JOURNAL OF COMPUTATIONAL PHYSICS 30, 145–148 (1979)

Note

A Fast Poisson-Solver for Large Grids

One of the simplest fast direct methods for solving the discrete Poisson equation is adapted for use in very large problems where the right-hand side and solution fields must be stored on disc. The performance of the algorithm described here is compared with that of a technique due to Schumann for a three-dimensional Poisson problem on a $(128)^3$ grid.

1. *Introduction*

Fast direct methods ("Poisson-solvers") [2, 3, 4, 5] are well-established for the solution of the discrete Poisson (or simple Helmholtz) equation in the case of moderate-sized problems which can easily be accommodated within the fast random-access memory of a computer. The purpose of this note is to demonstrate that one of the simplest such algorithms can easily be extended to very large problems for which only a small proportion of the gridpoint values can be held in memory at any one time.

The scheme described here was developed for use in a large numerical weather prediction model. Implementation of a semi-implicit time integration algorithm [1] requires the solution at each timestep of a three-dimensional discrete elliptic equation in spherical coordinates, with grid dimensions up to 360×180 in the horizontal, and 15 vertical levels. Diagonalization of the vertical part of the finite-difference operator reduces the problem to a set of Helmholtz equations, each of which must be solved over a horizontal grid covering the surface of the sphere.

To simplify the discussion, we present the scheme here in the context of a two-dimensional Poisson problem in Cartesian geometry. The extension to three dimensions is immediate, and results are included of an experiment to test the scheme on the solution of a discrete three-dimensional Poisson equation over a $128 \times 128 \times 128$ grid.

2. *Algorithm*

Suppose that we wish to solve the discrete Poisson equation (using centered second-order differences) over the grid $(i, j\colon\ 0 \leqslant i \leqslant N,\ 0 \leqslant j \leqslant M)$ with homogeneous Dirichlet boundary conditions and unit gridlength. Let

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

0021-9991/79/010145-04$02.00/0

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

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

with $\mathbf{x}_0 = \mathbf{x}_M = \mathbf{0}$, where A is the tridiagonal matrix of order $(N-1)$ with constant diagonal term -4, and 1's on the sub- and superdiagonals.

Let S be the matrix representation of a Fourier sine transform; thus S is a square matrix of order $(N-1)$ with elements given by $S = (s_{ij})$, where $s_{ij} = (2/N)\sin(ij\pi/N)$. With this scaling, $S^{-1} = (N/2)S$. Denote the components of the vectors $\hat{\mathbf{x}}_j = S\mathbf{x}_j$ and $\hat{\mathbf{b}}_j = S\mathbf{b}_j$ by $\hat{x}_{k,j}$ and $\hat{b}_{k,j}$ $(1 \leqslant k \leqslant N-1)$, respectively.

Then the basic FFT method for solving the system (1) proceeds as follows:

(1) For each line j, $1 \leqslant j \leqslant M-1$, calculate $\hat{\mathbf{b}}_j = S\mathbf{b}_j$, using Fast Fourier Transform (FFT) techniques;

(2) For each wave number k, $1 \leqslant k \leqslant N-1$, solve the tridiagonal system

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

with $\hat{x}_{k,0} = \hat{x}_{k,M} = 0$, where $\lambda_k = 2\cos(k\pi/N) - 4$;

(3) For each line j, $1 \leqslant j \leqslant M-1$, calculate $\mathbf{x}_j = S^{-1}\hat{\mathbf{x}}_j$.

Suppose now that the right-hand side is stored on disc, with the data partitioned into $(M-1)$ records each corresponding to a vector $\mathbf{b}_j$, and that the solution is to be written to disc in the same format. Steps (1) and (3) of the algorithm outlined above can easily be implemented, since we can read in one record at a time, multiply the vector by S or S^{-1}, and write out the resulting record. However, step (2) appears to be more difficult, since for each tridiagonal system we require one element from each record j, $1 \leqslant j \leqslant M-1$. In effect we need to reorder the data by columns rather than rows for step (2), and then to reverse the reordering ready for step (3). Schumann [6] has devised a "fast matrix transpose" algorithm for externally stored data, which can be used for this purpose [7].

There is, however, a much simpler solution. We solve each tridiagonal system (2) by means of the following algorithm, based on Gaussian elimination:

$$\begin{aligned}
&\omega_{k,0} = 0;\ \omega_{k,j} = (\lambda_k - \omega_{k,j-1})^{-1}, && 1 \leqslant j \leqslant M-1; \\
&g_{k,0} = 0;\ g_{k,j} = \omega_{k,j}(\hat{b}_{k,j} - g_{k,j-1}), && 1 \leqslant j \leqslant M-1; \\
&\hat{x}_{k,M} = 0;\ \hat{x}_{k,j} = g_{k,j} - \omega_{k,j}\hat{x}_{k,j+1}, && M-1 \geqslant j \geqslant 1.
\end{aligned} \tag{3}$$

In the following algorithm description, it is helpful to define

$$\boldsymbol{\omega}_j = \begin{bmatrix} \omega_{1,j} \\ \omega_{2,j} \\ \vdots \\ \omega_{N-1,j} \end{bmatrix}, \qquad \mathbf{g}_j = \begin{bmatrix} g_{1,j} \\ g_{2,j} \\ \vdots \\ g_{N-1,j} \end{bmatrix},$$

where the first subscript of the vector components corresponds to the wave number index k. The coefficients $\omega_{k,j}$ can be precomputed and stored on disc in the same format as the other fields, so that each record contains a vector $\boldsymbol{\omega}_j$. Also, we use the notation **ab** to denote componentwise multiplication of two vectors.

For a problem in memory, it is natural to solve one tridiagonal system at a time; however, as they are independent they may be solved in parallel. This feature is useful for implementation on parallel computers [3] or vector machines.

Moreover, by solving them in parallel the sine transforms and the solutions of tridiagonal systems can be interleaved in the following manner:

(a) Forward sweep: initialize by setting $\mathbf{g}_0 = \mathbf{0}$. Then for each line j, $1 \leqslant j \leqslant M - 1$:

(1) Read in the vector $\mathbf{b}_j$, and multiply by S to form $\hat{\mathbf{b}}_j$.

(2) Calculate $\hat{\mathbf{b}}_j - \mathbf{g}_{j-1}$.

(3) Read in the coefficient vector $\boldsymbol{\omega}_j$ and calculate $\mathbf{g}_j = \boldsymbol{\omega}_j(\hat{\mathbf{b}}_j - \mathbf{g}_{j-1})$.

(4) Write $\mathbf{g}_j$ to disc and proceed to line $(j + 1)$.

(b) Backward sweep: initialize by setting $\hat{\mathbf{x}}_M = \mathbf{0}$. Then for each line j, $M - 1 \geqslant j \geqslant 1$:

(1) Read in the coefficient vector $\boldsymbol{\omega}_j$ and calculate $\boldsymbol{\omega}_j\hat{\mathbf{x}}_{j+1}$.

(2) Read in $\mathbf{g}_j$ and calculate $\hat{\mathbf{x}}_j = \mathbf{g}_j - \boldsymbol{\omega}_j\hat{\mathbf{x}}_{j+1}$.

(3) Multiply $\hat{\mathbf{x}}_j$ by S^{-1} to form $\mathbf{x}$, .

(4) Write $\mathbf{x}_j$ to disc and proceed to line $(j - 1)$.

At the end of the backward sweep, the complete solution has been written to disc, the whole process requiring 6 $(M - 1)$ input/output (I/O) operations, where an operation consists of reading or writing one record. This compares with $(6M + 5M \log_2 M)$ I/O operations (for $M = N$) in the algorithm suggested by Schumann [7], and for a 128×128 problem represents an 85 % reduction in I/O requirements.

The coefficient vectors $\boldsymbol{\omega}_j$ may be generated and written to disc during the forward sweep, without altering the number of I/O operations. Unfortunately it is not possible to economize on I/O by generating them again during the backward sweep, since the backward recursion corresponding to Eq. (3) is numerically ill-conditioned.

The minimum number of memory locations required for data is $2(N - 1)$, excluding memory for the I/O routines. In order to achieve a reasonable degree of parallelism between computation and I/O operations, it is necessary to allow a further $2(N - 1)$ memory locations for buffering.

The reverse reading of the records from disc storage is most easily accomplished by random or direct access input routines, but can be performed with sequential I/O routines using the equivalent of the Fortran BACKSPACE feature.

For computer systems with virtual storage or a paging mechanism, the I/O operations could be performed by the operating system automatically, and indeed efficiently, provided that the vectors $\mathbf{x}_j$, etc., have been carefully ordered.

By allowing a small number of extra lines in memory at a time, Hockney's FACR (1) algorithm [4, 5] can also be implemented in this way, provided that the tridiagonal systems are again solved by Gaussian elimination. The method also extends immediately to the three-dimensional discrete Poisson equation, which can be solved by performing a two-dimensional FFT on each plane, solving tridiagonal systems in the third direction, and then performing a two-dimensional inverse FFT on each plane; it is simply necessary to replace lines by planes in the algorithm described above.

3. *Implementation*

To demonstrate the effectiveness of the algorithm, two Fortran programs were written to solve the three-dimensional Poisson equation over a $128 \times 128 \times 128$ grid. One implemented the scheme presented here, while the other used a generalization of Schumann's transpose algorithm in which the fields were treated as $N \times N$ matrices with vectors of length N as matrix elements. In both cases the right-hand side and solution fields were stored on disc as 128 records each of dimension 128×128, and space was allowed for four such records to be held in memory simultaneously. Full use was made of *I/O* routines permitting data transfer and computation to proceed in parallel. Both programs were run on a CDC 6600.

The algorithm described here required 340 sec of CPU time, and 195 sec of *I/O* time; almost complete overlapping was achieved, so that the optimum elapsed time was also about 340 sec.

Schumann's algorithm required 380 sec of CPU time, the increase being mainly due to the data transfers within memory required during the transpose phases. The *I/O* time was 1,350 sec, with an optimum elapsed time in the region of 1,500 sec.

The right-hand side was derived from a prescribed solution field consisting of random numbers in the interval $[-1, +1]$, against which the computed solutions were checked. The maximum error was 1.14×10^{-12} (in both cases, as the programs were identical in terms of floating-point computation).

References

1. D. M. Burridge and J. Haseler, to appear.
2. B. L. Buzbee, G. H. Golub, and C. W. Nielson, *SIAM J. Numer. Anal.* **7** (1970), 627–656.
3. B. L. Buzbee, *IEEE Trans. Comput.* **C-22** (1973), 793–796.
4. R. W. Hockney, *J. Assoc. Comput. Mach.* **12** (1965), 95–113.
5. R. W. Hockney, The potential calculation and some applications, *in* "Methods in Computational Physics," Vol. 9, 135–211, Academic Press, New York, 1970.
6. U. Schumann, *Angew. Inform.* **5** (1972), 213–216.
7. U. Schumann, *IEEE Trans. Comput.* **C-22** (1973), 542–544.

Received: June 23, 1977; revised: November 22, 1977

David M. Burridge and Clive Temperton

European Centre for Medium Range Weather Forecasts
Bracknell, Berkshire, United Kingdom