JOURNAL OF COMPUTATIONAL PHYSICS 20, 171–182 (1976)

A Direct Method for the Solution of Poisson's Equation with Neumann Boundary Conditions on a Staggered Grid of Arbitrary Size

U. SCHUMANN

Institut für Reaktorentwicklung, Kernforschungszentrum Karlsruhe
75 Karlsruhe, Postfach 3640, Federal Republic of Germany

AND

ROLAND A. SWEET

Natural and Physical Science Division, University of Colorado at Denver, Denver, Colorado 80202, and The National Center for Atmospheric Research, Boulder, Colorado 80303, U.S.A.*

Received June 11, 1975

A method based on cyclic reduction is described for the solution of the discrete Poisson equation on a rectangular two-dimensional staggered grid with an arbitrary number of grid points in each direction. Neumann boundary conditions are assumed in one direction and any boundary condition may be used in the other direction. The coefficients of the equation can be functions of the latter direction so that, e.g., non-equidistant grid spacings or non-Cartesian coordinates can be used. Poisson's equation with these boundary conditions describes, e.g., the pressure field of an incompressible fluid flow within rigid boundaries. Numerical results are reported for a FORTRAN subroutine using the method. For an $M \times N$ grid the operation count is proportional to $MN \log_2 N$, and about MN storage locations are required.

1. INTRODUCTION

The pressure field in an incompressible fluid flow is described by Poisson's equation with Neumann boundary conditions. This equation is a consequence of the Navier–Stokes equations and the equation of continuity. For three-dimensional problems the finite difference solution of the Navier–Stokes equations is

* The National Center for Atmospheric Research is sponsored by the National Science Foundation.

easiest if a staggered grid is used [1]. This paper describes a fast direct method to solve the finite difference form of Poisson's equation

$$-(a(x)\,u_{xx} + b(x)\,u_x + c(x)\,u) - u_{yy} = f(x, y) \tag{1a}$$

on a rectangular $R = \{(x, y)\colon 0 < x < X, 0 < y < Y\}$ with Neumann boundary conditions in the y-direction

$$\partial u/\partial y = 0 \qquad \text{on} \quad \{(x, y)\colon 0 < x < X, y = 0, Y\}, \tag{1b}$$

and any boundary conditions in the x-direction (including Neumann, Dirichlet, or periodic conditions) yielding coefficients which are independent of y. We define the staggered grid

$$\begin{aligned} R_h = \{(x_i, y_j)\colon\ & x_i = (i - \tfrac{1}{2})\,\Delta x,\ i = 0, 1, 2, \ldots, M + 1, \\ & y_j = (j - \tfrac{1}{2})\,\Delta y,\ j = 0, 1, 2, \ldots, N + 1\}, \end{aligned}$$

where M, Δx, N, and Δy are chosen so that $M\,\Delta x = X$ and $N\,\Delta y = Y$. We require $N \geqslant 2$ and $M \geqslant 2$. It is not necessary that the x_i are spaced equidistantly, however.

A second-order finite difference approximation to (1a) on R_h and the replacement of (1b) (e.g., for $y = 0$ by $[u(x_i, y_0) - u(x_i, y_1)]/\Delta y = 0$) result in the following linear system for v_{ij}, which is the approximation to $u(x_i, y_j)$:

$$\begin{bmatrix} A - I & -I & & & \\ -I & A & -I & & \\ & \cdot & \cdot & \cdot & \\ & & -I & A & -I \\ & & & -I & A - I \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N-1} \\ v_N \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix}, \tag{2}$$

where A is an $M \times M$ matrix of the following form:

$$A = \begin{bmatrix} \beta_1 & \gamma_1 & & & & \alpha_1 \\ \alpha_2 & \beta_2 & \gamma_2 & 0 & & \\ & \cdot & \cdot & \cdot & & \\ & 0 & \alpha_{M-1} & \beta_{M-1} & \gamma_{M-1} & \\ \gamma_M & & & \alpha_M & \beta_M & \end{bmatrix}, \qquad v_j = \begin{bmatrix} v_{1j} \\ v_{2j} \\ \vdots \\ v_{Mj} \end{bmatrix}, \qquad f_j = \begin{bmatrix} f_{1j} \\ f_{2j} \\ \vdots \\ f_{Mj} \end{bmatrix}, \tag{3}$$

and

$$f_{ij} = (\Delta y)^2 f(x_i, y_j).$$

(The entries f_{1j} and f_{Mj} will require modification depending on the boundary conditions on the x-boundaries). For Neumann or Dirichlet boundary conditions

in the x-direction, α_1 and γ_M are zero. They are nonzero for periodic boundary conditions.

A similar problem has been solved by Sweet [2] basically using the Buneman algorithm [3]. The present algorithm is a generalization in the following respects:

(a) There is *no* restriction on the number of intervals N and M (except $N \geqslant 2$, $M \geqslant 2$). In [2], N was required to be of the form $2^{k+1} + 1$ with k any nonnegative integer value.

(b) It allows a more general form of discretization in the x-direction. Instead of using only Cartesian equidistant grids, we can also apply the present method to non-Cartesian (e.g., cylindrical) coordinates and to nonequidistant spacings.

The present method can be extended to more than two dimensions. This will be discussed in Section 3.

The solution of Poisson's equation in three space dimensions by sine-cosine-transformation combined with Gaussian elimination has been described by Williams [4]; for this method, $(N - 1)$ must be a multiple of four. Kau and Peskin [5] and Schumann [6] also consider the three-dimensional problem, but with periodic boundary conditions in two directions and Neumann boundary conditions in only one direction. These studies use fast Fourier transform (FFT) in the planes bounded by periodic boundary conditions and N should be a power of two therefore; Gaussian elimination is used in the third direction. A program to solve Poisson's equation with Neumann boundary conditions for $N = 2^l 3^m 5^n + 1$ has been developed by Sweet [7]; it is not applicable for the present problem as it is designed for a *non*staggered grid. Iterative methods are discussed, e.g., in [8, 9].

The operation count of the present direct method is proportional to $MN \log_2 N$, and about MN storage locations are required. It is, therefore, considerably more effective than iterative methods unless a very good guess is available. In [2], it has been shown that the cyclic reduction method is faster than those based on FFT for $N = 2^{k+1}$. For general N the difference in speed between the present method and methods based on FFT is even larger.

2. Cyclic Reduction Scheme

2.1. *General Case*

We describe the solution of (2) for general N by cyclic reduction with the Buneman variation [3] for numerical stability. For $N = 2^{k+1} + 1$ the present scheme reduces to that described in [2].

Let

$$A^{(0)} = A, \quad K^{(0)} = A - I, \quad B^{(0)} = A - I, \quad C^{(0)} = I,$$
$$p^{(0)} = 0, \quad q_j^{(0)} = f_j, \quad N_0 = N, \quad J_0 = N \tag{4}$$

where I is the unit matrix. At the rth step of the *reduction process* we have the $N_r \times N_r$ system

$$\begin{bmatrix} K^{(r)} & -I & & & \\ -I & A^{(r)} & -I & & \\ & \cdot & \cdot & \cdot & \\ & & -I & A^{(r)} & -I \\ & & & -I & B^{(r)}/C^{(r)} \end{bmatrix} \begin{bmatrix} v_1 \\ v_{1+2^r} \\ \vdots \\ v_{J_r-2^r} \\ v_{J_r} \end{bmatrix} = \begin{bmatrix} K^{(r)}p_1^{(r)} + q_1^{(r)} \\ A^{(r)}p_{1+2^r}^{(r)} + q_{1+2^r}^{(r)} \\ \vdots \\ A^{(r)}p_{J_r-2^r}^{(r)} + q_{J_r-2^r}^{(r)} \\ (B^{(r)}/C^{(r)})\, p_{J_r}^{(r)} + q_{J_r}^{(r)} \end{bmatrix}, \tag{5}$$

understanding $B^{(r)}/C^{(r)}$ as $B^{(r)}(C^{(r)})^{-1}$. The f_j on the right-hand side has been split into two arrays for stability reasons [2, 3]. If $N_r > 2$, we reduce the number of unknowns to approximately half by eliminating every second unknown. For this purpose we multiply all rows at $j = 1, 1 + 2^{r+1}, 1 + 2 \cdot 2^{r+1}, \ldots$ by $A^{(r)}$. To each of these equations we then add the previous and succeeding equations except when N_r is even, in which case we add $A^{(r)}C^{(r)}/B^{(r)}$ times the last equation. From this we find

$$K^{(r+1)} = K^{(r)}A^{(r)} - I, \tag{6}$$

$$A^{(r+1)} = (A^{(r)})^2 - 2I, \tag{7}$$

and

$$\frac{B^{(r+1)}}{C^{(r+1)}} = \frac{B^{(r)}(A^{(r)})^2 - B^{(r)} - A^{(r)}C^{(r)}}{B^{(r)}} \quad \text{if } N_r \text{ even}, \tag{8}$$

$$\frac{B^{(r+1)}}{C^{(r+1)}} = \frac{B^{(r)}A^{(r)} - C^{(r)}}{C^{(r)}} \quad \text{if } N_r \text{ odd}, \tag{9}$$

$$\begin{aligned} N_{r+1} &= N_r/2 && \text{if } N_r \text{ even}, \\ &= (N_r + 1)/2 && \text{if } N_r \text{ odd}, \end{aligned} \tag{10}$$

$$\begin{aligned} J_{r+1} &= J_r - 2^r && \text{if } N_r \text{ even}, \\ &= J_r && \text{if } N_r \text{ odd}. \end{aligned} \tag{11}$$

Instead of computing these matrices directly, which would destroy their tri-

diagonal nature, we note the following factorized forms (the factorization of $A^{(r)}$ is developed in [3], that of $K^{(r)}$ in [2]):

$$A^{(r)} = \prod_{j=1}^{2^r} \left(A - 2\cos\left[\frac{(2j-1)\pi}{2^{r+1}}\right] I\right), \tag{12}$$

$$K^{(r)} = \prod_{j=1}^{2^r} \left(A - 2\cos\left[\frac{(2j-1)\pi}{(2^{r+1}+1)}\right] I\right). \tag{13}$$

From (4), (8), and (9) we see that $B^{(r)}$ and $C^{(r)}$ are polynomials, say b_r and c_r in the matrix A. If k_r is the degree of b_r and l_r the degree of c_r, we find:

$$k_0 = 1, \qquad l_0 = 0. \tag{14a}$$

$$\begin{aligned} N_r \text{ is odd:} &\quad k_{r+1} = k_r + 2^r, \quad l_{r+1} = l_r\,. \\ N_r \text{ is even:} &\quad k_{r+1} = k_r + 2^{r+1}, \quad l_{r+1} = k_r\,. \end{aligned} \tag{14b}$$

By induction it can be shown that

$$k_r = l_r + 2^r \quad \text{and} \quad l_r \leqslant 2^r - 1, \quad k_r \leqslant 2^{r+1} - 1 \quad \text{for} \quad r = 0, 1, 2, \ldots\,. \tag{15}$$

Using the substitution $x = 2\cos\theta$, we find:

$$b_r(x) = \frac{\cos[(k_r + \frac{1}{2})\,\theta]}{\cos(\theta/2)}, \qquad c_r(x) = \frac{\cos[(l_r + \frac{1}{2})\,\theta]}{\cos(\theta/2)}. \tag{16}$$

Using (4), (8), (9), and (14a, b) we can verify this by induction. From (16) we determine the roots θ_j of these polynomials and get:

$$B^{(r)} = \prod_{j=1}^{k_r} \left(A - 2\cos\left[\frac{(2j-1)\pi}{(2k_r+1)}\right] I\right), \tag{17}$$

$$\begin{aligned} C^{(r)} &= I, && l_r = 0 \\ &= \prod_{j=1}^{l_r} \left(A - 2\cos\left[\frac{(2j-1)\pi}{(2l_r+1)}\right] I\right), && l_r > 0. \end{aligned} \tag{18}$$

The new right-hand side vectors are found from:

$$p_1^{(r+1)} = p_1^{(r)} + (K^{(r)})^{-1}\,(q_1^{(r)} + p_{1+2^r}^{(r)}), \tag{19a}$$

$$q_1^{(r+1)} = q_{1+2^r}^{(r)} + p_1^{(r+1)}, \tag{19b}$$

$$p_j^{(r+1)} = p_j^{(r)} + (A^{(r)})^{-1}\,(p_{j-2^r}^{(r)} + p_{j+2^r}^{(r)} + q_j^{(r)}), \tag{20a}$$

$$q_j^{(r+1)} = q_{j-2^r}^{(r)} + q_{j+2^r}^{(r)} + 2p_j^{(r+1)}, \tag{20b}$$

$$j = 1 + 2^{r+1},\ 1 + 2\cdot 2^{r+1},\ 1 + 3\cdot 2^{r+1},\ldots,\ R_r\,.$$

If N_r is odd, $R_r = J_r - 2^{r+1}$ and

$$p_{J_{r+1}}^{(r+1)} = p_{J_r}^{(r+1)} = p_{J_r}^{(r)} + (B^{(r)}/C^{(r)})^{-1}\,(q_{J_r}^{(r)} + p_{J_r-2^r}^{(r)}), \tag{21a}$$

$$q_{J_{r+1}}^{(r+1)} = q_{J_r}^{(r+1)} = q_{J_r-2\cdot}^{(r)} + p_{J_r}^{(r+1)}. \tag{21b}$$

If N_r is even, $R_r = J_r - 2^{r+1} - 2^r$ and

$$p_{J_{r+1}}^{(r+1)} = p_{J_r-2^r}^{(r+1)} = p_{J_r-2^r}^{(r)} + (A^{(r)})^{-1}\,(q_{J_r-2^r}^{(r)} + p_{J_r}^{(r)} + p_{J_r-2\cdot 2^r}^{(r)}), \tag{22a}$$

$$q_{J_{r+1}}^{(r+1)} = q_{J_r-2^r}^{(r+1)} = q_{J_r-2\cdot 2^r}^{(r)} + \frac{A^{(r)}C^{(r)}}{B^{(r)}}\,(q_{J_r}^{(r)} + p_{J_r-2^r}^{(r+1)}) + p_{J_r-2^r}^{(r+1)}. \tag{22b}$$

The above reduction process is used for $r = 0, 1, 2,\ldots, k, k+1$ where the last reduction (for $r = k+1$, which defines k) results in the 2×2 system

$$\begin{bmatrix} K^{(r)} & -I \\ -I & B^{(r)}/C^{(r)} \end{bmatrix} \begin{bmatrix} v_1 \\ v_{J_r} \end{bmatrix} = \begin{bmatrix} K^{(r)}p_1^{(r)} + q_1^{(r)} \\ (B^{(r)}/C^{(r)})\,p_{J_r}^{(r)} + q_{J_r}^{(r)} \end{bmatrix}, \tag{23}$$

where $J_r = 1 + 2^r$. We then eliminate the last unknown and get:

$$\frac{E}{C^{(r)}}\,v_1 = \frac{E}{C^{(r)}}\,p_1^{(r+1)} + q_1^{(r+1)}, \tag{24}$$

where

$$E = K^{(r)}B^{(r)} - C^{(r)} \tag{25}$$

is a polynomial e in A and $p_1^{(r+1)}$, $q_1^{(r+1)}$ are defined as in (19a, b). The polynomial e is found, as above, to be

$$e(x) = \frac{\sin(k_r+1)\theta}{\sin\theta}\,\frac{\sin 2^r\theta}{\sin\theta}\cdot(2\cos\theta - 2), \tag{26}$$

so that

$$E = \prod_{j=1}^{k_r}\left[A - 2\cos\left(\frac{j\pi}{k_r+1}\right)I\right]\prod_{j=1}^{2^r-1}\left[A - 2\cos\left(\frac{j\pi}{2^r}\right)I\right][A - 2I]. \tag{27}$$

Note that $A - 2I$ may be singular, e.g., if Neumann or periodic boundary conditions are used in the x-direction; in this case the right-hand side f_j of (2) must satisfy a consistency condition [2].

From Eq. (24) we can evaluate the first vector of unknowns v_1 by using an algorithm of Swartztrauber [10, p. 1143] which efficiently solves linear systems of the form

$$\prod_{i=1}^{\alpha} (A - \lambda_i I)x = \prod_{j=1}^{\beta} (A - \mu_j I)b.$$

Thereafter, using (23) we determine v_{J_r} from

$$\frac{B^{(r)}}{C^{(r)}} v_{J_r} = \frac{B^{(r)}}{C^{(r)}} p_{J_r}^{(r)} + q_{J_r}^{(r)} + v_{J_r - 2^r}. \tag{28}$$

The *backsubstitution process* then follows for $r = k, k-1, \ldots, 0$, where we determine

$$v_j = p_j^{(r)} + (A^{(r)})^{-1} (q_j^{(r)} + v_{j-2^r} + v_{j+2^r}), \tag{29}$$

for $j = 1 + 2^r, 1 + 3 \cdot 2^r, 1 + 5 \cdot 2^r, \ldots, J_{r+1} - 2^r$. Then, we set

$$\begin{aligned} J_r &= J_{r+1} && \text{if} \quad J_{r+1} + 2^r > N, \\ &= J_{r+1} + 2^r && \text{if} \quad J_{r+1} + 2^r \leqslant N. \end{aligned} \tag{30}$$

If J_r is larger than J_{r+1} we again use (28) and then reduce r by 1 to continue the backsubstitution until $r = 0$.

2.2. *Special Handling of* $N = 2^{k+1} + 1$

If $N - 1$ is a power of two (say, 2^{k+1}), the above algorithm is identical to that given in [2] for $r = 0, 1, 2, \ldots, k$. It is different, however, for $r = k + 1$. For $N - 1 = 2^{k+1}$ the reduction process at $r = k$ results in a remaining system for three unknowns which can be solved more efficiently by the method described in [2] than by that described here. Noting that now $B^{(r)} = K^{(r)}$ and $C^{(r)} = I$ we use the method of [2] for this special case. Here, we solve

$$v_{1+2^k} = p_{1+2^k}^{(k+1)} + M^{-1} q_{1+2^k}^{(k+1)}, \tag{31}$$

and

$$v_j = p_j^{(k)} + (K^{(k)})^{-1} (q_j^{(k)} + v_{1+2^k}) \qquad j = 1, J_k. \tag{32}$$

As developed in [2], we use the factorization

$$M = \left[\prod_{j=1}^{2^k} \left(A - 2\cos\left[\frac{2j\pi}{(2^{k+1}+1)}\right] I\right)\right]\left[\prod_{j=0}^{2^k-1} (A - 2\cos(j\pi/2^k)\, I)\right]. \tag{33}$$

2.3. *Special Handling of $N_r = 3$ after Some r Reduction Steps*

If $N - 1$ is not a power of two, but falls into the range

$$N_2 + 2 \leqslant N \leqslant \tfrac{3}{2}N_2\,, \tag{34}$$

where N_2 is the largest power of two less than N, then after some (say, r) reduction steps we reach a situation where $N_r = 3$, but $B^{(r)} \neq K^{(r)}$. In these cases we may again use a special algorithm which is less effective than that for $N - 1 = 2^{r+1}$ but faster than the general method described in Section 2.1. The general method must be used for the remaining cases because after some r reductions we arrive at $N_r = 4$.

In the case $N_r = 3$, we have the remaining system

$$\begin{bmatrix} K^{(r)} & -I & \\ -I & A^{(r)} & -I \\ & -I & B^{(r)}/C^{(r)} \end{bmatrix} \begin{pmatrix} v_1 \\ v_{1+2^r} \\ v_{J_r} \end{pmatrix} = \begin{pmatrix} K^{(r)} p_1^{(r)} + q_1^{(r)} \\ A^{(r)} p_{1+2^r}^{(r)} + q_{1+2^r}^{(r)} \\ (B^{(r)}/C^{(r)})\, p_{J_r}^{(r)} + q_{J_r}^{(r)} \end{pmatrix}, \tag{35}$$

where $J_r = 1 + 2^{r+1}$. Multiplying the first equation by $B^{(r)}$, the second by $B^{(r)}K^{(r)}$ abd the last by $K^{(r)}C^{(r)}$ and then summing over all these equations, we eliminate the first and last unknowns and find:

$$\begin{aligned} v_{1+2^r} = p_{1+2^r}^{(r)} + D^{-1}\{&K^{(r)}C^{(r)}(p_{1+2^r}^{(r)} + q_{J_r}^{(r)}) + B^{(r)}(q_1^{(r)} + p_{1+2^r}^{(r)}) \\ &+ B^{(r)}K^{(r)}(q_{1+2^r}^{(r)} + p_1^{(r)} + p_{J_r}^{(r)})\}, \end{aligned} \tag{36}$$

where

$$D = A^{(r)}B^{(r)}K^{(r)} - B^{(r)} - K^{(r)}C^{(r)}. \tag{37}$$

Using the polynomial representations of $A^{(r)}$ and $K^{(r)}$, given in [2], and those of $B^{(r)}$ and $C^{(r)}$, as given in (16), the corresponding polynomial d for D is easy to find:

$$d(x) = \frac{\sin[(k_r + 2^r + 1)\,\theta]}{\sin\theta}\,\frac{\sin(2^r\theta)}{\sin\theta}\,(2\cos\theta - 2). \tag{38}$$

Using the roots of this polynomial we write

$$D = \left\{ \prod_{j=1}^{k_r+2^r} \left[A - 2\cos\left(\frac{j\pi}{k_r + 2^r + 1} \right) I \right] \right\} \left\{ \prod_{j=1}^{2^r-1} \left[A - 2\cos\left(\frac{j\pi}{2^r} \right) I \right] \right\} (A - 2I). \tag{39}$$

With v_{1+2^r} from Eq. (36) we compute

$$v_1 = p_1^{(r)} + (K^{(r)})^{-1}(q_1^{(r)} + v_{1+2^r}), \tag{40}$$

$$v_{J_r} = p_{J_r}^{(r)} + (B^{(r)}/C^{(r)})^{-1}(q_{J_r}^{(r)} + v_{1+2^r}), \tag{41}$$

and, thereafter, all other unknowns are computed by the backsubstitution process described above.

The solution of Eq. (36) may be viewed as the solution of three separate linear systems

$$Dx_1 = K^{(r)}C^{(r)}b_1, \qquad Dx_2 = B^{(r)}b_2, \qquad \text{and} \qquad Dx_3 = B^{(r)}K^{(r)}b_3$$

each with the same coefficient matrix D which must be inverted. To solve these efficiently we treat the solution as though we were inverting one matrix, D, for three different right-hand sides (rhs) simultaneously. This inversion requires two additional work arrays of length M to store the rhs and two additional work arrays of the same length to do the computations. In this case only $3 \cdot 2^r + 2k_r < 7 \cdot 2^r$ inversions of tridiagonal systems are needed. This must be compared to the $8 \cdot 2^r + 3 \cdot k_r < 14 \cdot 2^r$ inversions required if we, instead, further reduce Eq. (35) to a system of two unknowns and then solve as described in Section 2.1. Here, a savings of up to 55% for the last three unknowns over the time required for the general case makes this special treatment worthwhile.

The algorithm used to solve (36) and the work arrays can be used to solve (20a) and (29) in groups of three. This is possible since, for all values of j that are considered in these equations, the same matrix $A^{(r)}$ must be inverted. Use of this method reduces the computing time by about 12%.

Actually the workspace needed to solve (36) in parallel is sufficient to solve (20a) and (29) in groups of six. However, an additional routine would be required for this purpose. Moreover, for small r, not all the space of the array p is needed, and this space could be used to further speed up the algorithm. Such an approach requires the use of an algorithm that solves any number of parallel tridiagonal systems. However, use of the additional loops needed to achieve this flexibility is so time-consuming that these refinements are not worthwhile.

2.4. *Elimination of the Work Array p*

As shown in [3], for $(N - 1) = 2^{k+1}$, it is possible to avoid storing the auxiliary array p and instead to use its definition implicitly. When $N - 1$ is not a power of

two, no suitable method has been found to completely avoid storing of the p-array. However, we can reduce the size needed for this array so that it is no larger than $[\log_2 N]M$, where $[x]$ means the integer portion of x. For most values of N the size of the array is even much smaller than $[\log_2 N]M$. For this purpose, we use:

$$p_j^{(0)} = 0 \qquad \text{for} \quad 1 \leqslant j \leqslant N, \tag{42}$$

and for $r > 0$:

$$p_1^{(r)} = q_1^{(r)} - q_{1+2^{r-1}}^{(r-1)}, \tag{43}$$

$$p_j^{(r)} = \tfrac{1}{2}[q_j^{(r)} - q_{j-2^{r-1}}^{(r-1)} - q_{j+2^{r-1}}^{(r-1)}]$$
$$j = 1 + 2^{r+1}, 1 + 2 \cdot 2^{r+1}, 1 + 3 \cdot 2^{r+1}, \ldots, R_r, \tag{44}$$

$$p_{J_r}^{(r)} = q_{J_r}^{(r)} - q_{J_r-2^{r-1}}^{(r-1)} \qquad \text{if} \quad N_{r-1} \text{ is odd;} \tag{45}$$

otherwise we use the result of Eq. (22) as obtained for the previous step. The value of the last p array can be destroyed and its space can be used again if N_r is odd; otherwise, it must be retained until we reach the same level during the back-substitution phase. During the reduction phase we use a special variable to store the information on whether N_{r-1} is odd or even. During the backsubstitution phase we find that N_{r-1} is even if $J_r + 2^{r-1} \leqslant N$ and odd otherwise.

3. Performance Experiences and Generalization

The algorithm has been coded in FORTRAN and compared to the algorithm described in [2] which was also coded in FORTRAN. For $N = 2^{k+1} + 1$ both algorithms used the same computation time (144 msec for $M = N = 65$ on the NCAR Control Data 7600). The algorithm presented here would be somewhat slower because of the more general x-dependence in Eq. (1a); this has been balanced by solving the tridiagonal systems in groups of three as described in Section 2.3. However, the present program needs four more working arrays of size M, and a small amount of additional space is needed to store the roots of the polynomials. The length of the storage space for the roots is L, and L must be

$$\begin{aligned} L &= N - 2 && \text{for} \quad N = N_2 + 1 \text{ (as in [2]),} \\ L &= [3.75N] - 5 && \text{for} \quad N_2 + 1 < N \leqslant \tfrac{3}{2}N_2, \\ L &= 2N - 3 && \text{for} \quad \tfrac{3}{2}N_2 < N \leqslant 2N_2. \end{aligned}$$

Here, N_2 is the largest power of two less than or equal to N.

The relative computing time for $M = N$ is plotted in Fig. 1. We see that this time for any value of N is, at most, 25% larger than for $N = 2^k + 1$. Copies of the FORTRAN subroutine POISSN using the present algorithm are available from the authors.

If we generalize the method to three dimensions, the elements α_i, β_i, γ_i in matrix A (Eq. (3)) themselves become matrices. We then have to solve block-tridiagonal systems. If Cartesian coordinates, equidistant grid spacings and Neumann boundary conditions are used in at least two of the three dimensions,

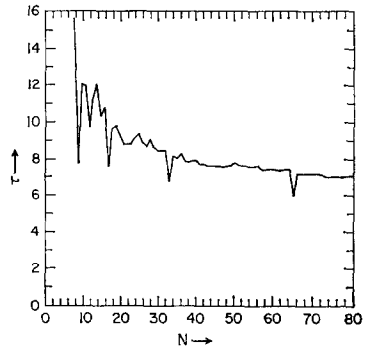

FIG. 1. Relative computing time $\tau = T/(M \cdot N \cdot \log_2 N)$ where T is the computing time in 10^{-6} sec. on the NCAR Control Data 7600 and $M = N$. The minimum values of τ appear for $N = 2^{k+1} + 1$.

we can use the present algorithm in a recursive sense. If these conditions do not apply, we may use other direct methods [e.g., 7, 10] to solve these systems. The generalization of the present method to other boundary conditions is possible and will be described elsewhere.

ACKNOWLEDGMENT

This work was done while U. Schumann was on leave with the Advanced Study Program of the National Center for Atmospheric Research, Boulder, Colorado, U.S.A.

REFERENCES

1. F. H. HARLOW AND J. E. WELCH, *Phys. Fluids* **8** (1965), 2182–2189.
2. R. A. SWEET, *J. Computational Phys.* **12** (1973), 422–428.
3. B. L. BUZBEE, G. H. GOLUB AND C. W. NIELSON, *SIAM J. Numer. Anal.* **7** (1970), 627–656.
4. G. P. WILLIAMS, *J. Fluid Mech.* **37** (1969), 727–750.

5. C. J. KAU AND R. L. PESKIN, "Numerical Simulation of Turbulence and Diffusion in Three-Dimensional Flow," Techn. Report No. 101, Geophys. Fluid Dyn. Progr., Rutgers Univ. (1972).
6. U. SCHUMANN, "Ein Verfahren zur direkten numerischen Simulation turbulenter Strömungen in Platten- und Ringspaltkanälen und über seine Anwendung zur Untersuchung von Turbulenzmodellen," Dissertation TH Karlsruhe, Report KFK 1854 (1973).
7. R. A. SWEET, *SIAM J. Numer. Anal.* **11** (1974), 506–520.
8. A. E. FANNING AND T. J. MUELLER, *J. Computational Phys.* **13** (1973), 450–454.
9. M. J. O'CARROLL, *J. Inst. Math. Appl.* **11** (1973), 343–350.
10. P. N. SWARTZTRAUBER, *SIAM J. Numer. Anal.* **11** (1974), 1136–1150.