A direct solver for elliptic PDEs in three dimensions based on hierarchical merging of Poincaré-Steklov operators

S. Hao[1], *P.G. Martinsson*[2]

Abstract: A numerical method for variable coefficient elliptic PDEs on three dimensional domains is described. The method is designed for problems with smooth solutions, and is based on a multidomain spectral collocation discretization scheme. The resulting system of linear equations can very efficiently be solved using a nested dissection style direct (as opposed to iterative) solver. This makes the scheme particularly well suited to solving problems for which iterative solvers struggle; in particular for problems with oscillatory solutions. A principal feature of the scheme is that once the solution operator has been constructed, the actual solve is extremely fast. An upper bound on the asymptotic cost of the build stage of $O(N^{4/3})$ is proved (for the case where the PDE is held fixed as N increases). The solve stage has close to linear complexity. The scheme requires a relatively large amount of storage per degree of freedom, but since it is a high order scheme, a small number of degrees of freedom is sufficient to achieve high accuracy. The method is presented for the case where there is no body load present, but it can with little difficulty be generalized to the non-homogeneous case. Numerical experiments demonstrate that the scheme is capable of solving Helmholtz-type equations on a domain of size $20\times20\times20$ wavelengths to three correct digits on a modest personal workstation, with $N \approx 2 \cdot 10^6$.

1. Introduction

The manuscript describes an algorithm for solving the boundary value problem

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

where $\Omega = [0,\,1]^3$ is the unit cube, and where A is an elliptic partial differential operator

$$\begin{aligned}[Au](\boldsymbol{x}) = &-c_{11}(\boldsymbol{x})[\partial_1^2\, u](\boldsymbol{x}) - c_{22}(\boldsymbol{x})[\partial_2^2\, u](\boldsymbol{x}) - c_{33}(\boldsymbol{x})[\partial_3^2\, u](\boldsymbol{x}) \\ &- 2c_{12}(\boldsymbol{x})[\partial_1\partial_2\, u](\boldsymbol{x}) - 2c_{13}(\boldsymbol{x})[\partial_1\partial_3\, u](\boldsymbol{x}) - 2c_{23}(\boldsymbol{x})[\partial_2\partial_3\, u](\boldsymbol{x}) \\ &+ c_1(\boldsymbol{x})[\partial_1\, u](\boldsymbol{x}) + c_2(\boldsymbol{x})[\partial_2\, u](\boldsymbol{x}) + c_3(\boldsymbol{x})[\partial_3\, u](\boldsymbol{x}) + c(\boldsymbol{x})u(\boldsymbol{x}).\end{aligned} \tag{2}$$

We assume that all coefficients in (2) are smooth functions, and that the matrix

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{12} & c_{22} & c_{23} \\ c_{13} & c_{23} & c_{33} \end{bmatrix}$$

is positive definite throughout the domain to ensure the ellipticity of the equation. The generalizations to problems with a body load, to different boundary conditions, and to general domains are straight-forward, as was demonstrated for problems in two dimensions in [24, 23, 18].

The method proposed is a generalization to three dimensions of a technique that was described for problems in two dimensions in [23, 11, 12, 24]. The idea is to explicitly build an approximation to the solution operator of (1) via a a hierarchical divide-and-conquer approach. This makes the scheme a *direct* solver, as opposed to more commonly used *iterative* solvers. The domain Ω is recursively split in halves to create a tree of boxes, cf. Figure 2. The splitting continues until each box is small enough that the solution, and its first and second derivatives, can accurately be resolved on a local tensor product grid of $p \times p \times p$ Chebyshev nodes (where, say, $p = 10$

[1]Dept. of Applied Mathematics, 526 UCB, University of Colorado at Boulder, Boulder, CO 80309-0526, USA.
[2]Corresponding author. Dept. of Applied Mathematics, 526 UCB, University of Colorado at Boulder, Boulder, CO 80309-0526, USA.

or $p = 15$). The PDE (1) is enforced via collocation to locally compute a solution operator for each leaf. Then in a single pass up through the tree of boxes, solution operators are constructed hierarchically by "gluing together" the solution operators of the two children of each box.

The solver we propose has three principal advantages over standard solvers for (1): (i) Since the solver is *direct*, the method is particularly well suited for problems for which efficient preconditioners are difficult to find, such as, e.g., problems with oscillatory solutions. (ii) In situations where a sequence of equations involving the same operator but different boundary data needs to be solved, the method is *very* fast. Once the solution operator has been built, we demonstrate that problems with 10^6 degrees of freedom can be solved in about 1 second on a standard workstation. (iii) Even though the solver is direct, its asymptotic complexity is at worst $O(N^{4/3})$ (as opposed to $O(N^2)$ complexity for classical nested dissection).

The idea of building a solution operator via a hierarchical process on a binary tree of boxes is directly analogous to classical "nested dissection" and "multifrontal" solvers [9, 10, 8]. The asymptotic complexity for these classical methods is $O(N^2)$ for the first solve, and then, once the solution operator has been built, $O(N^{4/3})$ for subsequent solves. The principal reason for the unfavorable scaling is the need to perform operations such as matrix inversion and matrix-matrix multiplication on dense matrices that are defined on the interfaces of the boxes. Since the number of interface nodes for two boxes close to the top of the tree is $O(N^{2/3})$, this leads to $O(N^2)$ overall complexity. It has recently been observed [19, 26, 31, 13, 22] that by exploiting internal structure in these dense matrices, the overall complexity can be reduced to linear, or close to linear, for both the build and the solve stage. Specifically, the structure exploited is that the off-diagonal blocks of these dense matrices have low numerical rank, and can be represented using so called "data sparse" formats such as, e.g., the $\mathcal{H}$-matrix format of Hackbusch and co-workers [16, 15, 3, 2]. The direct solver described here follows exactly the same idea as these linear complexity nested dissection schemes, but has a powerful advantage: While the performance of existing nested dissection schemes deteriorates very rapidly when the order of the discretization is increased, see [11, Table 3], the scheme proposed here allows very high order methods to be used with essentially no additional cost for the direct solver. (Loosely speaking, in the nested dissection method, the "separators" required to split the mesh into disconnected pieces grow thicker as the order is increased. In contrast, they stay razor thin in our version, regardless of the order.)

Like existing direct solvers for 3D problems, the method proposed here has two disadvantages in that it requires a fairly large amount of storage, and in that the time required to initially build the solution operators is comparatively large. In situations where an equation needs to be solved only once, and iterative methods converge rapidly, the method proposed here will in consequence not be competitive to techniques such as, e.g., multigrid. Our target is instead situations where iterative methods converge slowly, or where the cost of building the solution operator can be amortized over a large number of solves. Moreover, the method proposed involves less communication than iterative methods (since each solve involves only a single pass through the hierarchical tree, as opposed to one pass per iteration for iterative methods), which opens a path to the construction of solvers that can efficiently be parallelized.

2. Outline of proposed solver

The method described is based on a hierarchical subdivision of the computational domain, as illustrated in Figure 2. The solver consists of two stages. In the "build stage," we first process all of the leaf boxes in the tree. For each such box, a local solution operator and an approximation to the local Dirichlet-to-Neumann (DtN) operator are built. The build stage then continues with an upwards pass through the tree (going from smaller boxes to larger) where for each parent box, we construct approximations to its local solution operator and its

local DtN operator by "merging" the corresponding operators for its children. The end result of the "build stage" is a hierarchical representation of the overall solution operator for (1). Once this solution operator is available, the "solve stage" takes as input a given boundary data f, and constructs an approximation to the solution u valid throughout the domain through a single pass downwards in the tree (going from larger boxes to smaller).

The scheme described is a collocation scheme that uses for its collocation points tensor product grids of $q \times q$ Gaussian nodes (a.k.a. Legendre nodes) placed on each face of each leaf box in the domain. In addition to this global grid of Gaussian nodes, the computations executed on each leaf also rely on a local $p \times p \times p$ tensor product grid of Chebyshev nodes. Typically, $p = q + 1$ or $p = q + 2$.

The manuscript is structured as follows: Section 3 describes how a local spectral collocation method can be used to construct the local "solution operators" on a leaf in the tree. Section 4 describes how to construct a solution operator for a parent node, given the solution operators for its two children. Section 5 describes how the techniques in Sections 3 and 4 can be combined to form a direct solver for (1) based on a multidomain spectral discretization. The method described in Sections 3 – 5 has complexity $O(N^2)$ for the build stage. In Sections 6 and 7 we then describe how to accelerate the build stage to $O(N^{4/3})$ complexity by exploiting internal structure in the DtN operators. Finally, Section 8 illustrates the performance of the algorithm with several of numerical examples, and Section 9 summarizes the key features and limitations of the algorithm, and possibly extensions.

3. Leaf computation

3.1. **Overview.** This section describes a spectral method for computing a numerical approximation to the Dirichlet-to-Neumann operator associated with the equation (1) for a box domain Ω. In this section, we use a *global* spectral method to construct this operator. Later in this report, we will apply the techniques developed in this section to construct DtN operators for every leaf box in the hierarchical tree.

We represent potentials and fluxes on the six sides of the box Ω by collocation on tensor product grids of Gaussian nodes on each side. To be precise, fix a small positive integer q (say $q = 5$ or $q = 10$), and place on each side q^2 nodes in a tensor product grid. Let $\{\boldsymbol{z}_i\}_{i=1}^{6q^2}$ denote these points. Furthermore, let $\mathbf{u} \in \mathbb{R}^{6q^2}$ denote approximations to the potential at these points,

$$\mathbf{u}(i) \approx u(\boldsymbol{z}_i), \qquad i = 1, 2, \ldots, 6q^2.$$

Let $\mathbf{v} \in \mathbb{R}^{6q^2}$ denote approximations to the fluxes at the Gaussian nodes,

$$\mathbf{v}(i) \approx \begin{cases} \partial_1 u(\boldsymbol{z}_i), & \text{when } \boldsymbol{z}_i \text{ lies on a face parallel to the } x_2\text{-}x_3 \text{ plane,} \\ \partial_2 u(\boldsymbol{z}_i), & \text{when } \boldsymbol{z}_i \text{ lies on a face parallel to the } x_1\text{-}x_3 \text{ plane,} \\ \partial_3 u(\boldsymbol{z}_i), & \text{when } \boldsymbol{z}_i \text{ lies on a face parallel to the } x_1\text{-}x_2 \text{ plane.} \end{cases}$$

Our objective is now to build a $6q^2 \times 6q^2$ matrix $\mathbf{T}$ such that

$$\mathbf{v} = \mathbf{T}\mathbf{u}. \tag{3}$$

In order to build $\mathbf{T}$, we will work with a tensor product grid of $p \times p \times p$ Chebyshev nodes inside Ω (typically, $p = q + 1$). We proceed through the following steps: Given the vector $\mathbf{u}$ of tabulated Dirichlet data on the Gaussian nodes, we first interpolate to get Dirichlet data on the Chebyshev nodes that lie on the boundary. We then solve the PDE (1) using a spectral collocation method in the interior of the box, see [27]. This gives us the values of u tabulated at all interior nodes. We use spectral differentiation to construct the boundary fluxes, tabulated on the boundary Chebyshev nodes. Finally, we interpolate back from the Chebyshev nodes to the boundary Gaussian nodes. The combination of these four linear maps defines the matrix $\mathbf{T}$.

3.2. **Local spectral solver on the Chebyshev grid.** With p the order of the local Chebyshev approximation, as defined in Section 3.1, let $\{\boldsymbol{x}_k\}_{k=1}^{p^3}$ denote the points in a $p \times p \times p$ tensor product grid of Chebyshev nodes. Let $\tilde{\mathbf{u}} \in \mathbb{R}^{p^3}$ denote a vector holding approximations to u at $\{\boldsymbol{x}_k\}_{k=1}^{p^3}$ and let $\mathbf{D}^{(1)}$, $\mathbf{D}^{(2)}$, $\mathbf{D}^{(3)}$ denote the spectral differentiation matrices corresponding to the partial differential operators $\partial/\partial x_1$, $\partial/\partial x_2$ and $\partial/\partial x_3$. Then the operator in (1) can be approximated via a $p^3 \times p^3$ matrix

$$\begin{aligned}\mathbf{A} = &-\mathbf{C}_{11}(\mathbf{D}^{(1)})^2 - \mathbf{C}_{22}(\mathbf{D}^{(2)})^2 - \mathbf{C}_{33}(\mathbf{D}^{(3)})^2 \\ &- 2\mathbf{C}_{12}\mathbf{D}^{(1)}\mathbf{D}^{(2)} - 2\mathbf{C}_{13}\mathbf{D}^{(1)}\mathbf{D}^{(3)} - 2\mathbf{C}_{23}\mathbf{D}^{(2)}\mathbf{D}^{(3)} \\ &+ \mathbf{C}_1\mathbf{D}^{(1)} + \mathbf{C}_2\mathbf{D}^{(2)} + \mathbf{C}_3\mathbf{D}^{(3)} + \mathbf{C},\end{aligned}$$

where $\mathbf{C}_{ij}$ are diagonal matrices with entries $\{c_{i,j}(\boldsymbol{x}_k)\}_{k=1}^{p^3}$, and $\mathbf{C}_i$ and $\mathbf{C}$ are defined analogously.

Partition the index set of Chebyshev nodes as

$$\{1, 2, \ldots, p^3\} = I_{\mathrm{i}} \cup I_{\mathrm{e}}$$

where I_{i} lists the $(p-2)^3$ interior nodes and I_{e} lists the $p^3 - (p-2)^3$ exterior nodes. Then partition the vector $\tilde{\mathbf{u}}$ according to its value at internal and external nodes via

$$\tilde{\mathbf{u}}_{\mathrm{i}} = \tilde{\mathbf{u}}(I_{\mathrm{i}}) \quad \text{and} \quad \tilde{\mathbf{u}}_{\mathrm{e}} = \tilde{\mathbf{u}}(I_{\mathrm{e}}).$$

We also partition the $\mathbf{A}$ into four parts via

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

Following [27], we discretize (1) by enforcing a collocation condition at all interior nodes,

$$\mathbf{A}_{\mathrm{i,i}}\tilde{\mathbf{u}}_{\mathrm{i}} + \mathbf{A}_{\mathrm{i,e}}\tilde{\mathbf{u}}_{\mathrm{e}} = \mathbf{0}. \tag{4}$$

In constructing a DtN operator, we consider the Dirichlet data, as represented in $\tilde{\mathbf{u}}_{\mathrm{e}}$ as given, and seek to determine the values of the potential at the interior nodes. Solving (4), we find that

$$\tilde{\mathbf{u}}_{\mathrm{i}} = \mathbf{S}\,\tilde{\mathbf{u}}_{\mathrm{e}},$$

where the *solution operator* $\mathbf{S}$ is defined via

$$\mathbf{S} = -\mathbf{A}_{\mathrm{i,i}}^{-1}\mathbf{A}_{\mathrm{i,e}}. \tag{5}$$

3.3. **Interpolation between Gaussian and Chebyshev grids.** Construction of DtN operator consists for three steps:

Step 1 – re-tabulation from Gaussian nodes to Chebyshev nodes: Given a vector $\mathbf{u}_{\mathrm{e}} \in \mathbb{R}^{6q^2}$ of potential values tabulated on the Gaussian nodes on each face, form for each face the corresponding interpolating polynomial (this is the unique polynomial consisting of a sum of tensor products of polynomials of degree at most $q-1$). Then evaluate these six different polynomials at the exterior Cheybshev nodes. Let $\mathbf{L}_{\mathrm{G2C}}$ denote the matrix that achieves the interpolation.

Step 2 – solve and differentiate on the Chebyshev grids: Once the vector $\tilde{\mathbf{u}}_{\mathrm{e}}$ of Dirichlet data on the Chebyshev grid is available, we solve (1) using the technique in Section 3.2. Once the solution is available at the full $p \times p \times p$ Chebyshev grid, we use spectral differentiation to obtained the values of the boundary fluxes on each of the 6 faces. Let $\mathbf{V}$ denote the resulting linear map.

Step 3 – re-tabulation from Chebyshev nodes back to Gaussian nodes: In the final step, we simply interpolate the values of the boundary fluxes from the exterior Chebyshev nodes to the exterior Gaussian nodes. Let this interpolation matrix be denoted $\mathbf{L}_{\mathrm{C2G}}$.

Putting everything together, we form the DtN operator $\mathbf{T}$ as a product of three matrices

$$\underset{6q^2 \times 6q^2}{\mathbf{T}} = \underset{6q^2 \times 6p^2}{\mathbf{L}_{\mathrm{C2G}}} \quad \underset{6p^2 \times (6p^2 - 12p + 8)}{\mathbf{V}} \quad \underset{(6p^2 - 12p + 8) \times 6q^2}{\mathbf{L}_{\mathrm{G2C}}}, \tag{6}$$

where each matrix corresponds to one of the three steps described above.

4. Merge two DtN operators.

In this section, we describe how to construct the DtN operator for the PDE (1) under the assumption that the domain Ω_τ is a rectangular box formed by the union of two smaller rectangular boxes Ω_{σ_1} and Ω_{σ_2}, cf. Figure 1,

$$\Omega_\tau = \Omega_{\sigma_1} \cup \Omega_{\sigma_2}.$$

We further assume that the DtN operators $\mathbf{T}^{\sigma_1}$ and $\mathbf{T}^{\sigma_2}$ for the two "children" boxes have already been computed using the techniques described in Section 3. In this section, we describe how these two operators can be combined in a "merge operation" to form the DtN operator $\mathbf{T}^\tau$ associated with the union box.

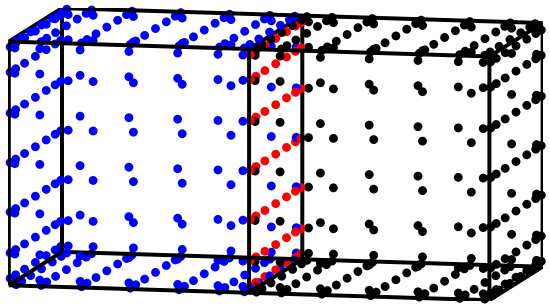


Figure 1. Illustration of the "merge step" described in Section 4, where the two boxes Ω_{σ_1} and Ω_{σ_2} are fused to form the larger rectangular domain Ω_τ. With the notation as in Section 4, the interpolation points are colored blue for J_1, black for J_2, and red for J_3.

We suppose that on each of the 11 faces involved in this geometry, there is a local tensor product grid of $q \times q$ Gaussian nodes. (Observe that the merge operation involves only sets of $q \times q$ Gaussian nodes on faces of boxes. The interior $p \times p \times p$ grids of Chebyshev nodes are used only for the leaf computations.) We partition these nodes into three sets:

- J_1 Boundary nodes of Ω_{σ_1} but not boundary nodes of Ω_{σ_2}.
- J_2 Boundary nodes of Ω_{σ_2} but not boundary nodes of Ω_{σ_1}.
- J_3 Boundary nodes shared by Ω_{σ_1} and Ω_{σ_2}. (These are interior nodes of Ω_τ).

Let $\mathbf{u}$ denote the vector holding tabulated solution values and let $\mathbf{v}$ denote the vector holding boundary fluxes values as in Section 3. Letting $\mathbf{u}_{\mathrm{i}}$ and $\mathbf{u}_{\mathrm{e}}$ denote the solution tabulated on the interior and exterior nodes of Ω_τ, we have

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

By the definitions of DtN operators for σ_1 and σ_2, we have

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{T}^{\sigma_1}_{1,1} & \mathbf{T}^{\sigma_1}_{1,3} \\ \mathbf{T}^{\sigma_1}_{3,1} & \mathbf{T}^{\sigma_1}_{3,3} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_3 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \mathbf{v}_2 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{T}^{\sigma_2}_{2,2} & \mathbf{T}^{\sigma_2}_{2,3} \\ \mathbf{T}^{\sigma_2}_{3,2} & \mathbf{T}^{\sigma_2}_{3,3} \end{bmatrix} \begin{bmatrix} \mathbf{u}_2 \\ \mathbf{u}_3 \end{bmatrix}. \tag{7}$$

Eliminating $\mathbf{v}_3$ from the equations in (7), we find

$$\mathbf{T}^{\sigma_1}_{3,1}\mathbf{u}_1 + \mathbf{T}^{\sigma_1}_{3,3}\mathbf{u}_3 = \mathbf{T}^{\sigma_2}_{3,2}\mathbf{u}_2 + \mathbf{T}^{\sigma_2}_{3,3}\mathbf{u}_3.$$

Therefore,

$$\mathbf{u}_3 = (\mathbf{T}^{\sigma_1}_{3,3} - \mathbf{T}^{\sigma_2}_{3,3})^{-1}(-\mathbf{T}^{\sigma_1}_{3,1}\mathbf{u}_1 + \mathbf{T}^{\sigma_2}_{3,2}\mathbf{u}_2) = \mathbf{S}^{\tau}\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix}, \tag{8}$$

where the *solution operator* $\mathbf{S}^{\tau}$ for the box τ is defined by.

$$\mathbf{S}^{\tau} = (\mathbf{T}^{\sigma_1}_{3,3} - \mathbf{T}^{\sigma_2}_{3,3})^{-1}[-\mathbf{T}^{\sigma_1}_{3,1} \mid \mathbf{T}^{\sigma_2}_{3,2}]. \tag{9}$$

Observe that $\mathbf{S}^{\tau}$ has a role analogous to the solution operator we defined for a leaf, cf. (5): it maps potential values on the boundary to potential values on the interior.

Now that we have solved for $\mathbf{u}_3$, we can express the boundary fluxes $\{\mathbf{v}_1, \mathbf{v}_2\}$ as a function of $\{\mathbf{u}_1, \mathbf{u}_2\}$. To be precise, combining (7) and (8), we find

$$\begin{bmatrix}\mathbf{v}_1\\ \mathbf{v}_2\end{bmatrix} = \begin{bmatrix}\mathbf{T}^{\sigma_1}_{1,1}\mathbf{u}_1 + \mathbf{T}^{\sigma_1}_{1,3}\mathbf{u}_3\\ \mathbf{T}^{\sigma_1}_{2,2}\mathbf{u}_2 + \mathbf{T}^{\sigma_1}_{2,3}\mathbf{u}_3\end{bmatrix} = \begin{bmatrix}\mathbf{T}^{\sigma_1}_{1,1}\mathbf{u}_1 + \mathbf{T}^{\sigma_1}_{1,3}(\mathbf{T}^{\sigma_1}_{3,3} - \mathbf{T}^{\sigma_2}_{3,3})^{-1}(-\mathbf{T}^{\sigma_1}_{3,1}\mathbf{u}_1 + \mathbf{T}^{\sigma_2}_{3,2}\mathbf{u}_2)\\ \mathbf{T}^{\sigma_1}_{2,2}\mathbf{u}_2 + \mathbf{T}^{\sigma_1}_{2,3}(\mathbf{T}^{\sigma_1}_{3,3} - \mathbf{T}^{\sigma_2}_{3,3})^{-1}(-\mathbf{T}^{\sigma_1}_{3,1}\mathbf{u}_1 + \mathbf{T}^{\sigma_2}_{3,2}\mathbf{u}_2)\end{bmatrix} = \mathbf{T}^{\tau}\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix},$$

where the Dirichlet-to-Neumann operator $\mathbf{T}^{\tau}$ for τ is defined via

$$\mathbf{T}^{\tau} = \begin{bmatrix}\mathbf{T}^{\sigma_1}_{1,1} & 0\\ 0 & \mathbf{T}^{\sigma_2}_{2,2}\end{bmatrix} + \begin{bmatrix}\mathbf{T}^{\sigma_1}_{1,3}\\ \mathbf{T}^{\sigma_2}_{2,3}\end{bmatrix}\mathbf{S}^{\tau}. \tag{10}$$

Remark 1. *Formula (8) involves the inverse of the matrix $\mathbf{T}^{\sigma_1}_{3,3} - \mathbf{T}^{\sigma_2}_{3,3}$, which raises the questions of first whether this matrix is necessarily invertible, and second what its condition number might be. We are unfortunately not aware of any rigorous mathematical results that answer these questions. However, extensive numerical experiments indicate that for coercive elliptic problems (e.g. Laplace, Yukawa), no difficulties in this regard arise. For problems with oscillatory solutions (e.g. Helmholtz, time-harmonic Maxwell) the situation is different. In this case, the DtN operator itself is not guaranteed to exist, and is in practice often ill-conditioned. This problem can be eliminated by replacing the DtN operator by other Poincaré-Steklov operators that are guaranteed to be well-posed, such as, e.g., the impedance-to-impedance map, as described in* [12]*. To keep the presentation simple, we in this paper use the DtN operator exclusively, which from a practical point of view tends to work very well even for Helmholtz problems involving reasonably large domains, as shown in the numerical experiments in Section 8.*

5. Direct solver via composite spectral method

Now that we know how to construct the DtN operator for a leaf (Section 3), and how to merge the DtN operators of two neighboring patches to form the DtN operator of their union (Section 4), we are ready to describe the full hierarchical scheme for solving (1).

5.1. **Discretization.** To start with, partition the domain Ω into a hierarchical tree of boxes, as shown in Figure 2. We let $\Omega_1 = \Omega$ denote the entire domain, the root of the tree. Then cut this box into two halves $\Omega_1 = \Omega_2 \cup \Omega_3$. Then continue cutting into smaller boxes, until all boxes are small enough that the local solution can accurately be resolved using a local grid of $p \times p \times p$ Chebyshev nodes. The set of boxes that were never subdivided are called *leaves.* If α and β are two boxes whose union forms a box τ, we say that α and β are *siblings,* and are the *children* of τ. We assume the ordering is such that if α is the child of τ, then $\alpha > \tau$. For simplicity, we assume the tree is uniform so that all leaf nodes are on the same hierarchical level. Let $\ell = 0$ denote the coarsest level and $\ell = L$ denote the level with the leaf nodes. Figure 2 shows the boxes on levels 0 to 3.

Now fix a small integer q that denotes the local order of the approximation (picking q between 5 and 20 has proven to be good choices). On each face of the leaf boxes, we then place a $q \times q$ tensor product grid of Gaussian nodes, which will serve as collocation nodes. Let $\{\boldsymbol{z}_\ell\}_{\ell=1}^{N}$

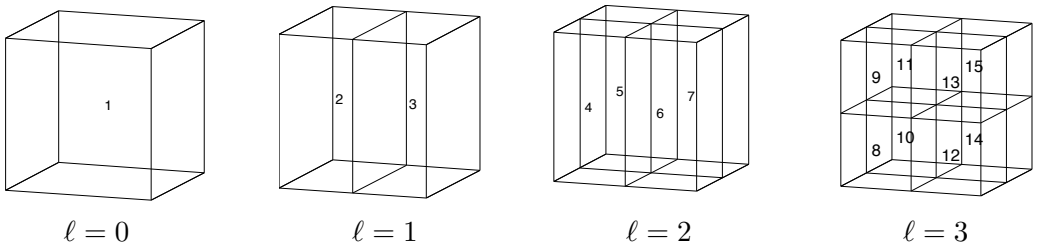


FIGURE 2. The rectangular domain Ω is split into $2 \times 2 \times 2$ leaf boxes. These are then organized into a binary tree of successively larger boxes as described in Section 5.1. Note that if box τ is the parent of a box σ, we have $\tau < \sigma$.

denote the collection of all collocation nodes on the boundaries. Let $\mathbf{u} \in \mathbb{R}^N$ be the vector holding approximation to the solution function u tabulated on $\{\boldsymbol{z}_\ell\}_{\ell=1}^N$. In other words,

$$\mathbf{u}(\ell) \approx u(\boldsymbol{z}_\ell).$$

5.2. **The full hierarchical scheme.** The algorithm we describe is a direct solver. It has a *build stage* where we construct the solution operator to the equation (1). Once this solution operator has been constructed, we can for any given boundary data f very rapidly construct the full solution u by applying the solution operator in a *solve stage*.

The build stage involves the following steps:

(1) *Discretization:* The domain is partitioned into a hierarchical tree as described in Section 5.1 and the global grid $\{\boldsymbol{z}_\ell\}_{\ell=1}^N$ is formed.
(2) *Leaf computation:* For each leaf box, a solution operator and DtN operator are constructed as described in Section 3. Recall that this computation relies on a local spectral discretization of (1) on the local $p \times p \times p$ Chebyshev grid.
(3) *Upwards pass:* We execute a single sweep through the tree, starting at the leaves and moving up to the root node (the entire domain). For each parent node, we construct its DtN operator and solution operator by "merging" the DtN operators of its two children, as described in Section 4.

The solve stage involves the following steps:

(1) *Downwards pass:* Given a vector holding the Dirichlet boundary data as input, we execute a sweep downwards through the tree, starting with the root (the entire domain) and working our way down towards the leaves. When processing a parent node, we know the value of the potential on its boundary nodes, and construct the potential at its interior nodes using the solution operator, as defined by (9).
(2) *Leaf processing:* At the end of the downwards sweep, we have constructed the potential at the Gaussian grids on all leaf edges. We can then reconstruct the potential at the interior Chebyshev nodes for every box by applying the local solution operator (5).

5.3. **Asymptotic complexity.** Let N be the total number of Chebyshev nodes used to discretize the three-dimensional rectangular domain Ω and suppose that we locally use Chebyshev grids with $p \times p \times p$ nodes. We further assume that the order of the Chebyshev and Gaussian meshes are similar, so that $q \approx p$. When analyzing the algorithm, we temporarily consider a scheme in which the hierarchical tree is an octree rather than a binary tree. The corresponding merge operator merges the DtN operators of the eight children of a parent box. Note that this change from a binary tree to an octree is merely a cosmetic change made to simplify the analysis. (The merge-8 operation can easily be executed as a sequence of three pair-wise merges.) At level

ℓ, a box in the octree has roughly $2^{-\ell}\,N^{1/3}$ discretization points along a side. Finally, observe that with L denoting the number of levels in the octree, we have $N \approx 8^L\,p^3$.

Let us start with analyzing the "build stage" where we build all solution operators. Here, we first process all leaves in the tree. Since computing the DtN operators for each leaf box involves inverting dense matrices of size $p^3 \times p^3$, and since there are N/p^3 leaves, the total cost for processing all leaves is

$$T_{\text{leaf}} \sim \frac{N}{p^3} \times (p^3)^3 \sim N p^6. \tag{11}$$

Next, we perform an upwards pass through the tree where we merge boxes in sets of eight. For each box τ on level ℓ, the operators $\mathbf{T}^\tau$ and $\mathbf{S}^\tau$ are constructed via (9) and (10). A box on level ℓ has roughly $2^{-\ell}\,N^{1/3}$ points along a side, which means the dense matrices involved in these formulas are of sizes $O(2^{-2\ell}\,N^{2/3}) \times O(2^{-2\ell}\,N^{2/3})$. Since there are $8^\ell = 2^{3\ell}$ boxes on level ℓ, and since the cost of matrix inversion is cubic in the matrix dimensions, the cost to process level ℓ is

$$T_\ell \sim 2^{3\ell} \times (2^{-2\ell} N^{2/3})^3 \sim 2^{-3\ell} N^2. \tag{12}$$

To summarize, the cost for the build stage is then

$$T_{\text{build}} = T_{\text{leaf}} + \sum_{\ell=0}^{L-1} T_\ell \sim N\,p^6 + \sum_{\ell=0}^{L-1} 2^{-3\ell} N^2 \sim N\,p^6 + N^2.$$

Next we consider the "solve stage" where we build a solution for a given set of boundary data. This stage consists of a downwards pass, where we at level ℓ need to apply dense matrices of size $O(2^{-2\ell}\,N^{2/3}) \times O(2^{-2\ell}\,N^{2/3})$. Since there are $2^{3\ell}$ boxes on level ℓ, this gives a total cost of processing level ℓ of $2^{3\ell} \times (2^{-2\ell} N^{2/3})^2 \sim 2^{-\ell} N^{4/3}$. For each leaf box on the bottom level, the solution operator is of size $O(p^3) \times O(p^2)$, so the cost of processing the leaves is $(N/p^3) \times p^3 \times p^2 \sim N\,p^2$. Consequently,

$$T_{\text{solve}} \sim N\,p^2 + \sum_{\ell=1}^{L} 2^{-\ell} N^{4/3} \sim N\,p^2 + N^{4/3}.$$

6. Fast algorithms for compressible matrices

The cost of the algorithm presented in Section 5.2 is dominated by constructing DtN operators at the top levels. The matrix operations involve inverting dense matrices of size $O(N^{2/3}) \times O(N^{2/3})$ where N is total number of discretization nodes, resulting in $O(N^2)$ total cost. However, there is internal structure in these dense matrices that can be explored to accelerate the computation. Specifically, the off-diagonal blocks of these matrices are rank-deficient to high precision and the diagonal blocks can be represented as *Hierarchical Block-Separable (HBS)* matrices. In this section, we describe the HBS matrix format.

Remark 2. *The format we refer to as HBS in this note is essentially identical to a format sometimes referred to as Hierarchically Semi-Separable (HSS), see, e.g.* [6, 4, 28]. *We prefer the term "block-separable" since these matrices are not really related to "semi-separable" matrices but the term HSS has stuck for historical reasons.*

6.1. Compressible matrices.

We first give definition of block separable matrix. Let $\mathbf{A}$ be a matrix of size $mp \times mp$ such that it is partitioned into $p \times p$ blocks, each of size $m \times m$, *i.e.*

$$\mathbf{A} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{A}_{1,2} & \dots & \mathbf{A}_{1,p} \\ \mathbf{A}_{2,1} & \mathbf{D}_2 & \dots & \mathbf{A}_{2,p} \\ \vdots & \vdots & & \vdots \\ \mathbf{A}_{p,1} & \mathbf{A}_{p,2} & \dots & \mathbf{D}_p \end{bmatrix} \tag{13}$$

We say that the matrix $\mathbf{A}$ is *block separable* if each off-diagonal block admits the factorization

$$\underset{m\times m}{\mathbf{A}_{\sigma,\tau}} = \underset{m\times k}{\mathbf{U}_{\sigma}} \quad \underset{k\times k}{\tilde{\mathbf{A}}_{\sigma,\tau}} \quad \underset{k\times m}{\mathbf{V}_{\tau}^{*}}, \quad \sigma,\tau \in \{1,2,\ldots,p\}, \quad \sigma \neq \tau, \tag{14}$$

where $\mathbf{U}$ and $\mathbf{V}$ are $m \times k$ matrices. Therefore, the matrix $\mathbf{A}$ admits the factorization

$$\underset{mp\times mp}{\mathbf{A}} = \underset{mp\times kp}{\mathbf{U}} \quad \underset{kp\times kp}{\tilde{\mathbf{A}}} \quad \underset{kp\times mp}{\mathbf{V}^{*}} + \underset{mp\times mp}{\mathbf{D}}, \tag{15}$$

where $\mathbf{U} = \operatorname{diag}(\mathbf{U}_1, \mathbf{U}_2, \ldots, \mathbf{U}_p)$, $\mathbf{V} = \operatorname{diag}(\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_p)$ and $\mathbf{D} = \operatorname{diag}(\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_p)$. Moreover,

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{A}}_{1,2} & \tilde{\mathbf{A}}_{1,3} & \ldots \\ \tilde{\mathbf{A}}_{2,1} & \mathbf{0} & \tilde{\mathbf{A}}_{2,3} & \ldots \\ \tilde{\mathbf{A}}_{3,1} & \tilde{\mathbf{A}}_{3,2} & \mathbf{0} & \ldots \\ \vdots & \vdots & \vdots & \end{bmatrix}.$$

Remark 3. *In constructing the factorization (14), we use the so-called interpolative decomposition (ID)* [7]*. For an $m \times n$ matrix $\mathbf{B}$ or rank k, the ID takes the form*

$$\underset{m\times n}{\mathbf{B}} = \underset{m\times k}{\mathbf{X}} \quad \underset{k\times k}{\mathbf{B}(I,\,J)} \quad \underset{k\times n}{\mathbf{Y}},$$

where I and J are index vectors pointing to k rows and columns of $\mathbf{B}$, respectively, and where the matrices $\mathbf{X}$ and $\mathbf{Y}$ are well-conditioned and have the $k \times k$ identity matrix as submatrices. When the ID is used in (14), the matrix $\tilde{\mathbf{A}}_{\sigma,\tau}$ will be simply a submatrix of $\mathbf{A}(I_\sigma, I_\tau)$. See [7, 20, 17, 21] *for details.*

6.1.1. *Hierarchically block-separable (HBS) matrices.* To give the definition of HBS matrices, we first define the binary tree structure associated with the discretization nodes with index $I = [1, 2, \ldots M]$. Basically, let I denote the root of the tree and partition the index set into two roughly equi-sized subsets level by level. Specifically, we call an index set as leaf node if there is no further split. For a non-leaf node τ we call two nodes σ_1 and σ_2 as the children of τ if $I_\tau = I_{\sigma_1} \cup I_{\sigma_2}$, and call τ the parent node of σ_1 and σ_2.

By now, we are ready to give definition of hierarchical block-separable with respect to a given binary tree associated with index set I. Let $\ell = 0, 1, \ldots, L$ denote the levels from the coarsest level to the finest level. A matrix is called a HBS matrix if it satisfies two conditions:

(1) For each leaf node pair $\{\tau, \tau'\}$ ($\tau \neq \tau'$) on level L, there exists integer k such that the off-diagonal blocks $\mathbf{A}_{\tau,\tau'}$ admits factorization

$$\underset{m\times m}{\mathbf{A}_{\tau,\tau'}} = \underset{m\times k}{\mathbf{U}_{\tau}} \quad \underset{k\times k}{\tilde{\mathbf{A}}_{\tau,\tau'}} \quad \underset{k\times m}{\mathbf{V}_{\tau}^{*}}. \tag{16}$$

(2) For off-diagonal blocks on level $\ell = L-1, L-2, \ldots, 1$, the rank-deficiency property at level ℓ can be constructed based on the next finer level $\ell+1$. Specifically, for any distinct non-leaf nodes τ and τ' on level ℓ with children σ_1, σ_2 and σ_1' and σ_2', define

$$\mathbf{A}_{\tau,\tau'} = \begin{bmatrix} \tilde{\mathbf{A}}_{\sigma_1,\sigma_1'} & \tilde{\mathbf{A}}_{\sigma_1,\sigma_2'} \\ \tilde{\mathbf{A}}_{\sigma_2,\sigma_1'} & \tilde{\mathbf{A}}_{\sigma_2,\sigma_2'} \end{bmatrix}.$$

There exists factorization

$$\underset{2k\times 2k}{\mathbf{A}_{\tau,\tau'}} = \underset{2k\times k}{\mathbf{U}_{\tau}} \quad \underset{k\times k}{\tilde{\mathbf{A}}_{\tau,\tau'}} \quad \underset{k\times 2k}{\mathbf{V}_{\tau}^{*}}. \tag{17}$$

Define

$$\mathbf{D}_\tau = \mathbf{A}(I_\tau, I_\tau)$$

for each leaf node τ, and define

$$\mathbf{B}_\tau = \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{A}}_{\sigma_1,\sigma_2} \\ \tilde{\mathbf{A}}_{\sigma_2,\sigma_1} & \mathbf{0} \end{bmatrix}$$

for non-leaf node τ with children σ_1 and σ_2. An HBS matrix $\mathbf{A}$ can then be fully described if

- for every leaf node, we are given the diagonal matrices $\mathbf{D}_\tau$, as well as column basis and row basis $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$;
- for every non-leaf node, we are given the interaction matrix $\mathbf{B}_\tau$ between the children of τ, as well as column basis and row basis $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$.

In other words, the HBS matrix $\mathbf{A}$ admits telescoping factorization given $\mathbf{U}$, $\mathbf{V}$, $\mathbf{D}$ and $\mathbf{B}$ hierarchically:

(1) $\tilde{\mathbf{A}}^{(0)} = \mathbf{B}^{(0)}$,

(2) $\underset{k2^\ell \times k2^\ell}{\tilde{\mathbf{A}}^{(\ell)}} = \underset{k2^\ell \times k2^{\ell-1}}{\mathbf{U}^{(\ell)}} \quad \underset{k2^{\ell-1} \times k2^{\ell-1}}{\tilde{\mathbf{A}}^{(\ell-1)}} \quad \underset{k2^{\ell-1} \times k2^\ell}{(\mathbf{V}^{(\ell)})^*} \quad + \quad \underset{k2^\ell \times k2^\ell}{\mathbf{B}^{(\ell)}} \quad$ for $\ell = 1, 2, \ldots, L-1$,

(3) $\underset{m2^L \times n2^L}{\mathbf{A}} = \underset{m2^L \times k2^L}{\mathbf{U}^{(L)}} \quad \underset{k2^L \times k2^L}{\tilde{\mathbf{A}}^{(L-1)}} \quad \underset{k2^L \times n2^L}{(\mathbf{V}^{(L)})^*} \quad + \quad \underset{m2^L \times m2^L}{\mathbf{D}^{(L)}}$.

6.2. **Fast algorithms on HBS matrices.** Fast algorithms on how to add two HBS matrices, apply HBS matrices to vector and invert HBS matrices are presented in [14]. Here we briefly summarize the fast matrix-vector multiplication and inversion algorithms.

6.2.1. *Matrix inversion.* The inverse of an HBS matrix can be rapidly constructed using a variation of the classical Sherman-Morrison-Woodbury formula by exploring the low-rank deficiency hierarchically. Here we provide a condensed description of the inversion procedure given in [14], which has asymptotic cost of $O(N\,k^2)$, for an $N \times N$ matrix whose off-diagonal blocks have rank k. Firstly, we define several matrices on each node τ

$$\hat{\mathbf{D}}_\tau = (\mathbf{V}_\tau^* \tilde{\mathbf{D}}_\tau^{-1} \mathbf{U}_\tau)^{-1}, \tag{18}$$

$$\mathbf{E}_\tau = \tilde{\mathbf{D}}_\tau^{-1} \mathbf{U}_\tau \hat{\mathbf{D}}_\tau, \tag{19}$$

$$\mathbf{F}_\tau = (\hat{\mathbf{D}}_\tau \mathbf{V}_\tau^* \tilde{\mathbf{D}}_\tau^{-1})^*, \tag{20}$$

$$\mathbf{G}_\tau = \tilde{\mathbf{D}}_\tau^{-1} - \tilde{\mathbf{D}}_\tau^{-1} \mathbf{U}_\tau \hat{\mathbf{D}}_\tau \mathbf{V}_\tau^* \tilde{\mathbf{D}}_\tau^{-1}. \tag{21}$$

In above equations, we define

$$\tilde{\mathbf{D}}_\tau = \mathbf{D}_\tau$$

if τ is a leaf node and define

$$\tilde{\mathbf{D}}_\tau = \begin{bmatrix} \hat{\mathbf{D}}_{\sigma_1} & \mathbf{B}_{\sigma_1,\sigma_2} \\ \mathbf{B}_{\sigma_2,\sigma_1} & \hat{\mathbf{D}}_{\sigma_2} \end{bmatrix}$$

if τ is a non-leaf node. Note that all the matrices $\{\hat{\mathbf{D}}_\tau, \mathbf{E}_\tau, \mathbf{F}_\tau, \mathbf{G}_\tau\}_\tau$ can be computed cheaply level by level. Using the formula

$$\mathbf{A}^{-1} = \mathbf{E}(\tilde{\mathbf{A}} + \hat{\mathbf{D}})^{-1} \mathbf{F}^* + \mathbf{G}, \tag{22}$$

we can express $\mathbf{A}^{-1}$ hierarchically via

$$(\tilde{\mathbf{A}}^{(\ell)} + \hat{\mathbf{D}}^{(\ell)})^{-1} = \mathbf{E}^{(\ell-1)} (\tilde{\mathbf{A}}^{(\ell-1)} + \hat{\mathbf{D}}^{(\ell-1)})^{-1} (\mathbf{F}^{(\ell-1)})^* + \mathbf{G}^{(\ell-1)} \tag{23}$$

for $\ell = L, L-1, \ldots, 2$, and

$$(\tilde{\mathbf{A}}^{(1)} + \hat{\mathbf{D}}^{(1)})^{-1} = \begin{bmatrix} \hat{\mathbf{D}}_2 & \mathbf{B}_{2,3} \\ \mathbf{B}_{3,2} & \hat{\mathbf{D}}_3 \end{bmatrix}^{-1} = \mathbf{G}^{(0)} = \mathbf{G}_1. \tag{24}$$

Algorithm 3 summarizes the inversion procedure of HBS matrices.

6.2.2. *Matrix-vector multiplication.* In this section, we briefly describe how to compute $\mathbf{y} = \mathbf{A}^{-1}\mathbf{x}$, for given $\mathbf{x}$, using the compressed representation of $\mathbf{A}^{-1}$ resulting from the inversion algorithm. Using equations (23) and (24), $\mathbf{y}$ can be computed hierarchically as

$$\begin{aligned}
\mathbf{y} = \mathbf{A}^{-1}\mathbf{x} &= (\tilde{\mathbf{A}}^{(L)} + \hat{\mathbf{D}}^{(L)})^{-1}\mathbf{x} \\
&= \mathbf{E}^{(L-1)}(\tilde{\mathbf{A}}^{(L-1)} + \hat{\mathbf{D}}^{(L-1)})^{-1}(\mathbf{F}^{(L-1)})^*\mathbf{x} + \mathbf{G}^{(L-1)}\mathbf{x} \\
&= \mathbf{E}^{(L-1)}\mathbf{E}^{(L-2)}(\tilde{\mathbf{A}}^{(L-2)} + \hat{\mathbf{D}}^{(L-2)})^{-1}(\mathbf{F}^{(L-2)})^*(\mathbf{F}^{(L-1)})^*\mathbf{x} + \mathbf{G}^{(L-2)}\mathbf{G}^{(L-1)}\mathbf{x} \\
&= \mathbf{E}^{(L-1)}\mathbf{E}^{(L-2)}\dots\mathbf{E}^{(1)}\mathbf{G}_1(\mathbf{F}^{(1)})^*\dots(\mathbf{F}^{(L-2)})^*(\mathbf{F}^{(L-1)})^*\mathbf{x} + \mathbf{G}^{(1)}\dots\mathbf{G}^{(L-2)}\mathbf{G}^{(L-1)}\mathbf{x}.
\end{aligned}$$

Details on the fast matrix-vector multiplication algorithm are given in Algorithm 4.

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, $\ell = L, L-1, \dots, 1$
  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
$\mathbf{G}_1 = \begin{bmatrix} \hat{\mathbf{D}}_2 & \mathbf{B}_{2,3} \\ \mathbf{B}_{3,2} & \hat{\mathbf{D}}_3 \end{bmatrix}^{-1}$.

6.2.3. *Asymptotic costs of HBS matrix algebra.* Let us analyze the asymptotic cost of matrix inversion and matrix-vector multiplication for an HBS matrix of size $M \times M$, with ranks of the off-diagonal blocks k. Let L denote the total number of levels, and suppose that $M \sim k\,2^L$. In order to execute the inversion algorithm described in Section 6.2.1, we need to compute all matrices $\hat{\mathbf{D}}_\tau, \mathbf{E}_\tau, \mathbf{F}_\tau, \mathbf{G}_\tau$ on each level ℓ. The cost is dominated by performing dense matrix inversion of matrices of size $2k \times 2k$ on each level, where there are 2^ℓ nodes totally. The total cost is therefore

$$T_{\text{inv}} \sim \sum_{\ell=1}^{L} 2^\ell (2k)^3 \sim 2^L\, k^3 \sim Mk^2.$$

Moreover, the total cost of fast matrix-vector multiplication is

$$T_{\text{multi}} = \sum_{\ell=1}^{L} 2^\ell (2k)^2 \sim 2^L k^2 \sim Mk.$$

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

7. Accelerating the direct solver

In this section, we describe how to accelerate the direct solver in Section 5.2 using the fast algorithms described in Section 6.2 to achieve $O(N^{4/3})$ complexity.

7.1. **Acceleration using structured matrix algebra.** Recall from Section 5.3 that the dominant cost of the basic scheme described in Section 5.2 derives from the need to perform matrix computations on dense matrices whose sizes get large as we approach to top of the tree. For instance, recall that the DtN operator for a node τ with children σ_1 and σ_2 is computed via the formula, cf. (10),

$$\mathbf{T}^\tau = \begin{bmatrix} \mathbf{T}_{1,1}^{\sigma_1} & 0 \\ 0 & \mathbf{T}_{2,2}^{\sigma_2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}_{1,3}^{\sigma_1} \\ \mathbf{T}_{2,3}^{\sigma_2} \end{bmatrix} (\mathbf{T}_{3,3}^{\sigma_1} - \mathbf{T}_{3,3}^{\sigma_2})^{-1} [-\mathbf{T}_{3,1}^{\sigma_1} \mid \mathbf{T}_{3,2}^{\sigma_2}].$$

The key observation is now that since $\mathbf{T}_{3,3}^{\sigma_1}$ and $\mathbf{T}_{3,3}^{\sigma_2}$ each represent (discretized) parts of the DtN operators for σ_1 and σ_2, they can be represented using the HBS format described in Section 6, which allows us to perform matrix inversion very efficiently. Analogously, for every parent node τ, the solution operator $\mathbf{S}_\tau$ is a discretization of the operator that maps Dirichlet boundary data for Ω_τ to the face shared by Ω_{σ_1} and Ω_{σ_2}, which is an interior plane of Ω_τ. Standard regularity results for elliptic PDEs indicate that this operator is globally of low rank.

Observe that the accelerated scheme makes no *detailed* à priori assumptions on the numerical ranks of the various matrices that arise. Instead, the scheme takes as input a tolerance ε, and numerically determines the optimal rank that achieves the specified tolerance. We use insights from regularity theory for PDEs only for two purposes: (1) As a guide on where to look for rank deficiencies. (2) In order to estimate the overall asymptotic complexity of the scheme.

Let us show in some additional detail how a merge operation gets executed once all discretized DtN operators are represented in the HBS format. Using the same notation as in Section 4, we suppose that τ is a parent box with children σ_1 and σ_2. We then have the DtN operators $\mathbf{T}^{\sigma_1}$ and $\mathbf{T}^{\sigma_2}$ available in DtN formats. From these representations, we extract the low rank factors associated with $\mathbf{T}^{\sigma_1}_{1,3}$, $\mathbf{T}^{\sigma_1}_{3,1}$, $\mathbf{T}^{\sigma_2}_{2,3}$ and $\mathbf{T}^{\sigma_2}_{3,2}$, and denote these by $\{\mathbf{Q}_{1,3}, \mathbf{R}_{1,3}\}$, $\{\mathbf{Q}_{3,1}, \mathbf{R}_{3,1}\}$, $\{\mathbf{Q}_{2,3}, \mathbf{R}_{2,3}\}$ and $\{\mathbf{Q}_{3,2}, \mathbf{R}_{3,2}\}$, respectively. The sub-matrices $\mathbf{T}^{\sigma_1}_{3,3}$ and $\mathbf{T}^{\sigma_2}_{3,3}$ are themselves in HBS format. Then the following computations will execute the merge:

(1) Add two HBS matrices $\mathbf{T}^{\sigma_1}_{3,3}$ and $-\mathbf{T}^{\sigma_2}_{3,3}$ resulting a new HBS matrix $\mathbf{T}^{\sigma_1}_{3,3} - \mathbf{T}^{\sigma_2}_{3,3}$.
(2) Invert the HBS matrix $\mathbf{T}^{\sigma_1}_{3,3} - \mathbf{T}^{\sigma_2}_{3,3}$.
(3) Apply the thin low rank factor $\begin{bmatrix}\mathbf{R}_{1,3}\\ \mathbf{R}_{2,3}\end{bmatrix}$ to the inverse (in HBS form). The resulting matrix together with another low rank factor $[-\mathbf{Q}_{3,1} \mid \mathbf{Q}_{3,2}]$ form the low rank approximation to the solution operator $\mathbf{S}^{\tau}$.
(4) Performing matrix products $\mathbf{T}^{\sigma_1}_{1,3}\,\mathbf{S}^{\tau}$ and $\mathbf{T}^{\sigma_2}_{2,3}\,\mathbf{S}^{\tau}$ are analogous, just exploiting all factors are low rank.
(5) Perform a low-rank update to the block-diagonal matrix $\begin{bmatrix}\mathbf{T}^{\sigma_1}_{1,1} & 0\\ 0 & \mathbf{T}^{\sigma_2}_{2,2}\end{bmatrix}$, whose blocks are provided in HBS-form to construct the new HBS matrix $\mathbf{T}^{\tau}$.

Remark 4. *For intermediate size problems, we can use simpler "data sparse" formats to store the matrices $\mathbf{T}^{\tau}$ encoding all DtN operators. For instance, each such matrix can be represented as a 6×6 block matrix, corresponding to interactions between the 6 faces of the box Ω_τ. Substantial gains are achieved by storing the six diagonal blocks of this matrix densely (without exploiting any internal structure), and the 30 off-diagonal blocks using simple low rank representations. A format such as this does not improve the asymptotic $O(N^2)$, but is far easier to code, and greatly accelerates the practical execution time for mid-size problems.*

7.2. **Asymptotic cost of the accelerated solver.** We use the same notation as in Section 5.3, with N denoting the total number of Chebyshev nodes used to discretize the three-dimensional rectangular domain Ω, and L denoting be the number of levels in the octree, so that there are 8^L leaf boxes. Since there are p^3 Chebyshev discretization nodes on each leaf box, we have $N \approx 8^L\, p^3$. We further assume that the order of the Chebyshev and Gaussian meshes are similar, $q \approx p$.

We first discuss the cost of the "build stage." When processing the leaves, we do not use structured matrix algebra, so the cost (11) remains valid. For the upwards pass where we merge boxes to construct the DtN operator $\mathbf{T}^{\tau}$ and the solution operator $\mathbf{S}^{\tau}$ for each parent node τ, we exploit that all matrices are stored either in HBS format, or as low-rank matrices. Recall that there are $2^{3\ell}$ boxes on level ℓ and that all matrices are now of size $M \times M$, with $M \sim 2^{-2\ell}N^{2/3}$ (recall that there are roughly $2^{-\ell}N^{1/3}$ points along a side of a box on level ℓ). Using that the time required to invert an $M \times M$ HBS matrix of rank k is $O(M\,k^2)$ (see Section 6.2.3), we then find that the cost to process level ℓ is

$$T_\ell \sim 2^{3\ell} \times 2^{-2\ell} N^{2/3} k^2 \sim 2^{\ell} N^{2/3} k^2.$$

This rank k depends on the level. In the present situation, the operator represented in the HBS format approximates an integral operator defined on a face discretized into $2^{-\ell}N^{1/3} \times 2^{-\ell}N^{1/3}$ points. It was shown in [5] the maximal rank k encountered in this situation satisfies

$$k \sim 2^{-\ell} N^{1/3}. \tag{25}$$

In consequence, the build stage has an overall asymptotic cost that is bounded by

$$T_{\text{build}} = T_{\text{leaf}} + \sum_{\ell=0}^{L-1} T_\ell \sim N\,p^6 + \sum_{\ell=0}^{L-1} 2^\ell\, N^{2/3}\,(2^{-\ell}N^{1/3})^2 \sim N\,p^6 + N^{4/3}. \tag{26}$$

Remark 5. *Observe that our derivation of a bound on the asymptotic complexity is slightly crude. We assumed that at level ℓ in the "build stage," the ranks used inside the HBS structure were constant across all levels in the HBS structure, and satisfy (25). In reality, the ranks vary across levels inside the HBS structure too, which means that the bound $M\,k^2$ is pessimistic. Performing a more careful analysis gets slightly involved given that there are two hierarchical structures nested inside each other. We expect however that the final result would be substantially better than $O(N^{4/3})$, and would follow very closely the estimates provided in* [29, 30, 25]*, where the use of structured matrix algebra to accelerate the classical nested dissection algorithm is analyzed in some detail.*

For the "solve stage," we execute a downwards pass through the tree. At level ℓ, we apply $2^{3\ell}$ solution operators of size size $O(2^{-2\ell}\,N^{2/3}) \times O(2^{-2\ell}\,N^{2/3})$ that have numerical rank $k \sim 2^{-\ell}\,N^{1/3}$. This results in a total time to process level ℓ of $2^{3\ell} \times 4^{-\ell}N^{2/3} \times 2^{-\ell}N^{1/3} \sim N$. For each leaf box on the bottom level, the solution operator is of size $O(p^3) \times O(p^2)$, so the cost of processing the leaves is $(N/p^3) \times p^3 \times p^2 \sim N\,p^2$. Consequently,

$$T_{\text{solve}} \sim N\,p^2 + \sum_{\ell=1}^{L} N \sim N\,p^2 + N\,\log N.$$

Remark 6. *In deriving the estimates on the asymptotic complexity, we assumed that the underlying equation is fixed as N is increased. This is a reasonable assumption for problems such as Laplace and low-frequency Helmholtz. However, when solving problems with oscillatory solutions, one often strives to keep the "number of points per wave-length" constant as N increases. In this case, the estimates given in this section will not hold.*

8. Numerical experiments

This section presents numerical experiments that illustrte the performance of the proposed direct solver. All the experiments are carried out on a personal work-station with an Intel Xeon E-1660 3.3GHz 6-core CPU, and 128GB of RAM. The experiments serve two purposes. The first is to systematically measure the speed and memory requirements (the amount of RAM used in the build stage) for different problems. The second is to measure the accuracy of the algorithm. Specifically, we report:

N_{tot}	Total number of Chebyshev discretization nodes.
T_{build}	Time for building the solution operator (seconds).
T_{solve}	Time to solve the equation once solution operator is built (seconds).
R	Amount of memory required at build stage to store the solution operator (GB).

For all experiments, we chose the compression parameter $\epsilon = 10^{-5}$. The code was implemented in Matlab, which means that while the timings reported should scale in a representative manner, the absolute times can probably be improved. (Additionally, memory usage is not quite optimal which means that we run out of memory sooner than what would happen in a careful implementation in Fortran or C.) Finally, due to the overhead cost of structured matrix algebra, we switch from dense matrix algebra to HBS matrix algebra only once the block sizes exceed a certain threshold. This accelerates the practical run time of the code, but does not influence the asymptotic cost estimate.

Example 8.1: Laplace problem with known exact solution We first consider the Laplace problem

$$\begin{cases} -\Delta u(\boldsymbol{x}) = 0 & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = f(\boldsymbol{x}) & \boldsymbol{x} \in \Gamma, \end{cases} \tag{27}$$

on the domain $\Omega = [0,\, 1]^3$. The number of Chebyshev nodes is fixed $p = 5$ on each panel while the number of panels on each side is increased. The boundary data is chosen to coincide with the known solution

$$u_{\text{exact}}(\boldsymbol{x}) = \frac{1}{4\pi|\boldsymbol{x} - \hat{\boldsymbol{x}}|}$$

where $\hat{\boldsymbol{x}} = (-2,\, -1,\, 0)$. To measure the accuracy, we present both the error E^{∞} in ℓ^{∞}-norm as well as the relative error E^{rel} given by

$$E^{\infty} = ||\mathbf{u}_{\text{approx}} - \mathbf{u}_{\text{exact}}||_{\ell^{\infty}} \quad \text{and} \quad E^{\text{rel}} = \frac{||\mathbf{u}_{\text{approx}} - \mathbf{u}_{\text{exact}}||_{\ell^{\infty}}}{||\mathbf{u}_{\text{exact}}||_{\ell^{\infty}}},$$

where $\mathbf{u}_{\text{exact}}$ and $\mathbf{u}_{\text{approx}}$ denote the vectors holding the exact and the computed solutions evaluated at all interior Chebyshev nodes. This is an artificial problem in the sense that it could very easily be solved to high precision using a single spectral grid. The purpose of this example is primarily to investigate the stability and scaling properties of the algorithm. Speed, memory and accuracy results are shown in Table 1. Note that the off-diagonal blocks of DtN operators at top levels are stored in low-rank form which saves roughly 1.6 times of the memory compared to dense form. If no low-rank computation is executed, the computation runs out of memory (over 128GB) when N is about $2 \cdot 10^6$. Figure 3 shows the scales of time spent at build stage and solve stage. For the very largest problem, the program started to store matrices on the hard drive, which explains the large jump in the solve time.

In the examples we could fit in RAM, very little benefit was seen for using structured matrix algebra, except at the very top levels. We would therefore not expect the measured behavior for T_{build} to reflect the asymptotic bound proven in Section 5.3.

N_{tot}	R (GB)	T_{build} (sec)	T_{solve} (sec)	E^{∞}	E^{rel}
4 913	0.04	0.97	0.004	1.20e-06	3.38e-05
35 937	0.52	20.34	0.032	1.45e-08	4.08e-07
274 625	6.33	522.78	0.24	5.48e-08	1.54e-07
2 146 689	76.59	17103.21	1121.0	6.51e-09	1.83e-07

TABLE 1. Results for solving Laplace's equation (27) in Example 8.1 with known exact solution.

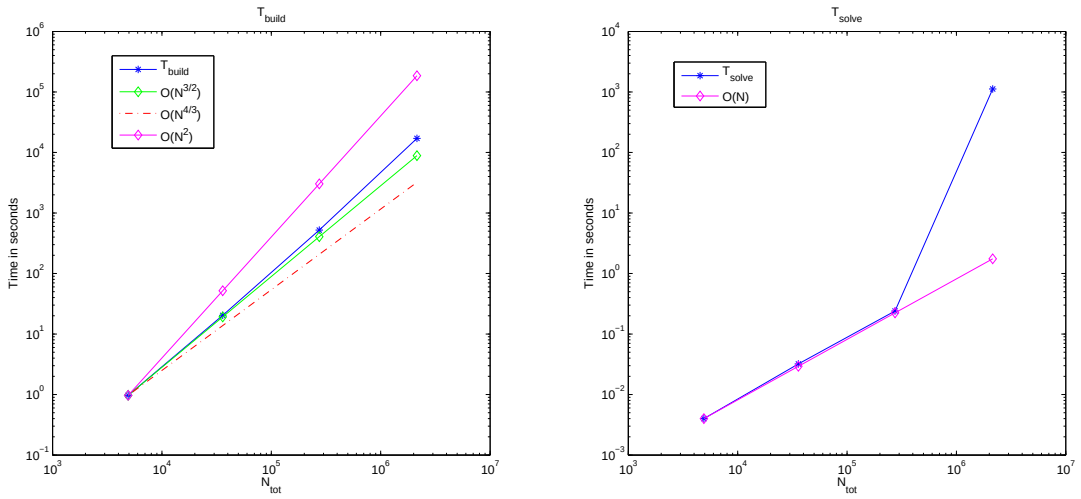
Example 8.2(a): Helmholtz problems with known exact solution and fixed number of Chebyshev nodes on each panel. We next consider a Helmholtz problem

$$\begin{cases} -\Delta u(\boldsymbol{x}) - \kappa^2 u(\boldsymbol{x}) = 0 & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = f(\boldsymbol{x}) & \boldsymbol{x} \in \Gamma, \end{cases} \tag{28}$$

on domain $\Omega = [0,\, 1]^3$. The number of Chebyshev nodes is fixed $p = 5$ on each panel the same as we did in Example 8.1. The boundary data is chosen to coincide with the known solution

$$u_{\text{exact}}(\boldsymbol{x}) = \frac{e^{i\kappa|\boldsymbol{x} - \hat{\boldsymbol{x}}|}}{4\pi|\boldsymbol{x} - \hat{\boldsymbol{x}}|}$$

where $\hat{\boldsymbol{x}} = (-2,\, -1,\, 0)$. Accuracy is measured in the same way as Example 8.1. Table 2 reports the results when $\kappa = 12.56$ such that there are $2 \times 2 \times 2$ wavelengths across the whole domain.



(a) (b)

FIGURE 3. (Example 8.1)(a) Time at build stage in seconds, (b) time at solve stage.

N_{tot}	R (GB)	T_{build} (sec)	T_{solve} (sec)	E^{∞}	E^{rel}
4 913	0.04	1.01	0.005	6.39e-03	1.70e-01
35 937	0.54	22.20	0.042	1.98e-03	5.60e-02
274 625	6.41	514.53	0.32	5.34e-04	1.53e-02
2 146 689	76.95	18605.66	1368	1.36e-04	3.81e-03

TABLE 2. Results for solving Helmholtz equation (28) in Example 8.2(a) with 2 wavelength in each direction.

Example 8.2(b): Helmholtz problems with known exact solution and fixed number of boxes on each side. In this example, we still solve the Helmholtz problem (28). We now keep the number of boxes constant, using an $8 \times 8 \times 8$ grid, and instead test what happens as the local discretization order is increased. (In other words, we do "p-refinement" rather than "h-refinement.") Note that in the following examples, we keep this problem setting in regards to what happens as N is increased. Table 3 reports the results when $\kappa = 62.8$ such that there are $10 \times 10 \times 10$ wavelengths across the whole domain. Table 4 reports an analogous experiment, but now for a domain of size $20 \times 20 \times 20$ wavelengths.

N_{tot}	R (GB)	T_{build} (sec)	T_{solve} (sec)	E^{∞}	E^{rel}
274 625	6.79	850.5	0.2	1.55e-03	4.28e-02
531 441	15.03	2427.2	0.6	6.51e-05	1.80e-03
912 673	29.51	4967.0	1.1	1.83e-06	5.06e-05
1 442 897	52.80	10133.1	2.7	3.57e-08	9.86e-07
2 146 689	89.15	19831.9	558.8	1.14e-08	3.15e-07

TABLE 3. Results for solving Helmholtz equation (28) in Example 8.2(b) with $10 \times 10 \times 10$ wavelength across the domain.

N_{tot}	R (GB)	T_{build} (sec)	T_{solve} (sec)	E^{∞}	E^{rel}
274 625	8.65	1034.3	0.2	1.34e+00	3.76e+01
531 441	18.40	2910.6	0.5	1.70e-01	4.78e+00
912 673	34.55	7573.7	1.1	7.50e-03	2.11e-01
1 442 897	59.53	14161.1	2.8	9.45e-04	2.65e-02
2 146 689	97.73	25859.3	978.7	5.26e-05	1.48e-03

TABLE 4. Results for solving Helmholtz equation (28) in Example 8.2(b) with $20 \times 20 \times 20$ wavelength across the domain.

Example 8.3: Laplace's equation with unknown exact solution. In this example, we solve the Laplace's equation (27) with Dirichlet boundary data given by

$$f(\boldsymbol{x}) = \cos(8\,x_1)(1 - 2\,x_2)\,e^{x_3}. \tag{29}$$

Since we have no knowledge of an exact solution, we report pointwise convergence. Letting u_{N_1} and u_{N_2} denote the value of u computed using N_1 and N_2 degrees of freedom where $N_2 > N_1$, we used

$$E^{\infty} = |u^{N_1}(\hat{\boldsymbol{x}}) - u^{N_2}(\hat{\boldsymbol{x}})| \quad \text{and} \quad E^{\text{rel}} = \frac{|u^{N_1}(\hat{\boldsymbol{x}}) - u^{N_2}(\hat{\boldsymbol{x}})|}{|u^{N_1}(\hat{\boldsymbol{x}})|}$$

as estimates for the pointwise errors at point $\hat{\boldsymbol{x}} = (0.5,\, 0.25,\, 0.75)$. Results are reported in Table 5.

Observe that the examples investigated in Examples 8.1 and 8.2(a/b) were slightly artificial in that the solutions are perfectly smooth even near the corners. The current example represents a more typical situation where the solution exhibits singular behavior near corners. However, as the results show, the scheme handles these relatively mild singularities very well, and converges rapidly.

N_{tot}	R (GB)	T_{build} (sec)	T_{solve} (sec)	$u_{\text{int}}(\boldsymbol{x})$	E^{∞}	E^{rel}
117 649	2.13	84.7	0.06	-0.32055891842	1.10e-06	3.44e-06
274 625	6.09	540.1	0.2	-0.32055781677	9.43e-08	2.94e-07
531 441	14.35	1517.3	0.4	-0.32055772259	2.56e-08	7.98e-08
912 673	29.11	2822.4	0.7	-0.32055769701	1.24e-07	3.87e-07
1 442 897	50.44	9130.9	1.4	-0.32055757286	7.34e-08	2.29e-07
2 146 689	86.12	18076.5	541.5	-0.32055776368		

TABLE 5. Results for Example 8.3: Solving Laplace's equation (27) with Dirichlet boundary data (29).

Example 8.4: Helmholtz equation with unknown exact solution. In this example, we solve the Helmholtz equation (28) with Dirichlet boundary data given in (29). The wavenumber is set $\kappa = 62.8$ such that there are $10 \times 10 \times 10$ wavelength across the domain. Table 6 presents the convergence as well as time and memory results. Pointwise errors are computed the same as Example 8.3.

Example 8.5: Variable coefficient Helmholtz. We solve the variable coefficient problem

$$\begin{cases} -\Delta u(\boldsymbol{x}) - \kappa^2(1 - b(\boldsymbol{x}))u(\boldsymbol{x}) = 0 & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = f(\boldsymbol{x}) & \boldsymbol{x} \in \Gamma, \end{cases} \tag{30}$$

N_{tot}	R (GB)	T_{build} (sec)	T_{solve} (sec)	$u_{\text{int}}(\boldsymbol{x})$	E^{∞}	E^{rel}
274 625	6.58	662.9	0.2	1.93279205417	1.87e+00	9.68e-01
531 441	15.13	1382.9	0.3	3.80381724805	9.55e-02	2.51e-02
912 673	30.25	2992.5	0.7	3.70818462313	2.91e-03	7.84e-04
1 442 897	55.23	7895.9	2.3	3.71109259971	6.16e-05	1.66e-05
2 146 689	89.15	21190.2	789.8	3.71103088491		

TABLE 6. Results for Example 8.4: Solving Helmholtz equation (28) with Dirichlet boundary data (29).

where $\Omega = [0,\, 1]^3$, where $\Gamma = \partial\Omega$ and where

$$b(\boldsymbol{x}) = (\sin(4\pi x_1)\, \sin(4\pi x_2)\, \sin(4\pi x_3))^2.$$

The Helmholtz parameter was chosen as $\kappa = 62.8$, corresponding to a domain of size $10 \times 10 \times 10$ wavelengths. The Dirichlet boundary data was given by in (29). Again we measure the pointwise errors since we do not know the exact solution. Results are reported in Table 7. We observe that the accuracy as almost as good as constant coefficient case.

N_{tot}	R (GB)	T_{build} (sec)	T_{solve} (sec)	$u_{\text{int}}(\boldsymbol{x})$	E^{∞}	E^{rel}
274 625	6.55	639.3	0.2	10.22765480303	6.13e-02	5.99e-03
531 441	15.15	1443.4	0.3	10.16634235402	8.69e-03	8.55e-04
912 673	30.35	3701.0	0.7	10.17503224623	4.56e-04	4.48e-05
1 442 897	55.39	5639.6	1.4	10.17548843592	1.35e-04	1.33e-05
2 146 689	89.27	20854.3	874.7	10.17535090141		

TABLE 7. Results for Example 8.5: Solving the variable coefficient problem (30).

Example 8.6: Constant convection diffusion problem. Our last example is to solve a convection diffusion problem

$$\begin{cases} -\Delta u(\boldsymbol{x}) - 1000\, \partial_3 u(\boldsymbol{x}) = 0 & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = f(\boldsymbol{x}) & \boldsymbol{x} \in \Gamma, \end{cases} \tag{31}$$

on domain $\Omega = [0,\, 1]^3$ with Dirichlet boundary data given in (29). Results are reported in Table 8. We observe that despite the very sharp gradients encountered in this example, the method is still capable of attaining more than 5 correct digits.

N_{tot}	R (GB)	T_{build} (sec)	T_{solve} (sec)	$u_{\text{int}}(\boldsymbol{x})$	E^{∞}	E^{rel}
117 649	2.41	136.0	0.08	-0.90989632585	3.33e-02	3.66e-02
274 625	6.65	524.9	0.1	-0.87662189265	2.30e-03	2.62e-02
531 441	15.33	1806.0	0.3	-0.87432365086	4.36e-05	4.99e-05
912 673	30.45	3524.9	0.7	-0.87428002798	1.23e-05	1.41e-05
1 442 897	55.35	6719.9	1.3	-0.87429233218	3.30e-06	3.77e-06
2 146 689	88.03	19313.7	656.2	-0.87429562890		

TABLE 8. Results for Example 8.6: Solving the constant convection problem (31).

9. Conclusions and Future work

We have described a numerical method for solving variable coefficient elliptic PDEs on rectangular domains in 3D. The scheme is based on a composite spectral discretization and is an extension of previous work for the two dimensional case [23, 11].

The technique is conceptually similar to the nested dissection method of George [10]. Like this classical direct solver, our solver relies on a hierarchical subdivision of the domain into a tree of nested sub-domains, in our case boxes. An approximation to the solution operator is built in a single sweep through the tree from smaller to larger boxes. In this sweep, we build for each box numerical approximations to its Dirichlet-to-Neumann (DtN) operator, and to a local "solution operator." However, in contrast to the classical nested dissection technique, our method retains high practical efficiency even for high order discretizations. We demonstrate that by exploiting that the matrix approximating each DtN operator can be represented in a "data sparse" format, the asymptotic complexity of the solver can be accelerated from $O(N^2)$ for the build stage to, at worst, $O(N^{4/3})$. Our work is in this regard related to recent work on accelerating nested dissection methods [19, 26, 31, 13, 22] but gives much higher accuracy for the same number of degrees of freedom, due to the use of high-order discretizations. Specifically, we solved a Helmholtz problem on a domain of size $20 \times 20 \times 20$ wave-lengths to three correct digits using 2.1M degrees of freedom, cf. Table 4.

A key advantage of the method proposed is that it involves less communication than competing iterative methods. Each solve involves only one or two sweeps through the hierarchical tree, which keeps computations highly localized. The development of parallel implementations of this algorithm is a work in progress, and will be reported at a later date.

The fact that the method can without difficulty handle high order local discretizations makes it particularly well suited for solving Helmholtz problems with oscillatory solutions, as was demonstrated in the numerical experiments in Section 8. It is hard to make a general recommendation for an "optimal choice" of the local discretization order — choosing a higher order p leads to needing fewer degrees of freedom overall, but on the other hand increases the storage requirements per degree of freedom. Moreover, the fact that the cost of the leaf computations scales as $N\,p^6$ puts a practical limit on how large p can realistically be. We have found that p in the range between 10 and 15 often works very well.

In this manuscript, we describe a basic version of the method applicable to simple rectangular domains. The extension to more general domains, including curved and infinite domains, is in principle straight-forward, as was demonstrated for problems in two dimensions in [24, 23, 18].

The scheme can easily be extended to cover the case of "body loads" (that is, the case $[Au](\boldsymbol{x}) = g(\boldsymbol{x})$ rather than $[Au](\boldsymbol{x}) = 0$). The corresponding extension for the 2D case was described in [24, 1]. The more general version that includes body loads has the same asymptotic scaling as the method reported here. In terms of absolute times, the cost of the build stage increases only very slightly. For the solve stage, one must decide whether the solution operators on the leaves should be stored or not. Storing them leads to very fast solves, but requires a lot of memory. If they are not stored, then memory requirements are similar to the scheme reported here, but then the solve stage becomes noticeably slower since the PDE needs to be solved on the fly on each leaf box.

Acknowledgements: The research reported was supported by DARPA, under the contract N66001-13-1-4050, and by the NSF, under the contract DMS-1407340. We also gratefully acknowledge very helpful suggestions by the anonymous referees.

References

[1] T. Babb, A. Gillman, S. Hao, and P.G. Martinsson, *An accelerated Poisson solver based on multidomain spectral discretization*, 2016, In review.

[2] Mario Bebendorf, *Hierarchical matrices*, Lecture Notes in Computational Science and Engineering, vol. 63, Springer-Verlag, Berlin, 2008, A means to efficiently solve elliptic boundary value problems. MR 2451321 (2009k:15001)

[3] Steffen Börm, *Efficient numerical methods for non-local operators*, EMS Tracts in Mathematics, vol. 14, European Mathematical Society (EMS), Zürich, 2010, $\mathcal{H}^2$-matrix compression, algorithms and analysis. MR 2767920

[4] S. Chandrasekaran, M. Gu, X.S. Li, and J. Xia, *Superfast multifrontal method for structured linear systems of equations*, SIAM J. Matrix Anal. Appl. **31** (2009), 1382 – 1411.

[5] Shiv Chandrasekaran, Patrick Dewilde, Ming Gu, and Naveen Somasunderam, *On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs*, SIAM Journal on Matrix Analysis and Applications **31** (2010), no. 5, 2261–2290.

[6] Shiv Chandrasekaran and Ming Gu, *A fast and stable solver for recursively semi-separable systems of equations*, Structured Matrices in Mathematics, Computer Science, and Engineering: Proceedings of an AMS-IMS-SIAM Joint Summer Research Conference, University of Colorado, Boulder, June 27–July 1, 1999 (Providence, RI) (Vadim Olshevsky, ed.), Contemporary Mathematics, vol. II, AMS Publications, 2001.

[7] H. Cheng, Z. Gimbutas, P.G. Martinsson, and V. Rokhlin, *On the compression of low rank matrices*, SIAM Journal of Scientific Computing **26** (2005), no. 4, 1389–1404.

[8] Timothy A Davis, *Direct methods for sparse linear systems*, vol. 2, Siam, 2006.

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

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

[11] A. Gillman and P. G. Martinsson, *A direct solver with $O(N)$ complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method*, SIAM Journal on Scientific Computing **36** (2014), no. 4, A2023–A2046.

[12] Adrianna Gillman, AlexH. Barnett, and Per-Gunnar Martinsson, *A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media*, BIT Numerical Mathematics **55** (2015), no. 1, 141–170 (English).

[13] Adrianna Gillman and Per-Gunnar Martinsson, *An $O(N)$ algorithm for constructing the solution operator to 2D elliptic boundary value problems in the absence of body loads*, Advances in Computational Mathematics **40** (2014), no. 4, 773–796 (English).

[14] Adrianna Gillman, Patrick Young, and Per-Gunnar Martinsson, *A direct solver $O(N)$ complexity for integral equations on one-dimensional domains*, Frontiers of Mathematics in China **7** (2012), 217–247, 10.1007/s11464-012-0188-3.

[15] Lars Grasedyck and Wolfgang Hackbusch, *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing **70** (2003), no. 4, 295–334.

[16] Wolfgang Hackbusch, *A sparse matrix arithmetic based on H-matrices; Part I: Introduction to H-matrices*, Computing **62** (1999), 89–108.

[17] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev. **53** (2011), no. 2, 217–288.

[18] T.S. Haut, T. Babb, P.G. Martinsson, and B.A. Wingate, *A high-order scheme for solving wave propagation problems via the direct construction of an approximate time-evolution operator*, IMA Journal of Numerical Analysis (2015), In press.

[19] Sabine Le Borne, Lars Grasedyck, and Ronald Kriemann, *Domain-decomposition based H-LU preconditioners*, Domain decomposition methods in science and engineering XVI, Springer, 2007, pp. 667–674.

[20] Edo Liberty, Franco Woolfe, P.G. Martinsson, Vladimir Rokhlin, and Mark Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA **104** (2007), no. 51, 20167–20172. MR MR2366406

[21] P. G. Martinsson and S. Voronin, *A CUR factorization algorithm based on the interpolative decomposition*, 2015, Arxiv.org report #1412.8447.

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

[23] ———, *A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method*, J. Comput. Phys. **242** (2013), 460–479.

[24] P.G. Martinsson, *The Hierarchical Poincare-Steklov (HPS) solver for elliptic PDEs: A tutorial*, 2015, arXiv preprint arXiv:1506.01308.

[25] P.G. Schmitz and L. Ying, *A fast direct solver for elliptic problems on cartesian meshes in 3D*, 2011, UT-Austin tech report.
[26] Phillip G. Schmitz and Lexing Ying, *A fast direct solver for elliptic problems on general meshes in 2D*, Journal of Computational Physics **231** (2012), no. 4, 1314 – 1338.
[27] L.N. Trefethen, *Spectral methods in matlab*, SIAM, Philadelphia, 2000.
[28] J. Xia, S. Chandrasekaran, M. Gu, and X.S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numerical Linear Algebra with Applications **17** (2010), no. 6, 953–976.
[29] Jianlin Xia, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM Journal on Scientific Computing **35** (2013), no. 2, A832–A860.
[30] Jianlin Xia, *Randomized sparse direct solvers*, SIAM Journal on Matrix Analysis and Applications **34** (2013), no. 1, 197–227.
[31] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S Li, *Fast algorithms for hierarchically semiseparable matrices*, Numerical Linear Algebra with Applications **17** (2010), no. 6, 953–976.