EFFICIENT ITERATIVE SOLUTION OF THE THREE-DIMENSIONAL HELMHOLTZ EQUATION

HOWARD C. ELMAN* AND DIANNE P. O'LEARY†

Abstract. We examine preconditioners for the discrete indefinite Helmholtz equation on a three-dimensional box-shaped domain with Sommerfeld-like boundary conditions. The preconditioners are of two types. The first is derived by discretization of a related continuous operator that differs from the original only in its boundary conditions. The second is derived by a block Toeplitz approximation to the discretized problem. The resulting preconditioning matrices allow the use of fast transform methods and differ from the discrete Helmholtz operator by an operator of low rank. We present experimental results demonstrating that when these methods are combined with Krylov subspace iteration, convergence rates depend only mildly on both the wave number and discretization mesh size. In addition, the methods display high efficiencies in an implementation on an IBM SP-2 parallel computer.

Key words. Helmholtz equation, preconditioning, iterative methods, parallel, fast transform methods.

AMS(MOS) subject classifications. 65F10, 65N22, 78A40, 65Y05.

1. Introduction.

The problem considered in this paper is to compute the numerical solution of the Helmholtz equation

$$-\Delta u - k^2 u = f. \tag{1}$$

This equation arises in numerous physical applications [9, pp. 640ff]. Here we consider a three-dimensional box-shaped domain $\Omega = (a_1, b_1) \times (a_2, b_2) \times (a_3, b_3) \subset \Re^3$, with Sommerfeld-like boundary conditions

$$u_n - iku = 0 \tag{2}$$

on $\partial\Omega$, which constitute an approximation to the Sommerfeld radiation condition

$$\lim_{r \to \infty} r(u_n - iku) = 0, \tag{3}$$

used in models of acoustic scattering [17].

Discretization of the problem (1)–(2) results in a linear system of equations

$$Au = f. \tag{4}$$

Since the problem is fully three-dimensional, any reasonable discretization will contain a large number of unknowns and require considerable storage. Direct methods based on Gaussian elimination with partial pivoting require a prohibitive amount of additional storage and thus have limited use. Multilevel methods suffer from the requirement

* Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, e-mail: elman@cs.umd.edu. This work was supported by the U. S. Office of Naval Research under grant N000149510338 and by the U. S. National Science Foundation under grant DMS-9423133.

† Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, e-mail: oleary@cs.umd.edu. This work was supported by the U. S. Office of Naval Research under grant N000149510338 and by a General Research Board Grant from the Office of Graduate Studies and Research of the University of Maryland.

that coarse spaces used must be fine enough to accurately represent the solution; see e.g. [5, 7, 19]. In addition, the complex symmetric coefficient matrix A typically has eigenvalues with both positive and negative real parts. This can cause difficulties for iterative solution methods, and preconditioning of the matrix is essential in order to attain efficiency.

In this paper, we propose solving the discrete Helmholtz equation using Krylov subspace iterative methods with a preconditioning methodology derived from fast direct methods. The basic principle behind fast direct solvers is to apply an inexpensive transformation to break a problem into a number of lower-dimensional but independent problems. Many solvers use fast Fourier transforms (FFT's) to achieve separation of variables and then solve the resulting set of decoupled problems using sparse matrix methods. Fast direct methods are standard tools for solving the Poisson equation on regular domains with Dirichlet, Neumann or periodic boundary conditions [6]; they can be adapted to other domains via capacitance matrix or embedding methods [14, 22]. They have been used for the three-dimensional Helmholtz equation with Dirichlet or Neumann boundary conditions on an irregular domain [21], and for the two-dimensional problem in polar coordinates with nonreflecting boundary conditions [18] (derived from a Dirichlet-to-Neumann mapping) in [11, 15]. In this work, we develop efficient solvers for problems with Sommerfeld-like boundary conditions on box-shaped domains. Combining our techniques with capacitance matrix methods would produce solvers for general geometries in Cartesian coordinates, including exterior problems.

Our idea generalizes some results developed for two-dimensional Helmholtz problems by Ernst and Golub [12]. (See also [8, §4] for variants applied to definite elliptic problems.) We approximate the discrete operator A with a matrix Q that can be treated with fast direct methods. For finite difference discretizations, we derive Q by defining and discretizing the differential operator in the same way as for A except that the boundary conditions on either two or four faces of Ω are replaced by more convenient ones (Dirichlet or Neumann). The resulting matrix Q differs from A by a (relatively) low-rank operator and can be used as a preconditioner for A, to accelerate the convergence of iterative solvers based on Krylov subspaces. We also develop variants of these ideas for finite element discretizations (on uniform grids), focusing on trilinear elements. Here, rather than explicitly modifying the boundary conditions to constuct Q, we use the fact that the discrete operator A is close to a block Toeplitz matrix and replace certain sub-blocks of A by Toeplitz approximations that are amenable to fast transforms. For both types of discretizations, we will demonstrate empirically that Q meets the requirements for an effective preconditioner:

- Applying the action of Q^{-1} to a vector is not too expensive. For our preconditioners, using Q^{-1} entails a set of FFT's together with solution of smaller dimensional problems (see Section 2).
- Q greatly reduces the number of iterations needed by Krylov subspace methods to solve (4).

In particular, we will show that for several choices of Q, the experimental convergence behavior of preconditioned restarted GMRES [23] depends only mildly on both the wave number k and the discretization mesh size. In addition, we will demonstate how the methods can be implemented on a parallel computer with high efficiency.

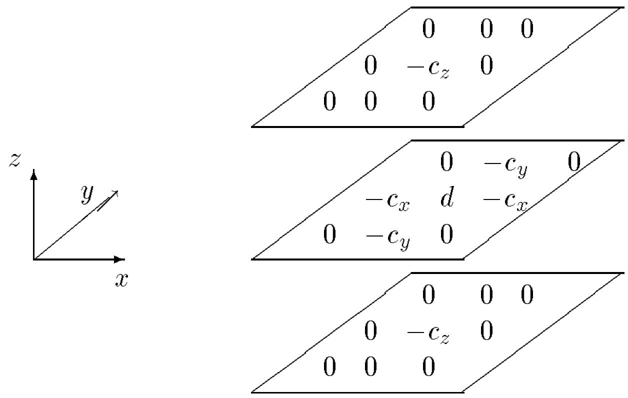
The paper is organized as follows. In Section 2, we show how fast transform methods can be used to generate preconditioning operators for finite difference discretizations of the Helmholtz equation and we develop variants applicable to low-order finite element discretizations, using trilinear elements as a specific example. In Section 3, we present the results of a series of numerical experiments demonstrating the performance of the preconditioners. In most cases, there is virtually no increase in iteration counts as the mesh size is refined for fixed wave numbers; especially for finite differences, there is only slight dependence on the wave numbers. In addition, we show that the new methods are more effective than a standard algebraic preconditioner based on symmetric successive overrelaxation (SSOR) [27]. In Section 4, we show how the methods can be implemented on parallel computers, and we demonstrate their parallel efficiency using experiments on a sixteen processor IBM SP-2. Finally, in Section 5, we make some concluding remarks.

2. The preconditioners. Good preconditioners for Krylov subspace iterations can be determined in two ways:

- preconditioners derived from operators related to the desired operator.
- preconditioners derived from matrices related to the desired matrix.

We will use both approaches in our work. For simplicity, we restrict our attention to $\Omega = (0,1)^3$, the unit box.

2.1. Preconditioners for finite difference discretizations. Given positive integers m_x, m_y, and m_z, let problem (1)–(2) be discretized by the seven-point (second order accurate) finite difference operator on a uniform mesh with cells of size $h_x \times h_y \times h_z$, where $h_x = 1/(m_x+1)$, $h_y = 1/(m_y+1)$, $h_z = 1/(m_z+1)$. Assume that the normal derivatives in the boundary conditions are approximated by one-sided differences and that the discrete boundary conditions are used to eliminate all unknowns on $\partial\Omega$. The resulting "stencil" at interior grid points is



where $c_x = \frac{1}{h_x^2}$, $c_y = \frac{1}{h_y^2}$, $c_z = \frac{1}{h_z^2}$, and $d = 2(c_x + c_y + c_z) - k^2$. The figure depicts the contributions to the stencil in a given x-y plane together with the contributions in the two neighboring x-y planes in the z-direction. For points adjacent to the x boundaries, the value

$$\gamma_x \equiv \frac{c_x(1 + ikh_x)}{1 + k^2h_x^2}$$

is subtracted from the center value of the stencil, and similarly for the y and z boundaries.

If the matrix problem is formed by ordering the unknowns by lines within the x-y planes, the resulting matrix A is block tridiagonal and can be written in tensor product form as

$$A = I_{m_z} \otimes I_{m_y} \otimes T^{(x)}_{m_x} + I_{m_z} \otimes T^{(y)}_{m_y} \otimes I_{m_x} + T^{(z)}_{m_z} \otimes I_{m_y} \otimes I_{m_x} - k^2 I_{m_x m_y m_z} .$$

Here, I_m denotes the identity matrix of size m, and

$$T^{(x)}_{m_x} = T_{m_x} + \gamma_x E_{m_x}$$

has size m_x with

$$T_{m_x} = c_x \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}, \quad E_{m_x} = \begin{bmatrix} 1 & & & & \\ & 0 & & & \\ & & \ddots & & \\ & & & 0 & \\ & & & & 1 \end{bmatrix} .$$

The matrices $T^{(y)}_{m_y}$ and $T^{(z)}_{m_z}$ are defined analogously.

It was shown by Ernst and Golub [12] that for the Helmholtz equation on a two-dimensional rectangular domain, an effective preconditioner can be devised by replacing Sommerfeld-like boundary conditions on two opposite edges by Dirichlet or Neumann conditions. Let us extend this idea to three-dimensional problems. If we replace the boundary conditions (2) at $x = 0$ and $x = 1$ by homogeneous Dirichlet conditions $u = 0$, then the problem is separable in the x-direction. Discretization yields a matrix

$$Q_d = I_{m_z m_y} \otimes T_{m_x} + P \otimes I_{m_x} , \tag{5}$$

where

$$P = I_{m_z} \otimes T^{(m_y)}_y + T^{(m_z)}_z \otimes I_{m_y} - k^2 I_{m_z m_y} .$$

The eigenvectors of T_{m_x} are the columns of the orthogonal matrix U_s corresponding to a discrete sine series representation. Therefore, the product $v = U_s^T w$ can be formed by computing the discrete Fourier sine transform of the vector w, and $w = U_s v$ is the inverse transform of the vector v. If D_s is the diagonal matrix of eigenvalues of T_{m_x}, then Q_d can be represented as

$$Q_d = (I_{m_z m_y} \otimes U_s)(I_{m_z m_y} \otimes D_d + P \otimes I_{m_x})(I_{m_z m_y} \otimes U_s^T) . \tag{6}$$

Reordering the rows and columns of the matrix $I_{m_z m_y} \otimes D_d + P \otimes I_{m_x}$ in (6) by lines within each y-z plane yields a matrix $\mathcal{P}$ containing m_x diagonal blocks, each with the same nonzero structure as a five-point finite difference discretization of a two-dimensional problem.

Using Q_d as a preconditioner entails applying the action of Q_d^{-1} to a vector v at each step of an iterative algorithm. The discussion above shows that this computation can be performed in the following sequence of steps:

1. Perform $m_x m_y$ sine transform operations in the x coordinate directions to compute $v_1 = (I_{m_z m_y} \otimes U_s^T)v$.
2. Solve m_x two-dimensional problems, one in each y-z plane, to compute v_2.
3. Perform $m_x m_y$ inverse sine transform operations in the x coordinate directions to compute $w = (I_{m_z m_y} \otimes U_s)v_2$.

The solution of the two-dimensional problems required in Step 2 can be done using a variety of techniques, including general sparse direct methods [10], band solvers, and domain decomposition.

Consider a variant of this preconditioner, derived using Dirichlet boundary conditions at two pairs of opposite faces of $\partial\Omega$: $x = 0, 1$ and $y = 0, 1$. The resulting preconditioning matrix is

$$(7)\ Q_{dd} = I_{m_z} \otimes I_{m_y} \otimes T_{m_x} + I_{m_z} \otimes T_{m_y} \otimes I_{m_x} + T_{m_z}^{(z)} \otimes I_{m_y} \otimes I_{m_x} - k^2 I_{m_x m_y m_z}\,.$$

A set of sine transforms in the x-direction still decouples the problem into m_x two-dimensional subproblems as in (6), but now if this is followed by a set of sine transforms in the y-direction, the result is $m_x m_y$ independent one-dimensional (tridiagonal) problems. Step 2 of the computation of the action of Q_{dd}^{-1} then has the following form:

2a. Perform $m_z m_x$ inverse sine transform operations in the y coordinate directions.

2b. Solve $m_x m_y$ tridiagonal systems in the z coordinate directions.

2c. Perform $m_z m_x$ sine transform operations in the y coordinate directions.

We will consider four preconditioning operators of this type:

- Q_d derived from Dirichlet boundary conditions at $x = 0, 1$.
- Q_{dd} derived from Dirichlet boundary conditions at $x = 0, 1$ $y = 0, 1$.
- Q_n derived from homogeneous Neumann boundary conditions at $x = 0, 1$. The fast solver here involves discrete cosine transforms and solution of m_x two-dimensional problems.
- Q_{nn} derived from homogeneous Neumann boundary conditions at $x = 0, 1$, $y = 0, 1$. The fast solver involves discrete cosine transforms in two directions plus solution of $m_x m_y$ one-dimensional (tridiagonal) problems.

If m_x and m_y are powers of two, then the time required to compute $Q_{dd}^{-1}v$ or $Q_{nn}^{-1}v$ is proportional to $m_x m_y m_z(\log m_x + \log m_y + 1)$. For $Q_d^{-1}v$ or $Q_n^{-1}v$, the time is proportional to $m_x m_y m_z \log m_x$ plus the time to solve the two-dimensional problems.

Clearly, it is possible to derive other variants of these ideas based on other boundary conditions. As long as the Sommerfeld-like boundary conditions are retained in at least one coordinate direction, the preconditioning operators that use any combination of Dirichlet and Neumann boundary conditions are all nonsingular for any value of k.

2.2. Preconditioners for a finite element discretization. The weak formulation of the Helmholtz equation (1)–(2) is to find $u \in \mathcal{S}$ such that

$$a(u, v) = (f, v) \qquad \text{for all } v \in \mathcal{S},$$

where

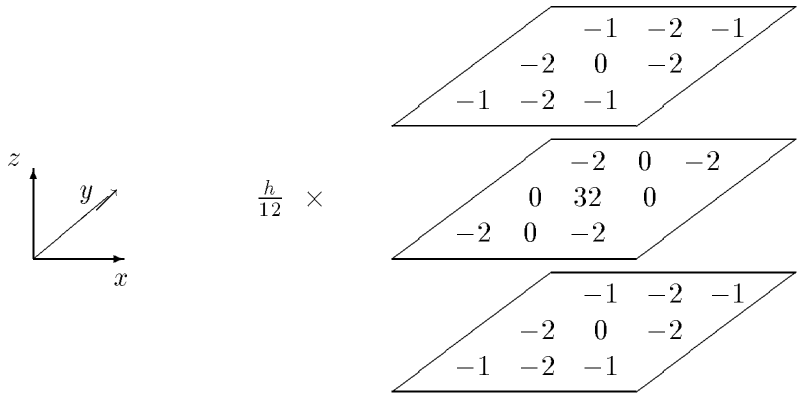
$$(8)\quad \begin{aligned} a(u, v) &= \int_\Omega (\nabla u \cdot \nabla \bar{v} - k^2 u\bar{v}) - ik\int_{\partial\Omega} u\bar{v} \\ (f, v) &= \int_\Omega u\bar{v} \end{aligned}$$

and $\mathcal{S}$ is an appropriate Sobolev space (often, $H^1(\Omega)$), depending on f. Note that the boundary conditions (2) are explicitly incorporated into the weak form.

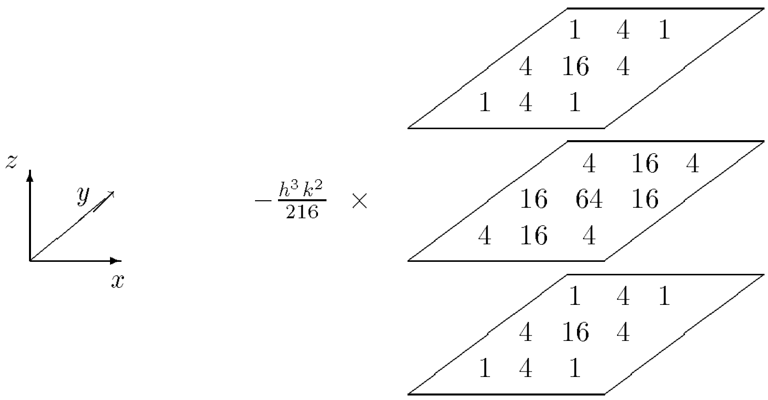
Let $\mathcal{S}_h$ denote the finite dimensional subspace of $\mathcal{S}$ determined from continuous piecewise trilinear basis functions on rectangular elements. The discrete weak form is to find $u_h \in \mathcal{S}_h$ such that

$$a(u_h, v) = (f, v) \qquad \text{for all } v \in \mathcal{S}_h.$$

Assuming uniform cells of size $h \times h \times h$ and using a natural ordering of unknowns, the resulting coefficient matrix A again has block tridiagonal structure. (Different values of h can be used in each coordinate direction; we restrict our attention to cubic cells only to simplify the notation.) We describe it using its stencil at interior mesh points, which consists of the contribution from the stiffness matrix (from $(\nabla u \cdot \nabla v)$),



together with the contribution from the mass matrix (from $k^2(u, v)$),



For the mesh points on the boundary, fewer elements contribute to the stiffness and mass matrices and the stencils are somewhat different. We omit a detailed description but observe that contributions from the boundary integral in (8) are pure imaginary since the basis functions are real. The matrix A has size m^3 where now $h = 1/(m-1)$. (This is slightly different from the relation between mesh size and number of grid points for finite differences, because here the unknowns on the boundary are included in the system.)

To develop preconditioners, one alternative is to simply use the matrices derived above for finite differences. That is, given a finite element grid with m^3 unknowns, let Q_d, Q_{dd}, Q_n, and Q_{nn} be the preconditioning matrices of the same size defined in Section 2.1.

An alternative and somewhat more successful approach is to derive preconditioners with tensor product and Toeplitz structure that match the finite element matrix except at the boundary. First, we recall that the sine transform diagonalizes matrices of the form

$$S_{\alpha,\beta} \equiv \alpha I + \beta \begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & 1 & 0 & 1 \\ & & & & 1 & 0 \end{bmatrix} \tag{9}$$

where α and β are arbitrary scalars.

Second, we observe that the finite element matrix is close to a matrix with tensor product structure. The contribution of the mass matrix can be expressed as $h^3k^2/216$ times

$$-S_{4,1} \otimes S_{4,1} \otimes S_{4,1} + B_m \, ,$$

where B_m is nonzero only in rows corresponding to boundary points. Similarly, the stiffness matrix is $h/12$ times

$$-S_{0,1} \otimes S_{2,1} \otimes S_{2,1} + 4S_{0,1} \otimes I \otimes I - I \otimes S_{0,1} \otimes S_{0,2} + 32\, I \otimes I \otimes I + B_s \, ,$$

where B_s is nonzero only in rows corresponding to boundary points.

Let us define a matrix $\widehat{Q}_{dd}$ that matches the finite element matrix in all rows except those corresponding to the x and y boundaries; in those rows, we simply neglect the contributions from B_m and B_s. The resulting preconditioner differs from the finite element matrix by a matrix of rank $4(m^2 - m)$. Since we have omitted the nonzeros in B_m and B_s corresponding to the x and y boundaries, and since each of the other matrices in the tensor product representation is diagonalized by the matrix U_s corresponding to the discrete sine transformation, we have an easy way to form the product $\widehat{Q}_{dd}^{-1} v$:

1. Perform m^2 sine transform operations in the x coordinate directions to compute $v_1 = (I_{m^2} \otimes U_s^T)v$.
2. Solve m two-dimensional problems, one in each y-z plane, to compute v_2:
   - 3a. Perform m^2 sine transform operations.
   - 3b. Solve m^2 tridiagonal systems.
   - 3c. Perform m^2 inverse sine transform operations
3. Perform m^2 inverse sine transform operations in the x coordinate directions to compute $w = (I_{m^2} \otimes U_s)v_2$.

A preconditioner $\widehat{Q}_{nn}$ can be defined in an analogous way, by changing the rows corresponding to the x and y boundaries in a way so that the discrete cosine transformation diagonalizes the resulting matrices in the tensor product formulation. Since the cosine transformation diagonalizes matrices of the form

$$C_{\alpha,\beta} \equiv \alpha I + \beta \begin{bmatrix} -1 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & 1 & 0 & 1 \\ & & & & 1 & -1 \end{bmatrix},$$

we choose α and β in the same way as for $S_{\alpha,\beta}$ in order to match the interior stencil coordinates. The resulting preconditioner differs from A in the same rows as Q_{nn}.

REMARK 2.1. It is also possible to define matrices $\hat{Q}_d$ and $\hat{Q}_n$ that match A except in rows corresponding to just one opposite pair of boundaries, as in the development of Q_d and Q_n above. Because these types of preconditioners were more costly for the finite difference examples (see Section 4), we did not implement these variants.

3. Experimental results. In this section, we present the results of numerical experiments in which the preconditioners described in Section 2 are used with iterative method GMRES(20) [23] to solve the discrete Helmholtz equation. Our concern here is the effect of mesh size and wave number on performance. Additional results showing behavior on a parallel computer are given in Section 4. In all tests, the linear system (4) is defined by choosing a discrete solution u, and the right hand side is then computed as $f = Au$. We consider two discrete solutions:

1. a vector with real and imaginary parts consisting of uniformly distributed random numbers in the interval $[-1, 1]$;
2. a smooth vector whose value at the mesh point with index (j, k, l) is $10a_{jkl} - i\, a_{jkl}$, where $a_{jkl} = 10,000j + 100k + l$.

The first case is designed to show the behavior for problems with non-smooth solutions, and the second case for problems with smooth solutions.

The iterative solver is used with right-oriented preconditioning; that is, GMRES is formally applied to the preconditioned system

$$AQ^{-1}\hat{u} = f, \quad u = Q^{-1}\hat{u}.$$

This ensures that the norm minimized by GMRES is independent of the choice of the preconditioner. The iteration is stopped when

$$\frac{\|r_j\|_2}{\|f\|_2} < 10^{-5}$$

where $r_j = f - Au_j$ is the residual for the iterate u_j, $u_0 \equiv 0$, and the norm is the usual vector Euclidean norm. All computations with these preconditioners were performed on an IBM SP-2 computer in double precision. (See Section 4 for details.)

We use a set of uniform three-dimensional grids of size $m \times m \times m$ for $20 \le m \le 80$, together with a variety of wave numbers k. For any k, accurate discrete solutions of (1) will be obtained only if the mesh is fine enough to resolve the features of the problem. A commonly used criterion is for the mesh to include at least ten grid points per wave, i.e., to require $k \le 2\pi/(10h)$. See [16] for rigorous justification of this criterion for one-dimensional problems, instead of the more stringent requirement that k^3h^2 be bounded. In the tabulated data shown below, results for problems with at least ten

Table 1

Iteration counts for GMRES(20) with preconditioners that combine one-dimensional transforms with two-dimensional sparse direct solves. Finite differences with non-smooth solution.

Q_d (sine + 2D solves)

	m			
k	20	40	60	80
1	11	11	12	–
5	9	10	11	–
10	9	11	11	–
20	12	12	12	–
30	13	14	14	–
40	12	18	17	–
50	16	19	24	–

Q_n (cosine+2D solves)

	m			
k	20	40	60	80
1	4	3	2	–
5	7	6	5	–
10	10	8	8	–
20	15	14	12	–
30	20	18	18	–
40	31	21	20	–
50	44	26	27	–

Table 2

Iteration counts for GMRES(20) with preconditioners that combine two sets of one-dimensional transforms with one-dimensional tridiagonal solves. Finite differences with non-smooth solution.

Q_{dd} (sine + 1D solves)

	m			
k	20	40	60	80
1	14	15	13	12
5	12	13	14	14
10	13	14	15	15
20	20	25	21	21
30	36	30	29	28
40	34	53	47	45
50	46	76	69	69

Q_{nn} (cosine + 1D solves)

	m			
k	20	40	60	80
1	5	4	3	3
5	10	9	8	8
10	19	17	16	15
20	50	41	38	35
30	87	76	77	69
40	133	96	106	95
50	264	142	174	165

Table 3

Iteration counts for GMRES(20) with preconditioners that combine two sets of one-dimensional transforms with one-dimensional tridiagonal solves. Finite differences with smooth solution.

Q_{dd} (sine + 1D solves)

	m			
k	20	40	60	80
1	16	23	36	42
5	13	18	23	30
10	14	20	26	30
20	18	22	28	32
30	26	37	36	40
40	29	55	55	53
50	24	75	69	81

Q_{nn} (cosine + 1D solves)

	m			
k	20	40	60	80
1	6	6	6	6
5	11	11	11	11
10	17	18	19	18
20	51	57	57	58
30	78	94	101	101
40	101	118	135	137
50	155	171	196	206

grid points per wave lie above the jagged line; data lying below these lines are included only to show trends and do not correspond to physically meaningful computations. Dashed lines (–) correspond to problems that are too large for practical computation on this configuration.

Tables 1 and 2 show results for non-smooth problems and finite difference discretizations. Table 1 shows the iteration counts required by GMRES(20) with the preconditioners that entail a set of trigonometric transforms together with direct solution of two-dimensional subproblems. The entries on the left are for the preconditioner Q_d, which uses sine transforms, and the entries on the right are for Q_n, which uses cosine tranforms. We will refer to these here as the "two-dimensional" solvers. Table 2 shows analogous results for Q_{dd} and Q_{nn}, where the two-dimensional subproblems are also treated using fast transforms; we will refer to these (less costly) preconditioners as the "one-dimensional" solvers. Table 3 shows analogous results for smooth problems. For the sake of brevity, here we only discuss the one-dimensional preconditioners Q_{dd} and Q_{nn}.

The following trends are evident in these tables:

1. For non-smooth problems, iteration counts are insensitive to mesh size. Indeed, the counts often decrease as h decreases. Performance is much better than would be expected from the rank $(m^2 - m)$ of the difference between A and the preconditioner Q.
2. Iteration counts for the methods based on the sine transpose increase very modestly with the wave number k; counts for the cosine transpose are more sensitive to k but they are smaller when k is small.
3. Fewer iterations are required with two-dimensional solvers (for which the boundary conditions determining Q are more like those determining A) than with the one-dimensional solvers. As we will see in Section 4, however, this is at the expense of significant extra work.
4. The counts for smooth problems are somewhat higher than in the non-smooth case. The qualitative dependence of performance on wave number is largely the same, as is the dependence on mesh size, except in the case of small wave numbers with sine transforms; here the iteration counts appear to grow roughly linearly with h^{-1}.

Tables 4, 5 and 6 show the iteration counts for the trilinear finite element discretization. Here we use $\hat{Q}_{dd}$ and $\hat{Q}_{nn}$ for the one-dimensional solvers; these were more effective than the variants Q_{dd} and Q_{nn}. Numbers with an asterisk correspond to the maximum number of iterations permitted. Many of the trends are the same as for finite differences: iteration counts for non-smooth problems are essentially independent of the mesh size; the sine-based methods are generally more effective than the cosine-based methods; and, for smooth problems, costs and dependence of the sine-based method on mesh size are somewhat greater. The sensitivity to wave number is considerably more pronounced than for finite differences, although for the larger values of k considered, the iteration counts decrease as the mesh is refined. This suggests that for these wave numbers the asymptotic behavior (as $h \to 0$) of the solvers is being approached only for the finest meshes considered here. In these tests, performance is less dependent on the smoothness of the solution than in the experiments with finite differences.

In a few tests without preconditioning, GMRES(20) required an average of eight times more steps than with the Q_{dd} preconditioner for non-smooth problems, and

Table 4

Iteration counts for GMRES(20) with preconditioners that combine one-dimensional transforms with two-dimensional sparse direct solves. Trilinear finite elements with non-smooth solution.

Q_d (sine + 2D solves)

	m			
k	20	40	60	80
1	24	25	26	–
5	32	29	28	–
10	48	37	34	–
20	87	65	58	–
30	300*	85	79	–
40	300*	146	89	–
50	300*	300*	158	–

Q_n (cosine + 2D solves)

	m			
k	20	40	60	80
1	23	22	22	–
5	32	30	29	–
10	51	41	39	–
20	85	67	59	–
30	300*	104	89	–
40	300*	245	127	–
50	300*	300*	300*	–

Table 5

Iteration counts for GMRES(20) with preconditioners that combine two sets of one-dimensional transforms with one-dimensional tridiagonal solves. Trilinear finite elements with non-smooth solution.

$\hat{Q}_{dd}$ (sine + 1D solves)

	m			
k	20	40	60	80
1	15	16	16	15
5	17	16	16	16
10	32	25	24	23
20	106	71	58	52
30	198	177	149	114
40	300*	292	222	171
50	300*	300*	300*	300*

$\hat{Q}_{nn}$ (cosine + 1D solves)

	m			
k	20	40	60	80
1	13	12	11	11
5	22	19	18	18
10	41	39	37	37
20	127	140	120	99
30	300*	299	236	206
40	300*	300*	300*	270
50	300*	300*	300*	300*

Table 6

Iteration counts for GMRES(20) with preconditioners that combine two sets of one-dimensional transforms with one-dimensional tridiagonal solves. Trilinear finite elements with smooth solution.

$\hat{Q}_{dd}$ (sine + 1D solves)

	m			
k	20	40	60	80
1	17	22	31	38
5	18	20	26	31
10	29	31	33	37
20	76	66	60	60
30	139	152	159	138
40	300*	234	200	178
50	300*	300*	300*	300*

$\hat{Q}_{nn}$ (cosine + 1D solves)

	m			
k	20	40	60	80
1	12	12	12	12
5	19	19	19	19
10	34	36	36	35
20	82	116	96	94
30	205	200	183	164
40	300*	294	277	225
50	300*	300*	300*	300*

Table 7
Iteration counts of GMRES(20) with SSOR ($\omega = 1$) preconditioning. Non-smooth solutions.

Finite differences

k	m = 20	40	60
1	26	19	20
5	26	28	24
10	37	49	50
20	76	78	87
30	200	103	103

Trilinear elements

k	m = 20	40	50
1	20	19	19
5	17	27	26
10	27	38	42
20	80	61	66
30	300*	90	85

Table 8
Iteration counts of GMRES(20) with SSOR ($\omega = 1$) preconditioning. Smooth solutions.

Finite differences

k	m = 20	40	60
1	52	200	300*
5	40	122	209
10	57	170	249
20	110	159	219
30	237	175	222

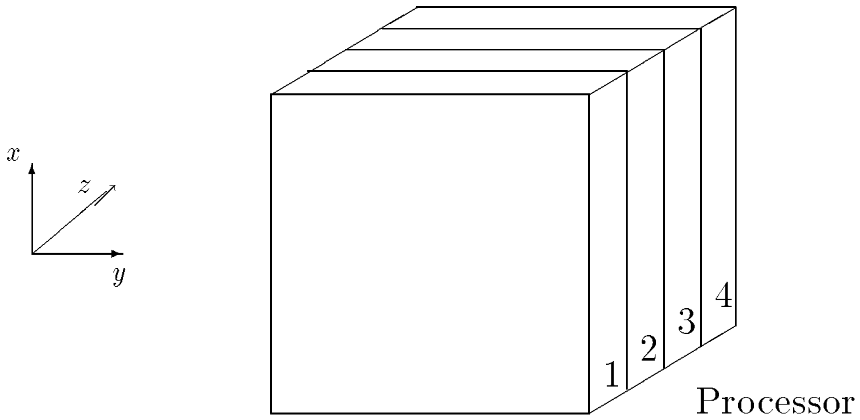
Trilinear elements

k	m = 20	40	50
1	26	99	122
5	24	62	97
10	37	78	113
20	96	99	117
30	300*	122	127

at least fifteen more steps for smooth problems. (These tests were for finite differences, $n = 20$ and 40 and $k \leq 20$; a maximum of 300 steps was used, and for the smooth problems the stopping criterion was not satisfied in a majority of the runs.) To compare the performance of the new preconditioners with a "standard" algebraic preconditioner, we show in Tables 7 and 8 iteration counts obtained using the SSOR preconditioner [27] (with $\omega = 1$) with GMRES(20). This method has been used to good effect for solving the Helmholtz equation with multiple right hand sides (derived e.g., from incident waves at different angles) in [13], cf. also [3]. These tests were run on a uniprocessor Sun SPARCstation 20 in Matlab, and we considered only mesh sizes less than or equal to $m = 60$. (Storage limitations permitted only $m \leq 50$ for trilinear elements.) In almost all cases, these iteration counts are larger than the analogous entries from Tables 1 – 6, and the growth in iteration counts with increasing wave numbers is considerably more pronounced. The differences are especially dramatic for smooth problems. (We also remark, however, that the SSOR method was surprisingly insensitive to mesh size in the non-smooth case. An analysis of behavior like this for a simple elliptic model problem is given in [20].) In addition, these tests were run with a "natural" ordering of the unknowns, which does not have efficient parallel implementation. Tests with point red-black ordering for the finite difference examples (for $n \leq 40$, $k \leq 20$ and non-smooth problems) required on average 65% more steps.

4. Parallel implementation and performance. The algorithms developed in Section 2 are well adapted for parallel computation in both shared memory and message-passing environments. We have implemented them on an IBM SP-2 computer assuming that the number of processors does not exceed m_x or m_z. In this section, we describe the implementation and present the results of experiments show-

FIG. 1. *Partitioning of the three-dimensional mesh among 4 processors.*



ing parallel performance.

4.1. Parallel implementation. For ease of illustration, we describe our implementation under the assumption that there are $p = 4$ processors. The mesh is partitioned among processors as in Figure 1; if m_z is divisible by p then each processor contains m_z/p blocks of $m_x m_y$ grid points oriented along the x-y plane. If m_z is not divisible by p, then the number of x-y planes assigned to each processor differs by at most one.

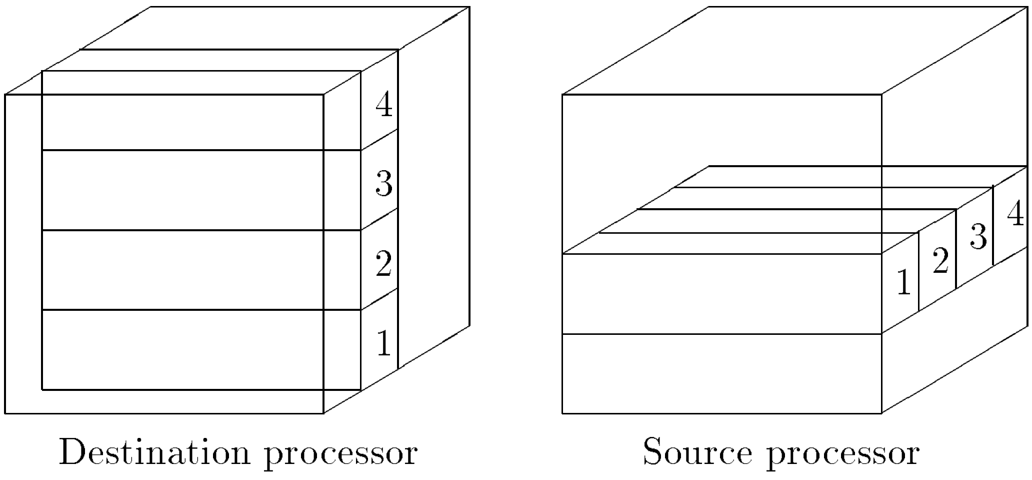
4.1.1. Parallel implementation of the iterative method. We use the original form of the restarted GMRES algorithm, based on the Arnoldi basis, as presented in [23]. With the partitioning described above, the GMRES algorithm is easy to parallelize: each processor is responsible for storing and updating at most

$$\hat{m}_z \equiv \left\lceil \frac{m_z}{p} \right\rceil m_x m_y$$

unknowns and for maintaining the factored form of a Hessenberg matrix of size $r \times r$ (using r steps of GMRES between restarts). Communication is necessary only for inner products and matrix-vector products. The cost per iteration is $O(\hat{m}_z)$ floating point operations plus accumulation of (on average) $(r+3)/2$ sums of scalars across processors, plus the cost of matrix-vector products and preconditioning.

4.1.2. Parallel implementation of matrix-vector products. Computation of products of the finite difference or finite element matrix with a vector requires that each processor send its highest-numbered x-y plane to the processor numbered one greater, and its lowest-numbered x-y plane to the processor numbered one less (if these processors exist). The stencil can then be applied to the local data. The cost per matrix multiply is $O(\hat{m}_z)$ floating point operations plus 2 sends and receives of $m_x m_y$ numbers per processor.

FIG. 2. *Destination and source processors for data movement, from the perspective of Processor 2.*



4.1.3. Parallel implementation of the preconditioning. The preconditioning computation is a somewhat more complex operation. Let

$$\hat{m}_x \equiv \left\lceil \frac{m_x}{p} \right\rceil m_y m_z \ .$$

We arrange the work in the chart below, estimating the cost assuming that m_x and m_y are powers of 2. Note that the costs of $\hat{Q}_{dd}$ and $\hat{Q}_{nn}$ are identical to those of Q_{dd}.

Operation	Cost
Each processor computes sine or cosine transforms in the x direction on its local data.	$O(\hat{m}_z \log m_x)$ operations.
The data is rearranged so that each processor has an approximately equal number of y-z planes. This requires a nontrivial amount of communication: the data movement from the perspective of Processor 2 is shown in Figure 2.	Each processor sends at most $\lceil \frac{m_z}{p} \rceil \lceil \frac{m_x}{p} \rceil m_y$ numbers to every other processor.
Each processor then solves its assigned two-dimensional problems. For preconditioners Q_{dd} and Q_{nn}, this entails sine or cosine transforms in the y direction followed by solution of tridiagonal systems, followed by inverse sine or cosine transforms. For Q_d and Q_n, a two-dimensional problem is solved in each of the y-z planes.	For Q_{dd} and Q_{nn}, the cost is the solution of $\lceil \frac{m_x}{p} \rceil$ problems of size $m_y m_z$. For Q_d and Q_n the cost is $O(\hat{m}_x(\log m_y + 1))$ operations.
The data is rearranged to its original configuration.	Each processor sends at most $\lceil \frac{m_z}{p} \rceil \lceil \frac{m_x}{p} \rceil m_y$ numbers to every other processor.
Each processor computes inverse sine or cosine transforms in the x direction.	$O(\hat{m}_z \log m_x)$ operations.

REMARK 4.1. For $m_x = m_y = m_z = m$, the total arithmetic cost for Q_{dd} or Q_{nn} is proportional to $\frac{m^3 \log m}{p}$, and the communication cost (assuming no contention for

TABLE 9

CPU times for solving the finite difference discretization with $k = 5$, for various grid sizes.

Q_d (sine + 2D solves)

	m		
Number of processors	16	32	64
1	1.89	25.37	–
2	.98	12.75	–
4	.57	6.72	–
8	.43	3.56	–
16	.52	2.14	27.72

Q_{dd} (sine + 1D solves)

	m		
Number of processors	16	32	64
1	1.62	13.12	–
2	.85	6.91	–
4	.47	3.74	28.24
8	.40	2.15	14.55
16	.52	1.37	8.14

messages sent simultaneously) is proportional to $\frac{m^3}{p}$. The communication cost is the same for the two-dimensional direct solvers Q_d or Q_n, and the smallest possible arithmetic cost is $O\left(\frac{m^4}{p}\right)$ if a nested dissection method is used [10]; this strategy would neglect any need for pivoting. For the tests described below, we used a bandsolver (which allows pivoting), resulting in cost proportional to $\frac{m^5}{p}$.

REMARK 4.2. For Q_{dd} and Q_{nn}, data movement can be completely masked by computation of the sine or cosine transforms in the y direction, provided communication speed is not too slow and the hardware supports overlapping of communication and computation. This option is not currently supported on the SP-2.

4.2. Parallel performance. We now describe the performance of the solvers using the transform-based preconditioners on a sixteen processor IBM SP-2 computer, a distributed memory machine with explicit message passing. The system contains sixteen RS6000/390 processing nodes running AIX, interconnected via a proprietary interprocessor communications switch. The computational component of the program was written in Fortran90 and compiled using the mpxlf90 compiler with the optimization (-O) switch. All modules were taken from off-the-shelf software and modified to conform to Fortran90: GMRES is derived from the TEMPLATES package [2]; the two-dimensional direct solvers use the LAPACK [1] bandsolver `cgbsv` and `cgbtrs`, as do the tridiagonal solvers used for the one-dimensional preconditioners; and the sine-transform and cosine-transform routines are from FFTPACK [26]. All tests used double precision complex floating point computations. Communication was performed using MPI [24] with nonblocking sends (MPI_ISEND) and blocking receives (MPI_RECV). Inner products were performed and broadcast using MPI_ALLREDUCE.

The results on parallel performance are summarized in Table 9 for finite differences and Table 10 for trilinear finite elements. Again, dashed lines correspond to problems that were too large. The entries show CPU times for solving the discrete problem with non-smooth solution on various grid sizes with the sine-based preconditioners. For an understanding of parallel performance it suffices to look at just one wave number, which in these cases was $k = 5$; results for the cosine-based preconditioners also lead to the same conclusions. We used grid sizes that are powers of two only for convenience so that the grid parameter m_z divides the number of processors; results when this is not the case are similar. The timings reflect the averages over three runs; these runs were made in time sharing mode and other users had access to the machine.

Table 10

CPU times for solving the trilinear finite element discretization with $k = 5$, for various grid sizes.

Q_d (sine + 2D solves)

Number of processors	m = 16	32	64
1	9.20	91.56	–
2	4.38	45.81	–
4	2.09	24.41	–
8	1.39	12.23	–
16	1.20	6.67	70.59

Q_{dd} (sine + 1D solves)

Number of processors	m = 16	32	64
1	3.41	24.82	–
2	1.71	12.70	–
4	.80	7.03	49.34
8	.65	3.71	25.10
16	.80	2.59	15.21

Table 11

Speedups for $m = 32$.

Number of processors	Fin. Diff. Q_d	Fin. Diff. Q_{dd}	Fin. Elem. Q_d	Fin. Elem. $\hat{Q}_{dd}$
2	1.99	1.90	2.00	1.95
4	3.78	3.51	3.91	3.53
8	7.13	6.10	7.49	6.69
16	11.86	9.58	13.73	9.58

It is evident from these data that all the methods display a large amount of parallelism. The average speedups in going from p to $2p$ processors for $m = 32$ is 1.77 for the one-dimensional solvers and 1.90 for the two-dimensional solvers; similar efficiences are observed for $m = 64$ and for $m = 16$ when p is small. The higher parallel efficiencies for the two-dimensional solvers reflect the larger ratio of arithmetic to communication for these preconditioners. Table 11 shows the total speedup, i.e., ratio of CPU time on one processor to CPU time on p processors, for $m = 32$. The typically larger speedups observed for the finite element problems also stem from the larger amount of arithmetic (caused by denser coefficient matrices) for these problems. There is some degradation of performance, especially for the one-dimensional solvers, when the number of processors increases. We attribute this to the decreased size (m^3/p^2) of the messages being sent by the preconditioner (see Section 4.1.3), coupled with the potential for contention for the communication switch and the relatively small amount of computation required by the one-dimensional solvers. For $p = 16$, the message sizes are 16 words for $m = 16$ and 128 words for $m = 32$. For the more compute-intensive two-dimensional solvers, there is little performance degradation. Despite this, overall costs of the one-dimensional solvers were significantly lower, even when these required more iterations (i.e., for finite differences).

Remark 4.3. In repeated examples of groups of three runs, we found some variation in the average CPU times, on the order of 10%. This explains some instances (e.g., finite elements, Q_d, $m = 16$) where doubling the processors led to speedups greater than two. These variations may derive from contention for the communication switch, although we also encountered some nontrivial variation on sixteen processors, when we occupied the whole machine.

5. Concluding remarks. The preconditioners presented here enable efficient parallel solution of the three-dimensional Helmholtz equation on large uniform grids. Performance of restarted GMRES with these preconditioners is relatively insensitive to discretization mesh size and wave number, and the algorithms are highly parallelizable. We have tested these methods on simple box-shaped domains. They can be adapted for use on exterior Helmholtz problems by applying the capacitance method of [21], perhaps using a nonuniform grid in the neighborhood of the scatterer. The techniques are potentially applicable in non-Cartesian coordinate systems, although the "fast" solvers in such settings are not nearly as fast, relying on generation and solution of general block tridiagonal systems [25]. We also expect them to perform well for more accurate local approximations to the radiation boundary conditions (3), of the type considered in [4].

Acknowledgements. We thank Alan Sussman for a great deal of advice on using the IBM-SP2, Ilya Zavorine for help with programming, and Olof Widlund for some helpful discussions.

REFERENCES

[1] E. Anderson et al., *LAPACK Users' Guide*, SIAM, Philadelphia, second ed., 1995.

[2] R. Barrett et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1993.

[3] A. Bayliss, C. I. Goldstein, and E. Turkel, *An iterative methods for the helmholtz equation*, J. Comput. Phys., 49 (1983), pp. 443–457.

[4] A. Bayliss, M. Gunzburger, and E. Turkel, *Boundary conditions for the numerical solution of elliptic equations in exterior domains*, SIAM J. Appl. Math., 42 (1982), pp. 430–451.

[5] J. H. Bramble, J. E. Pasciak, and J. Xu, *The analysis of multigrid algorithms for nonsymmetric and indefinite problems*, Math. Comp., 51 (1988), pp. 389–414.

[6] 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. on Numerical Analysis, 8 (1971), pp. 722–736.

[7] X.-C. Cai and O. B. Widlund, *Domain decomposition algorithms for indefinite elliptic problems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 243–258.

[8] R. H. Chan and M. K. Ng, *Conjugate gradient methods for Toeplitz systems*, SIAM Rev., 38 (1996), pp. 427–482.

[9] R. Dautray and J.-L. Lions, *Mathematical Analysis and Numerical Methods for Science and Technology*, vol. 1, Springer-Verlag, New York, 1990.

[10] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[11] O. Ernst, *Fast Numerical Solution of Exterior Helmholtz Problems with Radiation Boundary Condition by Imbedding*, PhD thesis, Stanford University, Program in Scientific Computing and Computational Mathematics, 1994.

[12] O. Ernst and G. H. Golub, *A domain decomposition approach to solving the Helmholtz equation with a radiation boundary condition*, tech. report, Computer Science Department, Stanford University, Stanford, CA Numerical Analysis Project NA-92-08, 1992.

[13] R. W. Freund and M. Malhotra, *A block-QMR Algorithm for non-Hermitian linear systems with multiple right hand sides*, Linear Algebra Appl., 254 (1997), pp. 197–257.

[14] J. A. George, *The use of direct methods for the solution of the discrete Poisson equation on non-rectangular regions*, tech. report, Computer Science Department, Stanford University, Stanford, CA STAN-CS-70-159, 1970.

[15] E. Heikkola, T. Rossi, P. Tarvainen, and Y. Kuznetsov, *Efficient preconditioners based on fictitious domains for elliptic fe-problems with lagrange multipliers*, Tech. Report 11/1996, University of Jyväskylä, 1996.

[16] F. Ihlenburg and I. Babuška, *Finite element solution of the Helmholtz with high wave number. Part I: The h-version of the FEM*, Computers Math. Applic., 30 (1995), pp. 9–37.

[17] M. C. Junger and D. Feit, *Sound, Structures and their Interaction*, MIT Press, Cambridge, MA, 2nd ed., 1986.

[18] J. B. Keller and D. Givoli, *Exact nonreflecting boundary conditions*, J. Comput. Phys., 82 (1989), pp. 172–192.

[19] M. Malhotra and P. M. Pinsky, *Parallel Preconditioning Based on h-Hierarchical Finite Elements with Applications to Acoustics*, Tech. Report SCCM 95-05, Scientific Computing and Computational Mathematics Program, Stanford University, 1995. To appear in Comp. Meths. Appl. Mech. Engrg.

[20] A. E. Naiman, I. M. Babuška, and H. C. Elman, *A note on conjugate gradient convergence*, Numerische Mathematik, 76 (1997), pp. 209–230.

[21] D. P. O'Leary and O. Widlund, *Capacitance matrix methods for the Helmholtz equation on general three dimensional regions*, Mathematics of Computation, 33 (1979), pp. 849–879.

[22] W. Proskurowski and O. Widlund, *On the numerical solution of Helmholtz's equation by the capacitance matrix method*, Mathematics of Computation, 30 (1976), pp. 433–468.

[23] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[24] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI: The Complete Reference*, The MIT Press, Cambridge, Massachusetts, 1996.

[25] P. Swarztrauber and R. Sweet, *Efficient Fortran subprograms for the solution of elliptic equations*, tech. report, National Center for Atmospheric Research, Boulder, CO, NCAR TN/IA-109; code available through http://www.netlib.org, 1975.

[26] P. N. Swarztrauber, *Vectorizing the fft's*, in Parallel Computations, G. Rodrigue, ed., Academic Press, New York, 1982, pp. 51–83.

[27] D. M. Young, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1970.