An $O(N)$ algorithm for constructing the solution operator to elliptic boundary value problems in the absence of body loads

Adrianna Gillman
Department of Mathematics
Dartmouth College
adrianna.gillman@Dartmouth.edu

Per-Gunnar Martinsson
Department of Applied Mathematics
University of Colorado at Boulder
martinss@colorado.edu

Abstract: The large sparse linear systems arising from the finite element or finite difference discretization of elliptic PDEs can be solved directly via, e.g., nested dissection or multifrontal methods. Such techniques reorder the nodes in the grid to reduce the asymptotic complexity of Gaussian elimination from $O(N^3)$ to $O(N^{1.5})$ for typical problems in two dimensions. It has recently been demonstrated that the complexity can be further reduced to $O(N)$ by exploiting structure in the dense matrices that arise in such computations (using, e.g., $\mathcal{H}$-matrix arithmetic). This paper demonstrates that such accelerated nested dissection techniques become particularly effective for boundary value problems without body loads when the solution is sought for several different sets of boundary data, and the solution is required only near the boundary. In this case, a modified version of the accelerated nested dissection scheme can execute any solve beyond the first in $O(N_{\text{boundary}})$ operations, where N_{boundary} denotes the number of points on the boundary. Typically, $N_{\text{boundary}} \sim N^{0.5}$. Numerical examples demonstrate the effectiveness of the procedure for a broad range of elliptic PDEs that includes both the Laplace and Helmholtz equations.

1. Introduction

1.1. Problem formulation.

This paper presents a fast technique for solving homogeneous boundary value problems (BVPs) of the form

$$\begin{aligned} -\Delta u(\boldsymbol{x}) + b(\boldsymbol{x})u_x(\boldsymbol{x}) + c(\boldsymbol{x})u_y(\boldsymbol{x}) + d(\boldsymbol{x})u(\boldsymbol{x}) &= 0 \qquad && \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) &= g(\boldsymbol{x}) && \boldsymbol{x} \in \Gamma, \end{aligned} \tag{1.1}$$

where $\Omega = [0,1]^2$ is the unit square in $\mathbb{R}^2$, where Γ is the boundary of Ω, and where b, c, and d are functions on Ω. We assume that the only information sought is the normal derivative of u at Γ. In other words, the objective is to construct an approximation to the Dirichlet-to-Neumann operator associated with the elliptic differential operator in (1.1).

The proposed technique is particularly efficient for situations where (1.1) needs to be solved for a sequence of different boundary data functions g. The method has two steps: (1) For a given set of functions b, c, and d, an approximate Dirichlet-to-Neumann is constructed at cost proportional to the number of degrees of freedom used to discretize Ω. (2) The solution $\partial u/\partial n$ for any given Dirichlet data g is constructed by application of the approximate Dirichlet-to-Neumann operator at cost proportional to the number of degrees of freedom used to discretize Γ (independent of the number of discretization points used for Ω).

1.2. Motivation.

While the present paper addresses the specific BVP (1.1) on a simple square domain, the technique can easily be extended for building solution operators to elliptic boundary value problems of the form

$$\begin{cases} Au(\boldsymbol{x}) = & 0 \qquad \boldsymbol{x} \in \Omega \\ Bu(\boldsymbol{x}) = & g(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Gamma, \end{cases} \tag{1.2}$$

where Ω is a domain in $\mathbb{R}^2$ or $\mathbb{R}^3$ with boundary Γ, where A is an elliptic partial differential operator, and where B is a trace operator (representing boundary conditions like Dirichlet, Neumann, mixed, etc.). The goal of this paper is to illustrate that when there is no body load and the solution u and/or its derivatives are sought only near the boundary, the relevant solution operator can be constructed at moderate cost, and applied almost instantaneously. This opens up the possibility of high accuracy computational simulations to be carried out in real time for 3D problems such as elasticity involving composite materials, electrostatics in domains with variable conductivity, acoustic and electromagnetic scattering problems (at long and intermediate wave-lengths at least), and many others.

1.3. **Discretization.** The method described is applicable to a variety of geometries and discretization schemes (finite elements, finite differences, etc.). For simplicity of presentation, we restrict our attention to the model problem where a square domain Ω is discretized via a finite difference scheme on a regular $n \times n$ square mesh. The resulting linear system takes the form

$$\mathsf{A}\boldsymbol{u} = \boldsymbol{b} \tag{1.3}$$

where A is an $n^2 \times n^2$ sparse matrix. For future reference, we let $N = n^2$ denote the total number of grid points.

Example: *When (1.1) represents the Laplace equation ($b = c = d = 0$) and the standard five-point finite difference stencil is used in the discretization,* A *consists of $n \times n$ blocks, each of size $n \times n$,*

$$\mathsf{A} = \begin{bmatrix} \mathsf{B} & -\mathsf{I} & 0 & 0 & \cdots \\ -\mathsf{I} & \mathsf{B} & -\mathsf{I} & 0 & \cdots \\ 0 & -\mathsf{I} & \mathsf{B} & -\mathsf{I} & \cdots \\ 0 & 0 & -\mathsf{I} & \mathsf{B} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}, \quad \textit{where} \quad \mathsf{B} = \begin{bmatrix} 4 & -1 & 0 & 0 & \cdots \\ -1 & 4 & -1 & 0 & \cdots \\ 0 & -1 & 4 & -1 & \cdots \\ 0 & 0 & -1 & 4 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix},$$

and where I *is the $n \times n$ identity matrix.*

1.4. **Existing fast solvers.** Existing techniques for solving (1.3) can roughly be divided into two categories:

Iterative methods construct a sequence of successively more accurate approximate solutions by applying the matrix A to a sequence of vectors. Since the $N \times N$ matrix A has $O(N)$ non-zero entries, the resulting solver has $O(N)$ complexity whenever convergence is fast. It is difficult to predict the convergence rate of iterative methods and often a customized pre-conditioner is required to accelerate the schemes.

Direct methods such as Gaussian elimination which compute a solution in a single shot are considered more stable and robust. Proper ordering of the nodes often allows Gaussian elimination to be executed at $O(N^{1.5})$ complexity [3], and the resulting "nested dissection" approach is quite competitive for moderate problem sizes (up to about $N \sim 10^6$). More recently, it has been shown that by exploiting additional structure in the coefficient matrix, the nested dissection method can be accelerated to (close to) linear complexity, [2, 6, 8].

1.5. **Novelty of the present work.** The techniques described in this paper are based on recent work [2, 4, 6, 7, 8] building linear complexity direct solvers for matrices arising from finite element and finite difference discretizations of elliptic partial differential equations. The core observation is that in situations where a BVP such as (1.1) needs to be solved for a sequence of different boundary data functions f, each solve beyond the first can be executed in *sub-linear* complexity. To be precise, the complexity of additional solves can be reduced to $O(N_{\text{boundary}})$ where N_{boundary} denotes the number of boundary nodes. In two dimensions, it is often the case that $N_{\text{boundary}} \sim N^{0.5}$ which means that each solve beyond the first has complexity $O(N^{0.5})$. Very substantial speed-ups can result. Section 6 shows that for a situation where $N = 16{,}000{,}000$ (and $N_{\text{boundary}} \approx 16{,}000$), the cost of executing an accelerated nested dissection scheme on a standard desktop computer is 8 minutes, while the cost of processing an additional boundary data function is only 0.1 seconds.

Technically, the method described is based on the classical nested dissection algorithm, and uses the same type of recursive subdivision of the grid via a sequence of minimum-cut partitions. For each one of the resulting sub-domains, we construct a reduced operator that "lives" on the boundary of the sub-domain, and can be conceptually thought of as a discrete analog of its Dirichlet-to-Neumann operator. These operators are in principle dense matrices, but by exploiting data sparsity as in [7], linear overall complexity is achieved.

1.6. **Outline of paper.** The paper presents a linear scaling variation of the nested dissection method which computes the Dirichlet-to-Neumann operator. The first step is to partition the domain into a quad-tree of nested boxes, see Section 2. For each of the smallest boxes, a solution operator is computed. Hierarchically merging these operators, moving up through the tree, the global Dirichlet-to-Neumann operator is computed, see Section 3. The solution operators have internal structure (see Section 4) which is exploited to improve the complexity of the method from $O(N^{1.5})$ to $O(N)$ see Section 5. Numerical experiments illustrate the performance and scaling of the proposed method, see Section 6.

2. Tree structure

When the box domain contains a large number of points, computing the solution operator directly is computationally prohibitive. Thus we tessellate the box into smaller boxes where the solution operator can be computed for little cost. In this section, we describe the simplest such tessellation.

Suppose the square domain Ω contains $N = n^2$ points ($n \times n$ grid). Let $N_{\rm leaf}$ denote a tuning parameter chosen so that a matrix of size $N_{\rm leaf} \times N_{\rm leaf}$ can be inverted quickly by brute force. The optimal choice of $N_{\rm leaf}$ depends on the computing environment, but we have found that $N_{\rm leaf} = 100$ is often a good choice. Let L be the smallest integer such that when Ω is partitioned into 4^L equisized boxes, each box contains no more than $N_{\rm leaf}$ points. These 4^L small boxes are called the *leaves* of the tree. Merge the leaves by sets of fours into boxes with twice the side length, to form the 4^{L-1} boxes that make up the next level in the tree. This process is repeated until Ω is recovered. We call Ω the *root* of the tree.

The set consisting of all boxes of the same size forms what we call a *level*. We label the levels using the integer $\ell = 0, 1, 2, \dots, L$, with $\ell = 0$ denoting the root, and $\ell = L$ denoting the leaves.

3. A variation of the nested dissection algorithm

This section describes an accelerated direct solver that is particularly suitable for what we call "pure" boundary value problems such as (1.1) in which there is no body load, and where the solution is sought only near the boundary. The idea is to construct a solution operator G that maps the given boundary data to the sought potential values (or flows) on the boundary. Letting $N_{\rm b}$ denote the number of nodes on the boundary of the domain, G is a dense $N_{\rm b} \times N_{\rm b}$ matrix.

Technically, the solution operator G is constructed via a divide-and-conquer approach (analogous to the one used in the classical nested dissection scheme): First a solution operator is constructed for each "leaf" in the quadtree described in Section 2, then solution operators for larger boxes are constructed via a hierarchical merging process in a single sweep through the tree, going from smaller to larger boxes.

For a grid with N nodes, the process described in this section requires $O(N^{1.5})$ operations to construct the solution operator, and then each subsequent solve (which consists merely of applying the solution operator) requires $O(N)$ operations. Techniques for accelerating these two costs to $O(N)$ and $O(N^{0.5})$, respectively, are then described in Sections 4 and 5.

3.1. **The solution operator and the Schur complement.** This subsection provides a precise definition of the concept of a "solution operator" associated with a subdomain P of the computational grid. For simplicity, we assume that P is a square or rectangular domain. We partition P into interior nodes and boundary nodes:

$$P = P_{\rm i} \cup P_{\rm b},$$

where $P_{\rm i}$ is defined as the set of nodes that have all four neighbors inside P, see Figure 1. (Note that the set P consists of all nodes at which the potential is unknown, and $P_{\rm b}$ is the outermost ring of these nodes, *not* the nodes at which Dirichlet data is prescribed.)

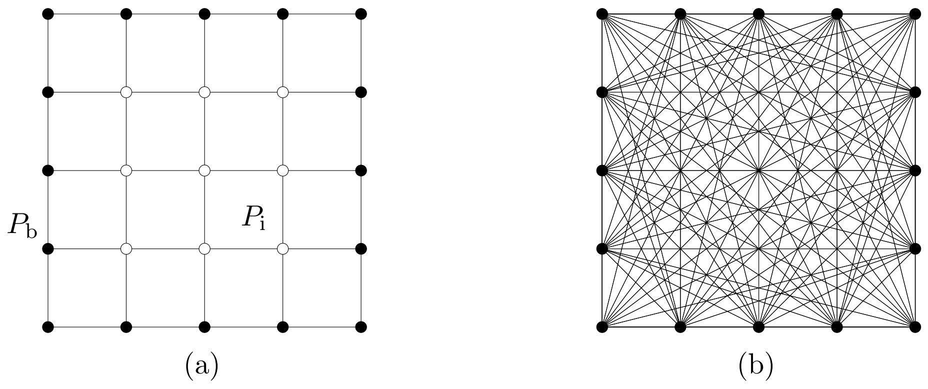

FIGURE 1. (a) Labeling of nodes for constructing the Schur complement of a leaf. $P_{\rm i}$ are the interior nodes (hollow), and $P_{\rm b}$ are the boundary nodes (solid). (b) After the merge, all internal nodes are "eliminated" but now all nodes communicate directly (i.e. the Schur complement S is dense).

Let $\boldsymbol{u}_{\rm b}$ and $\boldsymbol{u}_{\rm i}$ denote the potentials at the boundary nodes and the interior nodes, respectively. Reordering the equilibrium equation (restricted to P), we find that $\boldsymbol{u}_{\rm b}$ and $\boldsymbol{u}_{\rm i}$ must satisfy

$$\begin{bmatrix} \mathsf{A}_{\rm b,b} & \mathsf{A}_{\rm b,i} \\ \mathsf{A}_{\rm i,b} & \mathsf{A}_{\rm i,i} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_{\rm b} \\ \boldsymbol{u}_{\rm i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_{\rm b} \\ \boldsymbol{f}_{\rm i} \end{bmatrix}. \tag{3.1}$$

Since we assume that there is no body load, we know that $\boldsymbol{f}_{\rm i} = 0$. (Note that in general $\boldsymbol{f}_{\rm b} \neq \boldsymbol{0}$ since this term incorporates flows coming from neighboring domains.) Eliminating $\boldsymbol{u}_{\rm i}$ from (3.1), we therefore find

$$\mathsf{S}\,\boldsymbol{u}_{\rm b} = \boldsymbol{f}_{\rm b},$$

where S is the matrix

$$\mathsf{S} = \mathsf{A}_{\rm b,b} - \mathsf{A}_{\rm b,i}\,\mathsf{A}_{\rm i,i}^{-1}\,\mathsf{A}_{\rm i,b}. \tag{3.2}$$

We refer to S as the *Schur complement* associated with the subdomain P; the *solution operator* is then $\mathsf{G} = \mathsf{S}^{-1}$.

3.2. **Merging two Schur complements.** In this section, we present a technique for merging the Schur complements for two adjacent boxes. Let us call the two boxes $\Omega^{(\rm w)}$ and $\Omega^{(\rm e)}$ (for "west" and "east"). Further, let $P^{(\rm w)}$ and $P^{(\rm e)}$ denote the nodes on the boundaries of these two boxes, and let $\mathsf{S}^{(\rm w)}$ and $\mathsf{S}^{(\rm e)}$ denote Schur complements supported on these two sets of boundary nodes, see Figure 2.

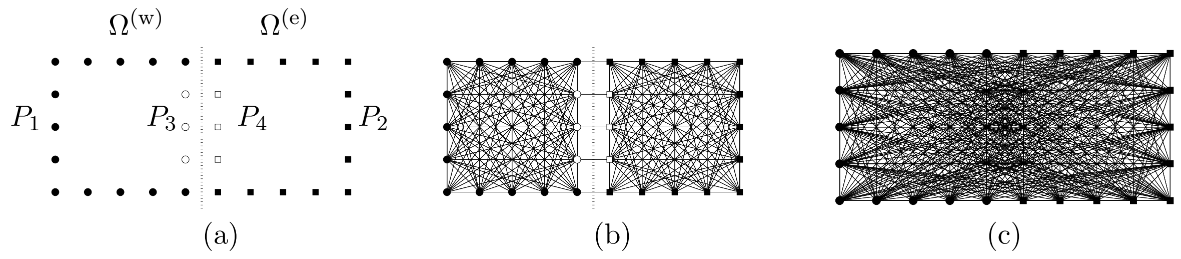

FIGURE 2. (a) Labeling of nodes for the merge operation described in Section 3. The nodes in P_1 and P_3 are round, and the nodes in P_2 and P_4 are square. The hollow nodes are interior to the union of the two boxes $\Omega^{(w)}$ and $\Omega^{(e)}$. (b) Connections between nodes before the merge. (c) Connections between nodes after eliminating the interior (hollow) nodes.

The objective of the merge is to eliminate the nodes that are now "interior" to the larger box formed by the union of the two smaller boxes; these nodes are marked as blue in Figure 2. To eliminate these points, we first partition the nodes in $P^{(w)}$ and $P^{(e)}$ so that

$$P^{(w)} = P_1 \cup P_3, \qquad \text{and} \qquad P^{(e)} = P_2 \cup P_4, \tag{3.3}$$

and so that $P_1 \cup P_2$ forms the boundary of the larger box, while the nodes in $P_3 \cup P_4$ are interior, see Figure 2. Partition the Schur complements $\mathsf{S}^{(w)}$ and $\mathsf{S}^{(e)}$ analogously:

$$\mathsf{S}^{(w)} = \begin{bmatrix} \mathsf{S}_{11} & \mathsf{S}_{13} \\ \mathsf{S}_{31} & \mathsf{S}_{33} \end{bmatrix}, \qquad \text{and} \qquad \mathsf{S}^{(e)} = \begin{bmatrix} \mathsf{S}_{22} & \mathsf{S}_{24} \\ \mathsf{S}_{42} & \mathsf{S}_{44} \end{bmatrix}.$$

Supposing that the interior edges are unloaded, equation (1.3) restricted to the union of the two boxes now reads

$$\left[\begin{array}{cc|cc} \mathsf{S}_{11} & \mathsf{A}_{12} & \mathsf{S}_{13} & 0 \\ \mathsf{A}_{21} & \mathsf{S}_{22} & 0 & \mathsf{S}_{24} \\ \hline \mathsf{S}_{31} & 0 & \mathsf{S}_{33} & \mathsf{A}_{34} \\ 0 & \mathsf{S}_{24} & \mathsf{A}_{43} & \mathsf{S}_{44} \end{array}\right] \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{u}_3 \\ \boldsymbol{u}_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \tag{3.4}$$

where A_{ij} are the relevant sub-matrices of the original discrete Laplacian A. From (3.4), one finds that the Schur complement of the union box is

$$\mathsf{S} = \begin{bmatrix} \mathsf{S}_{11} & \mathsf{A}_{12} \\ \mathsf{A}_{21} & \mathsf{S}_{22} \end{bmatrix} - \begin{bmatrix} \mathsf{S}_{13} & 0 \\ 0 & \mathsf{S}_{24} \end{bmatrix} \begin{bmatrix} \mathsf{S}_{33} & \mathsf{A}_{34} \\ \mathsf{A}_{43} & \mathsf{S}_{44} \end{bmatrix}^{-1} \begin{bmatrix} \mathsf{S}_{31} & 0 \\ 0 & \mathsf{S}_{42} \end{bmatrix}. \tag{3.5}$$

3.3. **The full algorithm.** For future reference, let us summarize the algorithm described:

(1) *Construct a quad-tree:* Partition the grid into a hierarchy of boxes as described in Section 2.

(2) *Process the leaves:* For each leaf box in the tree, construct its Schur complement as described in Section 3.1.

(3) *Hierarchical merge:* Loop over all levels of the tree, from finer to smaller. For each box on a level, compute its Schur complement by merging the (already computed) Schur complements of its children as described in Section 3.2.

(4) *Process the root of the tree:* After completing Step 3, the Schur complement for the entire domain is available. Invert (or factor) it to construct the solution operator.

Remark 3.1. For simplicity, the algorithm is described in a level-by-level manner (process all leaves first, then proceed one level at a time in going upwards). In fact, there is flexibility to travel through the tree in any order that ensures that no node is processed before its children. Since all Schur complements can be discarded once their information has been passed on to a parent, smarter orderings can greatly reduce the memory requirements.

3.4. **Asymptotic complexity of the algorithm.** As before, let $N = n^2$ denote the total number of points in the grid, let $N_{\rm leaf}$ denote the maximum number of points on a leaf, and let L denote the number of levels so that $N \leq 4^L\,N_{\rm leaf}$.

The cost to process one leaf in Step 2 in Section 3.3 is $O(N_{\rm leaf}^3)$. Since there are 4^L leaves, the total cost of Step 2 is therefore $4^L\,N_{\rm leaf}^3 \sim N\,N_{\rm leaf}^2$. Since $N_{\rm leaf}$ is a small constant number (in principle one could set $N_{\rm leaf} = 1$) the leaf processing cost is $O(N)$.

Next consider the cost of constructing the Schur complement of a box on level ℓ in executing Step 3 in Section 3.3. Note that all boxes involved have $O(n\,2^{-\ell})$ points along each side. Since some matrices in (3.5) are dense, the cost for each merge is proportional to $(n\,2^{-\ell})^3 = n^3\,2^{-3\ell}$. Since we need to compute $2^{2\ell}$ Schur complements on level ℓ, the total cost of Step 3 is then $\sum_{\ell=1}^{L} 2^{2\ell}\,n^3\,2^{-3\ell} = n^3\sum_{\ell=1}^{L} 2^{-\ell} \approx n^3$.

Since the cost of the final inversion/factorization in Step 4 is clearly n^3, the total cost of the algorithm in Section 3.3 is $O(n^3) = O(N^{1.5})$.

4. Compressible matrices

To improve the scaling of the nested dissection method, a more efficient technique for evaluating (3.5) will be implemented. We will exploit that while the matrices S_{ij} are all dense, they in the present context have additional structure: S_{ij} is when $i \neq j$ to high precision rank deficient, and S_{ii} has a structure that we call *Hierarchically Block Separable (HBS)*. This section briefly describes the HBS property, for details see [5]. We note that the HBS property is very similar to the concept of *Hierarchically Semi-Separable (HSS)* matrices [9, 1] which has previously been used in an analogous context [2]. Other researchers have used the somewhat related $\mathcal{H}$-matrix concept for similar purposes [6, 8].

4.1. **Block separable.** Let M be an $mp \times mp$ matrix that is blocked into $p\times p$ blocks, each of size $m\times m$. We say that M is "block separable" with "block-rank" k if for $\tau = 1,\,2,\,\dots,\,p$, there exist $n\times k$ matrices U_τ and V_τ such that each off-diagonal block $\mathsf{M}_{\sigma,\tau}$ of M admits the factorization

$$\begin{array}{ccccc} \mathsf{M}_{\sigma,\tau} & = & \mathsf{U}_\sigma & \tilde{\mathsf{M}}_{\sigma,\tau} & \mathsf{V}_\tau^*, \\ m\times m & & m\times k & k\times k & k\times m \end{array} \qquad \sigma,\tau\in\{1,\,2,\,\dots,\,p\},\qquad \sigma\neq\tau. \tag{4.1}$$

Observe that the columns of U_σ must form a basis for the columns of all off-diagonal blocks in row σ, and analogously, the columns of V_τ must form a basis for the rows in all the off-diagonal blocks in column τ. When (4.1) holds, the matrix M admits a block factorization

$$\begin{array}{ccccccc} \mathsf{M} & = & \mathsf{U} & \tilde{\mathsf{M}} & \mathsf{V}^* & + & \mathsf{D}, \\ mp\times mp & & mp\times kp & kp\times kp & kp\times mp & & mp\times mp \end{array} \tag{4.2}$$

where

$$\mathsf{U} = \mathrm{diag}(\mathsf{U}_1,\,\mathsf{U}_2,\,\dots,\,\mathsf{U}_p),\quad \mathsf{V} = \mathrm{diag}(\mathsf{V}_1,\,\mathsf{V}_2,\,\dots,\,\mathsf{V}_p),\quad \mathsf{D} = \mathrm{diag}(\mathsf{D}_1,\,\mathsf{D}_2,\,\dots,\,\mathsf{D}_p),$$

and

$$\tilde{\mathsf{M}} = \left[\begin{array}{cccc} 0 & \tilde{\mathsf{M}}_{12} & \tilde{\mathsf{M}}_{13} & \cdots \\ \tilde{\mathsf{M}}_{21} & 0 & \tilde{\mathsf{M}}_{23} & \cdots \\ \tilde{\mathsf{M}}_{31} & \tilde{\mathsf{M}}_{32} & 0 & \cdots \\ \vdots & \vdots & \vdots & \end{array}\right].$$

4.2. **Heirarchically Block-Separable.** Informally speaking, a matrix M is *Heirarchically Block-Separable* (HBS), if it is amenable to a *telescoping* block factorization. In other words, in addition to the matrix M being block separable, so is $\tilde{\mathsf{M}}$ once it has been reblocked to form a matrix with $p/2 \times p/2$ blocks. Likewise, the middle matrix from the block separable factorization of $\tilde{\mathsf{M}}$ will be block separable, etc.

For example, a "3 level" factorization of M is

$$\mathsf{M} = \mathsf{U}^{(3)}\big(\mathsf{U}^{(2)}\big(\mathsf{U}^{(1)}\,\tilde{\mathsf{M}}^{(0)}\,(\mathsf{V}^{(1)})^* + \mathsf{B}^{(1)}\big)(\mathsf{V}^{(2)})^* + \mathsf{B}^{(2)}\big)(\mathsf{V}^{(3)})^* + \mathsf{D}^{(3)}, \tag{4.3}$$

where the superscript denotes the level.

The HBS representation of an $N \times N$ matrix requires $O(Nk)$ to store and to apply to a vector. In addition, there exist an $O(Nk^2)$ inversion technique, see [5].

5. Structured matrix algebra in nested dissection

In this section, we apply the structured matrix techniques introduced in Section 4 to reduce the complexity of the solver of Section 3 from $O(N^{1.5})$ to $O(N)$. The key task that we need to accelerate is the construction of the Schur complement for a parent box from the Schur complements of its two children. The formula that need to be evaluated is, cf. (3.5),

$$\mathsf{S} = \begin{bmatrix} \mathsf{S}_{11} & \mathsf{A}_{12} \\ \mathsf{A}_{21} & \mathsf{S}_{22} \end{bmatrix} - \begin{bmatrix} \mathsf{S}_{13} & 0 \\ 0 & \mathsf{S}_{24} \end{bmatrix} \begin{bmatrix} \mathsf{S}_{33} & \mathsf{A}_{34} \\ \mathsf{A}_{43} & \mathsf{S}_{44} \end{bmatrix}^{-1} \begin{bmatrix} \mathsf{S}_{31} & 0 \\ 0 & \mathsf{S}_{42} \end{bmatrix}. \tag{5.1}$$

Observe that S_{13}, S_{31}, S_{24}, and S_{42} all have low (numerical) rank, so the second term in (5.1) is itself a low-rank matrix, and once it has been computed, the task of forming S consists merely of a low-rank update that is inexpensive.

Now the task of forming the second term in (5.1) is simple once we have computed the matrices

$$\begin{bmatrix} \mathsf{X}_3 \\ \mathsf{X}_4 \end{bmatrix} = \begin{bmatrix} \mathsf{S}_{33} & \mathsf{A}_{34} \\ \mathsf{A}_{43} & \mathsf{S}_{44} \end{bmatrix}^{-1} \begin{bmatrix} \mathsf{R}_3 \\ \mathsf{R}_4 \end{bmatrix}, \tag{5.2}$$

where R_3 and R_4 are defined by

$$\mathsf{R}_3 = \big[\mathsf{S}_{31}\ 0\big] \qquad \text{and} \qquad \mathsf{R}_4 = \big[0\ \mathsf{S}_{42}\big]. \tag{5.3}$$

It is easily seen that X_3 and X_4 defined by (5.2) can be expressed as

$$\mathsf{X}_3 = (\mathsf{S}_{44} - \mathsf{A}_{43}\mathsf{S}_{33}^{-1}\mathsf{A}_{34})^{-1}(\mathsf{R}_4 - \mathsf{A}_{43}\mathsf{S}_{33}^{-1}\mathsf{R}_3) \tag{5.4}$$

and

$$\mathsf{X}_4 = \mathsf{S}_{33}^{-1}\mathsf{R}_3 - \mathsf{S}_{33}^{-1}\mathsf{A}_{34}\mathsf{X}_4. \tag{5.5}$$

The formulas (5.4) and (5.5) can be rapidly evaluated since:

- S_{33} is an HBS matrix and so S_{33}^{-1} can be computed rapidly.
- A_{34} and A_{43} are anti-diagonal matrices. Thus the product $\mathsf{A}_{43}\mathsf{S}_{33}^{-1}\mathsf{A}_{34}$ is HBS and can be computed rapidly.
- $\mathsf{S}_{44} - \mathsf{A}_{43}\mathsf{S}_{33}^{-1}\mathsf{A}_{34}$ is the sum of two HBS matrices and is also HBS.
- The matrices R_3 and R_4 have low (numerical) rank.

6. Numerical experiments

In this section, we illustrate the capabilities of the proposed method for constructing solution operators for problems of the form

$$\begin{cases} -\Delta u(\boldsymbol{x}) + b(\boldsymbol{x})u_x(\boldsymbol{x}) + c(\boldsymbol{x})u_y(\boldsymbol{x}) + d(\boldsymbol{x})u(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Omega = [0,1]^2, \\ \qquad\qquad\qquad\qquad\qquad\qquad u(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma, \end{cases} \tag{6.1}$$

where b, c, and d are functions defined on Ω, and the boundary data g is defined on Γ. Section 6.1 substantiates the claims regarding asymptotic costs and reports on the accuracy of proposed method for several different model problems. Section 6.2 illustrates the scaling of the proposed technique with timing results.

For all problems, the domain is discretized with a uniform grid of $n \times n$ points so that the grid spacing is $h = 1/(n-1)$. Equation (6.1) is discretized with the finite difference scheme corresponding to the five point stencil. For example, when a node k is in the interior of Ω, the discretization of the differential operator in (6.1) is

$$\frac{1}{h^2}\big[4u(k) - u(k_n) - u(k_s) - u(k_w) - u(k_e)\big] + \\ \frac{1}{h}b(k)\big[u(k_e) - u(k_w)\big] + \frac{1}{h}c(k)\big[u(k_n) - u(k_s)\big] + d(k)u(k),$$

where k_e, k_n, k_w, k_s denote the grid points to the “east”, “north”, “west”, and “south” of k, respectively.

All experiments are run on a Lenovo laptop computer with 2.4GHz Intel i5 processor and 8GB of RAM. The method was implemented in Matlab. While this implementation is unoptimized, we believe it is sufficient for illustrating the potential of the proposed technique. Additionally, for all experiments the tolerance of the HSS representation of the Schur complement matrix is set to 10^{-7}. For all experiments, let $N = n^2$.

6.1. **Range of problems with optimal scaling.**

The proposed method for constructing Dirichlet-to-Neumann operators has been applied to several problems to investigate its asymptotic complexity. The problems are:

- *Laplace:* Let $b(\boldsymbol{x}) = c(\boldsymbol{x}) = d(\boldsymbol{x}) = 0$.
- *Diffusion-Convection I:* Let $c(\boldsymbol{x}) = d(\boldsymbol{x}) = 0$ and the convection in the x direction be constant: $b(\boldsymbol{x}) = 100$.
- *Diffusion-Convection II:* Same as *Diffusion-Convection I*, but with $b(\boldsymbol{x}) = 100$.
- *Diffusion-convection III:* Introduce a divergence free convection field by setting $b(\boldsymbol{x}) = 125\cos(4\pi y)$ and $c(\boldsymbol{x}) = 125\sin(4\pi x)$, and $d(\boldsymbol{x}) = 0$.
- *Diffusion-convection IV:* Introduce a convection field with sources and sinks by setting Let $b(\boldsymbol{x}) = 125\cos(4\pi x)$, $c(\boldsymbol{x}) = 125\sin(4\pi y)$, and $d(\boldsymbol{x}) = 0$.
- *Helmholtz I:* Consider the Helmholtz equation corresponding to a domain that is roughly 1.5×1.5 wavelengths large: $b(\boldsymbol{x}) = c(\boldsymbol{x}) = 0$ and $d(\boldsymbol{x}) = -100$.
- *Helmholtz II:* Consider the Helmholtz equation corresponding to a domain that is roughly 10×10 wavelengths large: $b(\boldsymbol{x}) = c(\boldsymbol{x}) = 0$ and $d(\boldsymbol{x}) = -4005$.
- *Helmholtz III:* Consider the Helmholtz equation near a resonance: $b(\boldsymbol{x}) = c(\boldsymbol{x}) = 0$ and $d(\boldsymbol{x}) = -\lambda_{10} + 10^{-5}$, where λ_{10} is the tenth eigenvalue of the discrete Laplace operator which are known analytically.
- *Helmholtz IV:* Consider a sequence of Helmholtz problems where the wave-number is increased to keep a constant 40 points per wave-length: $b(\boldsymbol{x}) = c(\boldsymbol{x}) = 0$ and $d(\boldsymbol{x}) = -\left(\frac{2\pi n}{40}\right)^2$.
- *Random Laplacian I:* Let the matrix A reflect an elliptic equilibrium problem on a network instead of a continuum PDE. In this case, the network is the square grid where each link is assigned a random *conductivity* between varying between 1 and 2. The potential at any single node is the weighted average of the potentials of its four neighbors, where the weights are the conductivities.
- *Random Laplacian II:* Same as *Random Laplacian I*, but now the conductivities vary between 1 and 1000.

Table 1 reports the amount of memory $M(n)$ in MB required to store the Dirichlet-to-Neumann operator for each problem as well as the ratio of $M(n)$ to n. The solution technique will scale linearly for the problems where the $M(n)$ grows linearly with n or $\frac{M(n)}{n}$ remains constant. Helmholtz IV is the only problem for which the method will not scale linearly.

Table 2 reports two errors:

e_1 - the l^2-error in the vector $\mathsf{S}^{-1}\,\boldsymbol{r}$ where $\boldsymbol{r}$ is a unit vector of random direction

e_2 - the l^2-error in the first column of S^{-1}

A slight loss in accuracy is observed for Helmholtz I, IV and Random II problems. There is a substantial loss in accuracy for the Helmholtz III problem. This is to be expected since the matrix A is close to being numerically singular.

Problem	$n = 256$	$n = 512$	$n = 1024$	$n = 2048$
Laplace	0.83 (3.2e-3)	1.62 (3.2e-3)	3.18 (3.1e-3)	6.27 (3.1e-3)
DiffConv I	0.91 (3.5e-3)	1.75 (3.4e-3)	3.32 (3.2e-3)	6.52 (3.2e-3)
DiffConv II	1.10 (4.3e-3)	1.84 (3.6e-3)	3.62 (3.5e-3)	6.87 (3.4e-3)
DiffConv III	0.86 (3.4e-3)	1.70 (3.3e-3)	3.32 (3.2e-3)	6.55 (3.3e-3)
DiffConv IV	0.97 (3.8e-3)	1.83 (3.6e-3)	3.43 (3.3e-3)	6.59 (3.2e-3)
Helmholtz I	0.86 (3.4e-3)	1.67 (3.3e-3)	3.25 (3.2e-3)	6.34 (3.1e-3)
Helmholtz II	1.04 (4.1e-3)	1.91 (3.7e-3)	3.56 (3.5e-3)	6.78 (3.3e-3)
Helmholtz III	0.86 (3.4e-3)	1.67 (3.3e-3)	3.29 (3.2e-3)	6.42 (3.1e-3)
Helmholtz IV	0.89 (3.5e-3)	1.74 (3.4e-3)	3.59 (3.5e-3)	7.89 (3.9e-3)
Random I	0.83 (3.2e-3)	1.64 (3.2e-3)	3.22 (3.1e-3)	6.34 (3.1e-3)
Random II	0.82 (3.2e-3)	1.64 (3.2e-3)	3.23 (3.2e-3)	6.36 (3.1e-3)

TABLE 1. Memory $M(n)$ in MB required to store the solution operator for the problems listed in Section 6.1. The quantity $\frac{M(n)}{n}$ is reported in parenthesis.

Problem	e_1	e_2
Laplace	1.08e-6	4.9e-8
DiffConv I	6.44e-6	1.71e-6
DiffConv II	1.44e-6	1.3e-8
DiffConv III	3.12e-6	2.49e-7
DiffConv IV	3.05e-6	2.49e-7
Helmholtz I	5.55e-5	8.00e-8
Helmholtz II	2.54e-6	4.53e-7
Helmholtz III	6.30e-2	1.389e-1
Helmholtz IV	2.86e-4	9.20e-5
Random I	5.94e-6	2.41e-6
Random II	2.35e-4	3.15e-4

TABLE 2. Errors e_1 and e_2 for the solution operator for the problems listed in Section 6.1.

6.2. **Performance.** The proposed method is applied to construct the solution operator resulting from the finite difference discretization of two problems: *Laplace* and *Helmholtz IV*.

In these experiment, we increase the number of discretization points from 512^2 to 4096^2. Table 3 reports:

T_{solve} - the time in seconds for constructing the Dirichlet-to-Neumann operator
T_{apply} - the time in seconds for applying the Dirichlet-to-Neumann operator to a vector

As predicted from the previous section, the time to construct the Dirichlet-to-Neuman operator for the *Laplace* problem grows linearly with N while the time to construct the operator for the *Helmholtz IV* problem does not. Notice the cost of applying the solution operator to a vector scales as $O(\sqrt{N})$ for both problems.

N	*Laplace*		*Helmholtz IV*	
	T_{solve} (sec)	T_{apply} (sec)	T_{solve} (sec)	T_{apply} (sec)
512^2	13.44	0.013	50.78	0.013
1024^2	45.25	0.027	193.58	0.027
2048^2	135.01	0.058	765.35	0.056
4096^2	450.73	0.107	3167.56	0.115

Table 3. Times for the approximation of the Dirichlet-to-Neumann operator for the *Laplace* and *Helmholtz IV* problems via the accelerated nested dissection method.

7. Conclusions

This paper presents a fast method for constructing the Dirichlet-to-Neumann operator for elliptic problems with no body loads. Numerical results indicate that the method scales linearly with the number of discretization points N for a variety of problems. Since application of the solution operator scales linearly with the number of boundary points (typically $O(\sqrt{N})$), constructing the solution for multiple right-hand sides is essentially free once the Dirichlet-to-Neumann operator is built. For a problem involving approximately 16 million unknowns, it takes about 8 minutes to build the solver, and 0.1 seconds to apply it to a right-hand side.

The fast direct solver described here relies on the intermediate dense matrices being compressible in the sense of being either of low rank, or having the HBS structure described in Section 4. It is currently not well understood exactly when this holds, but the numerical experiments in Section 6 indicate that the property is remarkably stable across a broad range of test problems.

Acknowledgements: The work reported was supported by NSF grants DMS0748488 and DMS0941476.

References

[1] S. Chandrasekaran and M. Gu, A divide-and-conquer algorithm for the eigendecomposition of symmetric block-diagonal plus semiseparable matrices, Numer. Math. **96** (2004), no. 4, 723–731.
[2] S. Chandrasekaran, M. Gu, X.S. Li, and J. Xia, Superfast multifrontal method for large structured linear systems of equations, SIAM J. Matrix Anal. Appl. **31** (2009), 1382–1411.
[3] A. George, Nested dissection of a regular finite element mesh, SIAM J. Numer. Anal. **10** (1973), 345–363.
[4] A. Gillman, Fast direct solvers for elliptic partial differential equations, Ph.D. thesis, University of Colorado at Boulder, Applied Mathematics, 2011.
[5] A. Gillman, P. Young, and P.G. Martinsson, A direct solver with $o(n)$ complexity for integral equations on one-dimensional domains, 2012, To appear in Frontiers of Mathematics in China.
[6] R. Kriemann L. Grasedyck and S. Le Borne, Domain decomposition based $\mathcal{H}$-LU preconditioning, Numer. Math. **112** (2009), no. 4, 565–600. MR MR2507619 (2010e:65200)

[7] P.G. Martinsson, A fast direct solver for a class of elliptic partial differential equations, J. Sci. Comput. **38** (2009), no. 3, 316–330. MR MR2475654 (2010c:65041)

[8] P. Schmitz and L. Ying, A fast direct solver for elliptic problems on general meshes in 2d, 2010, In review.

[9] Z. Sheng, P. Dewilde, and S. Chandrasekaran, Algorithms to solve hierarchically semi-separable systems, System theory, the Schur algorithm and multidimensional analysis, Oper. Theory Adv. Appl., vol. 176, Birkhäuser, Basel, 2007, pp. 255–294.