2014

A Direct Solver with O(N) Complexity for Variable Coefficient Elliptic PDEs Discretized via a High-Order Composite Spectral Collocation Method

A. Gillman
Dartmouth College

P. G. Martinsson
University of Colorado, Boulder

A direct solver with $O(N)$ complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method

A. Gillman
Department of Mathematics
Dartmouth College

P.G. Martinsson
Department of Applied Mathematics
University of Colorado at Boulder

Abstract: A numerical method for solving elliptic PDEs with variable coefficients on two-dimensional domains is presented. The method is based on high-order composite spectral approximations and is designed for problems with smooth solutions. The resulting system of linear equations is solved using a direct (as opposed to iterative) solver that has optimal $O(N)$ complexity for all stages of the computation when applied to problems with non-oscillatory solutions such as the Laplace and the Stokes equations. Numerical examples demonstrate that the scheme is capable of computing solutions with relative accuracy of 10^{-10} or better, even for challenging problems such as highly oscillatory Helmholtz problems and convection-dominated convection diffusion equations. In terms of speed, it is demonstrated that a problem with a non-oscillatory solution that was discretized using 10^8 nodes was solved in 115 minutes on a personal work-station with two quad-core 3.3GHz CPUs. Since the solver is direct, and the "solution operator" fits in RAM, any solves beyond the first are very fast. In the example with 10^8 unknowns, solves require only 30 seconds.

1. Introduction

1.1. **Problem formulation.** The paper describes a numerical method with optimal $O(N)$ complexity for solving boundary value problems of the form

$$\begin{cases} Au(\boldsymbol{x}) = 0 & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = f(\boldsymbol{x}) & \boldsymbol{x} \in \Gamma, \end{cases} \tag{1}$$

where Ω is a rectangle in the plane with boundary Γ, and where A is a coercive elliptic partial differential operator

$$[Au](\boldsymbol{x}) = -c_{11}(\boldsymbol{x})[\partial_1^2 u](\boldsymbol{x}) - 2c_{12}(\boldsymbol{x})[\partial_1\partial_2 u](\boldsymbol{x}) - c_{22}(\boldsymbol{x})[\partial_2^2 u](\boldsymbol{x}) + c_1(\boldsymbol{x})[\partial_1 u](\boldsymbol{x}) + c_2(\boldsymbol{x})[\partial_2 u](\boldsymbol{x}) + c(\boldsymbol{x})\, u(\boldsymbol{x}). \tag{2}$$

The methodology is based on a high order composite spectral discretization and can be modified to handle a range of different domains, including curved ones. For problems with smooth solutions, we demonstrate that the method can easily produce answers with ten or more correct digits.

The proposed method is based on a *direct solver* which in a single sweep constructs an approximation to the solution operator of (1). This gives the solver several advantages over established linear complexity methods based on *iterative solvers* (e.g. GMRES or multigrid), perhaps most importantly, the new method can solve problems for which iterative methods converge slowly or not at all. The direct solver has $O(N)$ complexity for all stages of the computation. A key feature is that once the solution operator has been built, solves can be executed extremely rapidly, making the scheme excel when solving a sequence of equations with the same operator but different boundary data.

1.2. **Outline of solution procedure.** The method in this paper is comprised of three steps:

(1) The domain is first tessellated into a hierarchical tree of rectangular patches. For each patch on the finest level, a local "solution operator" is built using a dense brute force calculation. The "solution operator" will be defined in Section 1.3; for now we simply note that it encodes all information about the patch that is required to evaluate interactions with other patches.

(2) The larger patches are processed in an upwards pass through the tree, where each parent can be processed once its children have been processed. The processing of a parent node consists of forming its solution operator by "gluing together" the solution operators of its children.
(3) Once the solution operators for all patches have been computed, a solution to the PDE can be computed via a downwards pass through the tree. This step is typically *very* fast.

1.3. **Local solution operators.** The "local solution operators" introduced in Section 1.2 take the form of discrete approximations to the *Dirichlet-to-Neumann*, or "DtN," maps. To explain what these maps do, first observe that for a given boundary function f, the BVP (1) has a unique solution u (recall that we assume A to be coercive). For $\boldsymbol{x} \in \Gamma$, let $g(\boldsymbol{x}) = u_n(\boldsymbol{x})$ denote the normal derivative in the outwards direction of u at $\boldsymbol{x}$. The process for constructing the function g from f is linear, we write it as

$$g = T\, f.$$

Or, equivalently,

$$T:\ u|_\Gamma \mapsto u_n|_\Gamma, \quad \text{where } u \text{ satisfies } Au = 0 \text{ in } \Omega.$$

From a mathematical perspective, the map T is a slightly unpleasant object; it is a hyper-singular integral operator whose kernel exhibits complicated behavior near the corners of Γ. A key observation is that in the present context, these difficulties can be ignored since we limit attention to functions that are smooth. In a sense, we only need to accurately represent the projection of the "true" operator T onto a space of smooth functions (that in particular do not have any corner singularities).

Concretely, given a square box Ω_τ we represent a boundary potential $u|_\Gamma$ and a boundary flux $u_n|_\Gamma$ via tabulation at a set of r tabulation points on each side. (For a leaf box, we use r Gaussian nodes.) The DtN operator T^τ is then represented simply as a dense matrix $\mathbf{T}^\tau$ of size $4r \times 4r$ that maps tabulated boundary potentials to the corresponding tabulated boundary fluxes.

1.4. **Computational complexity.** A straight-forward implementation of the direct solver outlined in Sections 1.2 and 1.3 in which all solution operators $\mathbf{T}^\tau$ are treated as general dense matrices has asymptotic complexity $O(N^{1.5})$ for the "build stage" where the solution operators are constructed, and $O(N \log N)$ complexity for the "solve stage" where the solution operator is applied for a given set of boundary data [16]. This paper demonstrates that by exploiting internal structure in these operators, they can be stored and manipulated efficiently, resulting in optimal $O(N)$ overall complexity.

To be precise, the internal structure exploited is that the off-diagonal blocks of the dense solution operators can to high accuracy be approximated by matrices of low rank. This property is a result of the fact that for a patch Ω_τ, the matrix $\mathbf{T}^\tau$ is a discrete approximation of the continuum DtN operator T^τ, which is an integral operator whose kernel is smooth away from the diagonal.

Remark 1.1. The proposed solver can with slight modifications be applied to non-coercive problems such as the Helmholtz equation. If the equation is kept fixed while N is increased, $O(N)$ complexity is retained. However, in the context of elliptic problems with oscillatory solutions, it is common to scale N to the wave-length so that the number of discretization points per wave-length is fixed as N increases. Our accelerated technique will in this situation lead to a practical speed-up, but will have the same $O(N^{1.5})$ asymptotic scaling as the basic method that does not use fast operator algebra.

1.5. **Prior work.** The direct solver outlined in Section 1.2 is an evolution of a sequence of direct solvers for integral equations dating back to [17] and later [9, 8, 10, 4, 3]. The common idea is to build a global solution operator by splitting the domain into a hierarchical tree of patches, build a local solution operator for each "leaf" patch, and then build solution operators for larger patches via a hierarchical merge procedure in a sweep over the tree from smaller to larger patches. In the context of integral equations, the "solution operator" is a type of scattering matrix while in the present context, the solution operator is a DtN operator.

The direct solvers [17, 9, 8, 10, 4], designed for dense systems, are conceptually related to earlier work on direct solvers for sparse systems arising from finite difference and finite element discretizations of elliptic PDEs such as the classical nested dissection method of George [6, 11] and the multifrontal methods by Duff and others [5]. These techniques typically require $O(N^{1.5})$ operations to construct the LU-factorization of a sparse coefficient matrix arising from the discretization of an elliptic PDE on a planar domain, with the dominant cost being the formation of Schur complements and LU-factorizations of dense matrices of size up to $O(N^{0.5}) \times O(N^{0.5})$. It was in the last several years demonstrated that these dense matrices have internal structure that allows the direct solver to be accelerated to linear or close to linear complexity, see, e.g., [22, 7, 13, 14, 19]. These accelerated nested dissection methods are closely related to the fast direct solver presented in this manuscript. An important difference is that the method in the present paper allows high order discretizations to be used without increasing the cost of the direct solver. To be technical, the solvers in [22, 7, 13, 14, 19] are based on an underlying finite difference or finite element discretization. High order discretization in this context leads to large frontal matrices (since the "dividers" that partition the grid have to be wide), and consequently very high cost of the LU-factorization.

Our discretization scheme is related to earlier work on spectral collocation methods on composite ("multi-domain") grids, such as, e.g., [12, 23], and in particular Pfeiffer *et al* [18]. For a detailed review of the similarities and differences, see [16].

An $O(N^{1.5})$ complexity version of the direct solver described in this paper was presented in [16] which in turn is based on [15]. In addition to the improvement in complexity, this paper describes a new representation of the local solution operators that leads to cleaner implementation of the direct solvers and allows greater flexibility in executing the leaf computation, see Remark 3.1.

1.6. **Outline of paper.** Section 2 introduces the mesh of Gaussian nodes that forms our basic computational grid. Sections 3, 4, and 5 describe a relatively simple direct solver with $O(N^{1.5})$ complexity. Sections 6, 7, and 8 describe how to improve the asymptotic complexity of the direct solver from $O(N^{1.5})$ to $O(N)$ by exploiting internal structure in certain dense matrices. Section 9 describes numerical examples and Section 10 summarizes the key findings.

2. Discretization

Partition the domain Ω into a collection of square (or possibly rectangular) boxes, called *leaf boxes*. On the edges of each leaf, place q Gaussian interpolation points. The size of the leaf boxes, and the parameter q should be chosen so that any potential solution u of (1), as well as its first and second derivatives, can be accurately interpolated from their values at these points ($q = 21$ is often a good choice). Let $\{\boldsymbol{x}_k\}_{k=1}^N$ denote the collection of interpolation points on all boundaries.

Next construct a binary tree on the collection of leaf boxes by hierarchically merging them, making sure that all boxes on the same level are roughly of the same size, cf. Figure 1. The boxes should be ordered so that if τ is a parent of a box σ, then $\tau < \sigma$. We also assume that the root of the tree (i.e. the full box Ω) has index $\tau = 1$. We let Ω_τ denote the domain associated with box τ.

With each box τ, we define two index vectors I_{i}^τ and I_{e}^τ as follows:

- I_{e}^τ A list of all *exterior* nodes of τ. In other words, $k \in I_{\text{e}}^\tau$ iff $\boldsymbol{x}_k$ lies on the boundary of Ω_τ.
- I_{i}^τ For a parent τ, I_{i}^τ is a list of all its *interior* nodes that are not interior nodes of its children. For a leaf τ, I_{i}^τ is empty.

Let $\mathbf{u} \in \mathbb{R}^N$ denote a vector holding approximations to the values of u of (1), in other words,

$$\mathbf{u}(k) \approx u(\boldsymbol{x}_k).$$

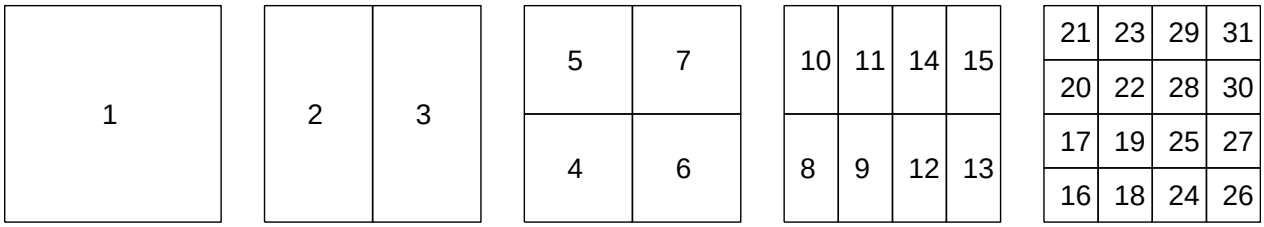

FIGURE 1. The square domain Ω is split into 4×4 leaf boxes. These are then gathered into a binary tree of successively larger boxes as described in Section 5.1. One possible enumeration of the boxes in the tree is shown, but note that the only restriction is that if box τ is the parent of box σ, then $\tau < \sigma$.

Finally, let $\mathbf{v} \in \mathbb{R}^N$ denote a vector holding approximations to the boundary fluxes of the solution u of (1), in other words

$$\mathbf{v}(k) \approx \begin{cases} \partial_2 u(\boldsymbol{x}_k), & \text{when } \boldsymbol{x}_j \text{ lies on a horizontal edge,} \\ \partial_1 u(\boldsymbol{x}_k), & \text{when } \boldsymbol{x}_j \text{ lies on a vertical edge.} \end{cases}$$

Note the $\mathbf{v}(k)$ represents an *outgoing flux* on certain boxes and an *incoming flux* on others. This is a deliberate choice to avoid problems with signs when matching fluxes of touching boxes.

3. Constructing the Dirichlet-to-Neumann map for a leaf

This section describes a spectral method for computing a discrete approximation to the DtN map T^τ associated with a leaf box Ω_τ. In other words, if u is a solution of (1), we seek a matrix $\mathbf{T}^\tau$ of size $4q \times 4q$ such that

$$\mathbf{v}(I_{\text{e}}^\tau) \approx \mathbf{T}^\tau \, \mathbf{u}(I_{\text{e}}^\tau). \tag{3}$$

Conceptually, we proceed as follows: Given a vector $\mathbf{u}(I_{\text{e}}^\tau)$ of potential values tabulated on the boundary of Ω_τ, form for each side the unique polynomial of degree at most $q-1$ that interpolates the q specified values of u. This yields Dirichlet boundary data on Ω_τ in the form of four polynomials. Solve the restriction of (1) to Ω_τ for the specified boundary data using a spectral method on a local tensor product grid of $q \times q$ *Chebyshev nodes.* The vector $\mathbf{v}(I_{\text{e}}^\tau)$ is obtained by spectral differentiation of the local solution, and then re-tabulating the boundary fluxes to the Gaussian nodes in $\{\boldsymbol{x}_k\}_{k \in I_{\text{e}}^\tau}$.

We give details of the construction in Section 3.2, but as a preliminary step, we first review a classical spectral collocation method for the local solve in Section 3.1

Remark 3.1. Chebyshev nodes are ideal for the leaf computations, and it is in principle also possible to use Chebyshev nodes to represent all boundary-to-boundary "solution operators" such as, e.g., $\mathbf{T}^\tau$ (indeed, this was the approach taken in the first implementation of the proposed method [16]). However, there are at least two substantial benefits to using Gaussian nodes that justify the trouble to retabulate the operators. First, the procedure for merging boundary operators defined for neighboring boxes is much cleaner and involves less bookkeeping since the Gaussian nodes do not include the corner nodes. (Contrast Section 4 of [16] with Section 4.) Second, and more importantly, the use of the Gaussian nodes allows for interpolation between different discretizations. Thus the method can easily be extended to have local refinement when necessary, see Remark 5.2.

3.1. Spectral discretization.

Let Ω_τ denote a rectangular subset of Ω with boundary Γ_τ, and consider the local Dirichlet problem

$$[Au](\boldsymbol{x}) = 0, \qquad \boldsymbol{x} \in \Omega_\tau \tag{4}$$

$$u(\boldsymbol{x}) = h(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma_\tau, \tag{5}$$

(a) (b)

FIGURE 2. Notation for the leaf computation in Section 3. (a) A leaf before elimination of interior (white) nodes. (b) A leaf after elimination of interior nodes.

where the elliptic operator A is defined by (2). We will construct an approximate solution to (4) using a classical spectral collocation method described in, e.g., Trefethen [21]: First, pick a small integer p and let $\{\boldsymbol{z}_k\}_{k=1}^{p^2}$ denote the nodes in a tensor product grid of $p \times p$ Chebyshev nodes on Ω_τ. Let $\mathbf{D}^{(1)}$ and $\mathbf{D}^{(2)}$ denote spectral differentiation matrices corresponding to the operators $\partial/\partial x_1$ and $\partial/\partial x_2$, respectively. The operator (2) is then locally approximated via the $p^2 \times p^2$ matrix

$$\mathbf{A} = -\mathbf{C}_{11}\big(\mathbf{D}^{(1)}\big)^2 - 2\mathbf{C}_{12}\mathbf{D}^{(1)}\mathbf{D}^{(2)} - \mathbf{C}_{22}\big(\mathbf{D}^{(2)}\big)^2 + \mathbf{C}_1\mathbf{D}^{(1)} + \mathbf{C}_2\mathbf{D}^{(2)} + \mathbf{C}, \tag{6}$$

where $\mathbf{C}_{11}$ is the diagonal matrix with diagonal entries $\{c_{11}(\boldsymbol{z}_k)\}_{k=1}^{p^2}$, and the other matrices $\mathbf{C}_{ij}$, $\mathbf{C}_i$, $\mathbf{C}$ are defined analogously.

Let $\mathbf{w} \in \mathbb{R}^{p^2}$ denote a vector holding the desired approximate solution of (4). We populate all entries corresponding to boundary nodes with the Dirichlet data from h, and then enforce a spectral collocation condition at the interior nodes. To formalize, let us partition the index set

$$\{1, 2, \ldots, p^2\} = J_{\mathrm{e}} \cup J_{\mathrm{i}}$$

in such a way that J_{e} contains the $4(p-1)$ nodes on the boundary of Ω_τ, and J_{i} denotes the set of $(p-2)^2$ interior nodes, see Figure 2(a). Then partition the vector $\mathbf{w}$ into two parts corresponding to internal and exterior nodes via

$$\mathbf{w}_{\mathrm{i}} = \mathbf{w}(J_{\mathrm{i}}), \quad \mathbf{w}_{\mathrm{e}} = \mathbf{w}(_{\mathrm{e}}).$$

J Analogously, partition $\mathbf{A}$ into four parts via

$$\mathbf{A}_{\mathrm{i,i}} = \mathbf{A}(J_{\mathrm{i}}, J_{\mathrm{i}}), \quad \mathbf{A}_{\mathrm{i,e}} = \mathbf{A}(J_{\mathrm{i}}, J_{\mathrm{e}}), \quad \mathbf{A}_{\mathrm{e,i}} = \mathbf{A}(J_{\mathrm{e}}, J_{\mathrm{i}}), \quad \mathbf{A}_{\mathrm{e,e}} = \mathbf{A}(J_{\mathrm{e}}, J_{\mathrm{e}}).$$

The potential at the exterior nodes is now given directly from the boundary condition:

$$\mathbf{w}_{\mathrm{e}} = \left[h(\boldsymbol{z}_k)\right]_{k \in J_{\mathrm{e}}}.$$

For the internal nodes, we enforce the PDE (4) via direct collocation:

$$\mathbf{A}_{\mathrm{i,i}}\,\mathbf{w}_{\mathrm{i}} + \mathbf{A}_{\mathrm{i,e}}\,\mathbf{w}_{\mathrm{e}} = \mathbf{0}. \tag{7}$$

Solving (7) for $\mathbf{w}_{\mathrm{i}}$, we find

$$\mathbf{w}_{\mathrm{i}} = -\mathbf{A}_{\mathrm{i,i}}^{-1}\,\mathbf{A}_{\mathrm{i,e}}\,\mathbf{w}_{\mathrm{e}}, \tag{8}$$

3.2. **Constructing the approximate DtN.** Now that we know how to approximately solve the local Dirichlet problem (4) via a local spectral method, we can build a matrix $\mathbf{T}^\tau$ such that (3) holds to high accuracy. The starting point is a vector $\mathbf{u}(I_\tau) \in \mathbb{R}^{4q}$ of tabulated potential values on the boundary of Ω_τ. We will construct the vector $\mathbf{v}(I_\tau) \in \mathbb{R}^{4q}$ via four linear maps. The combination of these maps is the matrix $\mathbf{T}^\tau$. We henceforth assume that the spectral order of the local Chebyshev grid matches the order of the tabulation on the leaf boundaries so that $p = q$.

Step 1 — re-tabulation from Gaussian nodes to Chebyshev nodes: For each side of Ω_τ, form the unique interpolating polynomial of degree at most $q-1$ that interpolates the q potential values on that side specified by $\mathbf{u}(I_{\mathrm{e}}^\tau)$. Now evaluate these polynomials at the boundary nodes of a $q \times q$ Chebyshev grid on Ω_τ. Observe that for a corner node, we may in the general case get conflicts. For instance, the potential at the south-west corner may get one value from extrapolation of potential values on the south border, and one value from extrapolation of the potential values on the west border. We resolve such conflicts by assigning the corner node the average of the two possibly different values. (In practice, essentially no error occurs since we know that the vector $\mathbf{u}(I_{\mathrm{e}}^\tau)$ tabulates an underlying function that is continuous at the corner.)

Step 2 — spectral solve: Step 1 populates the boundary nodes of the $q \times q$ Chebyshev grid with Dirichlet data. Now determine the potential at all interior points on the Chebyshev grid by executing a local spectral solve, cf. equation (8).

Step 3 — spectral differentiation: After Step 2, the potential is known at all nodes on the local Chebyshev grid. Now perform spectral differentiation to evaluate approximations to $\partial u/\partial x_2$ for the Chebyshev nodes on the two horizontal sides, and $\partial u/\partial x_1$ for the Chebyshev nodes on the two vertical sides.

Step 4 — re-tabulation from the Chebyshev nodes back to Gaussian nodes: After Step 3, the boundary fluxes on $\partial\Omega_\tau$ are specified by four polynomials of degree $q-1$ (specified via tabulation on the relevant Chebyshev nodes). Now simply evaluate these polynomials at the Gaussian nodes on each side to obtain the vector $\mathbf{v}(I_{\mathrm{e}}^\tau)$.

Putting everything together, we find that the matrix $\mathbf{T}^\tau$ is given as a product of four matrices

$$\underset{4q\times 4q}{\mathbf{T}^\tau} = \underset{4q\times 4q}{\mathbf{L}_4} \circ \underset{4q\times q^2}{\mathbf{L}_3} \circ \underset{q^2\times 4(q-1)}{\mathbf{L}_2} \circ \underset{4(q-1)\times 4q}{\mathbf{L}_1}$$

where $\mathbf{L}_i$ is the linear transform corresponding to "Step i" above. Observe that many of these transforms are far from dense, for instance, $\mathbf{L}_1$ and $\mathbf{L}_4$ are 4×4 block matrices with all off-diagonal blocks equal to zero. Exploiting these structures substantially accelerates the computation.

Remark 3.2. The grid of Chebyshev nodes $\{\boldsymbol{z}_k\}_{j=1}^{p^2}$ introduced in Section 3.1 is only used for the local computation. In the final solver, there is no need to store potential values at these grid points — they are used merely for constructing the matrix $\mathbf{T}^\tau$.

4. Merging two DtN maps

Let τ denote a box in the tree with children α and β. In this section, we demonstrate that if the DtN matrices $\mathbf{T}^\alpha$ and $\mathbf{T}^\beta$ for the children are known, then the DtN matrix $\mathbf{T}^\tau$ can be constructed via a purely local computation which we refer to as a "merge" operation.

We start by introducing some notation: Let Ω_τ denote a box with children Ω_α and Ω_β. For concreteness, let us assume that Ω_α and Ω_β share a vertical edge as shown in Figure 3, so that

$$\Omega_\tau = \Omega_\alpha \cup \Omega_\beta.$$

We partition the points on $\partial\Omega_\alpha$ and $\partial\Omega_\beta$ into three sets:

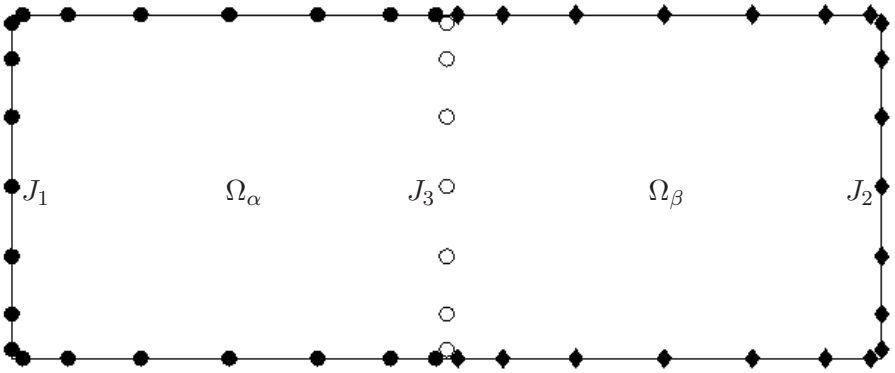

FIGURE 3. Notation for the merge operation described in Section 4. The rectangular domain Ω is formed by two squares Ω_α and Ω_β. The sets J_1 and J_2 form the exterior nodes (black), while J_3 consists of the interior nodes (white).

J_1 Boundary nodes of Ω_α that are not boundary nodes of Ω_β.
J_2 Boundary nodes of Ω_β that are not boundary nodes of Ω_α.
J_3 Boundary nodes of both Ω_α and Ω_β that are *not* boundary nodes of the union box Ω_τ.

Figure 3 illustrates the definitions of the J_k's. Let u denote a solution to (1), with tabulated potential values $\mathbf{u}$ and boundary fluxes $\mathbf{v}$, as described in Section 2. Set

$$\mathbf{u}_\mathrm{i} = \mathbf{u}_3, \qquad \text{and} \qquad \mathbf{u}_\mathrm{e} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix}. \tag{9}$$

Recall that $\mathbf{T}^\alpha$ and $\mathbf{T}^\beta$ denote the operators that map values of the potential u on the boundary to values of $\partial_n u$ on the boundaries of the boxes Ω_α and Ω_β, as described in Section 3. The operators can be partitioned according to the numbering of nodes in Figure 3, resulting in the equations

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{T}^\alpha_{1,1} & \mathbf{T}^\alpha_{1,3} \\ \mathbf{T}^\alpha_{3,1} & \mathbf{T}^\alpha_{3,3} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_3 \end{bmatrix}, \qquad \text{and} \qquad \begin{bmatrix} \mathbf{v}_2 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{T}^\beta_{2,2} & \mathbf{T}^\beta_{2,3} \\ \mathbf{T}^\beta_{3,2} & \mathbf{T}^\beta_{3,3} \end{bmatrix} \begin{bmatrix} \mathbf{u}_2 \\ \mathbf{u}_3 \end{bmatrix}. \tag{10}$$

Our objective is now to construct a solution operator $\mathbf{S}^\tau$ and a DtN matrix $\mathbf{T}^\tau$ such that

$$\mathbf{u}_3 = \mathbf{S}^\tau \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix} \tag{11}$$

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix} = \mathbf{T}^\tau \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix}. \tag{12}$$

To this end, we write (10) as a single equation:

$$\left[\begin{array}{cc|c} \mathbf{T}^\alpha_{1,3} & \mathbf{0} & \mathbf{T}^\alpha_{3,3} \\ \mathbf{0} & \mathbf{T}^\beta_{2,3} & \mathbf{T}^\beta_{3,3} \\ \hline \mathbf{T}^\alpha_{1,3} & -\mathbf{T}^\beta_{2,3} & \mathbf{T}^\alpha_{3,3} - \mathbf{T}^\beta_{3,3} \end{array}\right] \left[\begin{array}{c} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \hline \mathbf{u}_3 \end{array}\right] = \left[\begin{array}{c} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \hline \mathbf{0} \end{array}\right], \tag{13}$$

The last equation directly tells us that (11) holds with

$$\mathbf{S}^\tau = \big(\mathbf{T}^\alpha_{3,3} - \mathbf{T}^\beta_{3,3}\big)^{-1}\big[-\mathbf{T}^\alpha_{3,1} \mid \mathbf{T}^\beta_{3,2}\big]. \tag{14}$$

By eliminating $\mathbf{u}_3$ from (13) by forming a Schur complement, we also find that (12) holds with

$$\mathbf{T}^\tau = \begin{bmatrix} \mathbf{T}^\alpha_{1,1} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^\beta_{2,2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^\alpha_{1,3} \\ \mathbf{T}^\beta_{2,3} \end{bmatrix} \big(\mathbf{T}^\alpha_{3,3} - \mathbf{T}^\beta_{3,3}\big)^{-1}\big[-\mathbf{T}^\alpha_{3,1} \mid \mathbf{T}^\beta_{3,2}\big]. \tag{15}$$

5. The full hierarchical scheme

At this point, we know how to construct the DtN operator for a leaf (Section 3), and how to merge two such operators of neighboring patches to form the DtN operator of their union (Section 4). We are ready to describe the full hierarchical scheme for solving the Dirichlet problem (1). This scheme takes the Dirichlet boundary data f, and constructs an approximation to the solution u. The output is a vector $\mathbf{u}$ that tabulates approximations to u at the Gaussian nodes $\{\boldsymbol{x}_k\}_{k=1}^N$ on all interior edges that were defined in Section 2. To find u at an arbitrary set of target points in Ω, a post-processing step described in Section 5.3 can be used.

5.1. **The algorithm.** Partition the domain into a hierarchical tree as described in Section 2. Then execute a "build stage" in which we construct for each box τ the following two matrices:

- $\mathbf{S}^\tau$ For a parent box τ, $\mathbf{S}^\tau$ is a solution operator that maps values of u on $\partial\Omega_\tau$ to values of u at the interior nodes. In other words, $\mathbf{u}(I_{\text{i}}^\tau) = \mathbf{S}^\tau\,\mathbf{u}(I_{\text{e}}^\tau)$. (For a leaf τ, $\mathbf{S}^\tau$ is not defined.)
- $\mathbf{T}^\tau$ The matrix that maps $\mathbf{u}(I_{\text{e}}^\tau)$ (tabulating values of u on $\partial\Omega_\tau$) to $\mathbf{v}(I_{\text{e}}^\tau)$ (tabulating values of du/dn). In other words, $\mathbf{v}(I_{\text{e}}^\tau) = \mathbf{T}^\tau\,\mathbf{u}(I_{\text{e}}^\tau)$.

(Recall that the index vector I_{e}^τ and I_{i}^τ were defined in Section 2.) The build stage consists of a single sweep over all nodes in the tree. Any bottom-up ordering in which any parent box is processed after its children can be used. For each leaf box τ, an approximation to the local DtN map $\mathbf{T}^\tau$ is constructed using the procedure described in Section 3. For a parent box τ with children σ_1 and σ_2, the matrices $\mathbf{S}^\tau$ and $\mathbf{T}^\tau$ are formed from the DtN operators $\mathbf{T}^{\sigma_1}$ and $\mathbf{T}^{\sigma_2}$ via the process described in Section 4. Algorithm 1 summarizes the build stage.

Once all the matrices $\{\mathbf{S}^\tau\}_\tau$ have been formed, a vector $\mathbf{u}$ holding approximations to the solution u of (1) can be constructed for all discretization points by starting at the root box Ω and moving down the tree toward the leaf boxes. The values of $\mathbf{u}$ for the points on boundary of Ω can be obtained by tabulating the boundary function f. When any box τ is processed, the value of $\mathbf{u}$ is known for all nodes on its boundary (i.e. those listed in I_{e}^τ). The matrix $\mathbf{S}^\tau$ directly maps these values to the values of $\mathbf{u}$ on the nodes in the interior of τ (i.e. those listed in I_{i}^τ). When all nodes have been processed, approximations to u have constructed for all tabulation nodes on interior edges. Algorithm 2 summarizes the solve stage.

Remark 5.1. The merge stage is exact when performed in exact arithmetic. The only approximation involved is the approximation of the solution u on a leaf by its interpolating polynomial.

Remark 5.2. To keep the presentation simple, we consider in this paper only the case of a uniform computational grid. Such grids are obviously not well suited to situations where the regularity of the solution changes across the domain. The method described can *in principle* be modified to handle locally refined grids quite easily. A complication is that the tabulation nodes for two touching boxes will typically not coincide, which requires the introduction of specialized interpolation operators. Efficient refinement strategies also require the development of error indicators that identify the regions where the grid need to be refined. This is work in progress, and will be reported at a later date. We observe that our introduction of Gaussian nodes on the internal boundaries (as opposed to the Chebyshev nodes used in [16]) makes re-interpolation much easier.

5.2. **Asymptotic complexity.** In this section, we determine the asymptotic complexity of the direct solver. Let $N_{\text{leaf}} = 4q$ denote the number of Gaussian nodes on the boundary of a leaf box, and let q^2 denote the number of Chebychev nodes used in the leaf computation. Let L denote the number of levels in the binary tree. This means there are 4^L boxes. Thus the total number of discretization nodes N is approximately $4^L q = \frac{(2^L q)^2}{q}$. (To be exact, $N = 2^{2L+1}q + 2^{L+1}q$.)

ALGORITHM 1 (build solution operators)

This algorithm builds the global Dirichlet-to-Neumann operator for (1).
It also builds all matrices $\mathbf{S}^\tau$ required for constructing u at any interior point.
It is assumed that if node τ is a parent of node σ, then $\tau < \sigma$.

---

(1) **for** $\tau = N_{\text{boxes}}, N_{\text{boxes}} - 1, N_{\text{boxes}} - 2, \ldots, 1$
(2) **if** (τ is a leaf)
(3) Construct $\mathbf{S}^\tau$ via the process described in Section 3.
(4) **else**
(5) Let σ_1 and σ_2 be the children of τ.
(6) Split $I_{\text{e}}^{\sigma_1}$ and $I_{\text{e}}^{\sigma_2}$ into vectors I_1, I_2, and I_3 as shown in Figure 3.
(7) $\mathbf{S}^\tau = \left(\mathbf{T}_{3,3}^{\sigma_1} - \mathbf{T}_{3,3}^{\sigma_2}\right)^{-1} \left[-\mathbf{T}_{3,1}^{\sigma_1} \mid \mathbf{T}_{3,2}^{\sigma_2}\right]$
(8) $\mathbf{T}^\tau = \begin{bmatrix} \mathbf{T}_{1,1}^{\sigma_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_{2,2}^{\sigma_2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}_{1,3}^{\sigma_1} \\ \mathbf{T}_{2,3}^{\sigma_2} \end{bmatrix} \mathbf{S}^\tau.$
(9) Delete $\mathbf{T}^{\sigma_1}$ and $\mathbf{T}^{\sigma_1}$.
(10) **end if**
(11) **end for**

ALGORITHM 2 (solve BVP once solution operator has been built)

This program constructs an approximation $\mathbf{u}$ to the solution u of (1).
It assumes that all matrices $\mathbf{S}^\tau$ have already been constructed in a pre-computation.
It is assumed that if node τ is a parent of node σ, then $\tau < \sigma$.

---

(1) $\mathbf{u}(k) = f(\boldsymbol{x}_k)$ for all $k \in I_{\text{e}}^1$.
(2) **for** $\tau = 1, 2, 3, \ldots, N_{\text{boxes}}$
(3) $\mathbf{u}(I_{\text{i}}^\tau) = \mathbf{S}^\tau \, \mathbf{u}(I_{\text{e}}^\tau)$.
(4) **end for**

Remark: This algorithm outputs the solution on the Gaussian nodes on box boundaries. To get the solution at other points, use the method described in Section 5.3.

The cost to process one leaf is approximately $O(q^6)$. Since there are $\frac{N}{q^2}$ leaf boxes, the total cost of pre-computing approximate DtN operators for all the bottom level is $\frac{N}{q^2} \times q^6 \sim Nq^4$.

Next, consider the cost of constructing the DtN map on level ℓ via the merge operation described in Section 4. For each box on the level ℓ, the operators $\mathbf{T}^\tau$ and $\mathbf{S}^\tau$ are constructed via (14) and (14). These operations involve matrices of size roughly $2^{-\ell}N^{0.5} \times 2^{-\ell}N^{0.5}$. Since there are 4^ℓ boxes per level. The cost on level ℓ of the merge is

$$4^\ell \times \left(2^{-\ell}N^{0.5}\right)^3 \sim 2^{-\ell}N^{1.5}.$$

The total cost for all the merge procedures has complexity

$$\sum_{\ell=1}^{L} 2^{-\ell}N^{1.5} \sim N^{1.5}.$$

Finally, consider the cost of the downwards sweep which solves for the interior unknowns. For any non-leaf box τ on level ℓ, the size of $\mathbf{S}^\tau$ is $2^l q \times 2^l(6q)$ which is approximately $\sim 2^{-\ell}N^{0.5} \times 2^{-\ell}N^{0.5}$. Thus the cost of applying $\mathbf{S}^\tau$ is roughly $(2^{-\ell}N^{0.5})^2 = 2^{-2\ell}N$. So the total cost of the solve step has

complexity

$$\sum_{l=0}^{L-1} 2^{2\ell} 2^{-2\ell} N \sim N \log N.$$

In Section 8, we explain how to exploit structure in the matrices $\mathbf{T}$ and $\mathbf{S}$ to improve the computational cost of both the precomputation and the solve steps.

5.3. **Post-processing.** The direct solver in Algorithm 1 constructs approximations to the solution u of (1) at tabulation nodes at all interior edges. Once these are available, it is easy to construct an approximation to u at an arbitrary point. To illustrate the process, suppose that we seek an approximation to $u(\boldsymbol{y})$, where $\boldsymbol{y}$ is a point located in a leaf τ. We have values of u tabulated at Gaussian nodes on $\partial\Omega_\tau$. These can easily be re-interpolated to the Chebyshev nodes on $\partial\Omega_\tau$. Then u can be reconstructed at the interior Chebyshev nodes via the formula (8); observe that the local solution operator $-\mathbf{A}_{\mathrm{i,i}}^{-1}\mathbf{A}_{\mathrm{i,e}}$ was built when the leaf was originally processed and can be simply retrieved from memory (assuming enough memory is available). Once u is tabulated at the Chebyshev grid on Ω_τ, it is trivial to interpolate it to $\boldsymbol{y}$ or any other point.

6. Compressible matrices

The cost of the direct solver given as Algorithm 1 is dominated by the work done at the very top levels; the matrix operations on lines (7) and (8) involve dense matrices of size $O(N^{0.5}) \times O(N^{0.5})$ where N is the total number of discretization nodes, resulting in $O(N^{1.5})$ overall cost. It turns out that these dense matrices have internal structure that can be exploited to greatly accelerate the matrix algebra. Specifically, the off-diagonal blocks of these matrices are to high precision rank deficient, and the matrices can be represented efficiently using a hierarchical "data-sparse" format known as *Hierarchically Block Separable (HBS)* (and sometimes *Hierarchically Semi-Separable (HSS)* matrices [20, 1]). This section briefly describes the HBS property, for details see [8].

6.1. **Block separable.** Let $\mathbf{H}$ be an $mp \times mp$ matrix that is blocked into $p \times p$ blocks, each of size $m \times m$. We say that $\mathbf{H}$ is "block separable" with "block-rank" k if for $\tau = 1, 2, \ldots, p$, there exist $m \times k$ matrices $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ such that each off-diagonal block $\mathbf{H}_{\sigma,\tau}$ of $\mathbf{H}$ admits the factorization

$$\underset{m\times m}{\mathbf{H}_{\sigma,\tau}} = \underset{m\times k}{\mathbf{U}_\sigma} \underset{k\times k}{\tilde{\mathbf{H}}_{\sigma,\tau}} \underset{k\times m}{\mathbf{V}_\tau^*}, \qquad \sigma,\tau \in \{1, 2, \ldots, p\}, \quad \sigma \neq \tau. \tag{16}$$

Observe that the columns of $\mathbf{U}_\sigma$ must form a basis for the columns of all off-diagonal blocks in row σ, and analogously, the columns of $\mathbf{V}_\tau$ must form a basis for the rows in all the off-diagonal blocks in column τ. When (16) holds, the matrix $\mathbf{H}$ admits a block factorization

$$\underset{mp\times mp}{\mathbf{H}} = \underset{mp\times kp}{\mathbf{U}} \underset{kp\times kp}{\tilde{\mathbf{H}}} \underset{kp\times mp}{\mathbf{V}^*} + \underset{mp\times mp}{\mathbf{D}}, \tag{17}$$

where

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

and

$$\tilde{\mathbf{H}} = \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{H}}_{12} & \tilde{\mathbf{H}}_{13} & \cdots \\ \tilde{\mathbf{H}}_{21} & \mathbf{0} & \tilde{\mathbf{H}}_{23} & \cdots \\ \tilde{\mathbf{H}}_{31} & \tilde{\mathbf{H}}_{32} & \mathbf{0} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}.$$

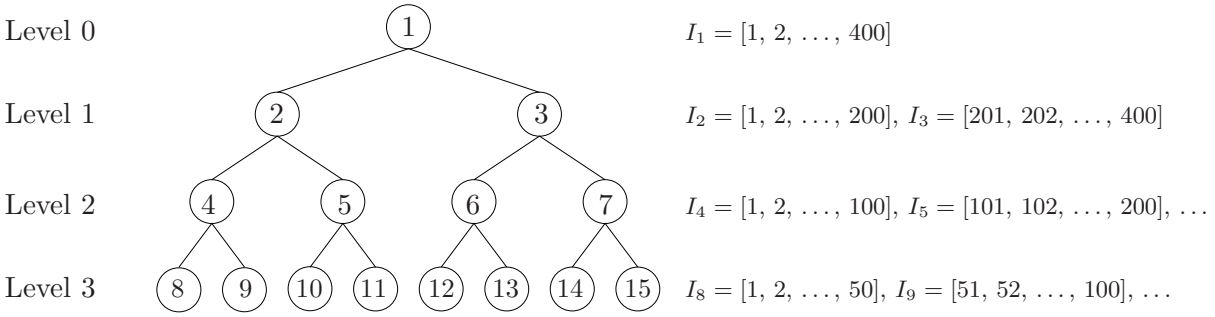

FIGURE 4. Numbering of nodes in a fully populated binary tree with $L = 3$ levels. The root is the original index vector $I = I_1 = [1, 2, \ldots, 400]$.

6.2. **Hierarchically Block-Separable.** Informally speaking, a matrix $\mathbf{H}$ is *Heirarchically Block-Separable* (HBS), if it is amenable to a *telescoping* block factorization. In other words, in addition to the matrix $\mathbf{H}$ being block separable, so is $\tilde{\mathbf{H}}$ 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 $\tilde{\mathbf{H}}$ will be block separable, etc.

In this section, we describe properties and the factored representation of HBS matrices. Details on constructing the factorization are provided in [8].

6.3. **A binary tree structure.** The HBS representation of an $M \times M$ matrix $\mathbf{H}$ is based on a partition of the index vector $I = [1, 2, \ldots, M]$ into a binary tree structure. We let I form the root of the tree, and give it the index 1, $I_1 = I$. We next split the root into two roughly equi-sized vectors I_2 and I_3 so that $I_1 = I_2 \cup I_3$. The full tree is then formed by continuing to subdivide any interval that holds more than some preset fixed number m of indices. We use the integers $\ell = 0, 1, \ldots, L$ to label the different levels, with 0 denoting the coarsest level. A *leaf* is a node corresponding to a vector that never got split. For a non-leaf node τ, its *children* are the two boxes σ_1 and σ_2 such that $I_\tau = I_{\sigma_1} \cup I_{\sigma_2}$, and τ is then the *parent* of σ_1 and σ_2. Two boxes with the same parent are called *siblings*. These definitions are illustrated in Figure 4.

6.4. **Definition of the HBS property.** We now define what it means for an $M \times M$ matrix $\mathbf{H}$ to be *hierarchically block separable* with respect to a given binary tree $\mathcal{T}$ that partitions the index vector $J = [1, 2, \ldots, M]$. For simplicity, we suppose that for every leaf node τ the index vector I_τ holds precisely m points, so that $M = m\,2^L$. Then $\mathbf{H}$ is HBS with block rank k if the following two conditions hold:

(1) Assumption on ranks of off-diagonal blocks at the finest level: For any two distinct leaf nodes τ and τ', define the $m \times m$ matrix

$$\mathbf{H}_{\tau,\tau'} = \mathbf{H}(I_\tau, I_{\tau'}). \tag{18}$$

Then there must exist matrices $\mathbf{U}_\tau$, $\mathbf{V}_{\tau'}$, and $\tilde{\mathbf{H}}_{\tau,\tau'}$ such that

$$\underset{m \times m}{\mathbf{H}_{\tau,\tau'}} = \underset{m \times k}{\mathbf{U}_\tau} \quad \underset{k \times k}{\tilde{\mathbf{H}}_{\tau,\tau'}} \quad \underset{k \times m}{\mathbf{V}^*_{\tau'}}. \tag{19}$$

(2) Assumption on ranks of off-diagonal blocks on level $\ell = L-1, L-2, \ldots, 1$: The rank assumption at level ℓ is defined in terms of the blocks constructed on the next finer level $\ell + 1$: For any distinct nodes τ and τ' on level ℓ with children σ_1, σ_2 and σ'_1, σ'_2, respectively, define

$$\mathbf{H}_{\tau,\tau'} = \begin{bmatrix} \tilde{\mathbf{H}}_{\sigma_1,\sigma'_1} & \tilde{\mathbf{H}}_{\sigma_1,\sigma'_2} \\ \tilde{\mathbf{H}}_{\sigma_2,\sigma'_1} & \tilde{\mathbf{H}}_{\sigma_2,\sigma'_2} \end{bmatrix}. \tag{20}$$

	Name:	Size:	Function:
For each leaf node τ:	$\mathbf{D}_\tau$	$m \times m$	The diagonal block $\mathbf{H}(I_\tau, I_\tau)$.
	$\mathbf{U}_\tau$	$m \times k$	Basis for the columns in the blocks in row τ.
	$\mathbf{V}_\tau$	$m \times k$	Basis for the rows in the blocks in column τ.
For each parent node τ:	$\mathbf{B}_\tau$	$2k \times 2k$	Interactions between the children of τ.
	$\mathbf{U}_\tau$	$2k \times k$	Basis for the columns in the (reduced) blocks in row τ.
	$\mathbf{V}_\tau$	$2k \times k$	Basis for the rows in the (reduced) blocks in column τ.

FIGURE 5. An HBS matrix $\mathbf{H}$ associated with a tree $\mathcal{T}$ is fully specified if the factors listed above are provided.

Then there must exist matrices $\mathbf{U}_\tau$, $\mathbf{V}_{\tau'}$, and $\tilde{\mathbf{H}}_{\tau,\tau'}$ such that

$$\underset{2k \times 2k}{\mathbf{H}_{\tau,\tau'}} = \underset{2k \times k}{\mathbf{U}_\tau} \; \underset{k \times k}{\tilde{\mathbf{H}}_{\tau,\tau'}} \; \underset{k \times 2k}{\mathbf{V}_{\tau'}^*}. \tag{21}$$

An HBS matrix is now fully described if the basis matrices $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ are provided for each node τ, and in addition, we are for each leaf τ given the $m \times m$ matrix

$$\mathbf{D}_\tau = \mathbf{H}(I_\tau, I_\tau), \tag{22}$$

and for each parent node τ with children σ_1 and σ_2 we are given the $2k \times 2k$ matrix

$$\mathbf{B}_\tau = \begin{bmatrix} 0 & \tilde{\mathbf{H}}_{\sigma_1,\sigma_2} \\ \tilde{\mathbf{H}}_{\sigma_2,\sigma_1} & 0 \end{bmatrix}. \tag{23}$$

Observe in particular that the matrices $\tilde{\mathbf{H}}_{\sigma_1,\sigma_2}$ are only required when $\{\sigma_1, \sigma_2\}$ forms a sibling pair. Figure 5 summarizes the required matrices.

6.5. **Telescoping factorization.** Given the matrices defined in the previous section, we define the following block diagonal factors:

$$\underline{\mathbf{D}}^{(\ell)} = \text{diag}(\mathbf{D}_\tau : \tau \text{ is a box on level } \ell), \qquad \ell = 0, 1, \ldots, L, \tag{24}$$

$$\underline{\mathbf{U}}^{(\ell)} = \text{diag}(\mathbf{U}_\tau : \tau \text{ is a box on level } \ell), \qquad \ell = 1, 2, \ldots, L, \tag{25}$$

$$\underline{\mathbf{V}}^{(\ell)} = \text{diag}(\mathbf{V}_\tau : \tau \text{ is a box on level } \ell), \qquad \ell = 1, 2, \ldots, L, \tag{26}$$

$$\underline{\mathbf{B}}^{(\ell)} = \text{diag}(\mathbf{B}_\tau : \tau \text{ is a box on level } \ell), \qquad \ell = 0, 1, \ldots, L-1, . \tag{27}$$

Furthermore, we let $\tilde{\mathbf{H}}^{(\ell)}$ denote the block matrix whose diagonal blocks are zero, and whose off-diagonal blocks are the blocks $\tilde{\mathbf{H}}_{\tau,\tau'}$ for all distinct τ, τ' on level ℓ. With these definitions,

$$\underset{m\,2^L \times n\,2^L}{\mathbf{H}} = \underset{m\,2^L \times k\,2^L}{\underline{\mathbf{U}}^{(L)}} \; \underset{k\,2^L \times k\,2^L}{\tilde{\mathbf{H}}^{(L)}} \; \underset{k\,2^L \times m\,2^L}{(\underline{\mathbf{V}}^{(L)})^*} + \underset{m\,2^L \times m\,2^L}{\underline{\mathbf{D}}^{(L)}}; \tag{28}$$

for $\ell = L-1, L-2, \ldots, 1$ we have

$$\underset{k\,2^{\ell+1} \times k\,2^{\ell+1}}{\tilde{\mathbf{H}}^{(\ell+1)}} = \underset{k\,2^{\ell+1} \times k\,2^{\ell}}{\underline{\mathbf{U}}^{(\ell)}} \; \underset{k\,2^{\ell} \times k\,2^{\ell}}{\tilde{\mathbf{H}}^{(\ell)}} \; \underset{k\,2^{\ell} \times k\,2^{\ell+1}}{(\underline{\mathbf{V}}^{(\ell)})^*} + \underset{k\,2^{\ell+1} \times k\,2^{\ell+1}}{\underline{\mathbf{B}}^{(\ell)}}; \tag{29}$$

and finally

$$\tilde{\mathbf{H}}^{(1)} = \underline{\mathbf{B}}^{(0)}. \tag{30}$$

7. Fast arithmetic operations on HBS matrices

Arithmetic operations involving dense HBS matrices of size $M \times M$ can often be executed in $O(M)$ operations. This fast matrix algebra is vital for achieving linear complexity in our direct solver. This section provides a brief introduction to the HBS matrix algebra. We describe the operations we need (inversion, addition, and low-rank update) in some detail for the single level "block separable" format. The generalization to the multi-level "hierarchically block separable" format is briefly described for the case of matrix inversion. A full description of all algorithms required is given in [7], which is related to the earlier work [2].

Before we start, we recall that a block separable matrix $\mathbf{H}$ consisting of $p \times p$ blocks, each of size $m \times m$, and with "HBS-rank" $k < m$, admits the factorization

$$\underset{mp \times mp}{\mathbf{H}} = \underset{mp \times kp}{\mathbf{U}} \quad \underset{kp \times kp}{\tilde{\mathbf{H}}} \quad \underset{kp \times mp}{\mathbf{V}^*} + \underset{mp \times mp}{\mathbf{D}}. \tag{31}$$

7.1. **Inversion of a block separable matrix.** The decomposition (31) represents $\mathbf{H}$ as a sum of one term $\mathbf{U}\tilde{\mathbf{H}}\mathbf{V}^*$ that is "low rank," and one term $\mathbf{D}$ that is easily invertible (since it is block diagonal). By modifying the classical Woodbury formula for inversion of a matrix perturbed by the addition of a low-rank term, it can be shown that (see Lemma 3.1 of [8])

$$\mathbf{H}^{-1} = \mathbf{E}\,(\tilde{\mathbf{H}} + \hat{\mathbf{D}})^{-1}\,\mathbf{F}^* + \mathbf{G}, \tag{32}$$

where

$$\hat{\mathbf{D}} = \left(\mathbf{V}^*\,\mathbf{D}^{-1}\,\mathbf{U}\right)^{-1}, \tag{33}$$

$$\mathbf{E} = \mathbf{D}^{-1}\,\mathbf{U}\,\hat{\mathbf{D}}, \tag{34}$$

$$\mathbf{F} = (\hat{\mathbf{D}}\,\mathbf{V}^*\,\mathbf{D}^{-1})^*, \tag{35}$$

$$\mathbf{G} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\,\mathbf{U}\,\hat{\mathbf{D}}\,\mathbf{V}^*\,\mathbf{D}^{-1}, \tag{36}$$

assuming the inverses in formulas (32) — (36) all exist. Now observe that the matrices $\hat{\mathbf{D}}$, $\mathbf{E}$, $\mathbf{F}$, and $\mathbf{G}$ can all easily be computed since the formulas defining them involve only block-diagonal matrices. In consequence, (32) reduces the task of inverting the big (size $mp \times mp$) matrix $\mathbf{H}$ to the task of inverting the small (size $kp \times kp$) matrix $\tilde{\mathbf{H}} + \hat{\mathbf{D}}$.

When $\mathbf{H}$ is not only "block separable", but "hierarchically block separable", the process can be repeated recursively by exploiting that $\tilde{\mathbf{H}} + \hat{\mathbf{D}}$ is itself amenable to accelerated inversion, etc. The resulting process is somewhat tricky to analyze, but leads to very clean codes. To illustrate, we include Algorithm 3 which shows the multi-level $O(M)$ inversion algorithm for an HBS matrix $\mathbf{H}$. The algorithm takes as input the factors $\{\mathbf{U}_\tau, \mathbf{V}_\tau, \mathbf{D}_\tau, \mathbf{B}_\tau\}_\tau$ representing $\mathbf{H}$ (cf. Figure 5), and outputs an analogous set of factors $\{\mathbf{E}_\tau, \mathbf{F}_\tau, \mathbf{G}_\tau\}_\tau$ representing $\mathbf{H}^{-1}$. With these factors, the matrix-vector multiplication $\mathbf{y} = \mathbf{H}^{-1}\mathbf{x}$ can be executed via the procedure described in Algorithm 4.

7.2. **Addition of two block separable matrices.** Let $\mathbf{H}^A$ and $\mathbf{H}^B$ be block separable matrices with factorizations

$$\mathbf{H}^A = \mathbf{U}^A\tilde{\mathbf{H}}^A\mathbf{V}^{A*} + \mathbf{D}^A, \qquad \text{and} \qquad \mathbf{H}^B = \mathbf{U}^B\tilde{\mathbf{H}}^B\mathbf{V}^{B*} + \mathbf{D}^B.$$

Then $\mathbf{H} = \mathbf{H}^A + \mathbf{H}^B$ can be written in block separable form via

$$\mathbf{H} = \mathbf{H}^A + \mathbf{H}^B = \left[\mathbf{U}^A\,\mathbf{U}^B\right] \begin{bmatrix} \tilde{\mathbf{H}}^A & 0 \\ 0 & \tilde{\mathbf{H}}^B \end{bmatrix} \left[\mathbf{V}^A\,\mathbf{V}^B\right]^* + \left(\mathbf{D}^A + \mathbf{D}^B\right). \tag{37}$$

To restore (37) to block separable form, permute the rows and columns of $\left[\mathbf{U}^A\,\mathbf{U}^B\right]$ and $\left[\mathbf{V}^A\,\mathbf{V}^B\right]$ to attain block diagonal form, then re-orthogonalize the diagonal blocks. This process in principle results in a matrix $\mathbf{H}$ whose HBS-rank is the sum of the HBS-ranks of $\mathbf{H}^A$ and $\mathbf{H}^B$. In practice, this

ALGORITHM 3 (inversion of an HBS matrix)

Given factors $\{\mathbf{U}_\tau, \mathbf{V}_\tau, \mathbf{D}_\tau, \mathbf{B}_\tau\}_\tau$ representing an HBS matrix $\mathbf{H}$, this algorithm constructs factors $\{\mathbf{E}_\tau, \mathbf{F}_\tau, \mathbf{G}_\tau\}_\tau$ representing $\mathbf{H}^{-1}$.

loop over all levels, finer to coarser, $\ell = L,\ L-1,\ \ldots,\ 1$
  loop over all boxes τ on level ℓ,
    if τ is a leaf node
      $\tilde{\mathbf{D}}_\tau = \mathbf{D}_\tau$
    else
      Let σ_1 and σ_2 denote the children of τ.
      $\tilde{\mathbf{D}}_\tau = \begin{bmatrix} \hat{\mathbf{D}}_{\sigma_1} & \mathbf{B}_{\sigma_1,\sigma_2} \\ \mathbf{B}_{\sigma_2,\sigma_1} & \hat{\mathbf{D}}_{\sigma_2} \end{bmatrix}$
    end if
    $\hat{\mathbf{D}}_\tau = \left(\mathbf{V}_\tau^* \tilde{\mathbf{D}}_\tau^{-1} \mathbf{U}_\tau\right)^{-1}$.
    $\mathbf{E}_\tau = \tilde{\mathbf{D}}_\tau^{-1} \mathbf{U}_\tau \hat{\mathbf{D}}_\tau$.
    $\mathbf{F}_\tau^* = \hat{\mathbf{D}}_\tau \mathbf{V}_\tau^* \tilde{\mathbf{D}}_\tau^{-1}$.
    $\mathbf{G}_\tau = \tilde{\mathbf{D}}_\tau^{-1} - \tilde{\mathbf{D}}_\tau^{-1} \mathbf{U}_\tau \hat{\mathbf{D}}_\tau \mathbf{V}_\tau^* \tilde{\mathbf{D}}_\tau^{-1}$.
  end loop
end loop
$\mathbf{G}_1 = \begin{bmatrix} \hat{\mathbf{D}}_2 & \mathbf{B}_{2,3} \\ \mathbf{B}_{3,2} & \hat{\mathbf{D}}_3 \end{bmatrix}^{-1}$.

ALGORITHM 4 (application of the inverse of an HBS matrix)

Given $\mathbf{x}$*, compute* $\mathbf{y} = \mathbf{H}^{-1}\mathbf{x}$ *using the factors* $\{\mathbf{E}_\tau, \mathbf{F}_\tau, \mathbf{G}_\tau\}_\tau$ *resulting from Algorithm 3.*

loop over all leaf boxes τ
  $\hat{\mathbf{x}}_\tau = \mathbf{F}_\tau^* \mathbf{x}(I_\tau)$.
end loop

loop over all levels, finer to coarser, $\ell = L,\ L-1,\ \ldots,\ 1$
  loop over all parent boxes τ on level ℓ,
    Let σ_1 and σ_2 denote the children of τ.
    $\hat{\mathbf{x}}_\tau = \mathbf{F}_\tau^* \begin{bmatrix} \hat{\mathbf{x}}_{\sigma_1} \\ \hat{\mathbf{x}}_{\sigma_2} \end{bmatrix}$.
  end loop
end loop

$\begin{bmatrix} \hat{\mathbf{y}}_2 \\ \hat{\mathbf{y}}_3 \end{bmatrix} = \mathbf{G}_1 \begin{bmatrix} \hat{\mathbf{x}}_2 \\ \hat{\mathbf{x}}_3 \end{bmatrix}$.

loop over all levels, coarser to finer, $\ell = 1,\ 2,\ \ldots,\ L-1$
  loop over all parent boxes τ on level ℓ
    Let σ_1 and σ_2 denote the children of τ.
    $\begin{bmatrix} \hat{\mathbf{y}}_{\sigma_1} \\ \hat{\mathbf{y}}_{\sigma_2} \end{bmatrix} = \mathbf{E}_\tau \hat{\mathbf{x}}_\tau + \mathbf{G}_\tau \begin{bmatrix} \hat{\mathbf{x}}_{\sigma_1} \\ \hat{\mathbf{x}}_{\sigma_2} \end{bmatrix}$.
  end loop
end loop

loop over all leaf boxes τ
  $\mathbf{y}(I_\tau) = \mathbf{E}_\tau \hat{\mathbf{q}}_\tau + \mathbf{G}_\tau \mathbf{x}(I_\tau)$.
end loop

rank increase can be combated by numerically recompressing the basis matrices, and updating the middle factor as needed. For details, as well as the extension to a multi-level scheme, see [2, 7].

7.3. **Addition of a block separable matrix with a low rank matrix.** Let $\mathbf{H}^B = \mathbf{QR}$ be a k-rank matrix where $\mathbf{Q}$ and $\mathbf{R}^*$ are of size $mp \times k$. We would like to add $\mathbf{H}^B$ to the block separable matrix $\mathbf{H}^A$. Since we already know how to add two block separable matrices, we choose to rewrite $\mathbf{H}^B$ in block separable form. Without loss of generality, assume $\mathbf{Q}$ is orthogonal. Partition $\mathbf{Q}$ into p blocks of size $m \times k$. The blocks make up the matrix $\mathbf{U}^B$. Likewise partition $\mathbf{R}$ into p blocks of size $k \times m$. The block matrix $\mathbf{D}^B$ has entries $\mathbf{D}_\tau = \mathbf{Q}_\tau \mathbf{R}_\tau$ for $\tau = 1, \ldots, p$. To construct the matrices $\mathbf{V}^B$, for each $\tau = 1, \ldots, p$, the matrix $\mathbf{R}_\tau$ is factorized into $\tilde{\mathbf{R}}_\tau \mathbf{V}_\tau *$ where the matrix $\mathbf{V}_\tau$ is orthogonal. The matrices $\tilde{\mathbf{R}}_\tau$ make up the entries of $\tilde{\mathbf{H}}^B$.

8. Accelerating the direct solver

This section describes how the fast matrix algebra described in Sections 6 and 7 can be used to accelerate the direct solver of Section 5 to attain $O(N)$ complexity. We recall that the $O(N^{1.5})$ cost of Algorithm 1 relates to the execution of lines (7) and (8) at the top levels, since these involve dense matrix algebra of matrices of size $O(N^{0.5}) \times O(N^{0.5})$. The principal claims of this section are:

- The matrices $\mathbf{T}^{\sigma_1}_{1,3}$, $\mathbf{T}^{\sigma_1}_{3,1}$, $\mathbf{T}^{\sigma_2}_{2,3}$, $\mathbf{T}^{\sigma_2}_{3,2}$ have low numerical rank.
- The matrices $\mathbf{T}^{\sigma_1}_{1,1}$, $\mathbf{T}^{\sigma_2}_{2,2}$, $\mathbf{T}^{\sigma_1}_{3,3}$, $\mathbf{T}^{\sigma_2}_{3,3}$ are HBS matrices of low HBS rank.

To be precise, the ranks that we claim are "low" scale as $\log(1/\nu) \times \log(m)$ where m is the number of points along the boundary of Ω_τ, and ν is the computational tolerance. In practice, we found that for problems with non-oscillatory solutions, the ranks are extremely modest: when $\nu = 10^{-10}$, the ranks range between 10 and 80, even for very large problems.

The cause of the rank deficiencies is that the matrix $\mathbf{T}^\tau$ is a highly accurate approximation to the Dirichlet-to-Neumann operator on Ω_τ. This operator is known to have a smooth kernel that is non-oscillatory whenever the underlying PDE has non-oscillatory solutions. Since the domain boundary $\partial\Omega_\tau$ is one-dimensional, this makes the expectation that the off-diagonal blocks have low rank very natural, see [8]. It is backed up by extensive numerical experiments (see Section 9), but we do not at this point have rigorous proofs to support the claim.

Once it is observed that all matrices in lines (7) and (8) of Algorithm 1 are structured, it becomes obvious how to accelerate the algorithm. For instance, line (7) is executed in three steps: (i) Add the HBS matrices $\mathbf{T}^{\sigma_1}_{3,3}$ and $-\mathbf{T}^{\sigma_2}_{3,3}$. (ii) Invert the sum of the HBS matrices. (iii) Apply the inverse (in HBS form) to one of the low rank factors of $\left[-\mathbf{T}^{\alpha}_{3,1} \;\middle|\; \mathbf{T}^{\beta}_{3,2}\right]$. The result is an approximation to $\mathbf{S}^\tau$, represented as a product of two thin matrices. Executing line (8) is analogous: First form the matrix products $\mathbf{T}^{\sigma_1}_{1,3}\mathbf{S}^\tau$ and $\mathbf{T}^{\sigma_2}_{2,3}\mathbf{S}^\tau$, exploiting that all factors are of low rank. Then perform a low-rank update to a block-diagonal matrix whose blocks are provided in HBS-form to construct the new HBS matrix $\mathbf{T}^\tau$.

Accelerating the solve stage in Algorithm 2 is trivial, simply exploit that the matrix $\mathbf{S}^\tau$ on line (3) has low numerical rank.

Remark 8.1. Some of the structured matrix operators (e.g. adding two HBS matrices, or the low-rank update) can algebraically lead to a large increase in the HBS ranks. We know for physical reasons that the output should have rank-structure very similar to the input, however, and we combat the rank-increase by frequently recompressing the output of each operation.

Remark 8.2. In practice, we implement the algorithm to use dense matrix algebra at the finer levels, where all the DtN matrices $\mathbf{T}^\tau$ are small. Once they get large enough that HBS algebra outperforms dense operations, we compress the dense matrices by brute force, and rely on HBS algebra in the remainder of the upwards sweep.

9. Numerical Examples

In this section, we illustrate the capabilities of the proposed method for the boundary value problem

$$\begin{cases} -\Delta u(\boldsymbol{x}) - c_1(\boldsymbol{x})\,\partial_1 u(\boldsymbol{x}) - c_2(\boldsymbol{x})\,\partial_2 u(\boldsymbol{x}) - c(\boldsymbol{x})\,u(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = f(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma, \end{cases} \tag{38}$$

where $\Omega = [0,1]^2$, $\Gamma = \partial\Omega$, and $c_1(\boldsymbol{x})$, $c_2(\boldsymbol{x})$, and $c(\boldsymbol{x})$ are smooth, cf. (2). The choice of the functions $c_1(\boldsymbol{x})$, $c_2(\boldsymbol{x})$, and $c(\boldsymbol{x})$ will vary for each example.

All experiments reported in this section were executed on a machine with two quad-core Intel Xeon E5-2643 processors and 128GB of RAM. The direct solver was implemented in Matlab, which means that the speeds reported can very likely be improved, although the asymptotic complexity should be unaffected.

In Section 9.1 we apply the direct solver to four problems with known analytic solutions. This allows us to very accurately investigate the errors incurred, but is arguably a particularly favorable environment. Section 9.2 presents results from more general situations where the exact solution is not known, and errors have to be estimated.

In all experiments, the number of Gaussian points per leaf edge q is fixed at 21, and 21×21 tensor product grids of Chebyshev points are used in the leaf computations. Per Remark 8.2, we switch from dense computations to HBS when a box has more than 2000 points on the boundary.

9.1. **Performance for problems with known solutions.** To illustrate the scaling and accuracy of the discretization technique, we apply the numerical method to four problems with known solutions. The problems are:

- ***Laplace*:** Let $c_1(\boldsymbol{x}) = c_2(\boldsymbol{x}) = c(\boldsymbol{x}) = 0$ in (38).
- ***Helmholtz I*:** Let $c_1(\boldsymbol{x}) = c_2(\boldsymbol{x}) = 0$, and $c(\boldsymbol{x}) = \kappa^2$ where $\kappa = 80$ in (38). This represents a vibration problem on a domain Ω of size roughly 12×12 wave-lengths. (Recall that the wave-length is given by $\lambda = \frac{2\pi}{\kappa}$.)
- ***Helmholtz II*:** Let $c_1(\boldsymbol{x}) = c_2(\boldsymbol{x}) = 0$, and $c(\boldsymbol{x}) = \kappa^2$ where $\kappa = 640$ in (38). This corresponds to a domain of size roughly 102×102 wave-lengths.
- ***Helmholtz III*:** We again set $c_1(\boldsymbol{x}) = c_2(\boldsymbol{x}) = 0$, and $c(\boldsymbol{x}) = \kappa^2$ in (38), but now we let κ grow as the number of discretization points grows to maintain a constant 12 points per wavelength.

The boundary data in (38) is chosen to coincide with the known solutions $u_{\text{exact}}(\boldsymbol{x}) = \log|\hat{\boldsymbol{x}} - \boldsymbol{x}|$ for the Laplace problem and with $u_{\text{exact}}(\boldsymbol{x}) = \mathrm{Y}_0(\kappa|\hat{\boldsymbol{x}} - \boldsymbol{x}|)$ for the three Helmholtz problems, where $\hat{\boldsymbol{x}} = (-2, 0)$, and where Y_0 denotes the 0'th Bessel function of the second kind.

In a first experiment, we prescribed the tolerance in the "fast" matrix algebra to be $\epsilon = 10^{-7}$. Table 1 reports the following quantities:

N	Number of Gaussian discretization points.
N_{tot}	Total number of discretization points. (N plus the number of Chebyshev points)
T_{build}	Time for building the solution operator.
T_{solve}	Time to solve for interior nodes.
T_{apply}	Time to apply the approximate Dirichlet-to-Neumann operator $\mathbf{T}^1$.
R	Amount of memory required to store the solution operator.
$E_{\text{pot}} =$	$\max_{k:\,\boldsymbol{x}_k \in \Omega} \left\{ \left| u_{\text{app}}(\boldsymbol{x}_k) - u_{\text{exact}}(\boldsymbol{x}_k) \right| \right\}$,

where u_{app} denotes the approximate solution constructed by the direct solver.

Our expectation is for all problems except *Helmholtz III* to exhibit optimal linear scaling for both the build and the solve stages. Additionally, we expect the cost of applying the Dirichlet-to-Neumann operator $\mathbf{T}^1$ to scale as $N^{0.5}$ for all problems except *Helmholtz III*. The numerical results clearly bear this out for *Laplace* and *Helmholtz I*. For *Helmholtz II*, it appears that linear scaling has not quite taken hold for the range of problem sizes our hardware could manage. The *Helmholtz III* problem clearly does not exhibit linear scaling, but has not yet reached its $O(N^{1.5})$ asymptotic at the largest problem considered, which was of size roughly 426×426 wave-lengths. We remark that the cost

	N_{tot}	N	T_{build} (seconds)	T_{solve} (seconds)	T_{apply} (seconds)	R (MB)	E_{pot}
Laplace	1815681	174720	91.68	0.34	0.035	1611.19	2.57e-05
	7252225	693504	371.15	1.803	0.104	6557.27	6.55e-05
	28987905	2763264	1661.97	6.97	0.168	26503.29	2.25e-04
	115909633	11031552	6894.31	30.67	0.367	106731.61	8.62e-04
Helmholtz I	1815681	174720	62.07	0.202	0.027	1611.41	6.86e-06
	7252225	693504	363.19	1.755	0.084	6557.12	7.47e-06
	28987905	2763264	1677.92	6.92	0.186	26503.41	1.55e-05
	115909633	11031552	7584.65	31.85	0.435	106738.85	1.45e-04
Helmholtz II	1815681	174720	93.96	0.29	0.039	1827.72	5.76e-07
	7252225	693504	525.92	2.13	0.074	7151.60	7.06e-07
	28987905	2763264	2033.91	8.59	0.175	27985.41	4.04e-06
Helmholtz III	1815681	174720	93.68	0.29	0.038	1839.71	1.29e-06
	7252225	693504	624.24	1.67	0.086	7865.13	1.21e-06
	28987905	2763264	4174.91	10.28	0.206	33366.45	1.76e-06

TABLE 1. Timing results in seconds for the PDEs considered in Section 9.1. For these experiments, $\epsilon = 10^{-7}$.

of the solve stage is tiny. For example, a problem involving 11 million unknowns (corresponding to approximately 100 million discretization points) takes 115 minutes for the build stage and then only 30 seconds for each additional solve. The cost for applying the Dirichlet-to-Neumann operator is even less at 0.36 seconds per boundary condition. Figure 6 illustrates the scaling via log-log plots.

In a second set of experiments, we investigated the accuracy of the computed solutions, and in particular how the accuracy depends on the tolerance ϵ in the fast matrix algebra. In addition to reporting E_{pot}, Table 2 reports

$$E_{\text{grad}} = \max_{k:\, \boldsymbol{x}_k \in \Gamma} \big\{ \big| u_{n,\text{app}}(\boldsymbol{x}_k) - u_{n,\text{exact}}(\boldsymbol{x}_k) \big| \big\},$$

where u_{app} denotes the approximate solution constructed by the direct solver for tolerances $\varepsilon = 10^{-7}$, 10^{-10}, and 10^{-12}. The number of discretization points was fixed problem to be $N = 693504$ ($N_{\text{tot}} = 7252225$).

	$\epsilon = 10^{-7}$		$\epsilon = 10^{-10}$		$\epsilon = 10^{-12}$	
	E_{pot}	E_{grad}	E_{pot}	E_{grad}	E_{pot}	E_{grad}
Laplace	6.55e-05	1.07e-03	2.91e-08	5.52e-07	1.36e-10	8.07e-09
Helmholtz I	7.47e-06	6.56e-04	5.06e-09	4.89e-07	1.38e-10	8.21e-09
Helmholtz II	7.06e-07	3.27e-04	1.42e-09	8.01e-07	8.59e-11	4.12e-08
Helmholtz III	1.21e-06	1.28e-03	1.85e-09	2.69e-06	1.63e-09	2.25e-06

TABLE 2. Errors for solving the PDEs in Section 9.1 with different user prescribed tolerances when the number of discretization points is fixed at $N = 693504$ ($N_{\text{tot}} = 7252225$).

The solution obtains (or nearly obtains) the prescribed tolerance while the normal derivative suffers from a three digit loss in accuracy. This loss is likely attributable to the unboundedness of the Dirichlet-to-Neumann map. The compressed representation captures the high end of the spectrum to the desired accuracy while the low end of the spectrum is captured to three digits less than the desired accuracy.

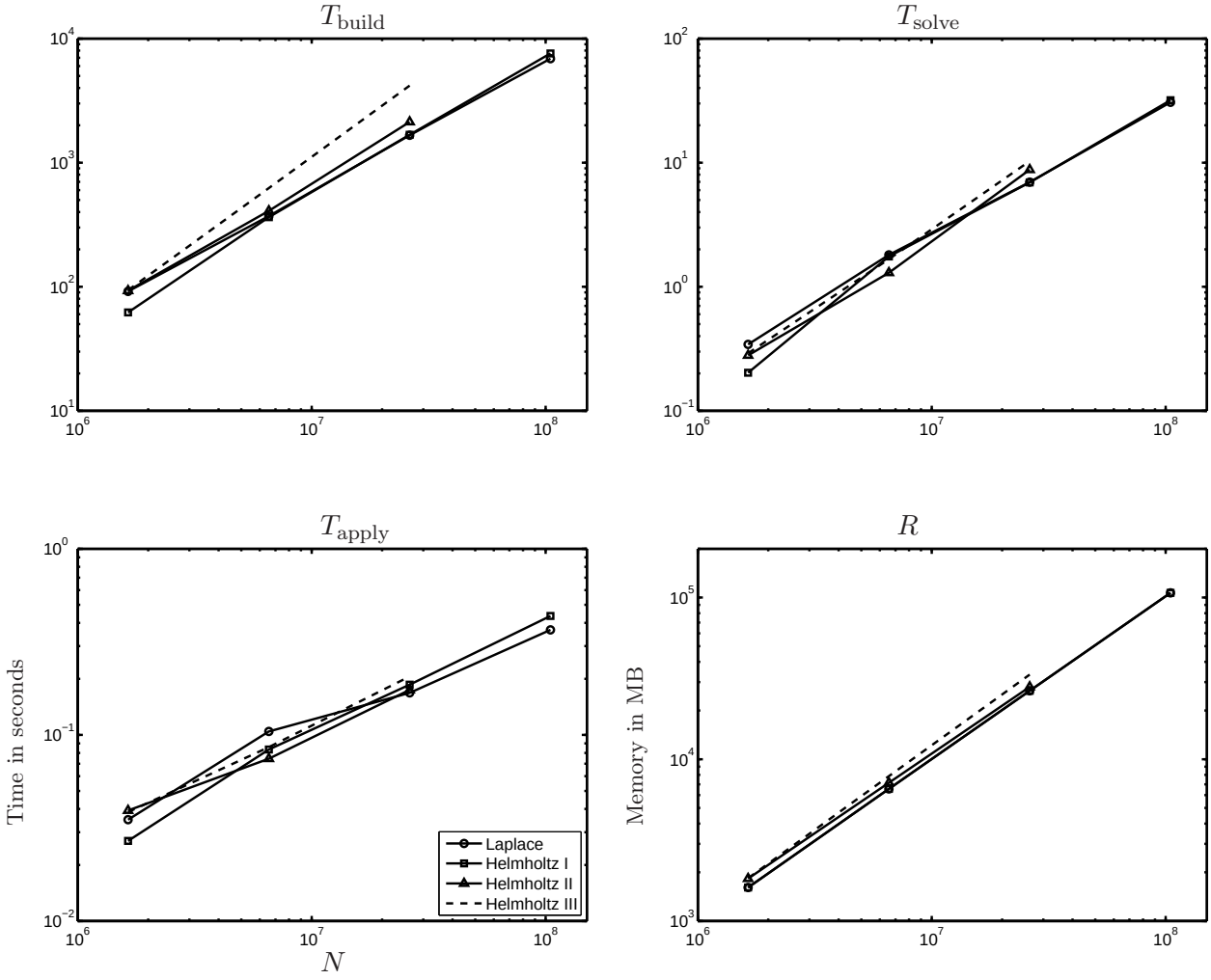

FIGURE 6. The first three graphs give the times required for building the direct solver (T_{build}), solving a problem (T_{solve}) and applying the approximate Dirichlet-to-Neumann operator on $\partial\Omega$ (T_{apply}). The fourth graph gives the memory R in MB required to store the solution operator.

9.2. **Convergence for unknown solutions.** In this section, we apply the direct solver to three problems for which we do not know an exact solution:

***Constant convection*:** Let the convection in the x_2 direction be constant by setting $c_1(\boldsymbol{x}) = 0$, $c_2(\boldsymbol{x}) = -100,000$ and set $c(\boldsymbol{x}) = 0$.

***Diffusion-Convection*:** Introduce a divergence free convection by setting $c_1(\boldsymbol{x}) = -10,000\cos(4\pi x_2)$, $c_2(\boldsymbol{x}) = -10,000\cos(4\pi x_1)$, and $c(\boldsymbol{x}) = 0$.

***Variable Helmholtz*:** Consider the variable coefficient Helmholtz problem where $c_1(\boldsymbol{x}) = 0$, $c_2(\boldsymbol{x}) = 0$, $c(\boldsymbol{x}) = \kappa^2(1 - (\sin(4\pi x_1)\sin(4\pi x_2))^2)$ and $\kappa = 640$.

For the three experiments, the boundary data is given by $f(\boldsymbol{x}) = \cos(2x_1)(1 - 2x_2)$.

	N_{tot}	N	$u^N(\hat{\boldsymbol{x}})$	E^N_{int}	$u^N_n(\hat{\boldsymbol{x}})$	E^N_{bnd}
Constant Convection	455233	44352	-0.474057246846306	0.477	-192794.835134257	824.14
	1815681	174720	-0.951426960146812	8.28e-03	-191970.688228300	1.47
	7252225	693504	-0.959709514830931	6.75e-10	-191972.166912008	0.365
	28987905	2763264	-0.959709515505929		-191972.532606428	
Variable Helmholtz	114465	11424	2.50679456864385	6.10e-02	-2779.09822864819	3188
	455233	44352	2.56780367343056	4.63e-07	409.387483435691	2.59e-02
	1815681	174720	2.56734097240752	1.77e-09	409.413356177218	3.30e-07
	7252225	693504	2.56734097418159		409.413355846946	
Diffusion-Convection	455233	44352	0.0822281612709325	5.04e-5	-35.1309711271060	2.23e-3
	1815681	174720	0.0822785917678385	2.67e-8	-35.1332056731696	7.57e-6
	7252225	693504	0.0822786184699243	5.41e-12	-35.1332132455725	2.11e-09
	28987905	2763264	0.0822786184753420		-35.1332132476795	

TABLE 3. Convergence results for solving the PDE's in Section 9.2 with a user subscribed tolerance of $\epsilon = 10^{-12}$.

	N_{tot}	N	T_{build} (seconds)	T_{solve} (seconds)	R (MB)
Constant Convection	455233	44352	21.04	0.85	683.25
	1815681	174720	176.09	3.47	2997.80
	7252225	693504	980.93	13.76	8460.94
	28987905	2763264	5227.52	77.03	48576.75
Variable Helmholtz	114465	11424	4.61	0.19	167.68
	455233	44352	42.72	1.110	774.34
	1815681	174720	450.68	4.54	3678.31
	7252225	693504	3116.57	17.64	15658.07
Diffusion-Convection	455233	44352	28.31	0.795	446.21
	1815681	174720	131.23	3.20	2050.20
	7252225	693504	906.11	17.12	8460.94
	28987905	2763264	4524.99	66.99	47711.17

TABLE 4. Times in seconds for solving the PDE's in Section 9.2 with a user subscribed tolerance of $\epsilon = 10^{-12}$.

To check for convergence, we post-process the solution as described in Section 5.3 to get the solution on the Chebyshev grid. Let u^N denote the solution on the Chebyshev grid. Likewise,let u^N_n denote the normal derivative on the boundary at the Chebyshev boundary points. We compare the solution and the normal derivative on the boundary pointwise at the locations

$$\hat{\boldsymbol{x}} = (0.75, 0.25) \quad \text{and} \quad \hat{\boldsymbol{y}} = (0.75, 0)$$

respectively, via

$$E^N_{\text{int}} = |u^N(\hat{\boldsymbol{x}}) - u^{4N}(\hat{\boldsymbol{x}})| \quad \text{and} \quad E^N_{\text{bnd}} = |u^N_n(\hat{\boldsymbol{y}}) - u^{4N}_n(\hat{\boldsymbol{y}})|.$$

The tolerance for the compressed representations is set to $\epsilon = 10^{-12}$. Table 3 reports the pointwise errors. We see that high accuracy is obtained in all cases, with solutions that have ten correct digits for the potential and about seven correct digits for the boundary flux.

The computational costs of the computations are reported in Table 4. The memory R reported now includes the memory required to store all local solution operators described in Section 5.3.

10. Conclusions

We have described a direct solver for variable coefficient elliptic PDEs in the plane, under the assumption that the solution and all coefficient functions are smooth. For problems with non-oscillatory solutions such as the Laplace and Stokes equations, the asymptotic complexity of the solver is $O(N)$, with a small constant of proportionality. For problems with oscillatory solutions, high practical efficiency is retained for problems of size up to several hundred wave-lengths.

Our method is based on a composite spectral discretization. We use high order local meshes (typically of size 21×21) capable of solving even very large scale problems to ten correct digits or more. The direct solver is conceptually similar to the classical nested dissection method [6]. To improve the asymptotic complexity from the classical $O(N^{1.5})$ to $O(N)$, we exploit internal structure in the dense "frontal matrices" at the top levels in a manner similar to recent work such as, e.g., [2, 7, 13, 14, 19]. Compared to these techniques, our method has an advantage in that high order discretizations can be incorporated without compromising the speed of the linear algebra. The reason is that we use a formulation based on Dirichlet-to-Neumann operators. As a result, we need high order convergence *only in the tangential direction* on patch boundaries.

The direct solver presented requires more storage than classical iterative methods, but this is partially off-set by the use of high-order discretizations. More importantly, the solver is characterized by very low data-movement. This makes the method particularly well suited for implementation on parallel machines with distributed memory.

Acknowledgments: The work was supported by NSF grants DMS-0748488 and CDI-0941476.

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, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl. **17** (2010), 953–976.

[3] Y. Chen, *Total wave based fast direct solver for volume scattering problems*, arXiv.org **1302.2101** (2013).

[4] Yu Chen, *A fast, direct algorithm for the Lippmann-Schwinger integral equation in two dimensions*, Adv. Comput. Math. **16** (2002), no. 2-3, 175–190, Modeling and computation in optics and electromagnetics.

[5] I.S. Duff, A.M. Erisman, and J.K. Reid, *Direct methods for sparse matrices*, Oxford, 1989.

[6] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. on Numerical Analysis **10** (1973), 345–363.

[7] A. Gillman, *Fast direct solvers for elliptic partial differential equations*, Ph.D. thesis, University of Colorado at Boulder, Applied Mathematics, 2011.

[8] A. Gillman, P. Young, and P.G. Martinsson, *A direct solver with $o(n)$ complexity for integral equations on one-dimensional domains*, Frontiers of Mathematics in China **7** (2012), no. 2, 217–247.

[9] Leslie Greengard, Denis Gueyffier, Per-Gunnar Martinsson, and Vladimir Rokhlin, *Fast direct solvers for integral equations in complex three-dimensional domains*, Acta Numer. **18** (2009), 243–275.

[10] K.L. Ho and L. Greengard, *A fast direct solver for structured linear systems by recursive skeletonization*, SIAM J Sci Comput **to appear** (2012).

[11] A. J. Hoffman, M. S. Martin, and D. J. Rose, *Complexity bounds for regular finite difference and finite element grids*, SIAM J. Numer. Anal. **10** (1973), 364–369.

[12] D.A. Kopriva, *A staggered-grid multidomain spectral method for the compressible navierstokes equations*, Journal of Computational Physics **143** (1998), no. 1, 125 – 158.

[13] Sabine Le Borne, Lars Grasedyck, and Ronald Kriemann, *Domain-decomposition based $\mathcal{H}$-LU preconditioners*, Domain decomposition methods in science and engineering XVI, Lect. Notes Comput. Sci. Eng., vol. 55, Springer, Berlin, 2007, pp. 667–674. MR 2334161

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

[15] ———, *A composite spectral scheme for variable coefficient helmholtz problems*, arXiv.org **1206.4136** (2012).

[16] ———, *A direct solver for variable coefficient elliptic pdes discretized via a composite spectral collocation method*, Journal of Computational Physics **242** (2013), no. 1, 460 – 479.

[17] P.G. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comp. Phys. **205** (2005), no. 1, 1–23.

[18] H.P. Pfeiffer, L.E. Kidder, M.A. Scheel, and S.A. Teukolsky, *A multidomain spectral method for solving elliptic equations*, Computer physics communications **152** (2003), no. 3, 253–273.

[19] P.G. Schmitz and L. Ying, *A fast direct solver for elliptic problems on general meshes in 2d*, Journal of Computational Physics **231** (2012), no. 4, 1314 – 1338.

[20] Zhifeng Sheng, Patrick Dewilde, and Shivkumar 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. MR MR2342902

[21] L.N. Trefethen, *Spectral methods in matlab*, SIAM, Philadelphia, 2000.

[22] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl. **31** (2009), no. 3, 1382–1411. MR 2587783 (2011c:65072)

[23] B. Yang and J.S. Hesthaven, *Multidomain pseudospectral computation of maxwell's equations in 3-d general curvilinear coordinates*, Applied Numerical Mathematics **33** (2000), no. 1 – 4, 281 – 289.