J Sci Comput (2009) 38: 316–330
DOI 10.1007/s10915-008-9240-6

A Fast Direct Solver for a Class of Elliptic Partial Differential Equations

Per-Gunnar Martinsson

Received: 20 September 2007 / Revised: 30 May 2008 / Accepted: 12 August 2008 /
Published online: 30 August 2008

Abstract We describe a fast and robust method for solving the large sparse linear systems that arise upon the discretization of elliptic partial differential equations such as Laplace's equation and the Helmholtz equation at low frequencies. While most existing fast schemes for this task rely on so called "iterative" solvers, the method described here solves the linear system directly (to within an arbitrary predefined accuracy). The method is described for the particular case of an operator defined on a square uniform grid, but can be generalized other geometries. For a grid containing N points, a single solve requires $O(N\log^2 N)$ arithmetic operations and $O(\sqrt{N}\log N)$ storage. Storing the information required to perform additional solves rapidly requires $O(N\log N)$ storage. The scheme is particularly efficient in situations involving domains that are loaded on the boundary only and where the solution is sought only on the boundary. In this environment, subsequent solves (after the first) can be performed in $O(\sqrt{N}\log N)$ operations. The efficiency of the scheme is illustrated with numerical examples. For instance, a system of size $10^6 \times 10^6$ is directly solved to seven digits accuracy in four minutes on a 2.8 GHz P4 desktop PC.

Keywords Fast solver · Direct method · Discrete Laplace operator · Hierarchically semi-separable matrix · H-matrix · Fast matrix algebra · Fast matrix inversion

1 Introduction

This paper describes a method for rapidly solving large systems of linear equations with sparse coefficient matrices. It is capable of handling the equations arising from the finite element or finite difference discretization of elliptic partial differential equations such as Laplace's equation, as well as the systems associated with heat conduction and random walks on certain networks. While most existing fast schemes for such problems rely on iterative solvers, the method described here solves the linear system directly (to within a

P.-G. Martinsson (✉)
Department of Applied Mathematics, University of Colorado at Boulder, Boulder, CO 80309-0526, USA
e-mail: martinss@colorado.edu

preset computational accuracy). This obviates the need for customized pre-conditioners, improves robustness in the handling of ill-conditioned matrices, and leads to dramatic speed-ups in environments in which several linear systems with the same coefficient matrix are to be solved.

The scheme is described for the case of equations defined on a uniform square grid. Extensions to more general grids, including those associated with complicated geometries and local mesh refinements are possible, as described in Sect. 6.

For a system matrix of size $N \times N$ (corresponding to a $\sqrt{N} \times \sqrt{N}$ grid), the scheme requires $O(N \log^2 N)$ arithmetic operations. For a single solve, only $O(\sqrt{N} \log N)$ storage is required. Moreover, for problems loaded on the boundary only, any solves beyond the first require only $O(\sqrt{N} \log N)$ arithmetic operations provided that only the solution on the boundary is sought. For problems loaded on the entire domain, it is still possible to perform very fast subsequent solves, but this requires $O(N \log N)$ storage. Numerical experiments indicate that the constants in these asymptotic estimates are quite moderate. For instance, to directly solve a system involving a $1000\,000 \times 1\,000\,000$ matrix to seven digits of accuracy takes about four minutes on a 2.8 GHz desktop PC with 512 Mb of memory. Additional solves beyond the first can be performed in 0.03 seconds (provided that only boundary data is involved).

The proposed scheme is conceptually similar to a couple of recently developed methods for accelerating domain decomposition methods such as nested dissection, [8] and [4]. The original nested dissection algorithm reduces a problem defined on a two dimensional domain in the plane to a sequence of problems defined on one dimensional domains. These problems involve dense coefficient matrices, but the reduction in dimensionality results in a decrease in the cost of a direct solve from $O(N^3)$ to $O(N^{3/2})$ for a grid containing N points. In [8] and [4] it is observed that these dense matrices in fact have internal structure, and that by exploiting this structure, it is possible to further reduce the computational cost. The scheme proposed here is similar to the schemes of [4, 8] in that it relies on a combination of a dimension-reduction technique, and fast algorithms for structured matrices to solve the resulting sequence of dense problems. However, it uses a different technique for dimension reduction, and a much simpler format for working with structured matrices than previous schemes. Its principal advantage over previous work is that what it actually computes is a sequence of Schur complements for successively larger parts of the computational domain. As a consequence, the scheme directly computes the solution operator that maps a boundary load to the solution on the boundary. Having access to this operator enables very fast solves in environments where a sequence of equations on the same computational grid are to be solved for a number of different boundary loads. The technique of hierarchical computation of Schur complements also appears to lead to improvements in robustness over competing methods.

A core advantage of the method of this paper, as well as the methods of [8] and [4], over existing methods such as multigrid or nested dissection, is that they combine the robustness of direct solvers with the almost linear asymptotic CPU time requirement of existing iterative solvers. Moreover, like all direct solvers, they perform exceptionally well in environments involving multiple right hand sides.

It is not fully understood what the exact range of applicability of the scheme proposed here is. It certainly works for positive definite symmetric matrices that arise upon the discretization of elliptic differential equations such as Laplace's equation or the equations of elasticity [2, 4]. Numerical experiments indicate that it also works for non-positive problems such as those arising from the discretization of the Helmholtz equation at low and intermediate frequencies, as well as for many non-symmetric problems such as those arising from the discretization of convection-diffusion problems (as long as the diffusion term is not too small). Moreover, it works for discrete Laplace operators on regular networks in two dimensions with no apparent constraints on regularity in the coefficients (see Sect. 7.2 for details).

The paper is structured as follows: Sect. 2 introduces a model problem that will be used to describe the method. Section 3 describes a very simple $O(N^2)$ direct solver. Section 4 describes some known algorithms for performing fast operations on matrices with off-diagonal blocks of low rank. Section 5 describes how the $O(N^2)$ scheme of Sect. 3 can be accelerated to $O(N \log^2 N)$ using the methods of Sect. 4. Section 6 describes how the technique can be modified to accommodate more general grids. Section 7 gives the results of numerical experiments. Section 8 summarizes the results presented.

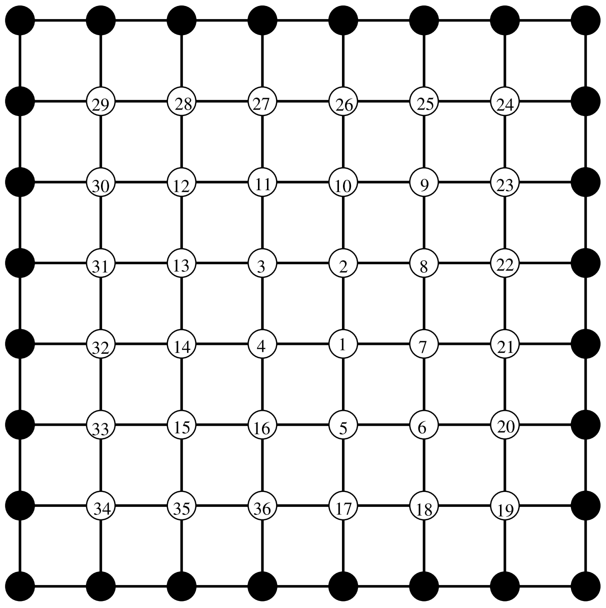
2 A Model Problem

We will describe the fast direct solver in the simplest possible setting by showing how it can be used to solve the linear system associated with the standard five-point stencil on a square uniform grid. This linear system is a standard discretization of Laplace's equation, but it can also be viewed as a first principles model of, *e.g.*, a random walk on the square grid. In this section, we introduce some notation, and formally describe the system matrix.

Letting m denote a positive integer, we consider a $(2m+2) \times (2m+2)$ uniform square grid, as illustrated in Fig. 1. As our model problem, we consider heat conduction on the grid. We give Dirichlet boundary conditions by prescribing the temperature at the $8m+4$ boundary nodes, and seek to determine the equilibrium temperature at the $(2m)^2$ internal nodes. Supposing that all links between nodes have unit conductivity, the vector x of nodal temperatures satisfies the equation

$$Ax = b, \tag{2.1}$$

Fig. 1 The computational grid for $N = 36$ (which is to say $m = 3$)



where A is the standard five-point stencil. Using the standard ordering of the nodes by column (*not* the ordering shown in Fig. 1), A is a block matrix consisting of $2m \times 2m$ blocks of size $2m \times 2m$, as follows:

$$A = \begin{bmatrix} D & -I & 0 & \cdots & 0 \\ -I & D & -I & & 0 \\ 0 & -I & D & & 0 \\ \vdots & & & & \vdots \\ 0 & & & & D \end{bmatrix}, \quad \text{where} \quad D = \begin{bmatrix} 4 & -1 & 0 & \cdots & 0 \\ -1 & 4 & -1 & & 0 \\ 0 & -1 & 4 & & 0 \\ \vdots & & & & \vdots \\ 0 & & & & 4 \end{bmatrix}. \tag{2.2}$$

The load vector b has contributions from the prescribed boundary data, and from any external heat source applied directly to the internal nodes (if applicable).

We will sometimes consider the more general system matrix A obtained by assigning different conductivities to the different links in the grid. We call such a matrix a "discrete Laplace operator" on the grid. Letting k denote one of the $(2m-2) \times (2m-2)$ nodes that do not connect to the boundary, the action of the discrete Laplace operator on x at k is

$$[Ax](k) = \sum_{j \in \mathbb{B}_k} \lambda_{kj} \big(x(k) - x(j)\big),$$

where $\mathbb{B}_k = \{k_{\rm s}, k_{\rm e}, k_{\rm n}, k_{\rm w}\}$ is an index set marking the nodes to the "south", "east", "north", and "west" of the node k, respectively, and λ_{kj} is the conductivity of the link between nodes k and j. (The standard five-point stencil in (2.2) is recovered by setting $\lambda_{kj} = 1$ for all connected nodes k and j.)

Generalizations to discrete Laplace operators on other grids is described in Sect. 6, while generalizations to matrices that arise upon the discretization of other elliptic differential operators are described in Sect. 7.2.

3 An Exact $O(N^2)$ Direct Solver

In this section we describe a method for directly solving the linear system (2.1) that relies on the sparsity pattern of the matrix only. In the absence of rounding errors, it would be exact. When the matrix A is of size $N \times N$, the method requires $O(N^2)$ floating point operations and $O(N)$ memory. This makes the scheme significantly slower than well-known $O(N^{3/2})$ schemes such as nested dissection [7]. (We mention that $O(N^{3/2})$ is optimal in this environment [11].) The merit of the scheme presented in this section is simply that it can straight-forwardly be accelerated to an $O(N \log^2 N)$ or possibly even $O(N)$ scheme, as shown in Sect. 5.

Ordering the N points in the grid in the spiral pattern shown in Fig. 1, the matrix A in (2.1) has the sparsity pattern indicated in Fig. 2 for $N = 100$. We next partition the grid into m concentric squares and collect the nodes into index sets $J_1, J_2, \ldots, J_m$ accordingly. In other words,

$$\begin{aligned} J_1 &= \{1, 2, 3, 4\}, \\ J_2 &= \{5, 6, \ldots, 16\}, \\ &\vdots \\ J_m &= \{(2m-2)^2 + 1, (2m-2)^2 + 2, \ldots, (2m)^2\}. \end{aligned}$$

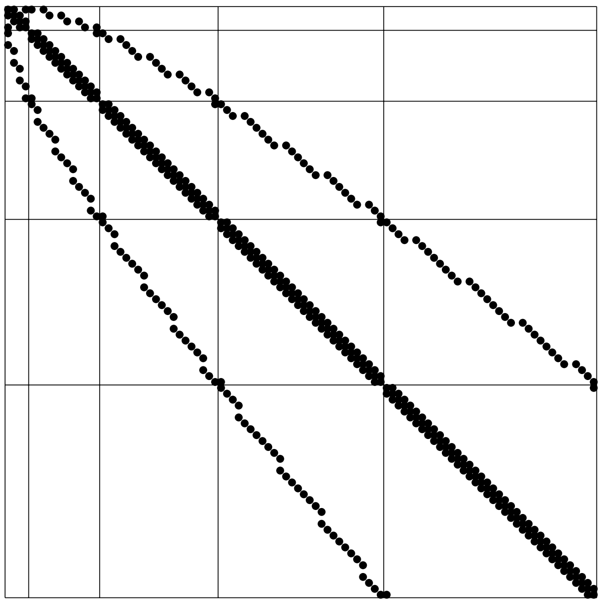


Fig. 2 The sparsity pattern of A in (2.1) for $N = 100$

For $\kappa, \lambda \in \{1, 2, \ldots, m\}$, we let $A_{\kappa\lambda}$ denote the submatrix of A formed by the intersection of the J_κ rows with the J_λ columns. The linear system (2.1) then takes on the block-tridiagonal form

$$\begin{bmatrix} A_{11} & A_{12} & 0 & 0 & \cdots & 0 \\ A_{21} & A_{22} & A_{23} & 0 & \cdots & 0 \\ 0 & A_{32} & A_{33} & A_{34} & \cdots & 0 \\ 0 & 0 & A_{43} & A_{44} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & A_{mm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_m \end{bmatrix}, \tag{3.1}$$

where x and b have been partitioned accordingly. The sparsity and block pattern of (3.1) for $m = 5$ (equivalently, $N = 100$) is illustrated in Fig. 2.

The blocked system of (3.1) can now easily be solved by eliminating the variables $x_1, x_2, \ldots, x_{m-1}$ one by one. Using the first row to eliminate x_1 from the second row, we obtain the following system of equations for the variables $x_2, \ldots, x_m$:

$$\begin{bmatrix} \tilde{A}_{22} & A_{23} & 0 & \cdots & 0 \\ A_{32} & A_{33} & A_{34} & \cdots & 0 \\ 0 & A_{43} & A_{44} & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & A_{mm} \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} \tilde{b}_2 \\ b_3 \\ b_4 \\ \vdots \\ b_m \end{bmatrix}, \tag{3.2}$$

where

$$\tilde{A}_{22} = A_{22} - A_{21} A_{11}^{-1} A_{12}$$

and

$$\tilde{b}_2 = b_2 - A_{21} A_{11}^{-1} b_1.$$

This elimination process is continued row by row until we obtain the following equation for x_m:

$$\tilde{A}_{mm} x_m = \tilde{b}_m. \tag{3.3}$$

Equation (3.3) is of size $(8m-4) \times (8m-4)$ and is solved directly to obtain x_m. Then x_{m-1} is determined by solving the equation

$$\tilde{A}_{m-1,m-1} x_{m-1} = \tilde{b}_{m-1} - A_{m-1,m} x_m.$$

The remaining x_j's are computed analogously. To summarize the entire process:

```
(1)  $\tilde{A}_{11} = A_{11}$ and $\tilde{b}_1 = b_1$.
(2)  for $\kappa = 2 : m$
(3)      $\tilde{A}_{\kappa\kappa} = A_{\kappa\kappa} - A_{\kappa,\kappa-1} \tilde{A}^{-1}_{\kappa-1,\kappa-1} A_{\kappa-1,\kappa}$
(4)      $\tilde{b}_{\kappa} = b_{\kappa} - A_{\kappa,\kappa-1} \tilde{A}^{-1}_{\kappa-1,\kappa-1} \tilde{b}_{\kappa-1}$
(5)  end
(6)  $x_m = \tilde{A}^{-1}_{mm} \tilde{b}_m$
(7)  for $\kappa = (m-1) : (-1) : 1$
(8)      $x_{\kappa} = \tilde{A}^{-1}_{\kappa\kappa} (\tilde{b}_{\kappa} - A_{\kappa,\kappa+1} x_{\kappa+1})$
(9)  end
```

We note that while all matrices $A_{\kappa\lambda}$ are sparse, the matrices $\tilde{A}_{\kappa\kappa}$ are dense. This means that the cost of inverting $\tilde{A}_{\kappa\kappa}$ in each step of the algorithm is $O(\kappa^3)$. (The remaining matrix-matrix operations involve matrices that are diagonal or tri-diagonal and have negligible costs in comparison to the matrix inversion.) The total cost T_{total} therefore satisfies

$$T_{\text{total}} \sim \sum_{\kappa=1}^{m} \kappa^3 \sim m^4 \sim N^2.$$

Remark 3.1 The matrix $\tilde{A}_{\kappa\kappa}$ is the Schur complement of the submatrix $A_{\kappa\kappa}$ in the system matrix of (3.1) obtained by eliminating all nodes interior to the nodes in the set J_κ. In applications where data is prescribed and sought at the boundary only, $\tilde{A}^{-1}_{mm}$ is the essential solution operator. For instance, suppose that in the heat conduction problem described in this section, we are interested in computing the fluxes through the boundary nodes. Once $\tilde{A}^{-1}_{mm}$ has been obtained, this computation can be executed in three inexpensive steps: (1) The prescribed temperatures are mapped to the vector b_m via a diagonal matrix. (2) The vector b_m is mapped to the equilibrium temperatures x_m on the boundary of the computational grid via application of $\tilde{A}^{-1}_{mm}$. (3) The flux through each boundary node is computed by adding the fluxes through each of the three bars that connect to it, which is easily done since the temperatures of these nodes are known. This computation costs only $O(N)$ operations, and requires only the storage of the $(8m-4) \times (8m-4)$ matrix $\tilde{A}^{-1}_{mm}$. Finally we note that by multiplying the three linear operators associated with the three steps, we obtain the so called "Dirichlet-to-Neumann" operator of the network.

4 Compressible Matrices and Fast Matrix Algebra

The most time-consuming part of the direct solver scheme described in Sect. 3 is the inversion of a sequence of dense intermediate matrices (the matrices $\tilde{A}_{\kappa\kappa}$). In many applications, these matrices have internal structure that enables the use of accelerated algorithms that rapidly perform the matrix inversion (to within a specified precision). In this section, we will describe that structure, and give a simple algorithm for computing an approximation to the inverse of such a matrix.

No claim is made that the methods described in this section are original; in fact, they are primitive versions of more sophisticated algorithms such as those described in [3, 9, 10]. A specific point of the paper is that a fast direct solver for many sparse matrices can be obtained via these very simple techniques. However, it should be noted that these techniques could easily be substituted by, *e.g.* the methods of [3] and that such a substitution would likely result in a gain in computational speed, see Sect. 4.3.

4.1 Compressible Matrices

Roughly speaking, we say that a matrix is "compressible" if it can be tessellated into submatrices in a pattern such as the one illustrated in Fig. 3, and each off-diagonal block in the tessellation can be approximated by a low-rank matrix.

In order to give a precise definition of the term "compressible", we let ε denote a computational accuracy, and recall that a matrix B is said to have ε-rank k if

$$\min_{\text{rank}(B_k)=k} \|B - B_k\| \leq \varepsilon.$$

Equivalently, a matrix B has ε-rank k if it has the most k singular values larger than ε. We let p denote the maximal rank allowed for the off-diagonal blocks, and we then define the "compressibility" property recursively by saying that a square matrix A is compressible if, upon partitioning it into four pieces of equal size,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

it is the case that A_{12} and A_{21} have ε-rank at most p, and A_{11} and A_{22} are "compressible".

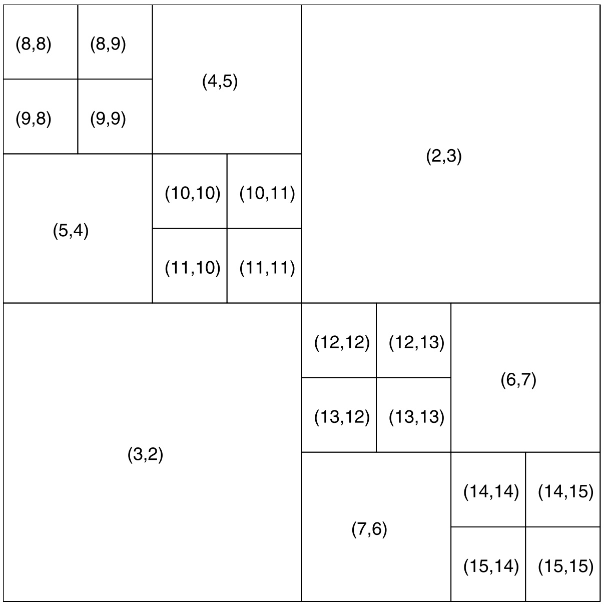


Fig. 3 Matrix tessellation. The diagonal blocks have full rank, while the off-diagonal blocks have rank at most p

An approximation to a compressible matrix can be stored using $O(pN\log N)$ real numbers, and a matrix-vector product involving a compressible matrix can be evaluated (approximately) using $O(pN\log N)$ arithmetic operations.

Remark 4.1 For notational simplicity, we assume throughout the paper that the ranks of all off-diagonal blocks are the same. It is however a simple matter to use adaptively tuned ranks when implementing the algorithm. The numerical examples given in Sect. 7 are all computed using variable ranks.

4.2 Inversion of Compressible Matrices

A recursive fast inversion scheme for compressible matrices can easily be derived from the following formula for the inverse of a 2×2 block matrix:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} X_{11} & -X_{11}A_{12}A_{22}^{-1} \\ -A_{22}^{-1}A_{21}X_{11} & A_{22}^{-1} + A_{22}^{-1}A_{21}X_{11}A_{12}A_{22}^{-1} \end{bmatrix}, \tag{4.1}$$

where

$$X_{11} = \left(A_{11} - A_{12}A_{22}^{-1}A_{21}\right)^{-1}.$$

From the formula (4.1), we immediately get the following recursive inversion scheme for compressible matrices:

```
(1)  function $B = invert\_matrix(A)$
(2)      if ($A$ is "small") then
(3)          Invert by brute force: $B = A^{-1}$
(4)      else
(5)          Split $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$.
(6)          $X_{22} = invert\_matrix(A_{22})$
(7)          $X_{11} = invert\_matrix(A_{11} - A_{12}X_{22}A_{21})$
(8)          $B = \begin{bmatrix} X_{11} & -X_{11}A_{12}X_{22} \\ -X_{22}A_{21}X_{11} & X_{22} + X_{22}A_{21}X_{11}A_{12}X_{22} \end{bmatrix}$.
(9)      end if
(10) end function
```

The efficiency of this algorithm is a consequence of the fact that the matrices A_{12} and A_{21} have low rank. As a result, the matrix-matrix multiplications that occur on lines (7) and (8) in fact consist simply of a small number of multiplications between compressible matrices and vectors. Moreover, the matrix additions in lines (7) and (8) are in fact low-rank updates to compressible matrices.

If the ranks of the off-diagonal blocks do not grow larger than p as the computation proceeds, then the computational complexity of the scheme described above is $O(p^2N\log^2 N)$. In the numerical experiments that we have performed, we determine the rank of each block adaptively at each step of the computation; the observed ranks have in all practical experiments been very small (*cf.* Table 2) and our CPU time measurements indicate $O(N\log^2 N)$ scaling with N of the direct solver.

Another way in which the computational scheme described here could fail is if errors aggregate during the course of the computation. In the computational experiments that we have performed, the errors appear to grow at worst as $\log N$ as the problem size is increased. Such error growth can very easily be dealt with by moderately decreasing ε as larger problems are considered.

4.3 Hierarchically Compressible Matrices

The performance of fast methods for structured matrices can often be improved from $O(N\log^2 N)$ to $O(N\log N)$ or $O(N)$. This is possible when the class of matrices under consideration satisfies the stronger compressibility criterion that it is possible to express the bases for the off-diagonal blocks hierarchically. In other words, it must be possible to express the basis vectors used at one level in terms of the basis vectors used at the next finer level. Using the matrix illustrated in Fig. 3 as an example, a basis for the column space of the block labelled $(4,5)$ is constructed from the bases for the column spaces of the blocks $(8,9)$ and $(9,8)$. An approximation to a matrix satisfying this stronger compressibility condition requires only $O(pN)$ storage, and can be applied to a vector in $O(pN)$ arithmetic operations.

An $O(N)$ inversion scheme for the "hierarchically compressible" matrices that arise when discretizing boundary integral equations in two dimensions is described in [12]. The concept of "hierarchically semi-separable" matrices described in [3–6] also exploits hierarchically defined basis functions to achieve high performance, as does the $\mathcal{H}^2$-matrix framework described in [1, 10].

5 A Fast Direct Solver

When the direct solver described in Sect. 3 is applied to a matrix A that arises from the discretization of an elliptic partial differential equation such as the Laplace equation, the intermediate matrices that arise in the computation are compressible in the sense described in Sect. 4, see [2, 4] and Remark 5.2. As a consequence, the scheme can be accelerated by replacing the dense matrix inversions by the fast matrix inversion scheme described in Sect. 4.2. Since the cost of inverting $\tilde{A}_{\kappa\kappa}$ at each step $\kappa = 1, 2, \dots, m$ is then $O(\kappa\log^2\kappa)$, the computational cost of the resulting scheme is

$$T_{\text{total}} \sim \sum_{\kappa=1}^{m} \kappa(\log\kappa)^2 \sim m^2(\log m)^2 \sim N(\log N)^2.$$

This scheme requires $O(N\log N)$ memory to store all information required to approximate the application of A^{-1}. Such a scheme has been implemented and the computational results are given in Sect. 7.

We note that the scheme is particularly memory efficient in environments where the solution is sought only on the boundary of the domain, *cf.* Remark 3.1. In such situations, the operators $\tilde{A}_{\kappa\kappa}$ need not be stored, and the direct solver requires only $O(\sqrt{N}\log N)$ memory. Moreover, if the domain is loaded only on the boundary, and if (2.1) is to be solved for several different boundary loads, subsequent solutions are obtained simply by applying the pre-computed operator $\tilde{A}_{mm}^{-1}$ at the modest cost of $O(\sqrt{N}\log N)$ operations.

We also note that while it would require $O(N)$ memory to store A itself, the scheme only accesses each entry of A once. Consequently, the elements of A can either be computed on the fly (if given by a formula), or fetched sequentially from slow memory ("tape").

Remark 5.1 A further improvement in computational speed can in principle be obtained whenever the intermediate matrices are compressible in the stronger sense described in

Sect. 4.3. In this case, the cost of inverting $\tilde{A}_{\kappa\kappa}$ is $O(\kappa)$ and so the total cost is

$$T_{\text{total}} \sim \sum_{\kappa=1}^{m} \kappa \sim m^2 \sim N.$$

Experimental data (described in Sect. 7) indicate that the matrices $\tilde{A}_{\kappa\kappa}$ are indeed "hierarchically compressible" and permit $O(N)$ inversion schemes. However, such a scheme has not yet been implemented.

Remark 5.2 We currently do not have a good understanding of which sparse matrices allow fast inversion and factorization. Experimental and theoretical results indicate that symmetric positive definite matrices arising from the discretization of elliptic PDEs certainly qualify [2, 4]. But neither symmetry nor ellipticity are necessary properties, as the numerical experiments described in Sect. 7.2 demonstrate.

6 Generalizations

The scheme presented can be adapted to more general operators in two and three dimensions. The generalization to other difference operators on uniform square grids in two dimensions is trivial. Other two-dimensional grids that are uniform in the sense that they can readily be partitioned into a sequence of concentric annuli can also be handled quite easily, and we expect the performance of such schemes to be similar to the performance reported in Sect. 7.

Many grids cannot in a natural way be partitioned into a sequence of concentric annuli; this is for instance the case with grids arising from adaptive mesh-refinement, or the discretization of complex geometries. For such cases, other domain decomposition techniques (based for instance on minimal graph cuts as in nested dissection) will probably perform better than the concentric annuli technique of this paper. However, the technique of computing Schur complements of the system matrix by successively merging larger and large patches of the computational grid should still be applicable. Work in this direction is currently under way.

7 Numerical Examples

In this section we describe the results of computational experiments that illustrate the capabilities of the method described in Sect. 5. Specifically, Sect. 7.1 describes a set of experiments that illustrate the computational speed of the method, while Sect. 7.2 focusses on the question of which matrices are amenable to the techniques described in this paper.

We use a fixed computational precision of $\varepsilon = 10^{-7}$ in all experiments. In other words, at each step of the calculation, off-diagonal blocks in the matrices were represented to within an absolute precision of ε. We let the ranks of approximation vary between blocks.

The code used is written in a Matlab-FORTRAN hybrid. It is highly non-optimized. The experiments were run on a 2.8 GHz Pentium 4 PC with 512 Mb of RAM.

7.1 Performance Test

The $O(N \log^2 N)$ numerical scheme described in Sect. 5 was implemented and tested on a conduction problem on a square uniform grid. We assigned each bar in the grid a conductivity drawn from a uniform random distribution on the interval [1/2, 1]. For a range of grid

Table 1 A summary of the computational experiment described in Sect. 7.1

N	T_{solve}	T_{apply}	M	e_1	e_2
10000	5.93e−1	2.82e−3	3.82e+2	2.61e−8	3.31e−8
40000	4.69e+0	6.25e−3	9.19e+2	4.71e−8	6.47e−8
90000	1.28e+1	1.27e−2	1.51e+3	7.98e−8	1.25e−7
160000	2.87e+1	1.38e−2	2.15e+3	9.02e−8	1.84e−7
250000	4.67e+1	1.52e−2	2.80e+3	1.02e−7	1.14e−7
360000	7.50e+1	2.62e−2	3.55e+3	1.37e−7	1.57e−7
490000	1.13e+2	2.78e−2	4.22e+3	–	–
640000	1.54e+2	2.92e−2	5.45e+3	–	–
810000	1.98e+2	3.09e−2	5.86e+3	–	–
1000000	2.45e+2	3.25e−2	6.66e+3	–	–

sizes between 50×50 and 1000×1000, we computed the operator $\tilde{A}_{mm}^{-1}$ described in Sect. 3. The computational cost, the amount of memory required, and the accuracies obtained are presented in Table 1. The following quantities are reported:

T_{solve}	Time required to construct $\tilde{A}_{mm}^{-1}$ (in seconds).
T_{apply}	Time required to apply $\tilde{A}_{mm}^{-1}$ (in seconds).
M	Memory required to construct $\tilde{A}_{mm}^{-1}$ (in kilobytes).
e_1	The l^2-error in the vector $\tilde{A}_{mm}^{-1} r$ where r is a unit vector of random direction.
e_2	The l^2-error in the first column of $\tilde{A}_{mm}^{-1}$.

The errors e_1 and e_2 were estimated by comparing the computed Schur complement to the result of using an iterative solver to solve the unreduced (2.1). (We let the iterative solver run until the residual had hit 10^{-14}.)

The numbers in Table 1 support the claims made regarding the asymptotic cost of the algorithm. They also indicate that when the local accuracy is kept fixed, the global error grows very slowly as the problem size increases.

7.2 Range of Applicability

Both theoretical and numerical results in the literature support the claim that matrices arising from the discretization of Laplace's equation (and closely related equations such as the equations of elasticity) should have inverses and Schur complements that are "compressible" in the sense described in Sect. 4. It is from a theoretical point of view less well understood to what extent similar results can be obtained for other classes of PDEs. In this section, we investigate via computational experiments whether the scheme described in Sect. 5 works for the matrices associated with the partial differential operator

$$[Du](x) = -\Delta u(x) + b(x)u_x(x) + c(x)u_y(x) + d(x)u(x) \tag{7.1}$$

acting on the domain $\Omega = [0,1] \times [0,1]$ with Dirichlet boundary data. The experiments indicate that generally speaking, the answer is yes as long as the magnitude of the functions b, c, and d, is not very large, but that issues of ill-conditioning can arise.

We discretized the domain Ω into $(2m+2) \times (2m+2)$ points organized in a uniform grid so that the grid spacing is $h = 1/(2m+1)$. For any interior grid point k, we discretized the operator (7.1) via the simplistic finite difference scheme:

$$[Au](k) = \frac{1}{h^2}\big[4u(k) - u(k_\mathrm{n}) - u(k_\mathrm{s}) - u(k_\mathrm{e}) - u(k_\mathrm{w})\big] \\ + \frac{1}{h}b(k)\big[u(k_\mathrm{e}) - u(k_\mathrm{w})\big] + \frac{1}{h}c(k)\big[u(k_\mathrm{n}) - u(k_\mathrm{s})\big] + d(k)u(k),$$

where $k_\mathrm{e}, k_\mathrm{n}, k_\mathrm{w}, k_\mathrm{s}$ denote the grid points to the "east", "north", "west", and "south" of k, respectively. We note that such a discretization scheme is not always a good one for an equation of the general form (7.1), especially when b and c are large. However, since our goal here is not to actually compute accurate solutions to any PDE, but rather to merely investigate the rank-structure of its inverse, this should not be of much significance.

All numerical experiments reported in this section were run on a grid with 400×400 internal grid points, and with a local truncation error of $\varepsilon = 10^{-7}$.

The performance of the scheme described in Sect. 5 was tested on eight different model problems:

PureLap: A pure Laplace problem. The functions b, c, and d were all identically zero, and the matrix A is the classical five-point stencil.

RandLap: This example stands out from the others in that it is not related to a discretization of (7.1). Instead, A is a discrete Laplace operator acting on a regular $(2m+2) \times (2m+2)$ square grid, as described at the end of Sect. 2. The conductivity of each bar was drawn from a uniform random distribution on the interval [0.01, 1].

ConstCon: A convection/diffusion problem with a constant convection term acting along the x_2-axis: $b \equiv 100$, $c = 0$, $d = 0$.

DivFrCon: A convection/diffusion problem with a divergence-free convection field: $b(x) = 125\cos(4\pi x_2)$, $c(x) = 125\sin(4\pi x_1)$, $d = 0$.

DivCon: A convection/diffusion problem with a convection field with sources and sinks: $b(x) = 125\cos(4\pi x_1)$, $c(x) = 125\sin(4\pi x_2)$, $d = 0$. We note that this boundary value problem is intrinsically highly ill-conditioned due to the existence of solutions representing purely internal flows between sources and sinks.

Helm100: A Helmholtz problem at a low frequency not close to a resonance: $b = c = 0$, $d = 100$. These coefficients translate to a computational domain roughly 1.5×1.5 wavelengths large.

HelmRes: A Helmholtz problem at a low frequency very close to a resonance: $b = c = 0$, $d = -\lambda_{10} + 10^{-5}$ where λ_{10} is the 10'th eigenvalue of the discrete Laplace operator. (Note that for the five-point stencil on a square regular grid, these are known analytically, and $\lambda_9 = \lambda_{10} = 167.77043\ldots$ is a double eigenvalue.)

Helm4000: A Helmholtz problem at an intermediate frequency not close to a resonance: $b = c = 0$, $d = 4000$. Note that $d = 4000$ corresponds to a domain roughly 10×10 wavelengths large. (The closest eigenvalues of the discrete Laplace operator are $\lambda_{300} = 3991.61850\ldots$ and $\lambda_{301} = 4028.67360\ldots$.)

The results of the experiments are reported in Table 2. The numbers r_j denote the average rank of the off-diagonal blocks of size $j \times j$ in $\tilde{A}_{mm}$. The number c_max is the maximum of

Table 2 Results of experiments designed to investigate the range of applicability of the fast direct solver. The quantities given are described in Sect. 7.2. (The meaning of the numbers in parenthesis is described in Remark 7.1)

	r_{50}	r_{100}	r_{200}	r_{400}	$c_{\max}$	E_1	E_2
PureLap	8 (16)	9 (19)	9 (24)	17 (33)	5.82e+000	2.84e−007	2.84e−007
RandLap	8 (15)	9 (18)	9 (23)	17 (32)	5.02e+001	1.18e−007	1.18e−007
ConstCon	6 (15)	7 (18)	7 (22)	15 (31)	5.81e+000	2.25e−006	2.25e−006
DivFrCon	7 (15)	8 (18)	9 (23)	16 (30)	5.81e+000	6.23e−007	6.23e−007
DivCon	7 (14)	7 (16)	7 (19)	13 (26)	8.56e+000	4.41e−003	4.39e−003
Helm100	8 (15)	8 (19)	9 (23)	16 (32)	5.74e+000	1.92e−007	1.92e−007
HelmRes	9 (16)	9 (20)	10 (25)	19 (34)	9.71e+003	3.25e−002	3.36e−002
Helm4000	6 (13)	6 (15)	6 (17)	11 (22)	5.16e+000	3.45e−008	3.50e−008

the condition numbers of the matrices $\tilde{A}_{\kappa\kappa}$ for $\kappa = 1, 2, \dots, m$. The numbers E_1 and E_2 denote estimates of the error measure

$$E = \frac{\|(\tilde{A}^{\varepsilon}_{mm})^{-1} - (\tilde{A}_{mm})^{-1}\|}{\|(\tilde{A}_{mm})^{-1}\|}, \tag{7.2}$$

where $\tilde{A}_{mm}$ is the exact Schur complement at level m, and $\tilde{A}^{\varepsilon}_{mm}$ is the result of the numerical algorithm described in Sect. 5. Of course, the exact matrix $\tilde{A}_{mm}$ is not available, but we computed the estimate E_1 of E by first simulating the action of $(\tilde{A}_{mm})^{-1}$ using an iterative solver for (2.1) and then using an inverse power iteration to estimate the operator norms in (7.2). The estimate E_2 is obtained by comparing the Schur complement resulting from the "fast" algorithm described in Sect. 5 with the unaccelerated method described in Sect. 3.

It is encouraging to see that in all environments tested, the interaction ranks r_j are very small. This indicates that the computational cost of the method described in Sect. 5 will scale almost linearly for a wide range of problems. The table also indicates that the method is highly accurate and stable in most environments. In particular, we note that for the very nearly resonant Helmholtz problem "HelmRes", we lose no more than 5 digits of accuracy (compared to the local truncation error) even though the physical problem being modelled amplifies applied loads by a factor of 10^4. Example "DivCon" illustrates a similar situation of a highly ill-conditioned physical problem resulting in a loss of accuracy. What is slightly worrying about this example, however, is that here all the intermediate Schur complements $\tilde{A}_{\kappa\kappa}$ are well-conditioned, meaning that we get no warning of a loss of computational accuracy. This indicates that while the fast solver described here may be applicable to a wide range of equations, it should in some environments be complemented with rigorous error estimation techniques.

Remark 7.1 The ranks r_j that are given in Table 2 refer to the rank in the matrix format described in Sect. 4.1. The numbers given in parentheses next to the values for r_j report the number of basis vectors that are required when the matrix is compressed using hierarchically defined bases, as described in Sect. 4.3 (this number is what Gu and Chandrasekaran call the "HSS-rank", see [3, 5]). Since these numbers are of moderate size, and grow only slowly with j, it appears likely that $O(N)$ methods are feasible for all environments tested. However, we note that the numbers in parenthesis are substantially larger than the numbers r_j, indicating that for moderate problem sizes, the gain could be smaller than one might have hoped.

8 Concluding Remarks

We have presented a scheme for rapidly performing direct solves on linear systems involving the large sparse matrices arising from the discretization of elliptic PDEs such as Laplace's equation, or the Helmholtz equation at low wave-numbers. The scheme is based on a combination between a technique for reducing a problem defined on a two-dimensional domain to a sequence of problems defined on one-dimensional subdomains, and the use of a fast method to invert the dense but structured matrices that represent the problem on each of the one-dimensional domains.

The scheme presented typically achieves an asymptotic computational complexity of $O(N \log^2 N)$, with a constant sufficiently small that a direct solve of a linear system involving a million by million sparse coefficient matrix can be performed in about four minutes on a modest desktop PC. The algorithm produces a Schur complement of the system matrix which can be used to solve additional linear problems involving the same coefficient matrix in 0.03 seconds (provided that data is given only on the boundary of the domain, and that the solution is sought only on the boundary). The computations cited produced results with seven correct digits.

The method described can be accelerated significantly (probably all the way to an $O(N)$ method) by using more efficient techniques for inverting structured matrices, such as those described in [12]. Work in this direction in progress.

Another way of potentially improving the algorithm is to use a different technique for reducing a problem on a two-dimensional domain to a sequence of problems defined on one-dimensional subdomains. One option would be to use a method based on minimal graph cuts such as nested dissection, as was done in [4]. This would likely result in a method that is more versatile in terms of which meshes can be handled, and might prove advantageous in terms of computational speed.

One feature of the present scheme that we believe is very attractive compared to competing methods is the aggregation of Schur complements of the original system matrix. This technique is inherently well-conditioned, and leads to conceptually simple algorithms that are robust when applied to a wide range of differential operators, as demonstrated in Sect. 7.2. It also has the advantage that it automatically produces a "reduced model" for the computational domain that directly maps a boundary load to the solution on the boundary in environments where the interior of the domain is unloaded; even for problems involving millions of degrees of freedom, this map can be evaluated in a couple of hundredths of a second on a regular desktop PC.

The method was developed as a technique for symmetric positive definite matrices such as those arising from the discretization of Laplace's equations and the equations of elasticity. In such environments, both theory and numerical experiments support our claims regarding stability, accuracy, and speed of the method. Numerical tests indicate that the method is in fact far more widely applicable, and will retain its performance in terms of speed and accuracy for a wide range of sparse matrices (see Sect. 7.2). However, examples do exist for which the method as currently implemented loses accuracy. Work aimed at determining in which environments the method can be made to work well is in progress.

Acknowledgements The idea of constructing a direct solver by computing a sequence of operators defined on concentric annuli was described to the author by Vladimir Rokhlin in 2003 in a conversation about constructing a fast direct solver for the Lippmann-Schwinger equation. The algorithm presented in this paper is directly inspired by that conversation. Moreover, the author is indebted to Mark Tygert for sharing his insights on fast methods.

References

1. Börm, S.: H^2-matrix arithmetics in linear complexity. Computing **77**(1), 1–28 (2006)
2. Börm, S.: Approximation of solution operators of elliptic partial differential equations by H and H^2-matrices. Tech. Report 85/2007, Max Planck Institute for Mathematics in the Sciences (2007)
3. Chandrasekaran, S., Gu, M., Li, X.S., Xia, J.: Some fast algorithms for hierarchically semiseparable matrices. Private Communication (2007)
4. Chandrasekaran, S., Gu, M., Li, X.S., Xia, J.: Superfast multifrontal method for structured linear systems of equations. Private Communication (2007)
5. Chandrasekaran, S., Gu, M., Lyons, W.: A fast adaptive solver for hierarchically semiseparable representations. Calcolo **42**(3–4), 171–185 (2005)
6. Chandrasekaran, S., Gu, M., Pals, T.: A fast ULV decomposition solver for hierarchically semiseparable representations. SIAM J. Matrix Anal. Appl. **28**(3), 603–622 (2006). (Electronic)
7. George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. **10**, 345–363 (1973)
8. Grasedyck, L., Kriemann, R., Le Borne, S.: Domain-decomposition based H-matrix preconditioners. In: Proceedings of DD16. LNSCE, vol. 55, pp. 661–668. Springer, Berlin (2006)
9. Hackbusch, W.: A sparse matrix arithmetic based on H-matrices. I. Introduction to H-matrices. Computing **62**(2), 89–108 (1999)
10. Hackbusch, W., Khoromskij, B., Sauter, S.A.: On H^2-matrices. In: Lectures on Applied Mathematics (Munich, 1999), pp. 9–29. Springer, Berlin (2000)
11. Hoffman, A.J., Martin, M.S., Rose, D.J.: Complexity bounds for regular finite difference and finite element grids. SIAM J. Numer. Anal. **10**, 364–369 (1973)
12. Martinsson, P.G., Rokhlin, V.: A fast direct solver for boundary integral equations in two dimensions. J. Comput. Phys. **205**(1), 1–23 (2005)