= 2I - A^2 \\
\underset{\sim}{p}_j^{(1)} &= A^{-1}\underset{\sim}{y}_j \\
\underset{\sim}{q}_j^{(1)} &= \underset{\sim}{y}_{j-1} + \underset{\sim}{y}_{j+1} - 2\,\underset{\sim}{p}_j^{(1)}
\end{aligned}
\tag{2}
$$

The identity matrix in the first equation of (2) has the same dimension as the A matrix. The calculation of $\underset{\sim}{p}_j^{(1)}$ requires the solution of the equation $A\,\underset{\sim}{p}_j^{(1)} = \underset{\sim}{y}_j$, which is a Poisson system of one dimension lower than the original system.

The iteration to perform the reduction of the system is the following:

$$
\begin{aligned}
A^{(k+1)} &= 2I - [A^{(k)}]^2 \\
\underset{\sim}{p}_j^{(k+1)} &= \underset{\sim}{p}_j^{(k)} - [A^{(k)}]^{-1}(\underset{\sim}{p}_{j-2^k}^{(k)} + \underset{\sim}{p}_{j+2^k}^{(k)} - \underset{\sim}{q}_j^{(k)}) \\
\underset{\sim}{q}_j^{(k+1)} &= \underset{\sim}{q}_{j-2^k}^{(k)} + \underset{\sim}{q}_{j+2^k}^{(k)} - 2\underset{\sim}{p}_j^{(k+1)}
\end{aligned}
\tag{3}
$$

where during the k^{th} iteration the indices j have the form $j = i2^{k+1}$, $1 \le i \le 2^{m-k} - 1$.

The matrix $A^{(k)}$ in (2) is not block tridiagonal, but it factors into block tridiagonal matrices according to the identity:

$$
A^{(k)} = -\prod_{j=1}^{2^k} (A + 2\,I\cos\theta_j^{(k)}) \tag{4}
$$

where

$$
\theta_j^{(k)} = (2j-1)\pi/2^{k+1}
$$

and A and I are the block matrices of M given in (1). Thus we can obtain $\underset{\sim}{p}_j^{(k+1)}$ in the k^{th} iteration of (3) by solving a sequence of 2^k Poisson problems of lower dimension.

Since we assume N has the form 2^m-1, after m-2 iterations of (3) the reduction allows us to write the single equation

$$A^{(m-1)}\underset{\sim}{x}_{2^{m-1}} = A^{(m-1)}\underset{\sim}{p}_{2^{m-1}}^{(m-1)} + \underset{\sim}{q}_{2^{m-1}}^{(m-1)} \qquad (5)$$

which can be solved for $\underset{\sim}{x}_{2^{m-1}}$ by using the factorization given in (4). At this point we can proceed with the back substitution using the iteration:

$$A^{(k)}\,(\underset{\sim}{x}_j - \underset{\sim}{p}_j^{(k)}) = \underset{\sim}{q}_j^{(k)} - (\underset{\sim}{x}_{j-2^k} + \underset{\sim}{x}_{j+2^k}) \qquad (6)$$

for $j = i\cdot 2^k$, and i odd in the interval $1 \le i \le 2^{m-k} - 1$. We solve (6) for $(\underset{\sim}{x}_j - \underset{\sim}{p}_j^{(k)})$, then use

$$\underset{\sim}{x}_j = \underset{\sim}{p}_j^{(k)} + (\underset{\sim}{x}_j - \underset{\sim}{p}_j^{(k)}) \qquad (7)$$

to solve for $\underset{\sim}{x}_j$. Boundary conditions force $\underset{\sim}{x}_0$ and $\underset{\sim}{x}_{2^m}$ to be zero in (6).

This concludes the general description of the algorithm. The data flow of the algorithm is the aspect that concerns us most in this paper since the challenging problem is to support the data flow when using a rotating auxiliary memory. In the next section we investigate various ways to carry out the computation described in (3) and (6) when the problem must reside on an auxiliary memory.

III. The solution of large Poisson problems

In this section we investigate a method for organizing a computation to solve Poisson's problem when the problem is too large for central memory. We assume that the data resides on an auxiliary memory such as a drum or fixed-head disk, so that the rotational position of the memory is the unique state variable that describes the state of the memory. Although the entire computation is too large for central memory, the central memory is at least large enough to contain several problems of lower dimension. For example, to solve a three-dimensional problem with a mesh size of 64 points in each dimension requires sufficient storage for 2^{18} mesh points. This is far too large to be contained in central memory for all but a very few computers. However, one two-dimensional subproblem contains only $2^{12} = 4096$ points, and can easily fit in central memory of typical scientific computers. In fact, eight to 16 subproblems might be able to reside simultaneously in a typical scientific computer memory. Large two-dimensional problems that require the techniques discussed in this section have 256 to 512 or more points in each dimension, or approximately 2^{16} to 2^{18} or more mesh points in total.

The reason that the direct method for solving Poisson's equation is somewhat challenging to implement on a computer with a rotating memory is that each point in the solution vector of the problem is influenced by every point of the right-hand side of (1). Thus, any implementation of any algorithm whatsoever requires data flow to go from every point to every other point, and this is contrary to the natural unidirectional flow of a drum memory. The strategy we choose is to use the unidirectional flow of infor-

mation of the drum for iterations (2) and (3) to collapse the data into a single problem of lower dimension. The back substitution calls for a reverse flow of information, which is impossible for a rotating disk. Here we make use of the natural periodicity of the drum to spread information as required for the back substitution. Latency cannot be made zero for the back substitution process as we describe it, but it can be kept to a small amount.

The method we use to organize data is shown in Fig. 1. Each record contains $\underset{\sim}{p}_j$ and $\underset{\sim}{q}_j$ for fixed j. The records are initialized with the value of $\underset{\sim}{y}_j$ in the j^{th} record, and subsequent copies of the j^{th} record contain $\underset{\sim}{p}_j$ and $\underset{\sim}{q}_j$ as they are produced during the evaluation of (2) and (3). When $\underset{\sim}{x}_j$ is calculated during the back substitution phase, it overwrites $\underset{\sim}{p}_j$ in one or more copies of the j^{th} record according to a scheme described later in this section.

Since the matrix $A^{(k)}$ is independent of j in (2) and (3), it is not necessary to store a copy of $A^{(k)}$ with each record, nor is it necessary to update $A^{(k)}$ for each j as indicated in (2) and (3). However, if boundary conditions are Neumann or periodic, and different faces of the boundary have different types of conditions, then the $A^{(k)}$ matrices may be dependent on j, and we must allow for the possibility of storing and recomputing $A^{(k)}$ for each j. The algorithm is valid for these boundary conditions with slight alterations in the factorization given in (4) and with slight changes to other minor details in the calculation.

The important aspect of the data organization given in Fig. 1

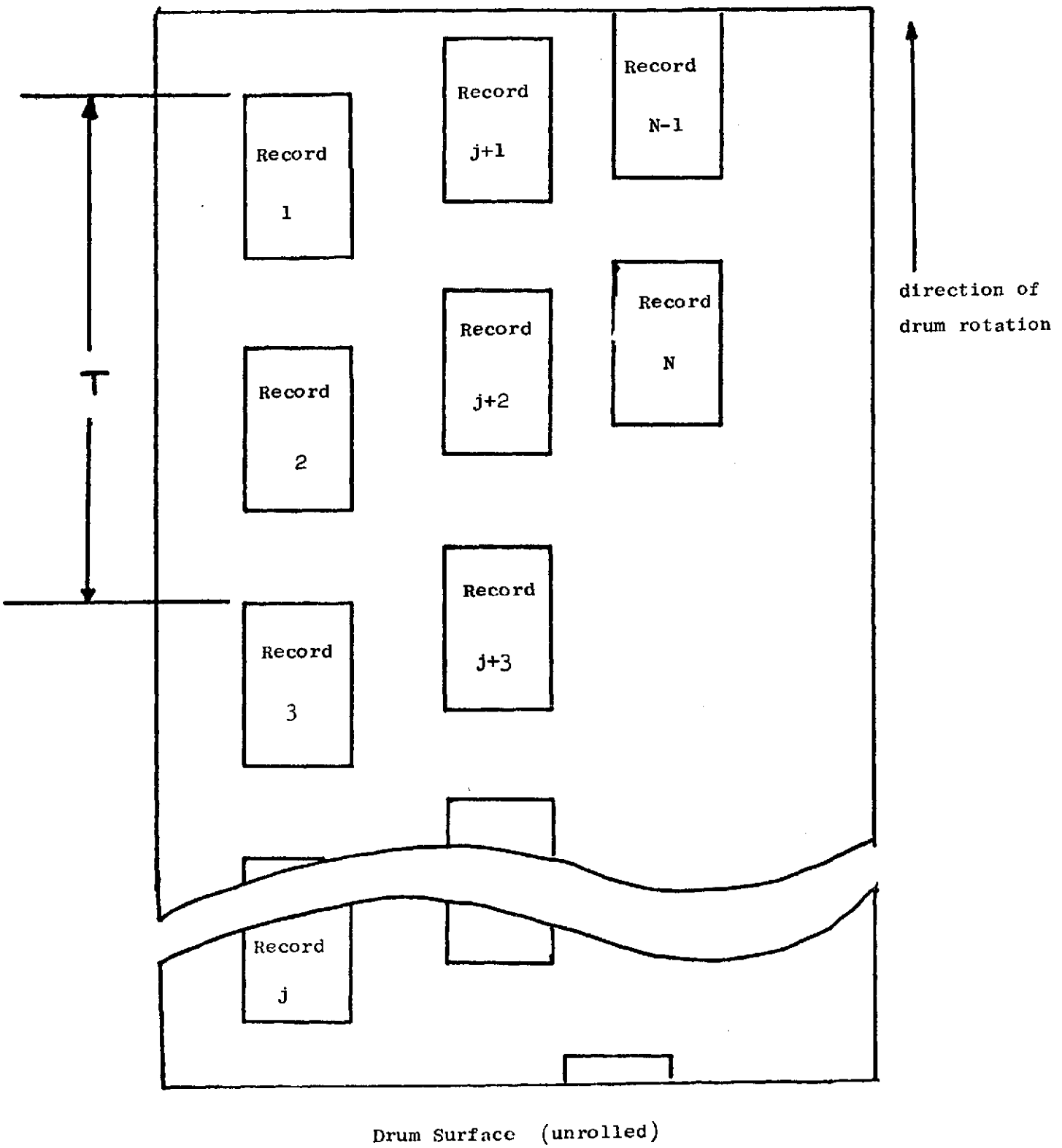

Fig. 1. Data Structure for Poisson solver. The interval T is equal to the drum rotation during the solution one Poisson system of lower-dimension.

is that the records are spaced around the drum so that the physical distance between records j and j+2 is just long enough to permit the computation in (2) to take place while the drum is rotating. As mentioned previously, the computation excludes the computation of $A^{(1)}$ if this matrix is the same for all j, as is the case for the stated boundary conditions. Fig. 2 shows the timing for the reading and writing of records during the computation of (2). Note that the first three records are read into memory and reside there while $\underset{\sim}{p}_2^{(1)}$ and $\underset{\sim}{q}_2^{(1)}$ are computed. During this period, the next two records are read into memory. At the close of the computation of $\underset{\sim}{p}_2^{(1)}$ and $\underset{\sim}{q}_2^{(1)}$, the data required to compute $\underset{\sim}{p}_4^{(1)}$ and $\underset{\sim}{q}_4^{(1)}$ are available in memory. These quantities are computed while the next two records are read into memory, and the record containing $\underset{\sim}{p}_2^{(1)}$ and $\underset{\sim}{q}_2^{(1)}$ is written back onto the drum. If simultaneous reading and writing of the drum is not permitted, then a scheme such as that shown in Fig. 3 is required. In this figure the initial configuration of records is such that two records are grouped together, followed by a space to allow the writing of a third record, and the pattern repeats around the drum. The distance between successive records in an adjacent pair is equal to the drum travel during one third of the calculation of (2) for one value of j, so that one pattern of two records and a blank record position passes under the read head during one iteration of (2). The timing in Fig. 3 shows that when the calculation of (2) has ended for one value of j, new data are available for the repetition of (2) for the next value of j. The output record is written during the blank position time between pairs of input records. Fig. 3 is essentially the same as Fig. 2 in all other respects.

Drum Position	Iteration 1			Iteration 2			Iteration 3		
	Read	Compute	Write	Read	Compute	Write	Read	Compute	Write
1	1	-	-	-	-	-	-	-	-
2	2	-	-	-	-	-	-	-	-
3	3	-	-	-	-	-	-	-	-
4	4	2	-	-	-	-	-	-	-
5	5	2	-	-	-	-	-	-	-
6	6	4	2	2	-	-	-	-	-
7	7	4	-	-	-	-	-	-	-
8	8	6	4	4	-	-	-	-	-
9	9	6	-	-	-	-	-	-	-
10	10	8	6	6	-	-	-	-	-
11	11	8	-	-	4	-	-	-	-
12	12	10	8	8	4	-	-	-	-
13	13	10	-	-	4	-	-	-	-
14	14	12	10	10	4	-	-	-	-
15	15	12	-	-	8	4	4	-	-
16	16	14	12	12	8	-	-	-	-
17	17	14	-	-	8	-	-	-	-
18	18	16	14	14	8	-	-	-	-
19	19	16	-	-	12	8	8	-	-
20	20	18	16	16	12	-	-	-	-
21	21	18	-	-	12	-	-	-	-
22	22	20	18	18	12	-	-	-	-
23	23	20	-	-	16	12	12	-	-
24	24	22	20	20	16	-	-	8	-
25	25	22	-	-	16	-	-	8	-
26	26	24	22	22	16	-	-	8	-
27	27	24	-	-	20	16	16	8	-
28	28	26	24	24	20	-	-	8	-
29	29	26	-	-	20	-	-	8	-
30	30	28	26	26	20	-	-	8	-
31	31	28	-	-	24	20	20	8	-
32	32	30	28	28	24	-	-	16	8
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 2. Timing for drum reads and writes when simultaneous reading and writing is allowed. Indices shown in figure give the value of j for each record.

Drum Position	Read	Compute	Write	Read	Write	Compute
1	1	-	-	-	-	-
2	-	-	-	-	-	-
3	2	-	-	-	-	-
4	3	-	-	-	-	-
5	-	2	-	-	-	-
6	4	2	-	-	-	-
7	5	2	-	-	-	-
8	-	4	2	2	-	-
9	6	4	-	-	-	-
10	7	4	-	-	-	-
11	-	6	4	4	-	-
12	8	6	-	-	-	-
13	9	6	-	-	-	-
14	-	8	6	6	-	-
15	10	8	-	-	4	-
16	11	8	-	-	4	-
17	-	10	8	8	4	-
18	12	10	-	-	4	-
19	13	10	-	-	4	-
20	-	12	10	10	4	-
21	14	12	-	-	8	4
22	15	12	-	-	8	-
23	-	14	12	12	8	-
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.

Fig. 3. Timing for drum reads and writes when simultaneous reading and writing is not allowed.

Idle computer time at the beginning of iterations in both figures is not lost. It can be used to complete computations of the prior iteration.

The interesting aspect of this data organization concerns what happens when (3) is evaluated. Note that the input data to (3) is the output data from (2) or from the previous iteration of (3). After reading the first three records to initiate the evaluation of (3) for the first value of j, the output data for each successive value of j requires two input records to be read. During each successive scan of the input data for an iteration of (3), the amount of data read reduces by a factor of 2, and the spacing between records read increases by a factor of 2. Thus it takes approximately twice as long to read two input records for (3) for k=2 as it takes to read two input records to evaluate (2). However, the time required to compute $\underset{\sim}{p}_j^{(2)}$ and $\underset{\sim}{q}_j^{(2)}$ is roughly double the time required to compute $\underset{\sim}{p}_1^{(1)}$ and $\underset{\sim}{q}_1^{(1)}$ because it is dominated by the time required to solve two problems of lower dimension, while the first iteration is dominated by the time required to solve a single system of lower dimension. This follows because A is block tridiagonal, while $A^{(1)}$ factors into two block tridiagonal systems, as given in (4).

The timing for two iterations of (3) is shown in Fig. 2. Note that for each successive iteration the space between input records doubles, but the computation time doubles as well. Thus if drum latency is negligible for (2), it is also negligible for every iteration of (3).

This completes the description of the reduction phase of the algorithm. During each iteration of this phase, drum latency can be made essentially zero. Between iterations latency depends on how closely the data comes to occupying an integral number of drum bands. If the data spans an integral

number of bands, then drum latency between iterations is also essentially zero. Note that a new iteration can begin at the first available value of j, and not necessarily at the least value of j since (3) can be done for j in any order. If data spans an integral number of bands except for a fraction α of a single band, then the latency during the reduction phase is approximately $\alpha[(\log_2 N) - 1]$ since there are $(\log_2 N) - 1$ transitions between iterations.

The reduction phase is relatively straightforward because there are no constraints on the placement of output data. We simply choose to write out each record as it is produced. The back substitution phase is more difficult to implement because the position of both the input and output data of each computation are fixed by the placement of data during the reduction phase. In particular, to solve for the odd unknowns we must substitute the computed values of the even unknowns into their respective positions between the odd records. Moreover, the data from which the even unknowns are computed are displaced in the direction of rotation from the final position for the even unknowns. During the back substitution phase the data flow must be in the reverse direction. Consequently, we have to move information against the natural direction of information flow of the drum.

Since mechanical drums cannot be rotated in one direction and then in the opposite direction, we must move the information backwards by moving it forward around the drum for almost a full revolution. A data file can be moved forward around the surface of a rotating drum provided there is sufficient buffer capacity in memory to hold portions of the file while the drum rotates. For the reduction phase we need only two buffers for data

being read, three buffers for the computation, and one buffer for data being written, giving a total of six buffers. We surely do not wish to increase the number of buffers to much more than this for back substitution. Instead we determine how records should be spaced around a band to take advantage of the periodicity of the drum and avoid extra buffers in core. We now show that record spacing should be chosen to be k/11 of a drum revolution where k is an integer in the range $1 \leq k \leq 11$.

Fig. 4 shows the spacing of records around the drum for the first reduction iteration and for the penultimate iteration of the back substitution. For the back substitution, records in which known values of $\underset{\sim}{x}_j$ have been stored are shown with underlines. Note that every other record contains a known value, and the unknowns that are computed for the particular iteration are interspersed between the unknowns. In doing the back substitution iteration as shown, the output records are even numbered and must be stored in the positions occupied by the even records of the first iteration. These are the records shown as the input records of the reduction phase shown in the figure. When the even-numbered records are stored as shown, the data for the last iteration are in position for a back-substitution computation with near-zero latency. By assumption, the known values of $\underset{\sim}{x}_j$ that appear as input to the back substitution iteration have been stored in their correct position during the previous back substitution iteration.

It is evident in Fig. 4 that during the reduction phase output records are displaced by four record positions from their corresponding input position, and during the back substitution the displacement is seven record

Read	Compute	Write	Read	Compute	Write
1	-	-			
2	-	-			
3	-	-			
4	2	-			
5	2	-	-	-	-
6	4	2	2	-	-
7	4	-	-	-	-
8	6	4	4*	-	-
9	6	-	-	2	-
10	8	6	6	2	-
11	8	-	-	2	-
12	10	8	8*	2	-
13	10	-	-	6	2*
14	12	10	10	6	-
15	12	-	-	6	4*
16	14	12	12*	6	-
17	14	-	-	10	6*
18	16	14	14	10	-
19	16	-	-	10	8*
20	18	16	16*	10	-
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

Fig. 4. Record positions for first reduction iteration and next to last back substitution. Asterisks indicate records containing known solutions.

positions. The total displacement is 11 records, and if this is equal to one drum revolution, the output of the back substitution iteration falls on the exact drum position for the final iteration to be done with minimum latency. Note that the number of buffers required for the back substitution phase is six, composed of two buffers for reading data, three for computation, and one for writing data. Also note that known $\underset{\sim}{x}_j$'s input to a reduction iteration are rewritten as output values as well as the $\underset{\sim}{x}_j$'s computed during the iteration.

Fig. 4 shows that the first reduction iteration, and the last two back substitutions can be done with zero latency if exactly 11/k records fit around the surface of one band for $1 \leq k \leq 11$. There remains to show that the latency for the other iterations is zero or near zero. This latency can be calculated easily by noting in Fig. 4 the several sources of the 11 record delay. During the reduction phase, after reading record 2, one record time is spent waiting for record 3, two record times for computation, and one record time for writing results. During the next iteration of the reduction, two record times are spent waiting for the last input operand, four for computation, and one for output. More generally during the k^{th} iteration of the reduction phase we have the following formula for the displacement of the output file relative to the input file:

$$D_R(k) = 2^{k-1} + 2^k + 1$$

where $D_R(k)$ is the displacement for iteration k of the reduction phase. The first term is the number of records spent waiting for the third operand after the second is read, the next term is the computation time,

and the third term is the time spent writing results. During the back substitution phase the displacement for reading the k^{th} iteration output to produce input for iteration k-1 is given by:

$$D_B(k) = 2^k + 2^{k+1} + 1.$$

The terms correspond to the terms in $D_R(k)$, and the first terms in $D_B(k)$ are double the values of the terms in $D_R(k)$. The total displacement is $D_R(k) + D_B(k) = 9 \cdot 2^{k-1} + 2$. When k = 1, the total displacement is 11, as indicated in Fig. 4. For k = 2, the displacement is 20, which is equal to -2 modulo 11. Thus when one band holds eleven records, the data for the second iteration is displaced only nine records around the drum during the reduction phase and back substitution processing, and this is not an entire drum revolution. A delay of two record times must be introduced into the computation somewhere, probably during the back substitution phase of the computation. Thus instead of writing the computed values of $\underset{\sim}{x}_j$ as soon as they are computed in the back substitution, they should be buffered and delayed two additional record times. Since output records are produced and written every four record times during this iteration, only one additional write buffer is necessary to achieve the necessary displacement

For k = 3, the total displacement is 36 = -6 mod 11. Here we must displace records by six record times, but output is written every eight record times during the back substitution for this iteration, so that only one buffer more than the original complement of six is required to achieve the required displacement of the output records. But this is the same requirement as for k = 2, so no extra buffers are required. For all back substitution iterations the output records are written at intervals greater than 11 and displacements need never exceed 11 so that a single extra write

buffer suffices for all iterations.

We now have two constraints on the arrangement of records on the drum. These are:

1. One record time should equal a half of the time required to solve a core-contained problem of lower dimension.

2. One record time should equal $k/11^{th}$ of a revolution time, for k an integer in the interval $1 \leq k \leq 11$.

To satisfy both constraints simultaneously, we suggest that the record displacement be selected according to the second constraint, which is problem independent, and that the problem size be designed to satisfy the first constraint. It is the usual case that the number of mesh points in each dimension may be selected with some flexibility provided that a sufficient number of points exist to give the desired accuracy. We suggest that if for a particular situation the solution of a core-contained problem does not take sufficient time to satisfy the first constraint, then the number of mesh points can be increased to lengthen this computation, and the total time to solve the entire problem does not increase. Increasing the number of mesh points merely decreases drum latency.

In the next section we look at specific examples to estimate the efficiency of this implementation for various problem sizes.

IV. Analysis of effectiveness of the algorithm

In this section we give figures for the drum latency anticipated as a function of problem size, and estimate the efficiency of the algorithm for various realistic sets of parameters. We also discuss the implication of electronic "drums", such as circulating memories using magnetic bubbles or charge-coupled diodes that may be available in several years.

First, to calculate the total drum latency for a computation, we note there are two sources of latency:

1. Latency from iteration to iteration because the data does not occupy an integral number of bands.
2. Latency during back substitution arising from the need to rewrite data in specific locations.

Each of these sources of latency can be identified with each iteration, and the total latency for any iteration cannot exceed ten records, or one record less than a full drum revolution. As a rough approximate we can estimate latency to be 5.5 record times, or one half a revolution per iteration. The number of iterations required is $2[\log_2 N) - 1]$ so a rough estimate of the latency in one calculation is $11[(\log_2 N) - 1]$ record times. The total time spent in a calculation with no latency is approximately N record times per iteration for $2(\log_2 N - 1)$ iterations, giving roughly $2N[(\log_2 N) - 1]$ record times. Then the fraction of additional time contributed by latency is:

$$\frac{\text{latent time}}{\text{active time}} \cong \frac{11}{2N}$$

This time becomes quite small as N increases.

Table I shows an exact calculation of the latency for several values of N. For this calculation the computation is assumed to begin as the first record passes the read head and terminates when the last record is written as output. During each record the computer is assumed either to be idle or computing, so that each record time contributes to latency or to active computing. In analyzing Table I, consider how large the problems are when the problem is three-dimensional. The problem of size 31 can be done comfortably by most large scientific computers, and is a useful size to attack. The problem of size 63 strains the capacity of all but the largest of presently available scientific computers. Note that the per cent latency is extremely low for this size of problem so that it appears to be quite feasible to use the scheme described here when such problems are attempted. Problems of size 127 are within reach of just a few super computers such as ILLIAC IV, and the very low loss of time due to latency makes this scheme quite attractive for such problems.

The major constraint of rotating memories in this drum allocation scheme concerns the need to have 11/k records per revolution in order to reduce latency during back substitution. Recall that data must be transmitted in the reverse direction of drum rotation, and we use the natural periodicity of the drum to accomplish this. If the number of records per revolution exceeds 11, then computation is degraded in two different ways. Both the latency and the buffer space increase as the number of records

Table I

N	Latent Records	Active Records	% Latent
15	19	98	19.4
31	40	258	15.5
63	40	642	6.23
127	55	1538	3.58

Drum latency for various problem sizes.

per revolution increases. There may be factors that dictate that record spacing be smaller than $1/11^{th}$ of a drum revolution, and this appears to be reasonable if the degradation due to latency and extra buffering is acceptable. It is unlikely that there is sufficient justification to exceed 11 records per revolution by a substantial amount.

Since typical drums rotate once every 10 to 40 msec., the record spacing for the solution of a core-contained two dimensional problem is $2/11^{ths}$ of this time, which is from 1.82 to 7.27 msec. Hockney [1970] reports times of 56 msec. and 196 msec. for solving a two-dimensional problem of size 32×32 and 64×64 respectively, on a CDC 6600. Modern computers such as ILLIAC IV have achieved speed increases from 10 to 30 times over the speed of a CDC 6600 so that Hockney's problem may be done in roughly 2 to 10 msec. on such computers. From these crude estimates we see that a problem of size 31 is likely to require at least $2/11^{ths}$ of a drum revolution and a problem of size 63 to require more than this. Consequently, the periodicity of 11 records per revolution is compatible with expected parameters for computer speed and drum revolution time when the core-contained problem is two-dimensional. In fact, it may be necessary to place 11/2 or 11/3 records around a drum band.

Present projections indicate that electronic memories may replace drum memories in future computers. Such electronic memories are likely to be circulating memories based upon a magnetic bubble or charge-coupled diode technology. Like drums, accesses for these electronic memories experience a latency while information is rotated into position. However,

latency is much less than for present drums, possibly of the order of 10 to 100 times less than the latency of mechanical drums. The limiting latency factor occurs when one record occupies a single band, at which points true zero latency is achieved because all iterations take an integral number of revolutions of the drum.

Electronic drums have an advantage not shared with mechanical drums. They can be reversed instantaneously and read in the opposite direction, if so designed. This feature can be used to great advantage with the algorithm cited here because the ideal way to perform the back substitution is to reverse the direction of rotation of the memory. For the back substitution all information flow should be in the direction opposite to the flow of information during the forward reduction. Moreover the spacing between records is correct to achieve minimum latency during the back substitution provided that output records produced during the forward iteration are delayed through buffering by $D_B(k) - D_R(k) = 3 \cdot 2^{k-1}$ record positions during the k^{th} iteration. The delay is achievable with the addition of single write buffer.

Before closing this section we should mention other algorithms that are susceptible to this type of data structuring for minimum latency operation. Cyclic odd-even reduction [Buzbee et al., 1970] is similar to Buneman's algorithm except that the forward reduction involves matrix multiplication rather than the solution of matrix equations. Like Buneman's algorithm, the computation time per iteration doubles with each iteration of the reduction process, and the number of input records decreases by roughly half, with the spacing between them doubling. The basic cycle time

for this iteration depends on the time required to multiply a block tridiagonal matrix by another matrix. However the basic cycle time for the back substitution is the time required to solve a block tridiagonal system in memory, which is the same basic cycle time as Buneman's algorithm. Because of the constraints of the drum memory, the reduction phase and back substitution phase must use the same basic cycle time, and the cycle time must therefore be the maximum of the matrix multiplication and matrix solution times. Therefore, we cannot take advantage of the faster time of matrix multiplication if we use cyclic odd-even reduction.

One advantage that does accrue to cyclic-odd even reduction is the fact that some computations can be avoided if a single equation is solved repeatedly for different right-hand sides. The drum algorithm can take advantage of this savings only in so far as the savings are realized in both the reduction and back substitution phases. Any savings realized in one phase and not the other is lost because of the constraint that records spacing produced during the reduction phase must be identical to the record spacing for back substitution.

V. Summary and comments

We have taken a highly regular Poisson problem and have shown how it can be solved on a computer with large rotating disk memory. The regularity of the problem is reflected in the regularity of data storage on the disk, and the result is the ability to solve the problem with very little time lost to drum latency. The regularity of the data structure is somewhat unexpected because computations change from iteration to iteration. The major issue that is settled here is that large Poisson problems can be solved effectively with a small high-speed memory and a large rotating drum memory. The algorithms used are efficient in that if all of memory were high-speed memory then the computation speed for this algorithm would be reasonably close to the speed of the best known algorithm for solving Poisson's equation, say to within a small constant factor independent of the size of the problem.

When we change from rectangular boundaries to something less structured, the present algorithm is not sufficient in itself to provide a solution. For such problems we have some doubt that a high-speed drum can be used effectively so as to make latency negligible. Periodic or other regular data structures are absolutely essential when large drum memories are used as a tightly coupled auxiliary memory, and such structures are present only when Poisson's equations are solved over highly regular regions.

In closing we should mention that we can predict the effectiveness of a minimum latency storage organization by comparing the computation time for the problem as we have described it to the computation time for

a problem in which random access is made to records on the drum, with a corresponding latency experienced for each record read. In the minimum-latency case each record requires one record time to read, and essentially zero latency time to access. In the random-access mode, each record on the average requires one record time to read and one half a drum revolution, or 5.5 records of latency for accessing the record. Thus the computation time for the random access problem is likely to be $5.5 + 1 = 6.5$ times as long as for the minimum latency code. For large problems the factor of 6.5 represents a very tangible savings in computation time.

References

Buneman, O., 1969. "A compact non-iterative Poisson solver," Report 294, Inst. for Plasam Res., Stanford University, Stanford, Calif., 1969.

Buzbee, B. L., Golub, G. H., and Nielson, C. W., 1970. "On direct methods for solving Poisson's equations," SIAM J. Numer. Anal., Vol. 7, No. 4, pp. 627-656, Dec. 1970.

Hockney, R. W., 1970. "The potential calculation and some applications," Methods Computational Phys., Vol. 9, pp. 135-211, 1970.

Young, D. M., 1971. Iterative Solution of Large Linear Systems, Academic Press, New York, 1971.