JOURNAL OF COMPUTATIONAL PHYSICS 34, 314–329 (1980)

On the FACR(l) Algorithm for the Discrete Poisson Equation

CLIVE TEMPERTON

European Centre for Medium Range Weather Forecasts,
Shinfield Park, Reading, Berkshire, U.K.

Received October 3, 1978

Direct methods for the solution of the discrete Poisson equation over a rectangle are commonly based either on Fourier transforms or on block-cyclic reduction. The relationship between these two approaches is demonstrated explicitly, and used to derive the FACR(l) algorithm in which the Fourier transform approach is combined with l preliminary steps of cyclic reduction. It is shown that the optimum choice of l leads to an algorithm for which the operation count per mesh point is almost independent of the mesh size. Numerical results concerning timing and round-off error are presented for the $N \times N$ Dirichlet problem for various values of N and l. Extensions to more general problems, and to implementation on parallel or vector computers are briefly discussed.

1. INTRODUCTION

In a previous paper [18], the author compared several direct methods for the solution of the discrete Poisson equation over an $N \times M$ rectangular grid, in terms of operation counts, storage requirements, speed, and accuracy. The methods considered included FFT-based algorithms, block-cyclic reduction (Buneman's algorithm), and FACR(1) algorithms, in which one preliminary step of block-cyclic reduction is used to halve either the number or the length of the Fourier transforms.

In this paper we consider the FACR(l) algorithm, in which l preliminary steps of block-cyclic reduction are carried out, the reduced system is solved by the FFT method and the solution is completed by l steps of block back-substitution. Both the basic FFT method and Buneman's algorithm are special cases of the FACR(l) algorithm. With the optimum value of l, the FACR(l) method is probably the fastest known numerically stable algorithm for solving the discrete Poisson equation over a rectangle; the algorithms of Lorenz [9] and Schroder *et al.* [10] are possible rivals.

Hockney [7] first introduced the FACR(l) algorithm, using l preliminary steps of a numerically unstable form of block-cyclic reduction, which nevertheless gave satisfactory results for small values of l. Swarztrauber [14] used a stable cyclic reduction scheme based on Variant 2 of Buneman's algorithm [2], derived the optimum value of l, and showed that this gave an operation count asymptotically proportional to $MN \log_2(\log_2 N)$.

In this paper we replace Variant 2 of Buneman's algorithm by Variant 1 for the cyclic reduction and repeat Swarztrauber's analysis under a rather different set of

0021-9991/80/030314–16$02.00/0

assumptions and definitions. Again it is shown that for optimum l the operation count is asymptotically proportional to $MN \log_2(\log_2 N)$; however, it will be demonstrated that for practicable grid sizes the operation count is effectively proportional to MN. It will also be shown that the FACR(l) algorithm arises quite naturally from a study of the relationship between the basic FFT and block-cyclic reduction methods. For simplicity we consider a rectangular grid (i, j): $0 \leqslant i \leqslant N$, $0 \leqslant j \leqslant M$, with unit gridlength, and with Dirichlet boundary conditions on all sides; the number of unknowns is thus $(N-1)(M-1)$. We assume for the time being that N and M are both powers of 2. Furthermore, it will be assumed that the discrete Poisson equation is to be solved a number of times with different right-hand sides but with the same grid and boundary conditions; and that core storage is plentiful, so that we can reserve separate arrays for the right-hand side and the solution (to permit the use of Variant 1 of Buneman's algorithm), and can also precalculate an array of coefficients to be used in the solution of tridiagonal systems by Gaussian elimination. Extensions of the algorithm to other boundary conditions, arbitrary N and M, and more stringent storage restrictions will be considered in Section 6.

2. Relationship between FFT and Block-Cyclic Reduction Methods

It is customary to refer to the FFT and block-cyclic reduction methods for solving the discrete Poisson equation as if they were unrelated; in fact the relationship between them is very close, as will be demonstrated here.

Ordering the variables by rows, and defining

$$\mathbf{x}_j = \begin{pmatrix} x_{1,j} \\ x_{2,j} \\ \vdots \\ x_{N-1,j} \end{pmatrix}, \qquad \mathbf{b}_j = \begin{pmatrix} b_{1,j} \\ b_{2,j} \\ \vdots \\ b_{N-1,j} \end{pmatrix},$$

the discrete Poisson equation can be written as a block-tridiagonal system

$$\mathbf{x}_{j-1} + A\mathbf{x}_j + \mathbf{x}_{j+1} = \mathbf{b}_j, \qquad 1 \leqslant j \leqslant M-1,$$

where

$$A = \begin{bmatrix} -4 & 1 & & & & \\ 1 & -4 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -4 & 1 \\ & & & & 1 & -4 \end{bmatrix}$$

and $\mathbf{x}_0 = \mathbf{x}_M = \mathbf{0}$. (Nonhomogeneous boundary conditions at $j = 0$, M can be handled by modifying $\mathbf{b}_1$ and $\mathbf{b}_{M-1}$, and at $i = 0$, N by modifying the first and last components of each $\mathbf{b}_j$.)

We first set out the cyclic odd-even reduction and factorization (CORF) algorithm for solving this system as given by Buzbee *et al.* [2].

Define $A^{(0)} = A$, $\mathbf{b}_j^{(0)} = \mathbf{b}_j$, $1 \leqslant j \leqslant M-1$.

Then after r reduction steps we have the block-tridiagonal system

$$\mathbf{x}_{j-h} + A^{(r)}\mathbf{x}_j + \mathbf{x}_{j+h} = \mathbf{b}_j^{(r)} \tag{1}$$

involving only those values of j which are multiples of $h = 2^r$, where $A^{(r)}$, $\mathbf{b}_j^{(r)}$ are defined recursively by

$$A^{(r+1)} = 2I - (A^{(r)})^2 \qquad (r \geqslant 0) \tag{2}$$

and

$$\mathbf{b}_j^{(r+1)} = \mathbf{b}_{j-h}^{(r)} + \mathbf{b}_{j+h}^{(r)} - A^{(r)}\mathbf{b}_j^{(r)} \tag{3}$$

for $j = 2h, 4h, \ldots, M-2h$, where $h = 2^r$.

In particular, after $k = (\log_2 M - 1)$ reduction steps the system has been reduced to a single equation:

$$A^{(k)}\mathbf{x}_h = \mathbf{b}_h^{(k)} \tag{4}$$

for $h = 2^k$. The remainder of the system can then be solved by k steps of back-substitution, using decreasing values of r; at each step we use Eq. (1) to obtain $\mathbf{x}_j$ for each j equal to an odd multiple of 2^r, the solution having already been found for all j equal to even multiples of 2^r.

Two aspects of this algorithm are particularly worthy of note: first, the matrices $A^{(r)}$ fill in rapidly as r increases, but, as shown in [2], the computations involving $A^{(r)}$ can be greatly simplified by using the factorization

$$A^{(r)} = -\prod_{j=1}^{2^r} (A + 2\cos\theta_j^{(r)} I) \qquad (r > 0), \tag{5}$$

where $\theta_j^{(r)} = (2j-1)\,\pi/2^{r+1}$.

Second, as also proved in [2], the algorithm is numerically highly unstable, and therefore of little practical use. Buneman's algorithm stabilizes the calculation by defining

$$\mathbf{b}_j^{(r)} = A^{(r)}\mathbf{p}_j^{(r)} + \mathbf{q}_j^{(r)}$$

and computing $\mathbf{p}_j^{(r)}$ and $\mathbf{q}_j^{(r)}$ rather than $\mathbf{b}_j^{(r)}$; all explicit multiplications by $A^{(r)}$ are thereby avoided.

We now consider an alternative method of stabilizing the CORF algorithm. The matrix A can be factorized as

$$A = S^{-1}\Lambda S$$

where S is the matrix representation of a Fourier sine transform; thus S is defined by $S = (s_{ij})$, where $s_{ij} = (2/N)\sin(ij\,\pi/N)$. (With this scaling, $S^{-1} = (N/2)S$.) Λ is a diagonal matrix of eigenvalues of A, defined by $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_{N-1})$, where $\lambda_j = 2\cos(j\pi/N) - 4$. It is easy to show that

$$A^{(r)} = S^{-1}\Lambda^{(r)}S, \tag{6}$$

where the diagonal matrices $\Lambda^{(r)}$ are defined recursively by

$$\Lambda^{(0)} = \Lambda,$$
$$\Lambda^{(r+1)} = 2I - (\Lambda^{(r)})^2, \qquad r \geqslant 0.$$

We can thus replace Eqs. (3), (4), and (1) by

$$\mathbf{b}_j^{(r+1)} = \mathbf{b}_{j-h}^{(r)} + \mathbf{b}_{j+h}^{(r)} - S^{-1}\Lambda^{(r)}S\mathbf{b}_j^{(r)} \tag{7}$$

($r = 0, 1, \ldots, \log_2 M - 2$; $h = 2^r$; $j = 2h, 4h, \ldots, M - 2h$),

$$\mathbf{x}_h = S^{-1}(\Lambda^{(k)})^{-1}\,S\mathbf{b}_h^{(k)} \tag{8}$$

($k = \log_2 M - 1$, $h = 2^k$), and

$$\mathbf{x}_j = S^{-1}(\Lambda^{(r)})^{-1}\,S[\mathbf{b}_j^{(r)} - \mathbf{x}_{j-h} - \mathbf{x}_{j+h}] \tag{9}$$

($r = \log_2 M - 2, \ldots, 1, 0$; $h = 2^r$; $j = h, 3h, \ldots, M - h$).

We will not discuss the numerical stability of the algorithm defined by Eqs. (7)–(9), but rather note that it requires almost twice as many sine transforms (and their inverses) as necessary. If we first compute

$$\hat{\mathbf{b}}_j^{(0)} = S\mathbf{b}_j, \qquad 1 \leqslant j \leqslant M - 1, \tag{10}$$

then the algorithm becomes

$$\hat{\mathbf{b}}_j^{(r+1)} = \hat{\mathbf{b}}_{j-h}^{(r)} + \hat{\mathbf{b}}_{j+h}^{(r)} - \Lambda^{(r)}\hat{\mathbf{b}}_j^{(r)} \tag{11}$$

($r = 0, 1, \ldots, \log_2 M - 2$; $h = 2^r$; $j = 2h, 4h, \ldots, M - 2h$),

$$\hat{\mathbf{x}}_h = (\Lambda^{(k)})^{-1}\,\hat{\mathbf{b}}_h^{(k)} \tag{12}$$

($k = \log_2 M - 1$, $h = 2^k$),

$$\hat{\mathbf{x}}_j = (\Lambda^{(r)})^{-1}\,[\hat{\mathbf{b}}_j^{(r)} - \hat{\mathbf{x}}_{j-h} - \hat{\mathbf{x}}_{j+h}] \tag{13}$$

($r = \log_2 M - 2, \ldots, 1, 0$; $h = 2^r$; $j = h, 3h, \ldots, M - h$), and finally

$$\mathbf{x}_j = S^{-1}\hat{\mathbf{x}}_j, \qquad 1 \leqslant j \leqslant M - 1. \tag{14}$$

But the algorithm defined by Eqs. (10)–(14) is precisely the "basic FFT" method, with the tridiagonal systems solved by (scalar) cyclic reduction. Thus the block-cyclic reduction method (stabilized by using the factorization of Eq. (6)) and the basic FFT method (with the tridiagonal systems solved by cyclic reduction) are equivalent. By the time the block-cyclic reduction method has been stabilized instead by Buneman's algorithm, and the basic FFT method has been modified to solve the tridiagonal systems by Gaussian elimination, the two resulting algorithms appear quite different; nevertheless, as the above argument shows, they are very closely related.

3. The FACR(l) Algorithm

Apart from considerations of stability, there is another motivation for factorizing $A^{(r)}$ using Eq. (6) rather than Eq. (5), for computing $\mathbf{b}_j^{(r)}$ during the reduction phase and $\mathbf{x}_j$ during the back-substitution phase. Using the factorization of Eq. (5), multiplication of a vector by $A^{(r)}$ requires 2^{r+1} additions and 2^r multiplications per component, while multiplication by $(A^{(r)})^{-1}$ requires 2^{r+1} additions and 2^{r+1} multiplications per component, assuming that the tridiagonal systems are solved by Gaussian elimination using precomputed coefficients. Using the factorization of Eq. (6), on the other hand, involves a sine transform, multiplication by a diagonal matrix, and an inverse sine transform, for multiplication by either $A^{(r)}$ or $(A^{(r)})^{-1}$. With the operation count given in [18] for a sine transform using a radix-2 FFT, this requires $3 \log_2 N + 5$ additions and $2 \log_2 N$ multiplications for each component of a vector of length $(N - 1)$, independently of the value of r. The "break-even" point is approximately $r = \log_2(\log_2 N) + 1$; for r larger than this it is faster to use Eq. (6) rather than Eq. (5) for the factorization of $A^{(r)}$.

The basic idea of the FACR(l) algorithm is to use Eq. (5) to factorize $A^{(r)}$ for small values of r, and to switch to Eq. (6) for larger values of r. To prevent the growth of round-off errors, the preliminary steps of block-cyclic reduction in the algorithm presented below are carried out using Variant 1 of Buneman's procedure.

For $1 \leqslant j \leqslant M - 1$, define $\mathbf{p}_j^{(0)} = \mathbf{0}$, $\mathbf{q}_j^{(0)} = \mathbf{b}_j$. Then for $0 \leqslant r \leqslant l - 1$, compute

$$\mathbf{p}_j^{(r+1)} = \mathbf{p}_j^{(r)} - (A^{(r)})^{-1} (\mathbf{p}_{j-h}^{(r)} + \mathbf{p}_{j+h}^{(r)} - \mathbf{q}_j^{(r)}), \tag{15}$$

$$\mathbf{q}_j^{(r+1)} = \mathbf{q}_{j-h}^{(r)} + \mathbf{q}_{j+h}^{(r)} - 2\mathbf{p}_j^{(r+1)}, \tag{16}$$

where $h = 2^r$, $j = 2h, 4h, \ldots, M - 2h$, $A^{(r)}$ is given by Eq. (5), and $\mathbf{p}_0^{(r)} = \mathbf{p}_M^{(r)} = \mathbf{0}$.

After l cyclic reduction steps, we have the following system:

$$\mathbf{x}_{j-h} + A^{(l)}\mathbf{x}_j + \mathbf{x}_{j+h} = A^{(l)}\mathbf{p}_j^{(l)} + \mathbf{q}_j^{(l)} \tag{17}$$

with $\mathbf{x}_0 = \mathbf{x}_M = \mathbf{0}$, involving only those values of j which are multiples of $h = 2^l$.

If we then define

$$\mathbf{y}_j = \mathbf{x}_j - \mathbf{p}_j^{(l)}, \tag{18}$$

$$\mathbf{g}_j = \mathbf{q}_j^{(l)} - (\mathbf{p}_{j-h}^{(l)} + \mathbf{p}_{j+h}^{(l)}), \tag{19}$$

Eq. (17) can be rewritten as

$$\mathbf{y}_{j-h} + A^{(l)}\mathbf{y}_j + \mathbf{y}_{j+h} = \mathbf{g}_j \tag{20}$$

for $h = 2^l$, $j = h, 2h,..., M - h$, with $\mathbf{y}_0 = \mathbf{y}_M = \mathbf{0}$.

Now defining $\hat{\mathbf{y}}_j = S\mathbf{y}_j$, $\hat{\mathbf{g}}_j = S\mathbf{g}_j$, and using Eq. (6) to factorize $A^{(l)}$, Eq. (20) becomes

$$\hat{\mathbf{y}}_{j-h} + \Lambda^{(l)}\hat{\mathbf{y}}_j + \hat{\mathbf{y}}_{j+h} = \hat{\mathbf{g}}_j\,. \tag{21}$$

As $\Lambda^{(l)}$ is diagonal, Eq. (21) represents a set of $(N - 1)$ independent tridiagonal systems which can easily be solved for $\hat{\mathbf{y}}_j$, $j = h, 2h,.... M - h$. From each $\hat{\mathbf{y}}_j$ we obtain $\mathbf{y}_j = S^{-1}\hat{\mathbf{y}}_j$ and hence $\mathbf{x}_j = \mathbf{y}_j + \mathbf{p}_j^{(l)}$.

The remaining $\mathbf{x}_j$ are then found in l steps of back-substitution: for $r = l - 1$, $l - 2,..., 0$ we solve the system

$$A^{(r)}(\mathbf{x}_j - \mathbf{p}_j^{(r)}) = \mathbf{q}_j^{(r)} - (\mathbf{x}_{j-h} + \mathbf{x}_{j+h}) \tag{22}$$

for $h = 2^r$, $j = h, 3h,..., M - h$, using once again the factorization of $A^{(r)}$ given by Eq. (5).

In the algorithm defined by Eqs. (15)–(22), l can take any value from 0 to $\log_2 M - 1$. For $l = 0$, we simply have the basic FFT method. For $l = 1$, we have a stabilized version of Hockney's FACR(1) algorithm [6, 7]. For $l = \log_2 M - 1$, the final step of cyclic reduction yields the system

$$A^{(l)}\mathbf{x}_h = A^{(l)}\mathbf{p}_h^{(l)} + \mathbf{q}_h^{(l)}$$

for $h = 2^l = M/2$, and Eq. (20) becomes simply

$$A^{(l)}\mathbf{y}_h = \mathbf{q}_h^{(l)} \tag{23}$$

and hence

$$\mathbf{y}_h = S^{-1}(\Lambda^{(l)})^{-1}\,S\mathbf{q}_h^{(l)}, \qquad h = M/2.$$

Buneman's algorithm (Variant 1) also yields Eq. (23) after $l = \log_2 M - 1$ reduction steps, the difference being that the system is solved by using Eq. (5) again to factorize $A^{(l)}$. Formally we can identify Buneman's algorithm with FACR($\log_2 M$), although there are only $\log_2 M - 1$ preliminary reduction steps.

For $l = 0$, only one array of size MN is needed, and the solution field can overwrite the right-hand side. For $l > 0$, the storage requirements for this algorithm are the same as for Buneman's algorithm (Variant 1). As pointed out in [18], this is only $3MN/2$; alternatively, the vectors $\mathbf{p}_j^{(r)}$ and $\mathbf{q}_j^{(r)}$ for $r > 0$ can share a second array of size MN which finally contains the solution field, and the right-hand side is preserved.

However, we show in the next section that if the tridiagonal systems are solved by Gaussian elimination using precomputed coefficients, then for $0 < l < \log_2 M$ there is a considerable saving in the storage requirements for these coefficients compared with both the basic FFT and Buneman's algorithms.

4. Operation Counts and Optimum l

Swarztrauber [14] presented an algorithm very similar to the one described above, the differences being that the block-cyclic reduction was performed using Variant 2 of Buneman's algorithm (in which the vectors $\mathbf{p}_j^{(r)}$ are eliminated) and the tridiagonal systems were solved by (scalar) cyclic reduction. In deriving an operation count and hence determining the optimum value of l, Swarztrauber defined an operation as consisting of a multiplication or division together with an addition or subtraction, and included only those operations which contributed toward the asymptotic operation count. In this paper we take a somewhat different viewpoint; we count additions and multiplications separately, and include all of them (apart from some lower-order terms of little significance).

In fact it turns out that for practicable grid sizes, there are approximately twice as many additions as multiplications, while the asymptotic operation count underestimates the actual operation count by at least 50 %.

In deriving an operation count we assume, following [18], that a tridiagonal system of order n (with unit subdiagonals and superdiagonals) can be solved in $2n$ additions and $2n$ multiplications using precomputed coefficients, while a sine transform of order n takes $(1.5 \log_2 n + 2.5)n$ additions and $(\log_2 n - 0.5)n$ multiplications.

During the rth preliminary step of cyclic reduction ($1 \leqslant r \leqslant l$), we have to solve $(M - 2^r)/2$ tridiagonal systems of order $(N - 1)$; altogether these contribute approximately lMN additions and lMN multiplications. Implementation of Eqs. (15) and (16) involves some extra additions: approximately $3MN/2$ for the first reduction ($r = 1$), since $\mathbf{p}_j^{(0)} = \mathbf{0}$ for all j, and $6MN/2^r$ for the rth reduction ($2 \leqslant r \leqslant l$). (Multiplication by 2 has been counted as an addition.) The total number of extra additions from this phase is thus approximately $MN(3/2 + 6\sum_{r=2}^{l} 2^{-r}) = MN(9/2 - 6/2^l)$.

After l steps of cyclic reduction, we are left with a system of order $L = (N - 1)(M/2^l - 1)$, defined by Eq. (17). Computation of the vectors $\mathbf{g}_j$ takes $2L$ additions; the sine transforms to find $\hat{\mathbf{g}}_j$ take $(1.5 \log_2 N + 2.5)L$ additions and $(\log_2 N - 0.5)L$ multiplications; the solution of tridiagonal systems for $\hat{\mathbf{y}}_j$ takes $2L$ additions and $2L$ multiplications; the inverse sine transforms for $\mathbf{y}_j$ take $(1.5 \log_2 N + 2.5)L$ additions and $(\log_2 N - 0.5)L$ multiplications; and finally the $\mathbf{x}_j$ are found after another L additions. Taking $L \sim 2^{-l}MN$, the contributions from this

phase of the algorithm are $2^{-l}MN(3\log_2 N + 10)$ additions and $2^{-l}MN(2\log_2 N + 1)$ multiplications.

Finally, during each of the l steps of back-substitution we have to solve $M/2$ tridiagonal systems of order $(N-1)$, contributing altogether approximately lMN additions and lMN multiplications. The extra additions required to implement Eq. (22) amount to $3(N-1)$ for each $\mathbf{x}_j$ found in the first $(l-1)$ steps of back-substitution, but only $2(N-1)$ for each $\mathbf{x}_j$ found in the last step (since $\mathbf{p}_j^{(0)} = \mathbf{0}$). The total number of extra additions from this phase is approximately $MN(5/2 - 3/2^l)$.

Summing up all of these contributions and making only a slight further approximation in each case, the operation count for the whole algorithm is approximately

$$[(2l + 7) + 2^{-l}(3\log_2 N)]\, MN \text{ additions} \tag{24}$$

and

$$[2l + 2^{-l}(2\log_2 N)]\, MN \text{ multiplications.} \tag{25}$$

Strictly speaking, these estimates are only valid for $1 \leqslant l \leqslant \log_2 M - 1$. From [18] we have the following operation counts: for the basic FFT method, FACR(0), $(3\log_2 N + 7)\, MN$ additions and $(2\log_2 N + 1)\, MN$ multiplications; for Buneman's algorithm, FACR($\log_2 M$), $(2\log M + 5)\, MN$ additions and $(2\log_2 M - 2)\, MN$ multiplications.

Differentiating (24) and (25) with respect to l, we find that the number of additions is minimized at $l \sim \log_2(\log_2 N)$, while the number of multiplications is minimized at $l \sim \log_2(\log_2 N) - \frac{1}{2}$. Taking $l \sim \log_2(\log_2 N)$ to be the optimum value for the whole algorithm, and substituting in (24) and (25), we obtain the following operation counts for the FACR(l) algorithm with optimum l:

$$[2\log_2(\log_2 N) + 10]\, MN \text{ additions}$$

and

$$[2\log_2(\log_2 N) + 2]\, MN \text{ multiplications.}$$

Several observations are in order here. First, the optimum value of l and the total number of operations per point depend only on N, the length of the Fourier transforms. Second, assuming that the range of practicable grid sizes is $16 \leqslant N \leqslant 256$, the operation count for FACR(l) with optimum l is 14–16 additions and 6–8 multiplications per point; hence the FACR(l) algorithm represents a very close approach to the elusive "stable $O(N^2)$ algorithm" [1, 4]. Even for $N = 4096$ the operation count is only 17 additions and 9 multiplications per point. Third, while the actual count for practicable grid sizes is 20–24 operations per point, the "asymptotic" count is $4\log_2(\log_2 N)$ or 8–12 operations per point, an underestimate by at least 50 %.

It has already been mentioned that this analysis is only valid for $1 \leqslant l \leqslant \log_2 M - 1$; in Table I the approximate numbers of additions and multiplications per point are presented for the $N \times N$ Dirichlet problem with $8 \leqslant N \leqslant 128$ and all possible values of l, including $l = 0$ (the basic FFT method) and $l = \log_2 N$ (Buneman's

algorithm). In deriving Table I, some lower-order terms, omitted in the foregoing analysis, were included. Note that for $N = 8$, Buneman's algorithm requires fewer operations than for any $l < \log_2 N$. For $N \geqslant 16$, the operation count is minimized at $l = 2$ or $l = 3$, in accordance with the analysis above; the minimum is quite shallow.

TABLE I

Number of Additions/Multiplications per Point for the $N \times N$ Dirichlet Problem

	N				
l	8	16	32	64	128
0	16/7	19/9	22/11	25/13	28/15
1	13/5	15/6	17/7	18/8	20/9
2	11/5	13/6	14/6	15/7	16/8
3	9/5	12/6	14/7	15/7	15/8
4	—	12/7	14/8	15/8	16/9
5	—	—	14/8	16/9	17/10
6	—	—	—	17/10	18/11
7	—	—	—	—	19/12

Throughout this discussion we have assumed that all tridiagonal systems are solved by Gaussian elimination using precomputed coefficients. It is of interest to determine the number of such coefficients which are required, since this affects both the storage requirements and the time taken for preprocessing. For the simple tridiagonal systems encountered here, a system of order n requires n precomputed coefficients, which can be calculated with $(n - 1)$ additions and n divisions.

In the reduction and back-substitution phases of the FACR(l) algorithm, we have to solve systems involving a total of $2^l - 1$ different tridiagonal matrices, each of order $(N - 1)$. In the remaining part of the algorithm, after the Fourier sine transforms have been performed, we have to solve $(N - 1)$ systems each of order $(M/2^l - 1)$. The total number of coefficients required is thus $\nu = (N - 1)\{2^l + M/2^l - 2\}$; in this case the total is also valid for $l = 0$ and $l = \log_2 M$. Differentiating ν with respect to l, we find that the minimum number of coefficients is $2(N - 1)(M^{1/2} - 1)$ at $l = \frac{1}{2}\log_2 M$.

The values of $\nu/(N - 1)$ are presented in Table II for the $N \times N$ Dirichlet problem $(8 \leqslant N \leqslant 128)$ and all possible values of l. For $l = 0$ and $l = \log_2 N$, the array of coefficients is the same size as the right-hand side and solution arrays, but for intermediate values of l the extra storage required is considerably less. Thus the FACR(l) algorithm with optimum l not only requires less computation than either the basic FFT method or Buneman's algorithm; if Gaussian elimination is used for solving the tridiagonal systems, then the optimum FACR(l) algorithm also requires fewer coefficients (and hence also less preprocessing) than either of the basic methods.

TABLE II

Number of Precomputed Coefficients ν Required for Tridiagonal Systems for the $N \times N$ Dirichlet Problem

	N				
l	8	16	32	64	128
0	7	15	31	63	127
1	4	8	16	32	64
2	4	6	10	18	35
3	7	8	10	14	22
4	—	15	15	18	22
5	—	—	31	32	34
6	—	—	—	63	64
7	—	—	—	—	127

[a] Tabulated value is $\nu/(N-1)$.

As outlined in Section 6, the number of coefficients can in fact be further reduced by almost a factor of 2 if a modified form of Gaussian elimination is used for the tridiagonal systems.

5. Numerical Experiments

A Fortran program (PSOLVE) was written to solve Poisson's equation over a rectangle under Dirichlet boundary conditions, using the FACR(l) algorithm as described in Section 3. The program incorporates a "radix $4+2$" FFT algorithm requiring no reordering [17], and won the "Poisson-solver contest" (on a 128×32 problem) organized at Karlsruhe in March 1977 [11]. A listing of PSOLVE (with some improvements incorporated since the Karlsruhe version) was included in [16].

Experiments were run to determine the speed and accuracy of the program on an $N \times N$ problem for various values of N and l. Hockney [8] has compared several Poisson-solver programs on various computers and finds that the fastest program varies from one machine to another, and even from one compiler to another on the same machine. Accordingly, PSOLVE has been run on various machines; in this paper we report results obtained on a CDC CYBER-175 and on an IBM 360/195, using the compilers FTN 4.6 and FORTX, respectively.

The CYBER-175 times are presetned in Table III. Buneman's algorithm is the fastest for $N = 8$ (as suggested by the operation counts in Table I) and also for $N = 16$. For $N \geqslant 16$, the optimum value of l is found to be 3 or 4 rather than 2 or 3 as predicted by the analysis of Section 4. The reason doubtless lies in the extra overheads incurred in calling the FFT subroutine; it is worth performing an extra step of cyclic reduction to halve the number of Fourier transforms, at the expense of a slightly

higher floating-point operation count. Similar observations were made in [18]. Taking the optimum value of l in each case, the CPU time per unknown is 1.00×10^{-5} sec at $N = 8$, and 1.07×10^{-5} sec at $N = 128$.

TABLE III

CYBER-175 CPU Times (Seconds) for the $N \times N$ Dirichlet Problem

	N				
l	8	16	32	64	128
0	2.24×10^{-3}	8.43×10^{-3}	2.61×10^{-2}	9.82×10^{-2}	3.69×10^{-1}
1	1.26×10^{-3}	4.89×10^{-3}	1.62×10^{-2}	6.24×10^{-2}	2.38×10^{-1}
2	7.35×10^{-4}	3.27×10^{-3}	1.19×10^{-2}	4.72×10^{-2}	1.84×10^{-1}
3	4.88×10^{-4}	2.49×10^{-3}	1.06×10^{-2}	4.34×10^{-2}	1.73×10^{-1}
4	—	2.21×10^{-3}	1.03×10^{-2}	4.44×10^{-2}	1.82×10^{-1}
5	—	—	1.05×10^{-2}	4.71×10^{-2}	1.98×10^{-1}
6	—	—	—	4.94×10^{-2}	2.15×10^{-1}
7	—	—	—	—	2.29×10^{-1}

TABLE IV

IBM 360/195 CPU Times (Seconds) for the $N \times N$ Dirichlet Problem

	N				
l	8	16	32	64	128
0	1.38×10^{-3}	5.75×10^{-3}	2.11×10^{-2}	9.04×10^{-2}	3.60×10^{-1}
1	9.38×10^{-4}	3.97×10^{-3}	1.53×10^{-2}	6.49×10^{-2}	2.63×10^{-1}
2	6.79×10^{-4}	3.22×10^{-3}	1.31×10^{-2}	5.54×10^{-2}	2.26×10^{-1}
3	5.91×10^{-4}	2.93×10^{-4}	1.31×10^{-2}	5.55×10^{-2}	
4	—	2.92×10^{-2}	1.37×10^{-2}	5.99×10^{-2}	
5	—	—	1.43×10^{-2}	6.48×10^{-2}	
6	—	—	—	6.86×10^{-2}	
7	—	—	—	—	

The IBM 360/195 CPU times are presented in Table IV. Again Buneman's algorithm is the fastest for $N = 8$; for $N = 16$ this is also true, but the time for $l = 3$ is almost identical. For $N \geqslant 16$, the optimum value of l is 2 or 3, in agreement with the theoretical estimate. Taking the optimum value of l in each case, the CPU time per unknown increases from 1.21×10^{-5} sec at $N = 8$ to 1.40×10^{-5} sec at $N = 128$.

Differing degrees of success in optimizing key sections of the program are reflected in the fact that at $l = 0$, where most of the work consists of Fast Fourier Transforms, the 360/195 appears to be the faster machine; while at $l = \log_2 N$, where most of the work consists of solving tridiagonal systems, the CYBER-175 appears to be faster.

Experiments were also carried out to investigate the accuracy of FACR(l) algorithms. For each value of N, a number of random "true" solutions were generated with values in the interval $[-1, +1]$. Corresponding right-hand sides were then computed using temporary double precision, for reasons set out in [18], and input to PSOLVE. The computed solutions were compared with the true solutions, and the mean maximum absolute errors for each value of N and l were then determined.

On the CYBER-175, the option is available at compile time to specify either truncated or rounded floating-point arithmetic; the effect of this choice on the accuracy of PSOLVE was also studied.

TABLE V

Mean Maximum Error for the $N \times N$ Dirichlet Problem[a]

	N				
	8	16	32	64	129
0	4.19×10^{-14}	1.38×10^{-13}	4.24×10^{-13}	2.16×10^{-12}	8.34×10^{-12}
1	3.02×10^{-14}	8.29×10^{-14}	3.59×10^{-13}	1.67×10^{-12}	7.62×10^{-12}
2	2.49×10^{-14}	5.40×10^{-14}	2.35×10^{-13}	7.57×10^{-13}	3.87×10^{-12}
3	2.42×10^{-14}	4.17×10^{-14}	1.09×10^{-13}	4.28×10^{-13}	1.99×10^{-12}
4	—	4.05×10^{-14}	6.55×10^{-14}	2.16×10^{-13}	9.40×10^{-13}
5	—	—	6.73×10^{-14}	1.22×10^{-13}	4.16×10^{-13}
6	—	—	—	1.14×10^{-13}	2.24×10^{-13}
7	—	—	—	—	1.71×10^{-13}

[a] CYBER-175, old version of sine transform, truncated arithmetic.

Table V shows the results obtained on the CYBER-175 using the Karlsruhe version of PSOLVE. For each value of N, the error decreases with increasing l, so that FACR($\log_2 N - 1$) and FACR($\log_2 N$) (i.e., Buneman's algorithm) are the most accurate in each case. (Here truncated arithmetic was used.) For fixed l, including $l = 0$ (the basic FFT algorithm), the errors are roughly proportional to N^2. For Buneman's algorithm (which does not use the FFT), however, the errors are roughly proportional to $N^{2/3}$.

In the Karlsruhe version of PSOLVE, the algorithm of Cooley, Lewis, and Welch [3] was used to convert real sine transforms of length N into complex transforms of length $N/2$. Since the sine transform is its own inverse (apart from a scaling factor), one can alternatively "invert" their algorithm and it turns out that by doing so the round-off error is considerably reduced. The reason is probably that the original algorithm involves a multiplication by $1/(\sin(j\pi/N))$, which is large for small j; in the inverted form this becomes a multiplication by $\sin(j\pi/N)$. The version of PSOLVE given in [16] uses the new version of the sine transform, an outline of which is given in the Appendix to this paper.

TABLE VI

Mean Maximum Error for the $N \times N$ Dirichlet Problem[a]

	N				
l	8	16	32	64	128
0	5.68×10^{-14}	1.14×10^{-13}	2.10×10^{-13}	4.30×10^{-13}	8.94×10^{-13}
1	3.38×10^{-14}	7.30×10^{-14}	1.22×10^{-13}	3.17×10^{-13}	5.89×10^{-13}
2	2.42×10^{-14}	4.73×10^{-14}	6.65×10^{-14}	2.05×10^{-13}	3.81×10^{-13}
3	2.42×10^{-14}	4.07×10^{-14}	6.59×10^{-14}	1.46×10^{-13}	2.85×10^{-13}
4	—	4.05×10^{-14}	6.64×10^{-14}	1.17×10^{-13}	2.29×10^{-13}
5	—	—	6.73×10^{-14}	1.11×10^{-13}	1.92×10^{-13}
6	—	—	—	1.14×10^{-13}	1.79×10^{-13}
7	—	—	—	—	1.71×10^{-13}

[a] CYBER-175, new version of sine transform, truncated arithmetic.

Following this discovery, the accuracy experiments described above were repeated; the results are presented in Table VI. For $l = \log_2 N$ (Buneman's algorithm), the results are of course the same as before. Otherwise (apart from $N = 8$, $l = 0$, and $l = 1$) the errors are reduced; for $N = 128$, $l = 0$ there is an order of magnitude reduction. For fixed l, the errors are now roughly proportional to N.

TABLE VII

Mean Maximum Error for the $N \times N$ Dirichlet Problem[a]

	N				
l	8	16	32	64	128
0	2.33×10^{-14}	5.84×10^{-14}	1.18×10^{-13}	2.38×10^{-13}	5.50×10^{-13}
1	1.97×10^{-14}	5.33×10^{-14}	9.45×10^{-14}	1.27×10^{-13}	3.45×10^{-13}
2	1.67×10^{-14}	4.48×10^{-14}	7.53×10^{-14}	1.05×10^{-13}	2.82×10^{-13}
3	1.49×10^{-14}	4.12×10^{-14}	7.37×10^{-14}	9.31×10^{-14}	2.55×10^{-13}
4	—	4.01×10^{-14}	7.20×10^{-14}	1.06×10^{-13}	2.42×10^{-13}
5	—	—	7.08×10^{-14}	1.04×10^{-13}	2.17×10^{-13}
6	—	—	—	1.04×10^{-13}	2.00×10^{-13}
7	—	—	—	—	1.87×10^{-13}

[a] CYBER-175, new version of sine transform, rounded arithmetic.

The version of PSOLVE incorporating the new sine transform was also tested on the CYBER-175 using rounded floating-point arithmetic. The results are shown in Table VII. It is seen that for $l = 0$ the errors are roughly halved by using the rounding option, while for $l = \log_2 N$ the errors are generally similar whichever of the two options is chosen.

TABLE VIII

Mean Maximum Error for the $N \times N$ Dirichlet Problem[a]

	N				
l	8	16	32	64	128
0	4.14×10^{-6}	7.20×10^{-6}	1.51×10^{-5}	2.98×10^{-5}	5.99×10^{-5}
1	2.53×10^{-6}	4.27×10^{-6}	1.05×10^{-5}	2.00×10^{-5}	3.38×10^{-5}
2	2.56×10^{-6}	4.05×10^{-6}	8.98×10^{-6}	1.41×10^{-5}	2.57×10^{-5}
3	2.54×10^{-6}	3.82×10^{-6}	8.08×10^{-6}	1.30×10^{-5}	2.44×10^{-5}
4	—	3.86×10^{-6}	7.87×10^{-6}	1.28×10^{-5}	2.35×10^{-5}
5	—	—	8.12×10^{-6}	1.33×10^{-5}	2.41×10^{-5}
6	—	—	—	1.37×10^{-5}	2.52×10^{-5}
7	—	—	—	—	2.58×10^{-5}

[a] IBM 360/195, new version of sine transform, truncated arithmetic.

Only the new version of PSOLVE was tested on the IBM 360/195; the results are presented in Table VIII. The errors are of course much larger, because of the difference in word length. (The mantissa of a single-precision floating-point number contains 24 bits on IBM machines, compared with 48 bits on CDC machines, and rounded floating-point arithmetic is not available as an option when compiling FORTRAN.) Again, for fixed l the error is roughly proportional to N.

Comparing the results of Table VIII with those of Table V of [18] for corresponding Assembler programs on an IBM 360/195, the decreased errors for FACR(0) reflect the improved sine transform used in PSOLVE. More surprisingly, the errors for PSOLVE with $l = \log_2 N$ are also considerably smaller than for the Assembler program implementing Buneman's algorithm. The reason may be that this algorithm consists largely of solving systems of the form

$$\prod_{i=1}^{2^r} (A - \lambda_i I)\,\mathbf{x} = \mathbf{b},$$

and the two programs solve such systems using the λ_i's in a different order. Schumann [11] pointed out the need for research into the dependence of round-off errors on the spectrum of the right-hand side of Poisson's equation; from the above result, it appears that the question of round-off errors in Poisson-solvers may be even more involved.

6. Generalizations

The FACR(l) algorithm developed above solves the discrete Poisson equation under a rather restrictive set of conditions; in this final section we indicate some generalizations.

First, the extension to the case $\Delta_i \neq \Delta_j$ is trivial, where Δ_i and Δ_j are respectively the grid lengths in the i and j directions. If N is not a power of 2, then a mixed-radix FFT [13, 17] must be used (for efficiency, N should be a product of small primes). The FACR(l) algorithm described above only requires that M be a multiple of 2^l; for arbitrary M, a generalized form of block-cyclic reduction is available [12, 15]. Other boundary conditions at $i = 0$, N can be incorporated by adding extra preprocessing and postprocessing options to the FFT [3, 14], and at $j = 0$, M by modifying the block-cyclic reduction process appropriately [2].

As mentioned at the end of Section 3, the FACR(l) algorithm based on Variant 1 of Buneman's algorithm requires (for $l > 0$) an auxiliary array of dimension approximately $MN/2$ even if the solution overwrites the right-hand side. If storage is so restricted that this is undesirable, then the version of the FACR(l) algorithm presented by Swarztrauber [14] and based on Variant 2 of Buneman's algorithm can be used instead; the additional computation required is small, especially if Variant 2 is implemented as suggested in [18].

Finally, the array of coefficients can be eliminated by using alternative methods (e.g., cyclic reduction) for solving the tridiagonal systems, at the expense of some extra computation. It has already been shown, however, that for l close to the optimum value the coefficient array is much smaller than the solution and right-hand side arrays. As noted in [18], the storage requirement for the coefficients can be almost halved by using "symmetric" Gaussian elimination [5]; with this modification available, it appears that using alternative methods for the tridiagonal systems will seldom be worthwhile.

In this paper we have considered the implementation of the FACR(l) algorithm on serial computers; on a parallel or vector computer we must also consider the effect of the choice of l on the degree of parallelism shown by the algorithm. For instance, at $l = 0$ the algorithm is highly parallel, since at each stage we are either performing $M - 1$ independent sine transforms or solving $N - 1$ independent tridiagonal systems. For $l > 0$, the degree of parallelism decreases at each stage of the reduction process. Thus the optimum value of l is likely to be smaller on parallel and vector computers than on serial computers.

APPENDIX: An Algorithm for the Sine Transform

Details of the improved sine transform referred to in Section 5 are given here. Given x_j, $1 \leqslant j \leqslant N - 1$, we wish to compute

$$b_k = \sum_{j=1}^{N-1} x_j \sin(jk\pi/N), \qquad 1 \leqslant k \leqslant N - 1.$$

(1) Set $y_0 = 0$,

$$y_j = \sin(j\pi/N)(x_j + x_{N-j}) + \tfrac{1}{2}(x_j - x_{N-j}), \qquad 1 \leqslant j \leqslant N - 1.$$

(2) Using a real periodic FFT (if necessary using the algorithm of [3] to convert to a complex FFT of length $N/2$), compute

$$\tilde{a}_k = \sum_{j=0}^{N-1} y_j \cos(2jk\pi/N), \qquad 0 \leqslant k \leqslant N/2$$

and

$$\tilde{b}_k = \sum_{j=0}^{N-1} y_j \sin(2jk\pi/N), \qquad 1 \leqslant k \leqslant N/2 - 1.$$

(3) Finally set

$$b_{2k} = \tilde{b}_k, \qquad 1 \leqslant k \leqslant N/2 - 1,$$
$$b_1 = \tfrac{1}{2}\tilde{a}_0,$$

and

$$b_{2k+1} = b_{2k-1} + \tilde{a}_k, \qquad 1 \leqslant k \leqslant N/2 - 1.$$

REFERENCES

1. R. E. BANK AND D. J. ROSE, *SIAM J. Numer. Anal.* **12** (1975), 529–540.
2. B. L. BUZBEE, G. H. GOLUB, AND C. W. NIELSON, *SIAM J. Numer. Anal.* **7** (1970), 627–656.
3. J. W. COOLEY, P. A. W. LEWIS, AND P. D. WELCH, *J. Sound Vib.* **12** (1970), 315–337.
4. F. W. DORR, *SIAM Rev.* **17** (1975), 412–415.
5. D. J. EVANS AND M. HATZOPOULOS, *Comput. J.* **19** (1976), 184–187.
6. R. W. HOCKNEY, *J. Assoc. Comput. Mach.* **12** (1965), 95–113.
7. R. W. HOCKNEY, The potential calculation and some applications, *in* "Methods of Computational Physics" (B. Alder, S. Fernbach, and M. Rotenberg, Eds.), Vol. 9, pp. 135–211, Academic Press, New York, 1970.
8. R. W. HOCKNEY, Computers, compilers and Poisson-solvers, *in* "Computers, Fast Elliptic Solvers and Applications (U. Schumann, Ed.), Advance Publications, London, 1978.
9. E. N. LORENZ, *Mon. Wea. Rev.* **104** (1976), 961–966.
10. J. SCHRÖDER, U. TROTTENBERG, AND H. REUTERSBERG, *Numer. Math.* **26** (1976), 429–459.
11. U. SCHUMANN, Report on the GAMM Workshop on fast solution methods for the discretized Poisson equation, *in* "Computers, Fast Elliptic Solvers and Applications" (U. Schumann, Ed.), Advance Publications, London, 1978.
12. U. SCHUMANN AND R. A. SWEET, *J. Computational Physics* **20** (1976), 171–182.
13. R. C. SINGLETON, *IEEE Trans. Audio Electroacoustics* **17** (1969), 93–103.
14. P. N. SWARZTRAUBER, *SIAM Rev.* **19** (1977), 490–501.
15. R. A. SWEET, *SIAM, Numer. Anal.* **14** (1977), 706–720.
16. C. TEMPERTON, "On the FACR (l) Algorithm for the Discrete Poisson Equation," ECMWF Research Department Internal Report No. 14, 1977.
17. C. TEMPERTON, "Mixed-Radix Fast Fourier Transforms without Reordering," ECMWF Technical Report No. 3, 1977.
18. C. TEMPERTON, *J. Computational Physics* **31** (1979), 1–20.