JOURNAL OF COMPUTATIONAL PHYSICS 31, 1–20 (1979)

Review

Direct Methods for the Solution of the Discrete Poisson Equation: Some Comparisons

CLIVE TEMPERTON

European Centre for Medium Range Weather Forecasts, Shinfield Park, Reading, Berkshire, United Kingdom

Received March 31, 1978; revised September 6, 1978

Some comparisons are presented between various algorithms for solving the discrete Poisson equation over a rectangle with Dirichlet boundary conditions on two opposite sides and either periodic or Dirichlet boundary conditions on the other two sides. The methods considered include those based on FFT in one-dimension, block-cyclic reduction, and variants of the FACR(1) algorithm in which FFT is combined with a preliminary cyclic reduction step. Comparisons are made in terms of detailed operation counts, execution time, and accuracy with respect to round-off error, for a wide range of grid sizes. The importance of careful programming is stressed, and some suggestions are offered for efficient implementation of the algorithms.

1. INTRODUCTION

During the past ten years, it has become increasingly popular to solve the discretized Poisson equation (and related problems) by direct rather than iterative methods. Such methods were at first applied only to the simple Poisson equation on a rectangular $N \times M$ grid, where N was restricted to the form $N = 2^k$ (or $N = 3 \times 2^k$ in Hockney's [12] important early paper). More recently, direct methods have been extended to other regular regions [20, 23], to arbitrary N [18, 25, 26], to irregular regions [4, 5, 28], and to general separable elliptic equations [21].

Many of the algorithms developed for the simple Poisson equation fall into two apparently distinct categories: those based on Fourier decomposition in one dimension, using fast Fourier transform (FFT) techniques, and those based on block-cyclic reduction (Buneman's algorithm). Both approaches are documented in some detail in the paper by Buzbee *et al.* [3]. It is natural to ask which of the two approaches leads to the fastest algorithms; from the available literature, the answer is not clear. For example, Sweet [24] found that for a particular problem, Buneman's algorithm was at least twice as fast as a method based on FFT, while Fischer *et al.* [10] found FFT methods to be the faster. The reasons for this confusion lie in the variety of methods available for carrying out component parts of the algorithms (principally fast Fourier transforms and the solution of tridiagonal systems), and in the assumptions made by different authors (e.g., whether or not any coefficients required in

0021-9991/79/040001-20$02.00/0

solving tridiagonal systems have been precalculated). Hockney [14] has also noted the influence of program design, compilers, and computer architecture on the relative performance of different algorithms. When comparing operation counts rather than execution times, there are further pitfalls; Hockney [12] pointed out the importance of counting additions as well as multiplications, and the danger of including only the highest-order terms. Unfortunately, not all subsequent authors have followed this advice.

In addition to the algorithms based on either Fourier analysis or cyclic reduction, we shall consider variants of the FACR(1) algorithm [12, 13] in which Fourier analysis is combined with one preliminary step of cyclic reduction. In most cases this is a faster method than either Fourier analysis or cyclic reduction on its own. The more general FACR(l) algorithm [13, 22] will be studied in a later paper [30] in which the relationship between the FFT-based and block-cyclic reduction algorithms will also be explicitly demonstrated. Other direct methods such as total reduction [16] and "marching" or "shooting" [1, 7] will not be considered here, though they may be competitive [17].

This paper has as its main aims the following: to establish reasonably accurate operation counts for a number of variants of the direct methods applied to a simple problem; to outline some alternative variants; to offer hints on how certain methods may be implemented most efficiently; to compare execution times; and also to compare the relative accuracy of different methods.

We consider the discretized Poisson equation on a rectangular $N \times M$ grid (i, j): $0 \leqslant i \leqslant N$, $0 \leqslant j \leqslant M$, and for simplicity we assume a unit grid length in each direction, so that the equation has the simple form:

$$x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1} - 4x_{i,j} = b_{i,j}, \tag{1}$$

where the boundary conditions at $i = 0, N$ are either Dirichlet,

$$x_{0,j} = x_{N,j} = 0, \qquad 0 \leqslant j \leqslant M,$$

or periodic,

$$x_{0,j} = x_{N,j}, \qquad 0 \leqslant j \leqslant M,$$

and the boundary conditions at $j = 0, M$ are Dirichlet,

$$x_{i,0} = x_{i,M} = 0, \qquad 0 \leqslant i \leqslant N.$$

We also assume that N is a power of 2, and that FFT's and block-cyclic reduction will normally be performed in the i-direction. Finally, we assume that the same problem has to be solved a number of times with different right-hand sides, and that sufficient core storage is available to precalculate and store any required coefficients; in some cases we will also wish to store both the right-hand side and the solution array separately. The consequences of altering some of these assumptions will also receive mention.

2. Preliminaries

Very careful operation counts for some of the algorithms which follow show that the number of additions or multiplications required is typically of the form

$$(M - 1)\{K_1 N \log_2 N + K_2 N + K_3 \log_2 N + K_4\},$$

where, however, the third and fourth terms of the sum are small compared to the first two, and tend to depend on programming details. We are therefore justified in neglecting them and simply establishing $K_1 \log_2 N + K_2$, the number of additions or multiplications per point.

As mentioned previously, the direct methods outlined here depend heavily on fast Fourier transforms and the solution of tridiagonal systems; we first establish operation counts for these procedures.

The experiments reported in Section 7 used a general-purpose Assembler Language FFT package with the following important characteristics:

(a) the number of data points (N) can be any number for a complex transform, and any even number for a real transform;

(b) all trigonometric function values are precalculated and picked out as required during the transform;

(c) the results emerge in the correct order, in contrast to most FFT programs which require somewhat tortuous logic to unscramble the transform (Sweet [24] found that this feature of most FFT programs significantly increased the execution time of an FFT-based Poisson-solver);

(d) factors of 2 are (as far as possible) grouped together into factors of 4.

The mixed-radix complex FFT was modeled on the algorithm given by Singleton [19] with appropriate modifications to eliminate reordering [29] and to pick out trigonometric function values rather than calculating them every time they are required. The additional manipulations required to use the FFT for real transforms are due to Cooley, Lewis, and Welch [6]; see also [22] and [30].

For general even N, factorized in the form

$$N = 2^p 3^q 4^r 5^s m_1^{t_1} m_2^{t_2} \cdots, \tag{2}$$

where $p \geqslant 1$ and m_i represents a general odd factor, the operation count for a real periodic transform is

$$N \left\{1.5p + 2.67q + 2.75r + 4s + 0.5 \sum_i (m_i + 5)\, t_i\right\} \text{ additions,}$$

$$N \left\{p + 2q + 1.5r + 3.2s + 0.5 \sum_i (m_i + 2)\, t_i - 1\right\} \text{ multiplications,}$$

while a real sine transform requires an extra $2.5N$ additions and $0.5N$ multiplications.

In the present analysis, we are restricting N to be a power of 2, and for simplicity we assume that the FFT is used in pure radix-2 mode, i.e., $r = 0$ in Eq. (2), so that our operation counts become the following:

for a periodic transform,

$$1.5 \log_2 N \text{ additions and } (\log_2 N - 1) \text{ multiplications per point;}$$

for a sine transform,

$$(1.5 \log_2 N + 2.5) \text{ additions and } (\log_2 N - 0.5) \text{ multiplications per point.}$$

Although these operation counts are typical of generally available real FFT routines, they can be improved upon even within the radix-2 formulation. Instead of proceeding via an artificial complex series of length $N/2$, it is possible to modify the complex FFT algorithm directly for the real periodic case. Thus the transform algorithm of Hockney [13] requires only $2.5 \log_2 N - 3.5$ operations per point, while Bergland [2] quotes an operation count of $1.5 \log_2 N - 2.5$ additions and $\log_2 N - 3.5$ multiplications per point for his algorithm.

We turn our attention now to the solution of tridiagonal systems. In the present problem, these are almost all of the simple form

$$x_{j-1} + \lambda x_j + x_{j+1} = b_j, \qquad 1 \leqslant j \leqslant M-1 \tag{3}$$

with $x_0 = x_M = 0$, and $|\lambda| \geqslant 2$. We use the following algorithm based on Gaussian elimination and given (for general diagonally dominant tridiagonal systems) by Varga [31, p. 195]:

$$\begin{array}{lll} \omega_1 = \lambda^{-1}; & \omega_j = (\lambda - \omega_{j-1})^{-1}, & 2 \leqslant j \leqslant M-1; \\ g_1 = \omega_1 b_1; & g_j = \omega_j(b_j - g_{j-1}), & 2 \leqslant j \leqslant M-1; \\ x_{M-1} = g_{M-1}; & x_j = g_j - \omega_j x_{j+1}, & M-2 \geqslant j \geqslant 1. \end{array}$$

Provided that the coefficients ω_j have been precalculated and stored, this algorithm requires only two additions and two multiplications per unknown. Other possibilities requiring less coefficient storage but more arithmetic include cyclic reduction [12, 13], methods based on the Toeplitz structure of the tridiagonal matrix [10], and on the factorization of the matrix into two rectangular matrices [8].

An interesting way of halving the number of coefficients while requiring no additional arithmetic arises from the "symmetric Gaussian elimination" or "folding" algorithm recently proposed by Evans and Hatzopoulos [9]; for tridiagonal systems of the form (3), this algorithm takes the simple form (for $M = 2m$):

$$\begin{array}{lll} \omega_1 = \lambda^{-1}; & \omega_j = (\lambda - \omega_{j-1})^{-1}, & 2 \leqslant j \leqslant m-1; \\ g_1 = \omega_1 b_1; & g_j = \omega_j(b_j - g_{j-1}), & 2 \leqslant j \leqslant m-1; \\ g_{M-1} = \omega_1 b_{M-1}; & g_j = \omega_{M-j}(b_j - g_{j+1}), & M-2 \geqslant j \geqslant m+1; \\ & x_m = (\lambda - 2\omega_{m-1})^{-1}(b_m - g_{m-1} - g_{m+1}); & \\ & x_j = g_j - \omega_j x_{j+1}, & m-1 \geqslant j \geqslant 1; \\ & x_j = g_j - \omega_{M-j} x_{j-1}, & m+1 \leqslant j \leqslant M-1. \end{array}$$

A slight modification is required if M is odd.

In certain circumstances, it is possible to save coefficient storage using the observation that if the system (3) is sufficiently diagonally dominant, then ω_j converges rather rapidly to a constant value. For example, with $\lambda = -4$, the sequence converges to within machine precision on a CDC 6600 (48-bit mantissa) at $j = 13$. Unfortunately, for the simple Poisson equation not all the tridiagonal systems encountered are sufficiently diagonally dominant; but for certain Helmholtz equations, storage may be saved in this way.

3. The Basic FFT Method

For clarity we derive briefly the basic FFT method for Poisson's equation with Dirichlet boundary conditions at $i = 0, N$. On each row, we express the solution values (including the prescribed boundary values) as sums of Fourier sine coefficients:

$$x_{i,j} = \sum_{k=1}^{N-1} \hat{x}_{k,j} \sin(ik\pi/N), \qquad 0 \leqslant i \leqslant N, \qquad 0 \leqslant j \leqslant M. \tag{4}$$

Substituting in Eq. (1), we obtain (after some manipulation) $(N - 1)$ tridiagonal systems each of the form:

$$\hat{x}_{k,j-1} + \lambda_k \hat{x}_{k,j} + \hat{x}_{k,j+1} = \hat{b}_{k,j}, \qquad 1 \leqslant k \leqslant N-1, \quad 1 \leqslant j \leqslant M-1, \tag{5}$$

where $\lambda_k = 2\cos(k\pi/N) - 4$ and

$$\hat{b}_{k,j} = (2/N) \sum_{i=1}^{N-1} b_{i,j} \sin(ik\pi/N). \tag{6}$$

The solution procedure thus has three stages. In the first stage, a sine transform is applied on each row of the right-hand side field to implement Eq. (6) and obtain the coefficients $\hat{b}_{k,j}$. Equation (5) is then solved for each value of k. In the third stage, an inverse sine transform is applied on each row to implement Eq. (4) and obtain the solution. In all we have to perform $2(M - 1)$ sine transforms of length N, and solve $(N - 1)$ tridiagonal systems of order $(M - 1)$.

For periodic boundary conditions at $i = 0, N$ we replace the sine transform by the full periodic transform:

$$x_{i,j} = \tfrac{1}{2}\bar{x}_{0,j} + \tfrac{1}{2}(-1)^i \bar{x}_{N/2,j} + \sum_{k=1}^{N/2-1} \{\bar{x}_{k,j}\cos(2ik\pi/N) + \hat{x}_{k,j}\sin(2ik\pi/N)\} \tag{7}$$

and obtain $(N/2 + 1)$ systems of the form

$$\bar{x}_{k,j-1} + \lambda_k \bar{x}_{k,j} + \bar{x}_{k,j+1} = \bar{b}_{k,j}, \qquad 0 \leqslant k \leqslant N/2, \quad 1 \leqslant j \leqslant M-1,$$

where

$$\lambda_k = 2\cos(2k\pi/N) - 4 \tag{8}$$

and

$$\bar{b}_{k,j} = (2/N) \sum_{i=0}^{N-1} b_{i,j} \cos(2ik\pi/N)$$

together with $(N/2 - 1)$ systems of the form

$$\hat{x}_{k,j-1} + \lambda_k \hat{x}_{k,j} + \hat{x}_{k,j+1} = \hat{b}_{k,j}, \qquad 1 \leqslant k \leqslant N/2 - 1, \quad 1 \leqslant j \leqslant M - 1,$$

where λ_k is again given by Eq. (8), and

$$\hat{b}_{k,j} = (2/N) \sum_{i=0}^{N-1} b_{i,j} \sin(2ik\pi/N).$$

In this case we have to perform $2(M - 1)$ periodic transforms of length N, and solve N tridiagonal systems of order $(M - 1)$.

The number of coefficients used in solving tridiagonal systems is $(N - 1)(M - 1)$ or $(N/2 + 1)(M - 1)$ depending on whether the boundary conditions at $i = 0$, N are Dirichlet or periodic. The solution can overwrite the right-hand side field; the only work space required is an area of N locations used by the FFT routine outlined in Section 2.

4. Block-Cyclic Reduction

Details of the block-cyclic reduction method (Buneman's algorithm) are given by Buzbee *et al.* [3] and need not be repeated here. There are two variants: Variant 1 requires two separate arrays, while in Variant 2 the solution can overwrite the original right-hand side field, using no additional work space.

To establish the operation count for Variant 1 with Dirichlet boundary conditions at $i = 0, N$, we first consider the number of tridiagonal systems which have to be solved. There are $(\log_2 N - 1)$ steps in the reduction phase; in the rth step we solve $(N/2^r - 1)$ systems of the form $A^{(r-1)}\mathbf{x} = \mathbf{b}$, where $A^{(r-1)}$ is the product of 2^{r-1} tridiagonal matrices. (See [3] for details). Solving for $\mathbf{x}_{N/2}$ requires $N/2$ tridiagonal solutions. There are then $(\log_2 N - 1)$ steps of the back-solution phase; in the rth step we solve 2^r systems of the form $A^{(s-1)}\mathbf{x} = \mathbf{b}$, where $s = \log_2 N - r$ and $A^{(s-1)}$ is the product of 2^{s-1} tridiagonal matrices. Altogether there are $N(\log_2 N - 1) + 1$ tridiagonal systems to be solved, each of order $(M - 1)$.

Second, we consider the extra additions required to calculate the vectors $\mathbf{p}_i^{(r)}$ and $\mathbf{q}_i^{(r)}$ at each reduction step and $\mathbf{x}_i$ during the back-solution steps, using the notation of [3] but replacing j by i to indicate that we are performing the block-cyclic reduction in the i-direction. A careful count shows that approximately $7N(M - 1)$ extra additions are required.

Turning now to Variant 2, the number of tridiagonal systems to be solved is exactly the same. The vectors $\mathbf{p}_i^{(r)}$ are eliminated; the equation given in [3] for $\mathbf{q}_i^{(r)}$ appears to require $12(M-1)$ extra additions, but this can be reduced to $8(M-1)$ by first defining

$$\bar{\mathbf{q}}_i^{(r-1)} = \mathbf{q}_{i-2h}^{(r-1)} - \mathbf{q}_{i-h}^{(r-2)} + \mathbf{q}_i^{(r-1)} - \mathbf{q}_{i+h}^{(r-2)} + \mathbf{q}_{i+2h}^{(r-1)}$$

and then computing

$$\mathbf{q}_i^{(r)} = \bar{\mathbf{q}}_i^{(r-1)} + (A^{(r-1)})^{-1}[\mathbf{q}_{i-3h}^{(r-2)} + \mathbf{q}_{i+3h}^{(r-2)} - \bar{\mathbf{q}}_i^{(r-1)} - \mathbf{q}_i^{(r-1)}],$$

where $h = 2^{r-2}$, and in the first step this reduces simply to

$$\mathbf{q}_i^{(1)} = \mathbf{b}_{i+1} + \mathbf{b}_{i-1} - 2A^{-1}\mathbf{b}_i\,.$$

Using this more economical form of Variant 2, the arithmetic involved in addition to the solution of tridiagonal systems amounts to approximately $9N(M-1)$ additions and $0.5N(M-1)$ multiplications.

For periodic boundary conditions at $i = 0, N$ the reduction and back-solution phases each have $\log_2 N$ steps; again, the details are given in [3]. In this case there are $N(\log_2 N + 1)$ tridiagonal systems to be solved, each of order $(M-1)$; while the extra work is approximately the same as in the Dirichlet case, for either variant. An alternative (and apparently faster) procedure for the periodic case has recently been developed by Sweet [26].

The number of coefficients used in solving tridiagonal systems is $(N-1)(M-1)$ or $(N+1)(M-1)$, depending on whether the boundary conditions at $i = 0, N$ are Dirichlet or periodic. When using Variant 1, the vectors $\mathbf{p}_i^{(r)}$ and $\mathbf{q}_i^{(r)}$, $r \geqslant 1$, can share an array of dimension $(M-1)N$ which becomes the solution array, so that the original right-hand side field is not destroyed. Alternatively, if the vectors $\mathbf{q}_i^{(r)}$ overwrite the right-hand side, then the auxiliary array containing the vectors $\mathbf{p}_i^{(r)}$ need only be half the size of the main array, so that Variant 1 in fact only requires 50 % more array space than Variant 2.

5. FACR(1) Methods

5.1. *Hockney's* FACR(1, *j*) *algorithm*

Hockney [12] pointed out that half the transforms in the basic FFT method could be eliminated by first performing one step of cyclic reduction in the j-direction, i.e., by eliminating all $x_{i,j}$ with j odd. The resulting equation is

$$\begin{aligned} x_{i,j-2} - x_{i-2,j} + 8x_{i-1,j} - 16x_{i,j} + 8x_{i+1,j} - x_{i+2,j} + x_{i,j+2} \\ = b_{i,j}^{(1)} = b_{i,j-1} - b_{i-1,j} + 4b_{i,j} - b_{i+1,j} + b_{i,j+1}\,. \end{aligned} \tag{9}$$

Again we can write $x_{i,j}$ as in Eq. (4) (this time for even j only) and we obtain (for Dirichlet boundary conditions at $i = 0, N$) tridiagonal systems of the form

$$\hat{x}_{k,j-2} + \lambda_k \hat{x}_{k,j} + \hat{x}_{k,j+2} = \hat{b}_{k,j}^{(1)}, \qquad 1 \leqslant k \leqslant N-1, \qquad j = 2, 4, \ldots, M-2,$$

where

$$\lambda_k = 2 - 4(\cos(k\pi/N) - 2)^2 \tag{10}$$

and

$$\hat{b}^{(1)}_{k,j} = (2/N) \sum_{i=1}^{N-1} b^{(1)}_{i,j} \sin(ik\pi/N). \tag{11}$$

Thus we first obtain $b^{(1)}_{i,j}$ for even j using Eq. (9); then perform the sine transform for even j using Eq. (11). Following the solution of $(N - 1)$ tridiagonal systems of order $(M/2 - 1)$, we perform inverse sine transforms (4) for even j, and finally the solution $x_{i,j}$ for odd j is obtained by solving tridiagonal systems along rows.

For periodic boundary conditions at $i = 0, N$ a completely analogous algorithm can be derived; the only difficulty is that a cyclic tridiagonal system of order N has to be solved for each odd-numbered line. A number of techniques are available, some of which were discussed by Temperton [27]; Algorithm 4 of that paper requires the least computation, namely, $3N$ additions and $2.5N$ multiplications, but as N is here assumed to be a power of 2, cyclic reduction is almost as efficient ($4N$ additions and $2N$ multiplications), and requires little or no coefficient storage.

For Hockney's method, the number of coefficients required for the tridiagonal solutions is $(M/2 - 1)(N - 1)$ or $(M/2 - 1)(N/2 + 1)$, depending on whether the boundary conditions at $i = 0, N$ are Dirichlet or periodic. As in the case of the basic FFT method, the solution can overwrite the right-hand side.

5.2. FACR(1, i)

In comparison with the basic FFT method, Hockney's algorithm halves the number of Fourier transforms required by forming a set of equations involving $x_{i,j}$ for even j only. An alternative strategy is to halve the length of the Fourier transforms by forming an analogous set involving $x_{i,j}$ for even i only. The resulting equations are of the form

$$\begin{aligned} x_{i-2,j} - x_{i,j-2} + 8x_{i,j-1} - 16x_{i,j} + 8x_{i,j+1} - x_{i,j+2} + x_{i+2,j} \\ = b^{(1)}_{i,j} = b_{i-1,j} - b_{i,j-1} + 4b_{i,j} - b_{i,j+1} + b_{i+1,j}\,. \end{aligned} \tag{12}$$

On each line we now have $(N/2 + 1)$ values of $x_{i,j}$ (including the boundary points), and these can be expressed as sums of $(N/2 - 1)$ sine coefficients:

$$x_{2i,j} = \sum_{k=1}^{N/2-1} \hat{x}_{k,j} \sin(2ik\pi/N), \qquad 0 \leqslant i \leqslant N/2, \qquad 0 \leqslant j \leqslant M. \tag{13}$$

Substituting in Eq. (12), we obtain $N/2 - 1$ pentadiagonal systems, each of which can be factorized into the form

$$\begin{aligned} \hat{y}_{k,j-1} + \lambda_{N-k}\hat{y}_{k,j} + \hat{y}_{k,j+1} &= \hat{b}^{(1)}_{k,j}\,, \\ \hat{x}_{k,j-1} + \lambda_k \hat{x}_{k,j} + \hat{x}_{k,j+1} &= \hat{y}_{k,j}\,, \end{aligned} \tag{14}$$

where $\lambda_k = 2\cos(k\pi/N) - 4$, and

$$\hat{b}_{k,j}^{(1)} = (4/N) \sum_{i=1}^{N/2-1} b_{2i,j}^{(1)} \sin(2ik\pi/N). \tag{15}$$

The algorithm thus proceeds as follows: the right-hand side of Eq. (12) is calculated at all grid points with i even. Equation (15) is then implemented; this involves $(M-1)$ sine transforms each of length $N/2$. Next the $(N/2-1)$ pentadiagonal systems, Eq. (14), are solved. $(M-1)$ inverse sine transforms each of length $N/2$ are then performed to obtain $x_{2i,j}$ and finally $x_{i,j}$ for i odd is obtained by solving simple tridiagonal systems in the j-direction.

For periodic boundary conditions at $i = 0, N$ a similar algorithm can be derived.

The number of coefficients required for the tridiagonal systems is the same as for the basic FFT method. Again, the solution can overwrite the right-hand side, and the only work space required is for FFT's.

5.3. FACR$(1, i+j)$: *Diagonal Cyclic Reduction*

Yet another way of combining the FFT method with one step of cyclic reduction is included here, not because it leads to a more efficient algorithm but because it has some interesting aspects, and because it has applications in the direct solution of Poisson's equation over irregular regions [28]. In Sections 5.1 and 5.2 we halved the number of unknowns by eliminating $x_{i,j}$ either for i odd or for j odd. A third alter-alternative is to eliminate $x_{i,j}$ for $(i+j)$ odd, resulting in the following equation:

$$\begin{aligned} x_{i,j+2} + x_{i,j-2} + x_{i-2,j} + x_{i+2,j} + 2(x_{i-1,j-1} + x_{i-1,j-1} + x_{i+1,j+1} + x_{i+1,j-1}) - 12x_{i,j} \\ = b_{i,j}^{(1)} = 4b_{i,j} - b_{i,j+1} + b_{i,j-1} + b_{i+1,j} + b_{i-1,j} \end{aligned} \tag{16}$$

with appropriate modifications near the boundaries. The retained points lie on alternate diagonals. For even j, we introduce the same sine summation as for FACR$(1, i)$:

$$x_{2i,j} = \sum_{k=1}^{N/2-1} \hat{x}_{k,j} \sin(2ik\pi/N), \qquad 0 \leqslant i \leqslant N/2, \qquad j \text{ even}, \tag{17}$$

while for odd j, we introduce the following modified summation:

$$x_{2i-1,j} = \sum_{k=1}^{N/2} \hat{x}_{k,j} \sin((2i-1)k\pi/N), \qquad 0 \leqslant i \leqslant N/2-1, \qquad j \text{ odd}. \tag{18}$$

Note that this series has an extra term $(k = N/2)$. Introducing (17) and (18) into Eq. (16) and performing the usual manipulations, we obtain $(N/2-1)$ pentadiagonal systems of order $(M-1)$, which factorize into the form:

$$\begin{aligned} \hat{y}_{k,j-1} + \lambda_k' \hat{y}_{k,j} + \hat{y}_{k,j+1} &= \hat{b}_{k,j}^{(1)}, \\ \hat{x}_{k,j-1} + \lambda_k'' \hat{x}_{k,j} + \hat{x}_{k,j+1} &= \hat{y}_{k,j}, \end{aligned} \tag{19}$$

where

$$\lambda'_k = 2\cos(k\pi/2N) - 4, \qquad \lambda''_k = 2\cos(k\pi/2N) + 4,$$
$$\hat{b}^{(1)}_{k,j} = (4/N) \sum_{i=1}^{N/2-1} b^{(1)}_{2i,j} \sin(2ik\pi/N) \tag{20}$$

for j even, and

$$\hat{b}^{(1)}_{k,j} = (4/N) \sum_{i=1}^{N/2} b^{(1)}_{2i-1,j} \sin((2i-1)k\pi/N) \tag{21}$$

for j odd.

In addition, for $k = N/2$ we obtain a tridiagonal system of order $M/2$ of a rather special form:

$$\begin{aligned} -15\hat{x}_{k,1} + \hat{x}_{k,3} &= \hat{b}^{(1)}_{k,1}, \\ \hat{x}_{k,j-2} - 14\hat{x}_{k,j} + \hat{x}_{k,j+2} &= \hat{b}^{(1)}_{k,j}, \qquad j = 3, 5, \ldots, M-3, \\ \hat{x}_{k,M-3} - 15\hat{x}_{k,M-1} &= \hat{b}^{(1)}_{k,M-1}, \end{aligned} \tag{22}$$

where

$$\hat{b}^{(1)}_{k,j} = -(4/N) \sum_{i=1}^{N/2} (-1)^i \, b^{(1)}_{2i-1,j}. \tag{23}$$

The algorithm is thus implemented as follows: first, $b^{(1)}_{i,j}$ is determined, using Eq. (16), at all points with $(i+j)$ even. The Fourier sine coefficients $\hat{b}^{(1)}_{k,j}$ are then computed using a simple sine transform, Eq. (20), on even lines and a shifted sine transform, Eqs. (21) and (23), on odd lines. The pentadiagonal systems (19) and the tridiagonal system (22) are solved for the sine coefficients $\hat{x}_{k,j}$. The solution at points with $(i+j)$ even is then obtained using a simple inverse sine transform (17) on even lines and the shifted inverse sine transform (18) on odd lines.

Finally the solution at points with $(i+j)$ odd is determined from the scalar equation

$$x_{i,j} = \tfrac{1}{4}(x_{i+1,j} + x_{i-1,j} + x_{i,j+1} + x_{i,j-1} - b_{i,j})$$

since all the quantities on the right-hand side are by now known.

The manipulations required to convert a real shifted sine transform into a real periodic transform are given by Swarztrauber [22].

The algorithm can be modified to eliminate $x_{i,j}$ for $(i+j)$ even; the roles of odd and even lines are then interchanged. An analogous algorithm can also be derived for periodic boundary conditions at $i = 0, N$.

The process of diagonal cyclic reduction is similar to the first step of the "total reduction" method of Schröder and Trottenberg [16].

6. Summary of Operation Counts

Using the results of Section 2, operation counts have been determined for all the algorithms outlined in Sections 3–5. These are summarized in Table I.

TABLE I

Summary of Operation Counts (per Point)

Boundary conditions at $i = 0, N$:	Dirichlet		Periodic	
	adds	mults	adds	mults
Buneman (Variant 1)	$2 \log_2 N + 5$	$2 \log_2 N - 2$	$2 \log_2 N + 9$	$2 \log_2 N + 2$
Buneman (Variant 2)	$2 \log_2 N + 7$	$2 \log_2 N - 1.5$	$2 \log_2 N + 11$	$2 \log_2 N + 2.5$
Basic FFT	$3 \log_2 N + 7$	$2 \log_2 N + 1$	$3 \log_2 N + 2$	$2 \log_2 N + 1$
Hockney's FACR$(1, j)$	$1.5 \log_2 N + 7.5$	$\log_2 N + 2$	$1.5 \log_2 N + 6$	$\log_2 N + 1.5$
FACR$(1, i)$	$1.5 \log_2 N + 7$	$\log_2 N + 2$	$1.5 \log_2 N + 4.5$	$\log_2 N + 0.5$
FACR$(1, i + j)$	$1.5 \log_2 N + 7.25$	$\log_2 N + 1.75$	$1.5 \log_2 N + 4.75$	$\log_2 N + 1.5$

For Dirichlet boundary conditions at $i = 0, N$ Buneman's algorithm has a lower operation count than the basic FFT method, while for periodic boundary conditions at $i = 0, N$ the reverse is true, except for very large values of N. (Note here the misleading effect of comparing the "asymptotic" operation counts of $4MN \log_2 N$ for Buneman's algorithm versus $5MN \log_2 N$ for the basic FFT method, regardless of boundary conditions.)

Combining the FFT method with one step of cyclic reduction to halve either the length or the number of the transforms, we obtain algorithms which require fewer operations than either Buneman's algorithm or the basic FFT method, except for Dirichlet boundary conditions on very small grids ($N \leqslant 16$) in which case the Buneman and FACR(1) algorithms have similar operation counts.

Replacing the radix-2 FFT by a radix-4 version yields a saving of approximately $0.25 \log_2 N$ additions and $0.5 \log_2 N$ multiplications per point for the basic FFT algorithm, under Dirichlet or periodic boundary conditions. For the FACR(1) algorithms, the saving is halved.

If a real FFT of the type developed by Bergland [2] is used instead of the half-length complex transform coupled with a preprocessing or postprocessing step [6], then a saving of five additions and four multiplications per point is realized for the basic FFT algorithm, or half this saving for the FACR(1) algorithms.

It is worth reiterating at this point that we have been solving all the tridiagonal systems by Gaussian elimination, using precalculated coefficients. If this is ruled out by lack of space, cyclic reduction may be used instead. To solve a tridiagonal system of order M by cyclic reduction requires approximately $4M$ additions and $2M$ multiof tridiagonal systems to be solved in Buneman's algorithm is approximately $N(\log_2 N - 1)$ for Dirichlet boundary conditions at $i = 0, N$, or $N(\log_2 N + 1)$ for periodic boundary conditions at $i = 0, N$; while in the FFT-based algorithms the number is approximately N, or at most $3N/2$. The consequences of changing from Gaussian elimination to cyclic reduction for simple tridiagonal systems are thus as follows: for either Variant of Buneman's algorithm, an extra $(2 \log_2 N - 1)$ (Dirichlet) or $(2 \log_2 N + 2)$ (periodic) additions per point; for basic FFT or FACR$(1, i+j)$, two extra additions per point; for Hockney's FACR$(1, j)$ algorithm, an extra two (Dirichlet) or one (periodic) additions per point; for FACR$(1, i)$, three extra additions per point. In this situation, the use of FFT-based methods appears even more advantageous.

7. Numerical Results

In the preceding sections, we have considered a number of algorithms in terms of operation counts; here we turn to their actual implementation on a computer. Now although it is clear that direct methods for the solution of Poisson's equation are much more efficient than simple iterative methods such as successive overrelaxation, they are considerably harder to program, and for their superiority to be fully realized it is important that they be efficiently coded. The author's personal preference is for the use of a low-level language, and the timings reported here relate to programs written in IBM Assembler Language. This preference was reinforced by the following experiment.

TABLE II

Execution Times (in msec on IBM 360/195) for 32×32 Dirichlet Problem by Buneman's Algorithm (Variant 1)

Language	Compiler	2-d indexing	1-d indexing
Fortran	*G*	58.5	39.6
Fortran	*H*	14.9	10.5
Fortran	*X*	14.2	9.4
Assembler	—	—	5.3

Two Fortran subroutines were written to implement Buneman's algorithm (Variant 1) for Poisson's equation on a rectangle with Dirichlet boundary conditions. One treated all the arrays as two-dimensional (i.e., doubly-subscripted), while the second treated them as one-dimensional (singly-subscripted). The two subroutines were then each compiled at levels *G*, *H*, and *X*. The Fortran style was intended to be as helpful as possible to the compiler, and as efficient as possible at run time (e.g., no branches to subroutines, or from one section of the program to another, apart from simple loops). The six resulting programs, together with a corresponding Assembler

Language subroutine, were then timed on the Dirichlet problem with $N = M = 32$. The results are given in Table II. The conclusions are clear; even the best Fortran subroutine took 75% longer than the Assembler version. If a high-level language must be employed, the indexing should be one dimensional, and an optimizing compiler should be used.

Hockney [14] has also compared several Poisson–solver routines on an IBM 360/195, both in Assembler Language and in Fortran (using various compilers), and obtained rather similar results.

Although we are assuming no storage limitations in this study, it is noteworthy that the length of the Assembler program was only 236 words, compared with 700–1000 words for the Fortran programs.

Assembler Language versions of four algorithms described in Sections 3–5 (the basic FFT algorithm, Buneman's algorithm Variant 1, Hockney's FACR$(1, j)$ algorithm and the FACR$(1, i)$ algorithm) were timed on $N \times N$ problems, with N ranging from 8 to 128, and with both Dirichlet and periodic boundary conditions at $i = 0, N$. The FFT package (Section 2) was used in its "radix 4 + 2" mode, thus improving slightly on the operation counts given in Table I. Also, the cyclic tridiagonal systems arising in Hockney's algorithm with periodic boundary conditions at $i = 0, N$ were solved using Algorithm 4 of [27]. The results are shown in Table III (Dirichlet boundary conditions at $i = 0, N$) and Table IV (periodic boundary conditions at $i = 0, N$).

TABLE III

Execution Times (sec) for Dirichlet Boundary Conditions at $i = 0, N$ (IBM 360/195)

Method	$N = 8$	$N = 16$	$N = 32$	$N = 64$	$N = 128$
Buneman	2.33×10^{-4}	1.12×10^{-3}	5.33×10^{-3}	2.54×10^{-2}	1.58×10^{-1}
Basic FFT	5.60×10^{-4}	2.24×10^{-3}	8.81×10^{-3}	3.94×10^{-2}	1.64×10^{-1}
FACR$(1, j)$	3.55×10^{-4}	1.42×10^{-3}	5.61×10^{-3}	2.45×10^{-2}	1.02×10^{-1}
FACR$(1, i)$	5.33×10^{-4}	1.69×10^{-3}	6.57×10^{-3}	2.61×10^{-2}	1.14×10^{-1}

TABLE IV

Execution Times (sec) for Periodic Boundary Conditions at $i = 0, N$ (IBM 360/195)

Method	$N = 8$	$N = 16$	$N = 32$	$N = 64$	$N = 128$
Buneman	3.89×10^{-4}	2.28×10^{-3}	1.10×10^{-2}	5.24×10^{-2}	4.44×10^{-1}
Basic FFT	4.79×10^{-4}	2.10×10^{-3}	7.68×10^{-3}	3.35×10^{-2}	1.55×10^{-1}
FACR$(1, j)$	3.26×10^{-4}	1.40×10^{-3}	5.31×10^{-3}	2.23×10^{-2}	1.01×10^{-1}
FACR$(1, i)$	5.07×10^{-4}	1.70×10^{-3}	7.21×10^{-3}	2.71×10^{-2}	1.36×10^{-1}

Several points are worthy of note. For Dirichlet boundary conditions, the fastest algorithm was Buneman's for $N \leqslant 32$, and Hockney's for $N \geqslant 64$. For periodic boundary conditions, the fastest was Hockney's throughout the range. The results are in line with the comparisons given in Section 6, though for Dirichlet boundary conditions the value of N at which Hockney's algorithm becomes faster than Buneman's is somewhat larger than predicted, presumably because of the overheads incurred in repeatedly calling a general-purpose FFT subroutine. FACR(1, i) is disappointing in comparison with Hockney's algorithm, especially for small values of N; again the reason lies in the extra overheads for the FFT. In terms of total execution time it is clearly faster to do $M/2$ transforms of length N rather than M transforms of length $N/2$, though the latter has a slightly lower floating-point operation count. This suggests that all the FFT-based algorithms could be made more efficient by performing the transforms in parallel rather than one at a time.

Note that, in agreement with the timings reported by Hockney [13], the execution time for the FFT-based algorithms is roughly proportional to N^2, the total number of points. For $N = 128$ with periodic boundary conditions at $i = 0, N$ the time for Buneman's algorithm is anomalously large; the solution of each tridiagonal system requires values spaced at intervals of 128 words, which causes memory bank conflicts on the 360/195. These could be eliminated either by adding a dummy value to each line, or by performing the block-cyclic reduction in the more conventional j-direction.

We consider now the question of the accuracy of the various algorithms. For each value of N and each set of boundary conditions, a random number generator was used to set up ten $N \times N$ "true" solutions with values in the interval $[-1, +1]$, from which corresponding right-hand sides were computed, using temporary double precision to eliminate round-off error from this stage of the procedure. (As shown in the Appendix, this is by no means a trivial refinement). Each algorithm was then used to recover the solution from the right-hand side, and the computed solution was compared with the original field to determine the maximum absolute point error. The maximum errors, meaned over ten solutions in each case, are shown in Tables V and VI.

Although the errors quoted in Tables V and VI fairly represent the error behavior of the Poisson-solver programs described here, they should not be taken as the best

TABLE V

Mean Maximum Errors for Dirichlet Boundary Conditions at $i = 0, N$

Method	$N = 8$	$N = 16$	$N = 32$	$N = 64$	$N = 128$
Buneman	4.48×10^{-6}	1.04×10^{-5}	2.16×10^{-5}	5.50×10^{-5}	1.04×10^{-4}
Basic FFT	5.05×10^{-6}	8.63×10^{-6}	2.91×10^{-5}	1.38×10^{-4}	4.59×10^{-4}
FACR(1, j)	4.78×10^{-6}	1.57×10^{-5}	4.79×10^{-5}	3.02×10^{-4}	9.57×10^{-5}
FACR(1, i)	7.06×10^{-6}	8.48×10^{-6}	2.07×10^{-5}	8.15×10^{-5}	3.00×10^{-4}

TABLE VI

Mean Maximum Errors for Periodic Boundary Conditions at $i = 0, N$

Method	$N = 8$	$N = 16$	$N = 32$	$N = 64$	$N = 128$
Buneman	7.63×10^{-6}	1.75×10^{-5}	3.60×10^{-5}	8.41×10^{-5}	1.85×10^{-4}
Basic FFT	4.48×10^{-6}	8.58×10^{-6}	1.55×10^{-5}	3.55×10^{-5}	7.34×10^{-5}
FACR$(1, j)$	5.13×10^{-6}	9.47×10^{-6}	2.01×10^{-5}	3.11×10^{-5}	8.72×10^{-5}
FACR$(1, i)$	7.25×10^{-6}	1.16×10^{-6}	2.27×10^{-5}	5.14×10^{-5}	1.12×10^{-4}

that can be achieved. As shown in [30], a revised implementation of the sine transform reduces the error of the FFT-based algorithms (in the case of Dirichlet boundary conditions at $i = 0, N$), while the error behavior of Buneman's algorithm can be improved by solving systems of the form

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

(A tridiagonal) using the roots λ_i in a different order. By these means the round-off errors in the case $N = 128$ are typically reduced by an order of magnitude.

8. Extensions

In Section 6, we noted the consequences of limiting the available core storage so that tridiagonal systems could no longer be solved by Gaussian elimination using precomputed coefficients; it was shown that under these circumstances the advantages of FFT-based methods over block-cyclic reduction became more pronounced. We now consider the effects of relaxing some of the constraints on the problem itself which were laid down in Section 1.

If the grid lengths in the i and j directions are unequal, the algorithms require only slight modification; the number of extra multiplications per point ranges from zero to two, depending on the algorithm used. It is generally more efficient to scale the problem to give unit grid length in the j-direction, so that the tridiagonal systems to be solved in that direction retain the form of Eq. (3), with 1's on the subdiagonals and superdiagonals of the corresponding tridiagonal matrices.

The operation counts for Neumann boundary conditions at $i = 0, N$ or at $j = 0, M$ are the same as for Dirichlet boundary conditions. Periodic boundary conditions at $j = 0, M$ require the solution of cyclic tridiagonal systems, and again it is the FFT-based algorithms, with fewer such systems to solve, which require less extra computation.

The most general form of elliptic equation which can readily be handled by the techniques developed in this paper is

$$\nabla^2\phi + \beta_j \delta_y \phi - \kappa_j \phi = b_{i,j}, \tag{24}$$

where $\bar{\nabla}^2$ and δ_y are the finite-difference analogs of ∇^2 and $\partial/\partial y$, and β and κ are functions of j only. Hockney's algorithm and FACR(1, $i+j$) become less straightforward, but for the remaining algorithms the only changes are to the tridiagonal systems in the j-direction. These now have the more general form

$$\mu_j x_{j-1} + \lambda_j x_j + \nu_j x_{j+1} = b_j, \qquad 1 \leqslant j \leqslant M-1$$

with $x_0 = x_M = 0$. It is possible (though not very easy) to solve such a system by cyclic reduction [11]; more suitable is Gaussian elimination using precomputed coefficients. This requires two additions, three multiplications, and two precomputed coefficients per unknown. With the increased operation count for the solution of each tridiagonal system, it is again the FFT-based methods, involving only about N such systems, which score over the block-cyclic reduction method. For the basic FFT method, the change from the simple Poisson equation (1) to the more general equation (24) increases the operation count by only one multiplication per point, though it doubles the number of precomputed coefficients required for Gaussian elimination.

Swarztrauber [21] has shown that block-cyclic reduction can be extended to equations even more general than Eq. (24), though the resulting algorithm is very complicated. Some extension is also possible for the FFT methods; see for example [15].

Finally in this section we relax the restriction that N be a power of 2. Sweet [25] has extended Buneman's algorithm to the case $N = 2^p3^q5^r \cdots$; the operation count rises rather rapidly as larger factors of N are included. A more flexible, simpler, and more efficient approach is to use FFT methods with a mixed-radix FFT (see Section 2). For particularly awkward values of N, e.g., large prime numbers, the rectangle may be embedded in a larger one with a more convenient value of N, using a capacity-matrix technique together with "fast embedding" [28].

However, for general N the most efficient method may not after all be FFT-based. A new block-cyclic reduction algorithm has recently been proposed [18, 26] which has an operation count approximately proportional to $MN \log_2 N$ for arbitrary M, N.

Throughout this paper it has been tacitly assumed that the most important factor in comparing algorithms is simply the operation count. Such is not necessarily the case when the algorithms are implemented on some recent computers. On a parallel machine, we need to consider the number of processors which can be simultaneously active, while on a vector computer such as the CRAY-1 we need to maximize the "vector length". In either case the important factor is the degree of parallelism inherent in the algorithm.

The basic FFT method for solving the discrete Poisson equation is a highly parallel algorithm; at each stage we are either performing a set of independent Fourier transforms, or solving a set of independent tridiagonal systems. For block-cyclic reduction the situation is not so favorable; at each step of the reduction process the number of independent systems to be solved is halved. It thus seems likely that on a parallel or vector machine, the FFT method will be substantially faster than block-cyclic reduction.

As a preliminary example of what can be achieved, a Fortran program implementing the basic FFT method solved the 128×128 Dirichlet problem on a CRAY-1 in 16 msec, at a rate of 39×10^6 floating point operations per second. Vectorization was achieved in the FFT phases by treating all rows simultaneously, and in the tridiagonal solution phase by treating all columns simultaneously.

9. Conclusions

Some of the most important conclusions of this study may be summarized as follows:

(1) In calculating operation counts for direct methods, it is important to include both additions and multiplications, and to include terms of order MN as well as those of order $MN \log_2 N$.

(2) Under the assumptions of Section 1, Buneman's algorithm (block-cyclic reduction) is faster than the basic FFT method for the Dirichlet problem, but the introduction of periodic boundary conditions in one direction tends to reverse the position.

(3) Buneman's algorithm (especially Variant 2) can be implemented with fewer operations than quoted, for example, by Hockney [13].

(4) FACR(1) algorithms, combining the FFT method with one preliminary level of cyclic reduction, are faster than eithei of the basic methods except for the Dirichlet problem on small grids, for which Buneman's algorithm remains the fastest. A forthcoming report [30] will examine FACR(l) algorithms, in which the FFT method is combined with l preliminary levels of cyclic reduction to give a further increase in speed.

(5) The cyclic reduction step for the FACR(1) algorithm can be incorporated in at least three different ways.

(6) Programming details are important; in particular, Fortran routines should treat the arrays as one dimensional rather than two dimensional, even when a sophisticated optimizing compiler is available.

(7) More complicated problems, and storage restrictions, tend to favor the FFT method since it requires fewer tridiagonal solutions than Buneman's algorithm.

APPENDIX: On Round-off Errors, Random Numbers, and Poisson-Solvers

In Section 7 we examined the accuracy of several algorithms with respect to round-off error using the following simple experimental procedure: a "true" solution $\mathbf{x}$ in the range $[-1, +1]$ was constructed using a random-number generator; a corresponding right-hand side $\mathbf{b} = A\mathbf{x}$ was then computed and input to the Poisson-solver. The maximum absolute difference (averaged over a number of trials) between the computed solution $\mathbf{x}'$ and the true solution $\mathbf{x}$ was taken as a measure of the round-off error of the Poisson-solver.

We demonstrate here the importance of using double precision to calculate $\mathbf{b}$ from $\mathbf{x}$, in order to confine round-off error to the Poisson-solver itself. Results are presented here for the FACR(l) routine PSOLVE described in [30] with $l = \log_2 N$ (i.e., the solution was obtained by Buneman's block-cyclic reduction algorithm), and summarized in Table VII.

TABLE VII

Mean Maximum Error for the $N \times N$ Dirichlet Problem

Precision of computations $\mathbf{b}$	$A^{-1}\mathbf{b}$	$N = 64$	$N = 128$
Single	Single	4.57×10^{-5}	1.64×10^{-4}
Double	Single	1.37×10^{-5}	2.58×10^{-5}
Single	Double	3.89×10^{-5}	1.55×10^{-4}

In the first series of experiments, all computations were performed in single precision, corresponding to the first row of Table VII. In particular it was noted that the errors grew like N^2. Recognizing that the computation of $\mathbf{b}$ was itself a source of round-off error, the experiments were repeated using double precision for this part of the procedure. As shown in the second row of Table VII, this resulted in a dramatic decrease in round-off error, especially for large grids; the errors now appeared to grow linearly with N.

Some insight into this result is provided by the following argument. By introducing round-off error into the computation of the right-hand side $\mathbf{b}$, we actually solve a slightly perturbed problem $A\mathbf{x}' = \mathbf{b} + \boldsymbol{\delta}$. Even if we had a "perfect" Poisson-solver which introduced no further round-off errors, the solution obtained would be apparently in error by $A^{-1}\boldsymbol{\delta}$. This situation can be realized by computing $\mathbf{b}$ in single precision as before, but then using a double-precision version of the Poisson-solver. The experiments were repeated using this version of the procedure, and yielded the results in the third row of Table VII; the errors are almost as large as those in the first row where a single-precision Poisson-solver was used.

Thus the errors in the first row of Table VII are dominated by the effect of round-off errors incurred in the computation of $\mathbf{b}$; those in the second row represent the round-off error committed by the Poisson-solver itself; while those in the third row represent the maximum change in the true solution caused by round-off errors in the computation of $\mathbf{b}$, i.e., the maximum difference between the true solutions $\mathbf{x} = A^{-1}\mathbf{b}$ and $\mathbf{x}' = A^{-1}(\mathbf{b} + \boldsymbol{\delta})$. This is now seen to be considerably larger than the error committed by the Poisson-solver itself.

When the above experiments were repeated on a CDC 6600, quite different results result, quoted in [17], that the errors appeared to grow linearly with N on a CDC 6600, but like N^2 on an IBM 360/195.) The explanation lay in the random number generator used to set up $\mathbf{x}$ on the CDC machine. The numbers generated were far from random in one important respect—the last 16 bits of their binary representation were all zeros, so that the calculation of $\mathbf{b}$ (involving only a few additions) could indeed be performed without round-off error. Although the random-number generator used in these experiments was written by the author, the CDC-supplied routine RANF turns out to have similar characteristics.

Acknowledgment

Some of this work was completed while the author was at the U.K. Meteorological Office, Bracknell.

References

1. R. E. Bank and D. J. Rose, Marching algorithms for elliptic boundary value problems. I: The constant coefficient case, *SIAM J. Numer. Anal.* **14** (1977), 792–829.
2. G. D. Bergland, A Fast Fourier Transform algorithm for real-valued series, *Comm. ACM* **11** (1968), 703–710.
3. B. L. Buzbee, G. H. Golub, and C. W. Nielson, On direct methods for solving Poisson's equations, *SIAM J. Numer. Anal.* **7** (1970), 627–656.
4. B. L. Buzbee, F. W. Dorr, J. A. George, and G. H. Golub, The direct solution of the discrete Poisson equation on irregular regions, *SIAM J. Numer. Anal.* **8** (1971), 722–736.
5. B. L. Buzbee and F. W. Dorr, The direct solution of the biharmonic equation on rectangular regions and the Poisson equation on irregular regions, *SIAM J. Numer. Anal.* **11** (1974), 753–763.
6. J. W. Cooley, P. A. W. Lewis, and P. D. Welch, The Fast Fourier Transform algorithm: programming considerations in the calculation of sine, cosine and Laplace transforms, *J. Sound Vib.* **12** (1970), 315–337.
7. F. W. Dorr, The direct solution of the discrete Poisson equation in $O(N^2)$ operations, *SIAM Rev.* **17** (1975), 412–415.
8. D. J. Evans, An algorithm for the solution of certain tridiagonal systems of linear equations, *Comput. J.* **15** (1972), 356–359.
9. D. J. Evans and M. Hatzopoulos, The solution of certain banded systems of linear equations using the folding algorithm, *Comput. J.* **19** (1976), 184–187.
10. D. Fischer, G. H. Golub, O. Hald, C. Leiva, and O. Widlund, On Fourier–Toeplitz methods for separable elliptic problems, *Math. Comp.* **28** (1974), 348–368.
11. D. Heller, Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems, *SIAM J. Numer. Anal.* **13** (1976), 484–495.
12. R. W. Hockney, A fast direct solution of Poisson's equation using Fourier analysis, *J. ACM* **12** (1965), 95–113.
13. 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, 1969.
14. R. W. Hockney, Computers, compilers and Poisson-solvers, *in* "Computers, Fast Elliptic Solvers and Applications" (U. Schumann, Ed.), Advance Publications, London, 1978.
15. R. C. LeBail, Use of Fast Fourier Transforms for solving partial differential equations in physics, *J. Computational Phys.* **9** (1972), 440–465.

16. J. Schröder and U. Trottenberg, Reduktionsverfahren für Differenzengleichungen bei Randwertaufgaben: I, *Numer. Math.* **22** (1973), 37–68.
17. 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.
18. U. Schumann and R. A. Sweet, A direct method for the solution of Poisson's equation with Neumann boundary conditions on a staggered grid of arbitrary size, *J. Computational Phys.* **20** (1976), 171–182.
19. R. C. Singleton, An algorithm for computing the mixed-radix Fast Fourier Transform, *IEEE Trans. Audio Electroacoustics* **17** (1969), 93–103.
20. P. N. Swarztrauber, The direct solution of the discrete Poisson equation on the surface of a sphere, *J. Computational Phys.* **15** (1974), 46–54.
21. P. N. Swarztrauber, A direct method for the discrete solution of separable elliptic equations, *SIAM J. Numer. Anal.* **11** (1974), 1136–1150.
22. P. N. Swarztrauber, The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle, *SIAM Rev.* **19** (1977), 490–501.
23. P. N. Swarztrauber and R. A. Sweet, The direct solution of the discrete Poisson equation on a disk, *SIAM J. Numer. Anal.* **10** (1973), 900–907.
24. R. A. Sweet, Direct methods for the solution of Poisson's equation on a staggered grid, *J. Computational Phys.* **12** (1973), 422–428.
25. R. A. Sweet, A generalized cyclic reduction algorithm, *SIAM J. Numer. Anal.* **11** (1974), 506–520.
26. R. A. Sweet, A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension, *SIAM J. Numer. Anal.* **14** (1977), 706–720.
27. C. Temperton, Algorithms for the solution of cyclic tridiagonal systems, *J. Computational Phys.* **19** (1975), 317–323.
28. C. Temperton, A fast Poisson-solver for an octagonal domain, *in* "Computers, Fast Elliptic Solvers and Applications" (U. Schumann, Ed.), Advance Publications, London, 1978.
29. C. Temperton, Mixed-radix Fast Fourier Transforms without reordering, ECMWF Technical Report No. 3.
30. C. Temperton, On the FACR(l) algorithm for the discrete Poisson equation, *J. Computational Phys.*, to appear. (Preliminary version available as ECMWF Research Department Internal Report No. 14).
31. R. S. Varga, "Matrix Iterative Analysis," Prentice-Hall, Englewood Cliffs, N.J., 1962.