Title

NUMERICAL SOLUTION OF HELMHOLTZ'S EQUATION BY IMPLICIT CAPACITANCE MATRIX METHODS

Author

Proskurowski, Wlodzimierz.

Publication Date

1977-02-01

Submitted to ACM Transactions on
Mathematical Software

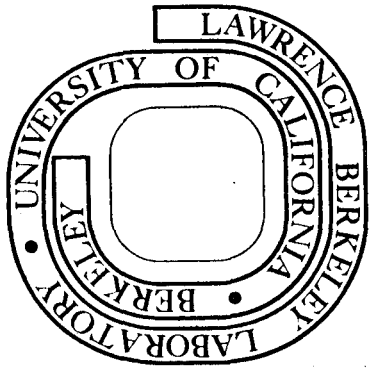
LBL-6402
Preprint C.1

NUMERICAL SOLUTION OF HELMHOLTZ'S EQUATION BY IMPLICIT CAPACITANCE MATRIX METHODS

Wlodzimierz Proskurowski

February 1977

Prepared for the U. S. Energy Research and
Development Administration under Contract W-7405-ENG-48



DISCLAIMER

NUMERICAL SOLUTION OF HELMHOLTZ'S EQUATION BY IMPLICIT CAPACITANCE MATRIX METHODS*

Wlodzimierz Proskurowski

Lawrence Berkeley Laboratory
University of California
Berkeley, California

February 1977

*This work was done with support from the U.S. Energy Research and Development Administration

ABSTRACT

The numerical solution of Helmholtz's equation in an arbitrary bounded plane region is considered. Variants of the capacitance matrix method are developed, which greatly reduce storage requirements. This allows the use of a very fine mesh with several hundred mesh points in each direction or the use of a computer with a small core storage.

1. INTRODUCTION

In recent years special techniques called the capacitance matrix method have been developed for the numerical solution of Helmholtz's equation in a general plane-bounded region Ω,

$$\Delta u + cu = f \text{ in } \Omega \quad ,$$

where c is a real constant, and either Dirichlet or Neumann conditions are specified on the boundary $\partial\Omega$. These methods make use of fast solvers in regions that allow for the separation of variables. Operation count, $\theta(n)$, for such fast solvers is proportional to $n^2 \log_2 n$, where n is the number of mesh points in each direction. For a detailed discussion of such methods and a history of their development, refer to Proskurowski and Widlund [8] and Widlund [11].

In this paper we confine ourselves to two dimensional bounded regions; for problems in three dimensions see O'Leary and Widlund [5].

The algorithms previously developed by us have a practical limitation on the number of mesh points, from memory considerations. Normally, one generates and stores a dense capacitance matrix C of the order of p, where p is the number of mesh points inside Ω adjacent to the boundary $\partial\Omega$. In this paper we develop an implicit method in which we avoid generating and storing the matrix C; see also O'Leary and Widlund [5]. Moreover, we exploit the fact that only a few mesh points from the closest vicinity of the boundary

$\partial\Omega$ are involved in the main part of the computation, that is, the capacitance matrix iterations. Using a solver on a rectangle, which takes the sparsity of the problem into account, developed originally by Banegas [1], we design an algorithm that requires only 32p storage locations for its main part. Only the last step, computing the final solution, is limited to memory requirements of a fast solver on a rectangle, i.e., $m \cdot n$ locations, where m and n are the number of mesh points in a rectangle in which the region Ω is imbedded. To remove that obstacle we propose a solver that requires only $2nm^{\frac{1}{2}}$ storage locations at the expense of some computational effort.

Numerical results from extensive experiments on the CDC 7600 computer are reported.

2. CAPACITANCE MATRIX METHODS AND POTENTIAL THEORY

In this section we give a brief review of the potential theoretical approach leading to capacitance matrix methods as described in Proskurowski and Widland [8]; see also Widlund [11].

We consider a problem on an arbitrary bounded plane region Ω. The region Ω is first imbedded in a larger region, a rectangle, and a uniform mesh is introduced with the same mesh size in the two coordinate directions. The boundary conditions on the rectangle can be of arbitrary type as long as they allow for the use of a fast solver; see Widlund [10] and Proskurowski and Widlund [8]. The set of mesh points is decomposed into three disjoint sets: Ω_h, $\partial\Omega_h$, and $(C\Omega)_h$. The set Ω_h is the set of interior mesh points, i.e., each of its members has all its immediate neighbors in the open set Ω. The remaining mesh points in Ω constitute $\partial\Omega_h$, the set of irregular mesh points, while the set $(C\Omega)_h$ contains all the remaining exterior mesh points. The discrete Laplacian is represented by the five-point formula for all points in $\Omega_h \cup (C\Omega)_h$. The data for the exterior points are extended in an arbitrary way; for the proof that the solution on $\Omega_h \cup \partial\Omega_h$ is independent of the solution and data on $(C\Omega)_h$, see Section 3 of Proskurowski and Widlund [8]. For the irregular points we must introduce a formula that also takes the boundary conditions on $\partial\Omega$ into account. We therefore combine the

discrete Laplacian with an interpolation formula. The important problem of scaling these auxiliary equations is treated in detail in Proskurowski and Widlund [8] and Shieh [9]. We will denote by A the $n^2 \times n^2$ matrix corresponding to the difference problem enlarged to a rectangle handling the given boundary conditions on $\partial\Omega$. The regularly structured problem for which a fast solver can be used is given by the $n^2 \times n^2$ matrix B representing the discrete Laplacian. With a proper ordering of equations, A and B differ only in rows corresponding to the irregular mesh points. For the Neumann problem we write $A=B+UV^T$, and for the Dirichlet problem $A=B+UZ^T$, where U, V, and Z have p columns, and p is the number of irregular mesh points. The matrix U represents an extension operator, which maps $\partial\Omega_h$ onto the whole rectangle. It retains the values on $\partial\Omega_h$ and makes the remaining values equal to zero. Its transpose, U^T, is a trace operator. Matrices V^T and $-Z^T$ are a compact representation of B-A, from which the zero rows corresponding to the regular mesh points have been deleted.

In potential theory the solution of the Neumann problem is given as a sum of a space potential u_s and a single layer potential of charge distribution at the boundary $\partial\Omega$:

$$u(x) = u_s(x) + \gamma(x). \tag{2.1}$$

A discrete analog to (2.1) is

$$u = Gf + GU\rho, \tag{2.2}$$

where each of the p columns of U represents a unit charge placed at an irregular point, where the discrete operator G plays the same role as the integral operator defined by the fundamental solution of the continuous problem (see Proskurowski and Widlund [8]), and ρ is determined by solving the capacitance matrix equation

$$C\rho = (I - V^T GU)\rho = V^T Gf = g \qquad (2.3)$$

where the p×p matrix C is the capacitance matrix and ρ is a vector of p components. A proper approach for the Dirichlet problem is the double-layer potential $\mathcal{W}$ of dipole density μ at the boundary $\partial\Omega$:

$$u(x) = u_s(x) + \mathcal{W}(x) \quad . \qquad (2.4)$$

A discrete analog to (2.4) is

$$u = Gf + GD\mu \ , \qquad (2.5)$$

where D has p columns, each of them representing a unit discrete dipole placed at an irregular point, and μ is the solution of

$$C\mu = (I+Z^T GD)\mu = -Z^T Gf = g, \qquad (2.6)$$

where μ is a vector of p components. Shieh [9] has shown that the capacitance matrix C is equal to K_h plus a matrix with a small condition number, where K_h is an approximation to the correct compact operator of the corresponding Fredholm integral equation of the second kind. The conjugate gradient method converges superlinearly for Fredholm integral equations of the second kind, as shown by Hayes [4]. Therefore, the conjugate gradient method applied for solving

eqs. (2.3) and (2.6) converges rapidly; in practice, it is independent of the size of the mesh; see also Proskurowski and Widlund [8]. In summary, the algorithm consists of the following steps:

1. Generate the capacitance matrix C.
2. Compute g.
3. Solve (2.3) and (2.6) by the conjugate gradient method.
4. Use the fast solver to obtain

$$u = G(f + U\rho) \quad \text{or} \quad u = G(f + D\mu) .$$

Another option for Step 3 is to factor C and solve (2.3) and (2.6) by Gaussian elimination. For the details of the algorithms and ways of fast generation of C, refer to Proskurowski and Widlund [8].

The total operation count for that algorithm is proportional to $n^2 \log_2 n$ and p^2.

Some alternatives to this algorithm that make it possible to avoid the explicit generation of C will be described in the next sections.

3. AN IMPLICIT CAPACITANCE MATRIX METHOD

Methods in which we explicitly generate, store, and possibly factor the capacitance matrix may become inefficient when the mesh is refined. The capacitance matrix is a dense, $p \times p$ matrix, where p is the number of irregular mesh points, which grows linearly with n, the number of mesh points in each coordinate direction. For example, somewhere between

the values of p equal to 150 and 200 the small core memory (SCM) for the CDC 7600 computer becomes saturated. The use of large core memory (LCM) would allow increasing the maximal values of p by a factor of 2 or slightly more, while for even larger p one must use a secondary memory device with a much longer access time. Therefore, we now present a method in which the capacitance matrix is used only implicitly without generating and storing it, thus saving p^2 memory locations at the expense of a small increase in computational effort; see also O'Leary and Widlund [5], Widlund [11] and an early paper of George [3].

We describe the method for the Dirichlet boundary conditions in which the proper Ansatz of double-layer potential is used. The Neumann boundary conditions, where the single-layer Ansatz is used and is a slightly simpler case, can be worked out in a similar way.

We once more write the capacitance matrix equation

$$C\mu = (I_p + Z^T GD) = Z^T Gf = g \ . \tag{3.1}$$

The capacitance matrix C can also be rewritten as

$$C = (I_p + Z^T GD) = U^T AGD \ , \tag{3.2}$$

which form we will subsequently use. Since matrix C is nonsymmetric and we intend to use the conjugate gradient method for solving (3.1), we reformulate it in terms of a least squares problem

$$C^T C\mu = C^T g \ . \tag{3.3}$$

Thus each step of the conjugate gradient method requires the computation of a matrix-vector product $C^T(Cx)$ for any vector x of length p given on the set of irregular mesh points, i.e.,

$$D^T G(U^T A)^T (U^T A) G D x \ . \tag{3.4}$$

Let us rewrite (3.4) as a sequence of equations.

$$\begin{aligned} x_1 &= Dx \ , \\ x_2 &= Gx_1 \ , \qquad \text{or } Bx_2 = x_1 \\ x_3 &= (U^T A)x_2 \ , \\ x_4 &= (U^T A)^T x_3 \ , \\ x_5 &= Gx_4 \ , \qquad \text{or } Bx_5 = x_4 \\ y &= D^T x_5 \ , \end{aligned} \tag{3.5}$$

Consequently, we first set to zero a large $n \cdot n$ array, then generate the mesh function Dx by distributing x onto the set of discrete dipoles. This step costs $2p$ multiplicative operations. Then we obtain $G(Dx)$ by using the fast solver at a cost proportional to $n^2 \log_2 n$ operations. U^T maps a mesh function defined for all mesh points into its restriction to irregular mesh points. Therefore, it is enough to apply the operator A to GDx only on the set of closest neighbors of irregular mesh points. Acting in this way we compute $U^T A(GDx)$ at the expense of $4p$ multiplicative operations. The part corresponding to the transpose of C is performed in a similar fashion. Thus, the vector $C^T Cx$ is obtained at a cost of two calls of the fast solver, proportional to $n^2 \log_2 n$ operations, plus a lower-order term, proportional to p operations. In our program some operations were repeated in order to save memory space.

Summing up, this method requires n^2+8p memory locations (n^2+10p for the Neumann problem) compared with p^2+n^2 plus a lower-order term proportional to p for the explicit capacitance matrix methods, as implemented in Proskurowski and Widlund [8].

On the other hand, the operation count for the present method is $(2k+3)\cdot\theta(n)$, where k is the number of the conjugate gradient iterations and is practically independent of n, the number of mesh points, and $\theta(n)$ is the cost of a fast Helmholtz solver proportional to $n^2\log_2 n$. In comparison, the cost for the explicit capacitance method with the conjugate gradient option is equal to $3.5\cdot\theta(n)+(2k+c)\cdot p^2$ operations, where c is a constant arising from the generation of the capacitance matrix.

An experimental comparison of the computation times for n=64, p=132 and a circular region is given in Section 7. It shows that whenever the capacitance matrix is small enough to fit into SCM, the explicit capacitance matrix methods are slightly faster. On the other hand, the present method does not make use of the translation invariance of the solution, which is exploited in our variant of the explicit capacitance matrix method, and alternative fast solvers might be easily used. A further development of this method is described in Section 5.

4. A HELMHOLTZ SOLVER THAT TAKES ADVANTAGE OF SPARSITY

Consider the Helmholtz equation $(-\Delta+c)u = f$ on a rectangular region with a $m\cdot n$ mesh. Let the mesh values of f and u be called sources and targets, respectively. Denote then by s the number of nonzero sources and by t the number of targets where the solution is required. Quite often there exist situation where either $s<<m\cdot n$ or $t<<m\cdot n$, or both. One such situ-

ation occurs while computing the discrete Green's function for the generation of the capacitance matrix, i.e., the case with s=1. A more general discussion of the applications for the present use is given in Section 5. We now describe how to make use of the sparsity of sources and/or targets to save memory space and possibly computational effort as well. This method was developed by Banegas [1] and we have been using in our experiments a considerably altered variant of her algorithm. This algorithm was meant to be compatible with the one using Fast Fourier Transform (FFT) as described in Proskurowski and Widlund [8]. Nevertheless, there is no difficulty in adapting it to alternative fast Helmholtz solvers, if necessary. We remark also that the only restriction on n is for it to be even.

First, recall the fast Helmholtz solver described in [8]. The solution there is obtained by applying the FFT in one coordinate direction, i.e., m times on vectors of length n, then by solving n very special tridiagonal systems of equations of order m by the Toeplitz method, and finally by using an inverse FFT on m vectors of length n. Memory requirement is here equal to $m \cdot n + 0(1)$ and the total operation count for this solver is $\theta(m,n) = \frac{9}{2} mn \log_2\left(\frac{n}{2}\right) + 15mn$, where each multiplication and each floating point addition is taken as a unit operation. We remark that the operation count always shows only a part of the actual computational expenses and is not an exact measure of it.

Let us now write the Fourier coefficients as inner products of the data vector $f^{(\ell)}$ with the eigenvectors $\phi^{(j)}$ of the matrix B representing the discrete Laplacian $(-\Delta+c)$:

$$\hat{f}_k^{(j)} = \frac{1}{n} \sum_{\ell=1}^{n} \phi_\ell^{(j)} \cdot f_k^{(\ell)} \quad , \tag{4.1}$$

where j=1,..., n and k=1,...,m.
In addition, the inverse Fourier coefficents can be written as

$$u_k^{(\ell)} = \sum_{j=1}^{n} \phi_\ell^{(j)} \cdot \hat{u}_k^{(j)} \quad , \tag{4.2}$$

where $\ell = 1,2,\ldots,n$ and $k = 1,2,\ldots,m$. It is easy to see that for sparse sources and targets the number of entries with double indexes (k,ℓ) is reduced to s and t, respectively. Consequently, the operation count for the summation formulas (4.1) and (4.2) is reduced considerably. Moreover, we may reorder the computation, performing it separately for each frequency j. We first compute the Fourier coefficients for all nonzero sources f_i, $i = 1,\ldots,s$ and simultaneously sum those having the same index k. Then we solve the intermediate tridiagonal matrix problem (with λ_j corresponding to each frequency j) with the previously computed Fourier coefficients as a right-hand side. Then finally we compute the inverse Fourier coefficients for all targets we need for the solution u_i, $i = 1,\ldots,t$, and sum simultaneously those having the same

index k. At this point locations used for the temporary values of $\hat{f}^{(j)}$ can be released and used for $\hat{f}^{(j+1)}$.

Thus, this procedure requires 3(s+t) locations for sources, targets, and their coordinates plus 2(m+n) locations for storing the Fourier coefficients temporarily and also sines and cosines. In the actual program we also store temporarily some indexes in order to avoid the repetition of computing them. In all, we use

$$4(s+t)+2(m+n) \text{ memory locations.} \tag{4.3}$$

If the sources and targets coincide, we could perform the computations in place (as we do in the fast solver using $m \cdot n$ locations), thereby further reducing these requirements to $4s+2(m+n)$ locations.

It is evident that for large s and t this procedure, which uses the conventional (i.e., slow) Fourier transform, will be much slower than a comparable solver using FFT. Now we will establish restrictions on s and t for this procedure to be competitive with a fast Helmholtz solver. The operation count is

$$\psi(n,m,s,t) = 3n \cdot (s+t) + 4mn. \tag{4.4}$$

Then $\psi = \theta$ for

$$s + t = \frac{3}{2} m \log_2 n/2 + \frac{11}{3} m. \tag{4.5}$$

For example, take m=n and s+t = 10n. This gives the ratio $\theta(n)/\psi(n) = \left(\frac{9}{2} \log_2 \frac{n}{2} + 15\right)/34$ which for n=64 is equal to 1.1. The corresponding ratio for execution times in numerical experiments (see Section 7) is very close to it.

5. AN IMPLICIT CAPACITANCE MATRIX METHOD USING A HELMHOLTZ SOLVER THAT EMPLOYS SPARSITY

We recall from Section 3 that the main computational effort (more than 90%) in an implicit capacitance matrix method goes for computing vectors $y = C^TCx = D^TG(U^TA)^T\ (U^TA)GDx$ during the conjugate gradient iterations. Moreover, a dominant part of this computation, also over 90%, is spent on a fast Helmholtz solver. We recall also that while distributing the discrete dipoles (D) and using Shortley-Weller stencils (U^TA), only the mesh points from a close neighborhood of the p irregular mesh points are involved in the computation. The values on the rest of the mesh points are set to zero. In (3.5) x_1 and x_4 are sources (s), and x_2 and x_5 targets (t) of the separable Helmholtz solver, in accordance with the notation introduced in Section 4.

A straightforward count for the Dirichlet problem gives $s < 5p$ for x_4 ($t < 5p$ for x_2) and $s=3p$ for x_1 ($t=3p$ for x_5). For the Neumann problem the corresponding values differ for x_1 ($s=p$) and x_5 ($t=p$), as we have here a single layer of charges instead of dipoles.

We can observe that the coordinates of the mesh points involved in computation are often repeated as we go from one irregular mesh point to the next. For example, for U^TA

only the layer of mesh points inside Ω at a distance not larger than $2h$ from the boundary $\partial\Omega$ is used in computation. This gives the value $s + t < 5p$ for the Dirichlet problem and $s + t \simeq 3p$ for the Neuman problem. A comparison with formula (4.4) shows that the use of Helmholtz solver with sparsity instead of a fast one should be favorable here also in the amount of computational effort.

When we use a two-dimensional array of entries the summation over the same coordinates (here double indexes) comes in a natural way. On the contrary, while using the Helmholtz solver described in Section 4, we work only with vectors of values and of coordinates of the entries. That is why we must construct an algorithm to recognize entries with the same coordinates in an effective way. To perform such a search in each conjugate gradient iteration would be costly. Therefore, in our computer implementation we preprocess the information about the irregular mesh points and their neighbors. It is performed only once at a cost proportional to the execution of p^2 logical IF statements. This constitutes only a small part of the total computational effort; less than 2.5% for meshes $n \geq 64$. Additional storage for two vectors of an approximate total length of $5p$ is required.

We now briefly repeat the implicit capacitance algorithm:

(a) Compute g_1 ,

(b) Compute $u_s = Gf$,

and $g = g_1 - u_s$ for $x \in \delta\Omega_h$,

(c) Solve for μ: $C^T C\mu = C^T g$,

(d) Compute $u = G(f + D\mu)$.

The capacitance matrix equation for the Dirichlet problem, in its normal form (c) can be solved at the cost of $2k\cdot(15np + 4mn)$ operations, where k is the number of conjugate gradient iterations, while using only 32p memory locations. For the Neumann problem the corresponding values are $2k\cdot(9np+4mn)$ and 25p.

Thus, if m=n, the cost of the main part of our Helmholtz solver is proportional to n^2, as $p = 0(n)$. This conjecture is fairly well confirmed by the experimental data in Table I.

Assume for the moment that we solve only the Laplace equation. Then the total memory requirements for the present algorithm are proportional to p. This allows us to use a very fine mesh or to employ a computer with a small core storage.

In a general case, i.e., when $f \neq 0$, we must also compute the term u_s denoted as the space potential in Section 2. There we have both s and t equal to almost m.n, and the sparse Helmholtz solver is quite ineffective (the operation count is proportional to $m\cdot n^2$); hence in most cases it cannot be recommended.

On the other hand a standard fast solver requires m·n memory locations, i.e., much more than needed for the rest of our algorithm. To resolve the last difficulty we designed a fast Helmholtz solver that requires $2nm^{1/2}$ memory locations, described in Section 6.

In the only process where a repetitive use of a Helmholtz solver is needed, (c), both sources and targets are sparse, as we have already seen. Therefore, a certain increase of computation time in (d) and partially in (b) in order to save storage (see Section 6) plays a lesser role in the total computational effort. Moreover, we can easily use large core memory (LCM) for the two-dimensional arrays needed in (d) and (b). LCM will be here accessed infrequently and therefore at a comparatively low cost. For CDC computers we can take advantage of the use of an inexpensive routine, MOVLEV, to manipulate the storage between SCM and LCM.[1]

There are also applications where the solution or its gradient are required only on $\partial\Omega_h$. In those cases the targets in (d) are also sparse. In order to retain flexibility we implemented the Helmholtz solver that employs sparsity together with the fast solver that uses FFT. Four different options were used for sparse and nonsparse sources and targets.

---

[1]The CDC 7600 has two types of cores storage:
1) SCM contains 65,536 decimal 60-bit words,
2) LCM consist of 512,000 decimal 60-bit words.

6. A METHOD OF SOLVING HELMHOLTZ'S EQUATION IN A RECTANGLE USING $2n^{3/2}$ MEMORY LOCATIONS

Standard fast methods for solving Helmholtz's equation on a rectangular region requires $n \cdot m + 0(1)$ memory locations: This poses a limitation on the size of the mesh if only fast core memory is to be used. If for large n and m a disk was used to store the $n \cdot m$ array, then the standard fast solver requires frequent manipulation to and from the disk with parts of the array. Large access time makes this procedure very expensive.

We now propose a method that somewhat lessens the restriction on the mesh at the expense of an increase in the number of operations.

Let us consider an infinite parallel strip with periodic boundary conditions. Denote by n the number of mesh points across the strip and impose free space boundary conditions. Denote then by m the number of mesh points along the strip. After the change of basis by the FFT on m lines of length n and a suitable permutation, we obtain n tridiagonal systems of equations of order m having the following special structure (see Proskurowski and Widlund [8]:

$$\begin{pmatrix} \mu & -1 & & & 0 \\ -1 & \lambda & -1 & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & \cdot \\ 0 & & & -1 & \lambda \quad -1 \\ & & & & -1 \quad \mu \end{pmatrix} \cdot \hat{x} = \hat{f} \, . \qquad (6.1)$$

For $|\lambda|>2$ we have the following explicit formula for the solution of our problem:

$$\hat{x}_i = \sigma \sum_j \mu^{-|i-j|} \cdot \hat{f}_j \quad , \quad i,j = 1,\ldots,m, \tag{6.2a}$$

where

$$\sigma = (\mu - \mu^{-1})^{-1} \quad ,$$

as can be verified by inspection.

For $|\lambda|=2$ we choose

$$\hat{x}_i = -\tfrac{1}{2} \sum_j (\lambda/2)^{|i-j+1|} \cdot |i-j| \cdot \hat{f}_j \quad , \tag{6.2b}$$

and for $|\lambda|<2$ we choose

$$\hat{x}_i = -\tfrac{1}{2} \sum_j [\sin(|i-j|\cdot\phi)/\sin\phi] \cdot \hat{f}_j \quad , \tag{6.2c}$$

where $\lambda = 2\cos\phi$ (see Proskurowski and Widlund [8]. Note that when $|\lambda|\to 2$ the expression in (6.2c) converges to the one in (6.2b).

We divide the strip lengthwise into k boxes $n\cdot m/k$ and find the solution x on the (k+1) lines connecting the boxes, in accordance with formulas (6.2). Then by taking the inverse FFT on those (k+1) lines, we obtain the solution x on them. We remark that this in itself is a cheap method of computing the solution to Helmholtz's equation on a sparse set of lines. Moreover, the summation in (6.2) needs to be taken only for those j for which $\hat{f}_j$ is nonzero, in the case of sparse data f.

Finally, we consider each $n\cdot m/k$ box as a separate problem with Dirichlet boundary conditions across the strip (the

values of x on (k+1) lines computed within the machine accuracy) and with periodic conditions on the shorter edges. By a suitable choice of the value m/k, we can use a standard fast method for each separate box.

Note that in order to save memory the procedures of computing $\hat{f}$ by taking FFT on each line and of computing x on (k+1) lines must be carried out simultaneously in the same loop for all i = 1,...,m:

(a) take the FFT: $f^{(i)} \to \hat{f}^{(i)}$

(b) for all chosen lines $\ell = 1,\ldots,k+1$
add the term $\hat{f}^{(i)}$ to $\hat{x}^{(\ell)}$, j=1,...,m,

(c) store $\hat{x}^{(i)}$, release $\hat{f}^{(i)}$.

Note also that the data f are required twice (from a function subprogram or from a disk): in step (a) above and while solving problems in each individual box. In both cases values of n consecutive locations of the n·m array are used simultaneously, which simplifies the access from and to a disk. No intermediate results are stored in an auxiliary memory in contrast to a straightforward extension of standard fast solvers to fine meshes.

Storage requirements for this algorithm are as follows: m temporary locations for μ^{-i}, i = 1,...,m-1 and σ, and n locations for $\hat{f}$, which all can be released after the first step performed; n(k+1) locations for the solution x on the

(k+1) chosen lines, and n·m/k locations needed for solving k problems on small boxes. Thus totally we need n·(m/k+k+1) memory locations. The operation count is consequently:

(a) FFT on m lines of length n: $(9/4)\cdot mn\log_2(n/2) + (11/2)mn$,

(b) computing $\hat{x}$ on (k+1) lines: $(k+1)mn$,

(c) inverse FFT on (k+1) lines: $(9/4)\cdot(k+1)n\log_2(n/2) + (11/2)\cdot n(k+1)$,

(d) solving the Dirichlet problem in k boxes:

$$k\left(\frac{9}{2}n\,\frac{m}{k}\log_2\frac{m}{2k} + 15\,\frac{m}{k}\cdot n\right).$$

Thus the total operation count is

$$\psi(m,n,k) = mn\left[\frac{9}{4}\log_2\left(\frac{n}{2}\right) + \frac{9}{2}\log_2\frac{m}{2k} + k + \frac{43}{2}\right]$$

plus a lower-order term. For comparison, the operation count for the standard fast solver is $\theta(m,n) = mn\left(\frac{9}{2}\log_2\frac{n}{2} + 15\right)$.

For $k = m^{1/2}$ the memory requirements are minimal and equal to $n(2m^{1/2} + 1)$. For simplicity, take m = n. Then for the optimal choice of k, equal to $n^{1/2}$, we have the following increase in computational effort

n	$\psi(n)/\theta(n)$	n^2	$2n^{3/2}$	32p
2^8	1.6	2^{16}	2^{13}	2^{14}
2^{10}	1.8	2^{20}	2^{16}	2^{16}
2^{12}	2.2	2^{24}	2^{19}	2^{18}

We tabulated also the memory requirement for the standard and present fast solvers, and the ones for the capacitance matrix iterations, i.e., n^2, $2n^{3/2}$, and 32p (p = 2n), respectively. The last two values are of the same order of magnitude for these meshes.

This solver has not yet been tested experimentally.

7. NUMERICAL EXPERIMENTS

In this section we report on results from a series of numerical experiments that were carried out on the CDC 7600 computer at the Lawrence Berkeley Laboratory. In our experiments we have used programs that are now obtainable from LBL's computer library [7].

For studying our capacitance matrix methods we have chosen problems with no discretization error and regions that are circular.

We will be using the following notation:

variant 1 – the capacitance matrix C is generated and factored; the capacitance matrix system is solved by Gaussian eliminations.

variant 2 – C is explicitly generated; the linear system with C is solved by the conjugate gradient method.

variant 3 – an implicit capacitance method; an FFT solver is used in each conjugate gradient iteration.

variant 4 – an implicit capacitance method; a solver that employs sparsity is used in each conjugate gradient iteration.

A comparison of the performance of all four variants is presented in Table II for a comparatively crude mesh, 64×64 points. Memory requirements for arrays in this case are roughly 24,000 locations for variants 1 and 2, while only 5,000 to 8,000 locations are needed for variants 3 and 4. If the

capacitance matrix fits into the fast memory, the variant 2 is the fastest one. The only exception is variant 4 for the Neumann problem where the number of sources and targets is quite low. If a repetitive use of a solver for a given geometry is needed (for instance in nonseparable problems — see Concus, Golub, and O'Leary [2], or in eigenvalue problems — see Proskurowski [6], variant 1 is recommended). Variant 1 is also superior when a very high accuracy of the solution is required. On the other hand one must keep in mind that most often the discretization error is larger than $1 \cdot 10^{-6}$. To obtain the solution with normalized ℓ_2-norm of the error $||\varepsilon||_2 = 1/n^{1/2}(\sum_i \varepsilon_i^2)^{1/2}$, of $1 \cdot 10^{-3}$ only six iterations for variants 2, 3 and 4 were needed with a subsequent saving in execution time down to 0.65 ÷ 0.71 sec for all these solvers.

In Table III we compare the separable solvers (on rectangular regions) we have used; the one using FFT and the other one with sparse sources and targets. For the latter we confined ourselves to the mesh of powers of 2 only for reasons of comparison. Note also that the number of sources (s) and targets (t) given here is typical for variant 4 and corresponds to experiments shown in Table I. Results of experiments with refined mesh confirm our conjecture that the execution time for the sparse solver is

proportional to n(s+t) and thus is essentially of the order of n^2 for variant 4, while the corresponding growth factor for variant 3 is $n^2 \log_2 n$.

The results given in Table I show that the number of conjugate gradient iterations remains almost constant when the mesh is refined. We can also see clearly that the execution time per iteration is proportional to n^2 for variant 4 and to $n^2 \log_2 n$ for variant 3. Hence, the total computational effort behaves similarly; when the number of inner points grows by a factor of 4.0 the total CPU time grows as 4.4 for variant 4 and 5.2 for variant 3.

We have also run experiments on very fine meshes for a Laplace equation using variant 4. The solution was obtained on a sparse set of mesh points close to $\partial\Omega$. We report here on one of these experiments with n = 650, p = 1468, s+t = 6728, and 212,201 mesh points inside Ω. The number of conjugate gradient iterations was 18 (the ℓ_2-norm of the error $3 \cdot 10^{-6}$) and the total execution time was 157.659 sec on CDC 7600 with the FTN4 (opt=2) compiler. Memory space needed for arrays here was ∿47,000 locations.

Additionally, we chose a circular region with a circular hole in its center. The diameter of the hole was 1.4 of the diameter of the outer circle. The rate of

convergence of the conjugate gradient iterations was in this case slower than for plain circles. Nevertheless, the influence of mesh refinement was insignificant; see Table IV. This seems to support our conviction that such behavior is to be expected for smooth regions, (i.e., without cusps, slits, etc.) of a different shape.

ACKNOWLEDGMENTS

I would like to thank Olof Widlund for his consistent support and advice throughout my work, Alexandra Banegas and Dianne P. O'Leary for making available early versions of their programs, and Paul Concus for comments on the manuscript.

REFERENCES

1. Banegas, A. Fast Poisson Solvers for Problems with Sparsity. (to be published).

2. Concus, P., Golub, G.H., and O'Leary, D.P. A Generalized Conjugate Gradient Method for the Solution of Elliptic Partial Differential Equations. In Proc. Symp. on Sparse Matrix Computation, Sept. 1975. J.R. Bunch and D.J. Rose, Eds., Academic Press, New York, 1976.

3. George, J.A. The Use of Direct Methods for the Solution of the Discrete Poisson Equation on Non-Rectangular Regions. Computer Science Dept. Report 159, Stanford University, 1970.

4. Hayes, R.M. Iterative Methods of Solving Linear Problems in Hilbert Space, Contributions to the Solution of Systems of Linear Eqns. and the Determination of Eigenvalues. O. Taussky, Ed. Nat. Bur. St. Appl. Math. Ser. 39 (1954), 71-103.

5. O'Leary, D.P. and Widlund, O. ERDA-NYU Report. (to be published).

6. Proskurowski, W. On the Numerical Solution of the Eigenvalue Problem of the Laplace Operator by a Capacitance Matrix Method. Lawrence Berkeley Laboratory Report 5396 (1976).

7. Proskurowski, W. Lawrence Berkeley Laboratory Computer Library Writeup (in preparation).

8. Proskurowski, W. and Widlund, O. On the Numerical Solution of Helmholtz's Equation by the Capacitance Matrix Method. Math. Comp. 30 (1976) 433-468.

9. Shieh, A. Fast Poisson Solver on Nonrectangular Domains. New York U., Ph.D. Thesis, June 1976.

10. Widlund, O. On the Use of Fast Methods for Separable Finite Difference Equations for the Solution of General Elliptic Problems. In Sparse Matrices and their Applications, D.J. Rose and R.A. Willoughby, Eds., Plenum Press, New York 1972.

11. Widlund, O. Capacitance Matrix Methods for Helmholtz's Equation on General Bounded Regions. In Proceedings from a meeting in Oberwolfach, Lecture Notes in Mathematics (July 1976), Springer-Verlag (to be published).

Table I. Influence of mesh refinements on the number of iterations and the execution time (CDC 7600 with FTN4 (OPT=2) compiler). The region was a circle.

	Variant 3 Dirichlet problem		Variant 4 Dirichlet problem			Variant 4 Neumman problem
Mesh	64×64	128×128	64×64	128×128	256×256	64×64
Number of boundary points	132	268	132	268	540	132
Number of mesh points inside Ω	1789	7209	1789	7209	28913	1789
Number of conjugate gradient iterations	12	14	13	15	16	8
ℓ_2-norm of the error	$2.3 \cdot 10^{-6}$	$2.0 \cdot 10^{-6}$	$1.0 \cdot 10^{-6}$	$1.0 \cdot 10^{-6}$	$1.3 \cdot 10^{-6}$	$1.2 \cdot 10^{-6}$
Total execution time, sec	1.243	6.473	1.232	5.470	24.158	0.602
Share of separable solvers	1.155	6.087	1.095	5.038	22.941	0.536
Time per iteration	0.090	0.432	0.085	0.324	1.290	0.061

Table II. CPU-time on CDC 7600 with FTN4 compiler (OPT=2) for all our solvers. The region was a circle, the mesh 64x64. There were 132 boundary mesh points and 1789 mesh points inside Ω.

	Variant 1 Dirichlet	Variant 2 Dirichlet	Variant 3 Dirichlet	Variant 4 Dirichlet	Variant 4 Neumann
Generation of C	0.455[a]	0.445	-	-	-
Factorization of C	0.602	-	-	-	-
Total execution time, sec	1.180[a]	0.855	1.243	1.147	0.602
Number of iterations	-	12	12	12	8
ℓ_2-norm of conjugate gradient residuals	-	$0.3 \cdot 10^{-6}$	$0.8 \cdot 10^{-6}$	$1.0 \cdot 10^{-6}$	$0.1 \cdot 10^{-6}$
ℓ_2-norm of the error	$1.2 \cdot 10^{-12}$	$0.8 \cdot 10^{-6}$	$2.3 \cdot 10^{-6}$	$3.2 \cdot 10^{-6}$	$1.2 \cdot 10^{-6}$
Time per iteration	-	0.033	0.090	0.085	0.061
Time to solve an additional problem	0.138	0.410	1.243	1.116	0.582

[a]Using single-layer Ansatz, one can generate C in 0.165 sec decreasing the total execution time to 0.905 sec.

Table III. CPU-time on CDC 7600 with FTN4 compiler (OPT=2) for solvers on rectangular regions.

	Mesh	Execution time in sec	Number of sources and targets
Solver using FFT	64 × 64	0.043	-
	128 × 128	0.203	-
	256 × 256	1.235[a]	-
Solver employing sparsity	64 × 64	0.27[b]	392
	64 × 64	0.385	600
	128 × 128	0.156	1224
	256 × 256	0.622	2472

[a]Computed while using LCM, which slows down the solver by 25-30%.

[b]This result is for the Neumann problem, all the rest for the Dirichlet problem.

Table IV. Influence of mesh refinements on the number of conjugate gradient iterations for a circular region with a circular hole in its center.

Mesh	16 × 16	32 × 32	64 × 64
Number of boundary points	44	84	168
Number of mesh points inside Ω	100	412	1680
Number of conjugate gradient iterations	16	16	16
ℓ_2-norm of conjugate gradient residuals	$1.7 \cdot 16^{-6}$	$1.7 \cdot 10^{-6}$	$1.6 \cdot 10^{-6}$

TECHNICAL INFORMATION DIVISION
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720