BIT Numerical Mathematics (2018) 58:851–879
https://doi.org/10.1007/s10543-018-0714-0

BIT

An accelerated Poisson solver based on multidomain spectral discretization

Tracy Babb[1] · **Adrianna Gillman**[2] · **Sijia Hao**[1] · **Per-Gunnar Martinsson**[3]

Received: 27 March 2017 / Accepted: 8 June 2018 / Published online: 5 July 2018

Abstract
This paper presents a numerical method for variable coefficient elliptic PDEs with mostly smooth solutions on two dimensional domains. The method works best for domains that can readily be mapped onto a rectangle, or a collection of nonoverlapping rectangles. The PDE is discretized via a multi-domain spectral collocation method of high local order (order 30 and higher have been tested and work well). Local mesh refinement results in highly accurate solutions even in the presence of local irregular behavior due to corner singularities, localized loads, etc. The system of linear equations attained upon discretization is solved using a direct (as opposed to iterative) solver with $O(N^{1.5})$ complexity for the factorization stage and $O(N \log N)$ complexity for the solve. The scheme is ideally suited for executing the elliptic solve required when parabolic problems are discretized via time-implicit techniques. In situations where the geometry remains unchanged between time-steps, very fast execution speeds are obtained since the solution operator for each implicit solve can be pre-computed.

Keywords Direct solver · High-order discretization · Multi-domain spectral method · Nested dissection · Reduction to interface · Implicit time stepping · Local refinement

Mathematics Subject Classification 65M70

Communicated by Elisabeth Larsson.

✉ Per-Gunnar Martinsson
martinsson@maths.ox.ac.uk

1 Department of Applied Mathematics, University of Colorado at Boulder, Boulder, CO, USA

2 Department of Computational and Applied Mathematics, Rice University, Houston, TX, USA

3 Mathematical Institute, University of Oxford, Oxford, UK

1 Introduction

This manuscript describes a direct solver for elliptic PDEs such as, e.g.,

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

where A is a variable coefficient elliptic differential operator

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

where Ω is a rectangular domain in $\mathbb{R}^2$ with boundary $\Gamma = \partial\Omega$, where all coefficient functions (c, c_i, c_{ij}) are smooth, and where f and g are given functions. The generalization to domains that are either unions of rectangles, or can via local parameterizations be mapped to a union of rectangles is relatively straight-forward [11, Sec. 6.4]. The technique is specifically developed to accelerate implicit time stepping techniques for parabolic PDEs such as, e.g., the heat equation

$$\begin{cases} \Delta u(\boldsymbol{x}, t) = \dfrac{\partial u}{\partial t}(\boldsymbol{x}, t), & \boldsymbol{x} \in \Omega,\ t > 0, \\ u(\boldsymbol{x}, t) = f(\boldsymbol{x}, t), & \boldsymbol{x} \in \Gamma,\ t > 0, \\ u(\boldsymbol{x}, 0) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega. \end{cases} \tag{3}$$

When (3) is discretized using an implicit time-stepping scheme (e.g. backwards Euler, Crank–Nicolson or the Transpose Method of Lines [1]), one is required to solve for each time-step an equation of the form (1), see Sect. 7.6. With the ability to combine very high order discretizations with a highly efficient means of time-stepping parabolic equations, we believe that the proposed method will be particularly well suited for numerically solving the Navier–Stokes equation at low Reynolds numbers.

The proposed solver is direct and builds an approximation to the solution operator of (1) via a hierarchical divide-and-conquer approach. It is conceptually related to classical nested dissection and multifrontal methods [2–4], but provides tight integration between the direct solver and the discretization procedure. Locally, the scheme relies on high order spectral discretizations, and collocation of the differential operator. We observe that while classical nested dissection and multifrontal solvers slow down dramatically as the discretization order is increased [6, Table 3], the proposed method retains high efficiency regardless of the discretization order. The method is an evolution of the scheme described in [10,11], and later refined in [5–7]. One novelty of the present work is that it describes how problems with body loads can be handled efficiently (the previous papers [5–7,11] consider the case where $g = 0$ in (1)). A second novelty is that local mesh refinement is introduced to enable the method to accurately solve problems involving concentrated loads, singularities at re-entrant corners, and other phenomena that lead to localized loss of regularity in the solution (in contrast, the previous papers [5–7,11] restrict attention to uniform grids).

The principal advantage of the proposed solver, compared to commonly used solvers for (1), is that it is *direct* (as opposed to *iterative*), which makes it particularly well suited for problems for which efficient pre-conditioners are difficult to find, such as, e.g., problems with oscillatory solutions. The cost to build the solution operator is in the most basic version of the scheme $O(N^{3/2})$, where N is the number of discretization points. However, the practical efficiency of the solver is very high and the superlinear scaling is hardly visible until $N > 10^7$. When the number of discretization points is higher than 10^7, the scheme can be modified to attain linear complexity by implementing techniques analogous to those described in [5]. Once the solution operator has been built, the time required to apply it to execute a solve given a boundary condition and a body load is either $O(N \log N)$ for the basic scheme, or $O(N)$ for the accelerated scheme, with a small scaling constant in either case. In Sect. 7, we demonstrate that even when $N = 10^6$, the time for solving (1) with a precomputed solution operator is approximately 1 s on a standard office laptop.

The discretization scheme we use is related to earlier work on spectral collocation methods on composite ("multi-domain") grids, such as, e.g., [9,14], and in particular Pfeiffer et al. [12]. The differences and similarities between the various techniques is discussed in detail in [11]. Our procedure is also conceptually related to so-called "reduction to the interface" methods, see [8] and the references therein. Such "interface" methods also use local solution operators defined on boundaries but typically rely on variational formulations of the PDE, rather than the collocation techniques that we employ.

The manuscript is organized as follows: Sect. 2 provides a high level description of the proposed method. Sections 3 and 4 describe the local discretization scheme. Section 5 describes the nested dissection type solver used to solve the system of linear equations resulting from the discretization. Section 6 describes how local mesh refinement can be introduced to the scheme. Section 7 provides results from numerical experiments that establish the efficiency of the proposed method.

2 Overview of algorithm

The proposed method is based on a hierarchical subdivision of the computational domain, as illustrated in Fig. 1 for the case of $\Omega = [0, 1]^2$. In the uniform mesh version of the solver, the tree of boxes is built by recursively splitting the original box in halves. 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 $p \times p$ Chebyshev nodes (where, say, $p = 10$ or $p = 20$).

Once the tree of boxes has been constructed, the actual solver consists of two stages. The first, or "build", stage consists of a single upwards pass through the tree of boxes, starting with the leaves and going up to larger boxes. On each leaf, we place a local $p \times p$ tensor product grid of Chebyshev nodes, and then discretize the restriction of (1) via a basic collocation scheme, as in [13]. By performing dense linear algebraic operations on matrices of size at most $p^2 \times p^2$, we form for each leaf a local solution operator and an approximation to the local Dirichlet-to-Neumann (DtN) operator, as described in Sect. 3. The build stage then continues with an upwards pass through

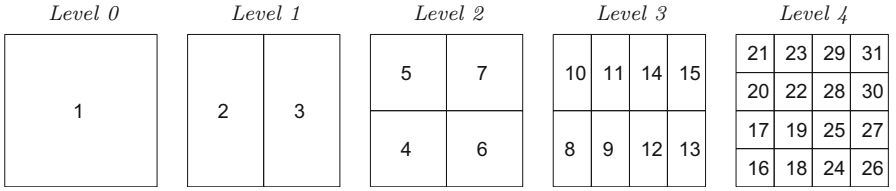

Fig. 1 The square domain Ω is split into 4×4 leaf boxes. These are then gathered into a binary tree of successively larger boxes as described in Sect. 2. One possible enumeration of the boxes in the tree is shown, but note that the only restriction is that if box τ is the parent of box σ, then $\tau < \sigma$

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, cf. Sect. 4. 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 a body load g, and constructs an approximation to the solution u valid throughout the domain via two passes through the tree: first an upwards pass (going from smaller boxes to larger) where "particular solutions" that satisfy the inhomogeneous equation are built, and then a downwards pass where the boundary conditions are corrected.

The global grid of collocation points used in the upwards and downwards passes is obtained by placing on the edge of each leaf a set of q Gaussian interpolation nodes (a.k.a. Legendre nodes). Observe that this parameter q is in principle distinct from the local parameter p which specifies the order of the local Chebyshev grids used to construct the solution operators on the leaves. However, we typically choose $p = q+1$ or $p = q + 2$.

3 Leaf computation

In this section, we describe how to numerically build the various linear operators (represented as dense matrices) needed for a given leaf Ω_τ in the hierarchical tree. To be precise, let u be the solution to the local equation

$$\begin{cases} [Au](\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega_\tau, \\ u(\boldsymbol{x}) = d(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma_\tau, \end{cases} \tag{4}$$

for some given (local) Dirichlet data d. We then build approximations to two linear operators that both take d and g as their inputs. The first operator outputs the local solution u on Ω_τ and the second outputs the boundary fluxes of u on Γ_τ.

3.1 Notation

We use two sets of interpolation nodes on the domain Ω_τ. First, let $\{\boldsymbol{y}_j\}_{j=1}^{4q}$ denote the nodes obtained by placing q Gaussian nodes on each of the four sides of Ω_τ. Next, let $\{\boldsymbol{x}_i\}_{i=1}^{p^2}$ denote the nodes in a $p \times p$ Chebyshev grid on Ω_τ. We partition the index

vector for the nodes in the Chebyshev grid as

$$\{1, 2, \ldots, p^2\} = I_{\text{ce}} \cup I_{\text{ci}}$$

so that I_{ce} holds the (Chebyshev) exterior nodes and I_{ci} holds the (Chebyshev) interior nodes. Let u_{c}, u_{ci}, u_{ce}, and u_{ge} denote vectors holding approximations to the values of the solution u at the interpolation nodes:

$$\begin{aligned} \mathsf{u}_{\text{c}} &\approx \{u(\boldsymbol{x}_i)\}_{i=1}^{p^2}, \quad \mathsf{u}_{\text{ci}} \approx \{u(\boldsymbol{x}_i)\}_{i\in I_{\text{ci}}}, \\ \mathsf{u}_{\text{ce}} &\approx \{u(\boldsymbol{x}_i)\}_{i\in I_{\text{ce}}}, \quad \mathsf{u}_{\text{ge}} \approx \{u(\boldsymbol{y}_j)\}_{j=1}^{4q}. \end{aligned}$$

Let $\mathsf{v}_{\text{ge}} \in \mathbb{R}^{4q}$ denote a vector holding boundary fluxes of u on the Gaussian grid, so that

$$\begin{aligned} \mathsf{v}_{\text{ge}}(j) &\approx [\partial_1 u](\boldsymbol{y}_{hj}) \quad \text{when } \boldsymbol{y}_j \text{ lies on a vertical boundary,} \\ \mathsf{v}_{\text{ge}}(j) &\approx [\partial_2 u](\boldsymbol{y}_{hj}) \quad \text{when } \boldsymbol{y}_j \text{ lies on a horizontal boundary.} \end{aligned}$$

The sign convention for boundary fluxes means that a positive flux sometimes represents flow into the box and sometimes out of the box. Finally, let d_{ge} and g_{ci} denote tabulations of the boundary data and the body load,

$$\mathsf{d}_{\text{ge}} = \{d(\boldsymbol{y}_j)\}_{j=1}^{4q}, \quad \mathsf{g}_{\text{ci}} = \{g(\boldsymbol{x}_i)\}_{i\in I_{\text{ci}}}.$$

Our objective is now to build the matrices that map $\{\mathsf{d}_{\text{ge}}, \mathsf{g}_{\text{ci}}\}$ to v_{ge} and u_{c}.

3.2 Discretization on the Cheyshev grid

In order to execute the local solve on Ω_τ of (4), we use a classical spectral collocation technique, as described, e.g., in [13]. To this end, let $\mathsf{D}^{(1)}$ and $\mathsf{D}^{(2)}$ denote the $p^2 \times p^2$ spectral differentiation matrices on the $p \times p$ Chebyshev grid (in other words, for any function u that is a tensor product of polynomials of degree at most $p-1$, the differentiation matrix *exactly* maps a vector of collocated function values to the vector of collocated values of its derivative). Further, let A denote the matrix

$$\mathsf{A} = -\mathsf{C}_{11}\left(\mathsf{D}^{(1)}\right)^2 - 2\mathsf{C}_{12}\mathsf{D}^{(1)}\mathsf{D}^{(2)} - \mathsf{C}_{22}\left(\mathsf{D}^{(2)}\right)^2 + \mathsf{C}_1\mathsf{D}^{(1)} + \mathsf{C}_2\mathsf{D}^{(2)} + \mathsf{C},$$

where C_{ij} are diagonal matrices with entries $\{c_{ij}(\boldsymbol{x}_k)\}_{k=1}^{p^2}$, and C_i and C are defined analogously. Next, partition the matrix A to separate interior and exterior nodes via

$$\mathsf{A}_{\text{ci,ci}} = \mathsf{A}(I_{\text{ci}}, I_{\text{ci}}), \quad \text{and} \quad \mathsf{A}_{\text{ci,ce}} = \mathsf{A}(I_{\text{ci}}, I_{\text{ce}}).$$

Collocating (4) at the interior nodes then results in the discretized equation

$$\mathsf{A}_{\text{ci,ci}}\,\mathsf{u}_{\text{ci}} + \mathsf{A}_{\text{ci,ce}}\,\mathsf{d}_{\text{ce}} = \mathsf{g}_{\text{ci}}, \tag{5}$$

where $\mathsf{d}_{\mathrm{ce}} = \{d(\boldsymbol{x}_i)\}_{i \in I_{\mathrm{ce}}}$ encodes the local Dirichlet data d.

3.3 Solving on the Chebyshev grid

While solving (5) gives the solution at the interior Chebychev nodes, it does not give a map to the boundary fluxes v_{ge} that we seek. These are found by following the classic approach of writing the solution as the superposition of the homogeneous and particular solutions. Specifically, the solution to (4) is split as

$$u = w + \phi$$

where w is a *particular solution*

$$\begin{cases} Aw(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega_\tau, \\ w(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Gamma_\tau, \end{cases} \tag{6}$$

and where ϕ is a *homogeneous solution*

$$\begin{cases} A\phi(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Omega_\tau, \\ \phi(\boldsymbol{x}) = d(\boldsymbol{x}), & \boldsymbol{x} \in \Gamma_\tau. \end{cases} \tag{7}$$

Discretizing (6) on the Chebyshev grid, and collocating at the internal nodes, we get the equation

$$\mathsf{A}_{\mathrm{ci,ce}}\mathsf{w}_{\mathrm{ce}} + \mathsf{A}_{\mathrm{ci,ci}}\mathsf{w}_{\mathrm{ci}} = \mathsf{g}_{\mathrm{ci}}.$$

Observing that $\mathsf{w}_{\mathrm{ce}} = 0$, the particular solution is given by

$$\mathsf{w}_{\mathrm{c}} = \begin{bmatrix} \mathsf{w}_{\mathrm{ce}} \\ \mathsf{w}_{\mathrm{ci}} \end{bmatrix} = \mathsf{F}_{\mathrm{c,ci}}\mathsf{g}_{\mathrm{ci}}, \quad \text{where} \quad \mathsf{F}_{\mathrm{c,ci}} = \begin{bmatrix} 0 \\ \mathsf{A}_{\mathrm{ci,ci}}^{-1} \end{bmatrix}. \tag{8}$$

Analogously, the discretization of (7) on the Chebyshev grid yields

$$\mathsf{A}_{\mathrm{ci,ce}}\phi_{\mathrm{ce}} + \mathsf{A}_{\mathrm{ci,ci}}\phi_{\mathrm{ci}} = 0.$$

Since $\phi_{\mathrm{ce}} = \mathsf{d}_{\mathrm{ce}}$, the homogeneous solution is given by

$$\phi_{\mathrm{c}} = \begin{bmatrix} \phi_{\mathrm{ce}} \\ \phi_{\mathrm{ci}} \end{bmatrix} = \begin{bmatrix} \mathsf{I} \\ -\mathsf{A}_{\mathrm{ci,ci}}^{-1}\mathsf{A}_{\mathrm{ci,ce}} \end{bmatrix} \mathsf{d}_{\mathrm{ce}}. \tag{9}$$

3.4 Interpolation and differentiation

Section 3.3 describes how to locally solve the BVP (4) on the Chebyshev grid via the superposition of the homogeneous and particular solutions. This computation assumes

that the local Dirichlet data d is given on the *Chebyshev* exterior nodes. In reality, this data will be provided on the Gaussian nodes, and we therefore need to introduce an interpolation operator that moves data between the different grids. To be precise, let $\mathsf{L}_{\mathrm{ce,ge}}$ denote a matrix of size $4(p-1) \times 4q$ that maps a given data vector d_{ge} to a different vector

$$\underset{4(p-1)\times 1}{\mathsf{d}_{\mathrm{ce}}} = \underset{4(p-1)\times 4q}{\mathsf{L}_{\mathrm{ce,ge}}} \underset{4q\times 1}{\mathsf{d}_{\mathrm{ge}}} \tag{10}$$

as follows: An entry of d_{ce} corresponding to an interior node is defined via a standard interpolation from the Gaussian to the Chebyshev nodes on the local edge. An entry of d_{ce} corresponding to a corner node is defined as the average value of the two extrapolated values from the Gaussian nodes on the two edges connecting to the corner (observe that except for the four rows corresponding to the corner nodes, $\mathsf{L}_{\mathrm{ce,ge}}$ is a 4×4 block diagonal matrix).

Combining (8), (9), and (10), the solution to (4) on the Chebyshev grid is

$$\mathsf{u}_{\mathrm{c}} = \mathsf{w}_{\mathrm{c}} + \phi_{\mathrm{c}} = \mathsf{F}_{\mathrm{c,ci}}\, \mathsf{g}_{\mathrm{ci}} + \mathsf{S}_{\mathrm{c,ge}}\, \mathsf{d}_{\mathrm{ge}}, \quad \text{where} \quad \mathsf{S}_{\mathrm{c,ge}} := \begin{bmatrix} \mathsf{I}_{\mathrm{ce,ce}} \\ -\mathsf{A}_{\mathrm{ci,ci}}^{-1}\mathsf{A}_{\mathrm{ci,ce}} \end{bmatrix} \mathsf{L}_{\mathrm{ce,ge}}. \tag{11}$$

All that remains is to determine the vector v_{ge} of boundary fluxes on the Gaussian nodes. To this end, let us define a combined interpolation and differentiation matrix $\mathsf{D}_{\mathrm{ge,c}}$ of size $4q \times p^2$ via

$$\mathsf{D}_{\mathrm{ge,c}} = \begin{bmatrix} \mathsf{L}_{\mathrm{loc}}\, \mathsf{D}_2(I_{\mathrm{s}}, :) \\ \mathsf{L}_{\mathrm{loc}}\, \mathsf{D}_1(I_{\mathrm{e}}, :) \\ \mathsf{L}_{\mathrm{loc}}\, \mathsf{D}_2(I_{\mathrm{n}}, :) \\ \mathsf{L}_{\mathrm{loc}}\, \mathsf{D}_1(I_{\mathrm{w}}, :) \end{bmatrix},$$

where $\mathsf{L}_{\mathrm{loc}}$ is a $q \times p$ interpolation matrix from a set of p Chebyshev nodes to a set of q Gaussian nodes, and where I_{s}, I_{e}, I_{n}, I_{w} are vectors of length p with entries corresponding to the points on the south, east, north, and west sides of the exterior nodes in the Chebyshev grid. By differentiating the local solution on the Chebyshev grid defined by (11), the boundary fluxes v_{ge} are given by

$$\mathsf{v}_{\mathrm{ge}} = \mathsf{H}_{\mathrm{ge,ci}}\, \mathsf{g}_{\mathrm{ci}} + \mathsf{T}_{\mathrm{ge,ge}}\, \mathsf{d}_{\mathrm{ge}}, \tag{12}$$

where

$$\mathsf{H}_{\mathrm{ge,ci}} = \mathsf{D}_{\mathrm{ge,c}}\, \mathsf{F}_{\mathrm{c,ci}} \quad \text{and} \quad \mathsf{T}_{\mathrm{ge,ge}} = \mathsf{D}_{\mathrm{ge,c}}\, \mathsf{S}_{\mathrm{c,ge}}. \tag{13}$$

4 Merging two leaves

Consider a rectangular box τ consisting of two leaf boxes α and β, and suppose that all local operators for α and β defined in Sect. 3 have been computed. Our objective is now to construct the Dirichlet-to-Neumann operator for τ from the local operators for its children. In this operation, only sets of Gaussian nodes on the boundaries will take part, cf. Fig. 2. We group these nodes into three sets, indexed by vectors J_1, J_2, and J_3, defined as follows:

J_1 Edge nodes of box α that are not shared with box β.
J_2 Edge nodes of box β that are not shared with box α.
J_3 Edge nodes that line the interior edge shared by α and β.

We also define

$$J^\tau_{\mathrm{ge}} = J_1 \cup J_2 \quad \text{and} \quad J^\tau_{\mathrm{gi}} = J_3$$

as the exterior and interior nodes for the parent box τ. Finally, we let $\mathsf{h}^\alpha, \mathsf{h}^\beta \in \mathbb{R}^{4q}$ denote two vectors that hold the boundary fluxes for the two local particular solutions w^α and w^β, cf. (12),

$$\mathsf{h}^\alpha_{\mathrm{ge}} = \mathsf{H}^\alpha_{\mathrm{ge,ci}}\, \mathsf{g}^\alpha_{\mathrm{ci}}, \quad \text{and} \quad \mathsf{h}^\beta_{\mathrm{ge}} = \mathsf{H}^\beta_{\mathrm{ge,ci}}\, \mathsf{g}^\beta_{\mathrm{ci}}. \tag{14}$$

Then the equilibrium equations for each of the two leaves can be written

$$\mathsf{v}^\alpha_{\mathrm{ge}} = \mathsf{T}^\alpha_{\mathrm{ge,ge}}\, \mathsf{u}^\alpha_{\mathrm{ge}} + \mathsf{h}^\alpha_{\mathrm{ge}}, \quad \text{and} \quad \mathsf{v}^\beta_{\mathrm{ge}} = \mathsf{T}^\beta_{\mathrm{ge,ge}}\, \mathsf{u}^\beta_{\mathrm{ge}} + \mathsf{h}^\beta_{\mathrm{ge}}. \tag{15}$$

Now partition the two equations in (15) using the notation shown in Fig. 2:

$$\begin{bmatrix} \mathsf{v}_1 \\ \mathsf{v}_3 \end{bmatrix} = \begin{bmatrix} \mathsf{T}^\alpha_{1,1} & \mathsf{T}^\alpha_{1,3} \\ \mathsf{T}^\alpha_{3,1} & \mathsf{T}^\alpha_{3,3} \end{bmatrix} \begin{bmatrix} \mathsf{u}_1 \\ \mathsf{u}_3 \end{bmatrix} + \begin{bmatrix} \mathsf{h}^\alpha_1 \\ \mathsf{h}^\alpha_3 \end{bmatrix}, \tag{16}$$

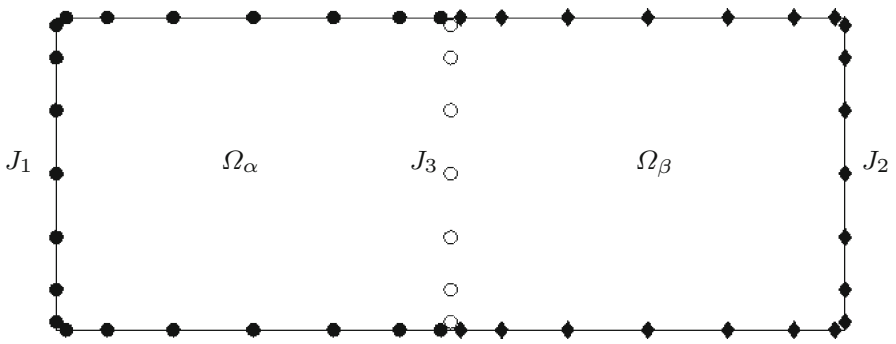

Fig. 2 Notation for the merge operation described in Sect. 4. Given two leaf boxes Ω_α and Ω_β, their union is denoted $\Omega_\tau = \Omega_\alpha \cup \Omega_\beta$. The sets J_1 (black circles) and J_2 (black diamonds) form the exterior nodes, while J_3 (white circles) consists of the interior nodes

$$\begin{bmatrix} v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} T^{\beta}_{2,2} & T^{\beta}_{2,3} \\ T^{\beta}_{3,2} & T^{\beta}_{3,3} \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \end{bmatrix} + \begin{bmatrix} h^{\beta}_2 \\ h^{\beta}_3 \end{bmatrix}. \tag{17}$$

[the subscript "ge" is suppressed in (16) and (17) since all nodes involved are Gaussian exterior nodes]. Combine the two equations for v_3 in (16) and (17) to obtain the equation

$$T^{\alpha}_{3,1}\, u_1 + T^{\alpha}_{3,3}\, u_3 + h^{\alpha}_3 = T^{\beta}_{3,2}\, u_2 + T^{\beta}_{3,3}\, u_3 + h^{\beta}_3.$$

This gives

$$u_3 = \left(T^{\alpha}_{3,3} - T^{\beta}_{3,3}\right)^{-1} \left(T^{\beta}_{3,2} u_2 - T^{\alpha}_{3,1} u_1 + h^{\beta}_3 - h^{\alpha}_3\right) \tag{18}$$

Using the relation (18) in combination with (16), we find that

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \left(\begin{bmatrix} T^{\alpha}_{1,1} & 0 \\ 0 & T^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} T^{\alpha}_{1,3} \\ T^{\beta}_{2,3} \end{bmatrix} \left(T^{\alpha}_{3,3} - T^{\beta}_{3,3}\right)^{-1} \left[-T^{\alpha}_{3,1} \mid T^{\beta}_{3,2} \right] . \right) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} h^{\alpha}_1 \\ h^{\beta}_2 \end{bmatrix} + \begin{bmatrix} T^{\alpha}_{1,3} \\ T^{\beta}_{2,3} \end{bmatrix} \left(T^{\alpha}_{3,3} - T^{\beta}_{3,3}\right)^{-1} \left(h^{\beta}_3 - h^{\alpha}_3\right).$$

We now define the operators

$$\begin{aligned} X^{\tau}_{\mathrm{gi,gi}} &= \left(T^{\alpha}_{3,3} - T^{\beta}_{3,3}\right)^{-1}, \\ S^{\tau}_{\mathrm{gi,ge}} &= \left(T^{\alpha}_{3,3} - T^{\beta}_{3,3}\right)^{-1} \left[-T^{\alpha}_{3,1} \mid T^{\beta}_{3,2} \right] = X^{\tau}_{\mathrm{gi,gi}} \left[-T^{\alpha}_{3,1} \mid T^{\beta}_{3,2} \right], \\ T^{\tau}_{\mathrm{ge,ge}} &= \begin{bmatrix} T^{\alpha}_{1,1} & 0 \\ 0 & T^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} T^{\alpha}_{1,3} \\ T^{\beta}_{2,3} \end{bmatrix} \left(T^{\alpha}_{3,3} - T^{\beta}_{3,3}\right)^{-1} \left[-T^{\alpha}_{3,1} \mid T^{\beta}_{3,2} \right] \\ &= \begin{bmatrix} T^{\alpha}_{1,1} & 0 \\ 0 & T^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} T^{\alpha}_{1,3} \\ T^{\beta}_{2,3} \end{bmatrix} S^{\tau}_{\mathrm{gi,ge}}. \end{aligned}$$

The approximate solution on the shared edge u^{τ}_{gi} can be constructed via an upward pass to compute the approximate boundary flux by

$$h^{\tau}_{\mathrm{ge}} = \begin{bmatrix} h^{\alpha}_1 \\ h^{\beta}_2 \end{bmatrix} + \begin{bmatrix} T^{\alpha}_{1,3} \\ T^{\beta}_{2,3} \end{bmatrix} w^{\tau}_{\mathrm{gi}}, \tag{19}$$

where $w^{\tau}_{\mathrm{gi}} = X^{\tau}_{\mathrm{gi,gi}}\left(h^{\beta}_3 - h^{\alpha}_3\right)$, followed by a downward pass

$$u^{\tau}_{\mathrm{gi}} = S_{\mathrm{gi,ge}} u^{\tau}_{\mathrm{ge}} + w^{\tau}_{\mathrm{gi}}.$$

Remark 1 (Physical interpretation of merge) The quantities $\mathsf{w}_{\text{gi}}^{\tau}$ and $\mathsf{h}_{\text{ge}}^{\tau}$ have a simple physical meaning. The vector $\mathsf{w}_{\text{gi}}^{\tau}$ introduced above is simply a tabulation of the particular solution w^{τ} associated with τ on the interior boundary Γ_3, and $\mathsf{h}_{\text{ge}}^{\tau}$ is the normal derivative of w^{τ}. To be precise, w^{τ} is the solution to the inhomogeneous problem, cf. (6)

$$\begin{cases} Aw^{\tau}(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega_{\tau}, \\ w^{\tau}(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Gamma_{\tau}. \end{cases} \tag{20}$$

We can re-derive the formula for $w|_{\Gamma_3}$ using the original mathematical operators as follows: First observe that for $\boldsymbol{x} \in \Omega^{\alpha}$, we have $A(w^{\tau} - w^{\alpha}) = g - g = 0$, so the DtN operator T^{α} applies to the function $w^{\tau} - w^{\alpha}$:

$$T_{31}^{\alpha}\left(w_1^{\tau} - w_1^{\alpha}\right) + T_{33}^{\alpha}\left(w_3^{\tau} - w_3^{\alpha}\right) = \left(\partial_n w^{\tau}\right)|_3 - \left(\partial_n w^{\alpha}\right)|_3$$

Use that $w_1^{\tau} = w_1^{\alpha} = w_3^{\alpha} = 0$, and that $(\partial_n w^{\alpha})|_3 = h_3^{\alpha}$ to get

$$T_{33}^{\alpha} w_3^{\tau} = \left(\partial_n w^{\tau}\right)|_3 - h_3^{\alpha}. \tag{21}$$

Analogously, we get

$$T_{33}^{\beta} w_3^{\tau} = \left(\partial_n w^{\tau}\right)|_3 - h_3^{\beta}. \tag{22}$$

Combine (21) and (22) to eliminate $(\partial_n w^{\tau})|_3$ and obtain

$$\left(T_{33}^{\alpha} - T_{33}^{\beta}\right) w_3^{\tau} = -h_3^{\alpha} + h_3^{\beta}.$$

Observe that in effect, we can write the particular solution w^{τ} as

$$w^{\tau}(\boldsymbol{x}) = \begin{cases} w^{\alpha}(\boldsymbol{x}) + \hat{w}^{\tau}(\boldsymbol{x}) & \boldsymbol{x} \in \Omega^{\alpha}, \\ w^{\beta}(\boldsymbol{x}) + \hat{w}^{\tau}(\boldsymbol{x}) & \boldsymbol{x} \in \Omega^{\beta}, \end{cases}$$

The function w^{τ} must of course be smooth across Γ_3, so the function $\hat{w}^{\tau}$ must have a jump that exactly offsets the discrepancy in the derivatives of w^{α} and w^{β}. This jump is precisely of size $h^{\alpha} - h^{\beta}$.

5 The full solver for a uniform grid

5.1 Notation

Suppose that we are given a rectangular domain Ω, which has hierarchically been split into a binary tree of successively smaller patches, as described in Sect. 2. We then define two sets of interpolation nodes. First, $\{\boldsymbol{x}_i\}_{i=1}^{M}$ denotes the set of nodes obtained by placing a $p \times p$ tensor product grid of Chebyshev nodes on each leaf in

the tree. For a leaf τ, let I_{c}^{τ} denote an index vector pointing to the nodes in $\{\boldsymbol{x}_i\}_{i=1}^{M}$ that lie on leaf τ. Thus the index vector for the set of nodes in τ can be partitioned into exterior and interior nodes as follows

$$I_{\mathrm{c}}^{\tau} = I_{\mathrm{ce}}^{\tau} \cup I_{\mathrm{ci}}^{\tau}.$$

The second set of interpolation nodes $\{\boldsymbol{y}_j\}_{j=1}^{N}$ is obtained by placing a set of q Gaussian ("Legendre") interpolation nodes on the edge of each leaf. For a node τ in the tree (either a leaf or a parent), let I_{ge}^{τ} denote an index vector that marks all Gaussian nodes that lie on the boundary of Ω_τ. For a parent node τ, let I_{gi}^{τ} denote the Gaussian nodes that are interior to τ, but exterior to its two children (as in Sect. 4).

Once the collocation points have been set up, we introduce a vector $\mathsf{u} \in \mathbb{R}^M$ holding approximations to the values of the potential u on the Gaussian collocation points,

$$\mathsf{u}(j) \approx u\left(\boldsymbol{y}_j\right), \quad j = 1, 2, 3, \ldots, M.$$

We refer to subsets of this vector using the short-hand

$$\mathsf{u}_{\mathrm{ge}}^{\tau} = \mathsf{u}\left(I_{\mathrm{ge}}^{\tau}\right), \text{ and } \quad \mathsf{u}_{\mathrm{gi}}^{\tau} = \mathsf{u}\left(I_{\mathrm{gi}}^{\tau}\right)$$

for the exterior and interior nodes respectively. At the very end of the algorithm, approximations to u on the local Chebyshev tensor product grids are constructed. For a leaf node τ, let the vectors $\mathsf{u}_{\mathrm{c}}^{\tau}$, $\mathsf{u}_{\mathrm{ce}}^{\tau}$, and $\mathsf{u}_{\mathrm{ci}}^{\tau}$ denote the vectors holding approximations to the potential on sets of collocation points in the Chebyshev grid marked by I_{c}^{τ}, I_{ce}^{τ}, and I_{ci}^{τ}, respectively. Observe that these vectors are *not* subvectors of u.

Before proceeding to the description of the algorithm, we introduce two sets of auxiliary vectors. First, for any parent node τ, let the vector $\mathsf{w}_{\mathrm{gi}}^{\tau}$ denote the computed values of the local particular solution w^τ that solves (6) on Ω_τ, as tabulated on the interior line marked by I_{gi}^{τ}. Also, define h^{τ} as the approximate boundary fluxes of w^τ as defined by (14) for a leaf and by (19) for a parent.

5.2 The build stage

Once the domain is partitioned into a hierarchical tree, we execute a "build stage" in which the following matrices are constructed for each box τ:

- S^{τ} For a box τ, the solution operator that maps Dirichlet data ψ on $\partial\Omega_\tau$ to values of u at the interior nodes. In other words, $\mathsf{u}_{\mathrm{c}}^{\tau} = \mathsf{S}_{\mathrm{c,ge}}^{\tau}\psi_{\mathrm{ge}}^{\tau}$ on a leaf or $\mathsf{u}_{\mathrm{gi}}^{\tau} = \mathsf{S}_{\mathrm{gi,ge}}^{\tau}\psi_{\mathrm{ge}}^{\tau}$ on a parent box.
- T^{τ} For a box τ, the matrix that maps Dirichlet data ψ on $\partial\Omega_\tau$ to the flux v on the boundary. In other words, $\mathsf{v}_{\mathrm{ge}}^{\tau} = \mathsf{T}_{\mathrm{ge,ge}}^{\tau}\psi_{\mathrm{ge}}$.
- F^{τ} For a leaf box, the matrix that maps the body load to the particular solution on the interior of the leaf assuming the Dirichlet data is zero on the boundary. In other words $\mathsf{w}_{\mathrm{c}}^{\tau} = \mathsf{F}_{\mathrm{c,ci}}^{\tau}\mathsf{g}_{\mathrm{ci}}$.
- H^{τ} For a leaf box, the matrix that maps the body load to the flux on the boundary of the leaf. In other words $\mathsf{h}_{\mathrm{ge}}^{\tau} = \mathsf{H}_{\mathrm{ge,ci}}^{\tau}\mathsf{g}_{\mathrm{ci}}^{\tau}$.

X^τ For a parent box τ with children α and β, X^τ maps the fluxes of the particular solution for the children on the interior of a parent to the particular solution on the interior nodes. In other words $\mathsf{w}_{\text{gi}} = \mathsf{X}^\tau_{\text{gi,gi}}(\mathsf{h}^\beta_3 - \mathsf{h}^\alpha_3)$.

The build stage consists of a single sweep over all nodes in the tree. Any ordering of the boxes in which a parent box is processed after its children can be used. For each leaf box τ, approximations S^τ and F^τ to the solution operators for the homogeneous and particular solutions are constructed. Additionally, approximations T^τ and H^τ to the local continuum operators that map boundary data and body load for a particular solution to the boundary fluxes of the resulting particular solution are constructed using the procedure described in Sect. 3. For a parent box τ with children α and β, we construct the solution operators $\mathsf{X}^\tau_{\text{gi,gi}}$ and $\mathsf{S}^\tau_{\text{gi,ge}}$, and the DtN operator $\mathsf{T}^\tau_{\text{ge,ge}}$ via the process described in Sect. 4. Algorithm 1 in Fig. 3 summarizes the build stage.

5.3 The solve stage

After the "build stage" described in Algorithm 1 has been completed, an approximation to the global solution operator of (1) has been computed, and represented through the various matrices (H^τ, F^τ, etc.) described in Sect. 5.2. Then given specific boundary data f and a body load g, the corresponding solution u to (1) can be found through a "solve stage" that involves two passes through the tree, first an upwards pass (from smaller to larger boxes), and then a downwards pass. In the upward pass, the particular solutions and normal derivatives of the particular solution are computed and stored in the vectors w and h respectively. Then by sweeping down the tree applying the solution operators S to the Dirichlet boundary data for each box τ and adding the particular solution, the approximate solution u is computed. Algorithm 2 summarizes the solve stage (Fig. 4).

We observe that the vectors $\mathsf{w}^\tau_{\text{gi}}$ can all be stored on a global vector $\mathsf{w} \in \mathbb{R}^N$. Since each boundary collocation node y_j belongs to precisely one index vector I^τ_{gi}, we simply find that $\mathsf{w}^\tau_{\text{gi}} = \mathsf{w}(I^\tau_{\text{gi}})$.

Remark 2 (Efficient storage of particular solutions) For notational simplicity, we describe Algorithm 2 (the "solve stage") in a way that assumes that for each box τ, we explicitly store a corresponding vector $\mathsf{h}^\tau_{\text{ge}}$ that represents the boundary fluxes for the local particular solution. In practice, these vectors can all be stored on a global vector $\mathsf{h} \in \mathbb{R}^N$, in a manner similar to how we store w. For any box τ with children α and β, we store on h the *difference* between the boundary fluxes, so that $\mathsf{h}(I^\tau_{\text{gi}}) = -\mathsf{h}^\alpha_3 + \mathsf{h}^\beta_3$. In other words, as soon as the boundary fluxes have been computed for a box α, we add its contributions to the vector $\mathsf{h}(I^\alpha_{\text{ge}})$ with the appropriate signs and then delete it. This becomes notationally less clear, but is actually simpler to code.

5.4 Algorithmic complexity

In this section, we determine the asymptotic complexity of the direct solver. The analysis is very similar to the analysis seen in [6] for no body load. Let $N_{\text{leaf}} = 4q$

ALGORITHM 1 (Build stage for problems with body load)

This algorithm builds all solution operators required to solve the non-homogeneous BVP (1). It is assumed that if node τ is a parent of node σ, then $\tau < \sigma$.

---

for $\tau = N_{\text{boxes}},\ N_{\text{boxes}} - 1,\ N_{\text{boxes}} - 2,\ \ldots,\ 1$
  if (τ is a leaf)

$$\mathbf{F}^{\tau}_{\text{c,ci}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{A}^{-1}_{\text{ci,ci}} \end{bmatrix} \quad \textit{[pot.]} \leftarrow \textit{[body load]}$$

$$\mathbf{H}^{\tau}_{\text{ge,ci}} = \mathbf{D}_{\text{ge,c}}\mathbf{F}^{\tau}_{\text{c,ci}} \quad \textit{[deriv.]} \leftarrow \textit{[body load]}$$

$$\mathbf{S}^{\tau}_{\text{c,ge}} = \begin{bmatrix} \mathbf{I} \\ -\mathbf{A}^{-1}_{\text{ci,ci}}\mathbf{A}_{\text{ci,ce}} \end{bmatrix} \mathbf{L}_{\text{ce,ge}} \quad \textit{[pot.]} \leftarrow \textit{[pot.] (solution operator)}$$

$$\mathbf{T}^{\tau}_{\text{ge,ge}} = \mathbf{D}_{\text{ge,c}}\mathbf{S}_{\text{c,ge}} \quad \textit{[deriv.]} \leftarrow \textit{[pot.] (DtN operator)}$$

  else

Let α and β be the children of τ.

Partition I^{α}_{ge} and I^{β}_{ge} into vectors I_1, I_2, and I_3 as shown in Figure 2.

$$\mathbf{X}^{\tau}_{\text{gi,gi}} = \left(\mathbf{T}^{\alpha}_{3,3} - \mathbf{T}^{\beta}_{3,3}\right)^{-1} \quad \textit{[pot.]} \leftarrow \textit{[deriv.]}$$

$$\mathbf{S}^{\tau}_{\text{gi,ge}} = \mathbf{X}^{\tau}_{\text{gi,gi}}\left[-\mathbf{T}^{\alpha}_{3,1} \mid \mathbf{T}^{\beta}_{3,2}\right] \quad \textit{[pot.]} \leftarrow \textit{[pot.] (solution operator)}$$

$$\mathbf{T}^{\tau}_{\text{ge,ge}} = \begin{bmatrix} \mathbf{T}^{\alpha}_{1,1} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^{\beta}_{2,2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \mathbf{S}^{\tau}_{\text{gi,ge}} \quad \textit{[deriv.]} \leftarrow \textit{[pot.] (DtN operator).}$$

  end if
end for

Fig. 3 Build stage

denote the number of Gaussian nodes on the boundary of a leaf box, and let p^2 denote the number of Chebychev nodes used in the leaf computation. In the asymptotic analysis, we set $p = q + 2$, so that $p \sim q$. Let L denote the number of levels in the binary tree. This means there are 2^L boxes. Thus the total number of discretization nodes N is approximately $2^L q^2$.

In processing a leaf, the dominant cost involves matrix inversion (or factorization followed by triangular solve) and matrix-matrix multiplications. The largest matrices encountered are of size $O(q^2) \times O(q^2)$, making the cost to process one leaf $O(q^6)$. Since there are N/q^2 leaf boxes, the total cost of pre-computing approximate DtN operators for all the bottom level is $\sim (N/q^2) \times q^6 \sim N\,q^4$.

Next, consider the process of merging two boxes, as described in Sect. 4. On level ℓ, there are 2^ℓ boxes, that each have $O(2^{-\ell/2}N^{0.5}))$ nodes along their boundaries (on level $\ell = 2$, there are 4 boxes that each have side length one half of the original side length; on level $\ell = 4$, there are 16 boxes that have side length one quarter of the original side length; etc.). The cost of executing a merge is dominated by the cost to perform matrix algebra (inversion, multiplication, etc) of dense matrices of size $2^{-\ell/2}N^{0.5} \times 2^{-\ell/2}N^{0.5}$. This makes the total cost for the merges in the upwards pass

ALGORITHM 2 (Solver for problems with body load)

This algorithm constructs an approximation $\mathbf{u}$ to the solution u of (1). It uses the matrices that represent the solution operator that were constructed using Algorithm 3. It is assumed that if node τ is a parent of node σ, then $\tau < \sigma$.

Upwards pass — construct all particular solutions:

for $\tau = N_{\text{boxes}}, N_{\text{boxes}} - 1, N_{\text{boxes}} - 2, \ldots, 1$
  if (τ is a leaf)
    # Compute the boundary fluxes of the local particular solution.
    $\mathbf{h}^{\tau}_{\text{ge}} = \mathbf{H}^{\tau}_{\text{ge,ci}}\, \mathbf{g}^{\tau}_{\text{ci}}$
  else
    Let α and β denote the children of τ.
    # Compute the local particular solution.
    $\mathbf{w}^{\tau}_{\text{gi}} = \mathbf{X}^{\tau}_{\text{gi,gi}} \left(-\mathbf{h}^{\alpha}_{3} + \mathbf{h}^{\beta}_{3}\right)$.
    # Compute the boundary fluxes of the local particular solution.
    $\mathbf{h}^{\tau}_{\text{ge}} = \begin{bmatrix} \mathbf{h}^{\alpha}_{1} \\ \mathbf{h}^{\beta}_{2} \end{bmatrix} + \begin{bmatrix} \mathbf{T}^{\alpha}_{1,3} \\ \mathbf{T}^{\beta}_{2,3} \end{bmatrix} \mathbf{w}^{\tau}_{\text{gi}}$.
  end if
end for

Downwards pass — construct all potentials:

 # Use the provided Dirichlet data to set the solution on the exterior of the root.
$\mathbf{u}(I^{1}_{\text{ge}}) = \{f(\boldsymbol{y}_k)\}_{k \in I^{1}_{\text{ge}}}$.
for $\tau = 1, 2, 3, \ldots, N_{\text{boxes}}$
  if (τ is a parent)
    # Add the homogeneous term and the particular term.
    $\mathbf{u}^{\tau}_{\text{gi}} = \mathbf{S}^{\tau}_{\text{gi,ge}}\, \mathbf{u}^{\tau}_{\text{ge}} + \mathbf{w}^{\tau}_{\text{gi}}$.
  else
    # Add the homogeneous term and the particular term.
    $\mathbf{u}^{\tau}_{\text{c}} = \mathbf{S}^{\tau}_{\text{c,ge}}\, \mathbf{u}^{\tau}_{\text{ge}} + \mathbf{F}^{\tau}_{\text{c,ci}}\, \mathbf{g}^{\tau}_{\text{ci}}$.
  end
end for

Fig. 4 Solve stage

$$\sum_{\ell=1}^{L} 2^{\ell} \times \left(2^{-\ell/2} N^{0.5}\right)^{3} \sim \sum_{\ell=1}^{L} 2^{\ell} \times 2^{-3\ell/2} N^{1.5} \sim N^{1.5} \sum_{\ell=1}^{L} 2^{-\ell/2} \sim N^{1.5}.$$

Finally, consider the cost of the solve stage (Algorithm 2). We first apply at each of the 2^L leaves the operators $\mathbf{H}^{\tau}_{\text{ge,ci}}$, which are all of size $4q \times (p-2)^2$, making the overall cost $\sim 2^L q^3 = N\, q$ since $p \sim q$ and $N \sim 2^L q^2$. In the upwards sweep, we apply at level ℓ matrices of size $O(2^{-\ell/2} N^{0.5}) \times O(2^{-\ell/2} N^{0.5})$ on 2^{ℓ} boxes, adding up to an overall cost of

$$\sum_{\ell=1}^{L} 2^{\ell} \times \left(2^{-\ell/2}\, N^{0.5}\right)^{2} \sim \sum_{\ell=1}^{L} 2^{\ell} \times 2^{-\ell}\, N \sim \sum_{\ell=1}^{L} N \sim NL \sim N \log N.$$

The cost of the downwards sweep is the same. However, the application of the matrices $\mathsf{F}^{\tau}_{\mathrm{c,ci}}$ at the leaves is more expensive since these are of size $O(q^2) \times O(q^2)$, which adds up to an overall cost of $2^L\, q^4 = N\, q^2$.

The analysis of the asymptotic storage requirements perfectly mirrors the analysis of the flop count for the solve stage, since each matrix that is stored is used precisely once in a matrix-vector multiplication. In consequence, the amount of storage required is

$$R \sim N\, q^2 + N\, \log N. \tag{23}$$

Remark 3 (A storage efficient version) The storage required for all solution operators can become prohibitive when the local order q is high, due to the term $N\, q^2$ in (23). One way to handle this problem is to not store the local solution operators for a leaf, but instead simply perform small dense solves each time the "solve stage" is executed. This makes the solve stage slower, obviously, but has the benefit of completely eliminating the Nq^2 term in (23). In fact, in this modified version, the overall storage required is $\sim NL \approx N\, \log_2(N/q^2)$, so we see that the storage costs decrease as q increases (as should be expected since we do all leaf computations from scratch in this case). Figure 9 provides numerical results illustrating the memory requirements of the various approaches.

6 Local refinement

When solving a boundary value problem like (1) it is common to have a localized loss of regularity due to, e.g., corners on the boundary, a locally non-smooth body load or boundary condition, or a localized loss of regularity in the coefficient functions in the differential operator. A common approach to efficiently regain high accuracy without excessively increasing the number of degrees of freedom used, is to locally refine the mesh near the troublesome location. In this manuscript, we assume the location is known and given, and that we manually specify the degree of local refinement. The difficulty that arises is that upon refinement, the collocation nodes on neighboring patches do not necessarily match up. To remedy this, interpolation operators are introduced to transfer information between patches (the more difficult problem of determining how to automatically detect regions that require mesh refinement is a topic of current research).

6.1 Refinement criterion

Suppose we desire to refine our discretization at some point $\hat{\boldsymbol{x}}$ in the computational domain (the point $\hat{\boldsymbol{x}}$ can be either in the interior or on the boundary). Consider as an example the situation depicted in Fig. 5. For each level of refinement, we split any leaf box that contains $\hat{\boldsymbol{x}}$ and any "close" leaf boxes into a 2×2 grid of equal-sized leaf boxes. In Fig. 5 we perform one level of refinement and find there are 6 leaf boxes

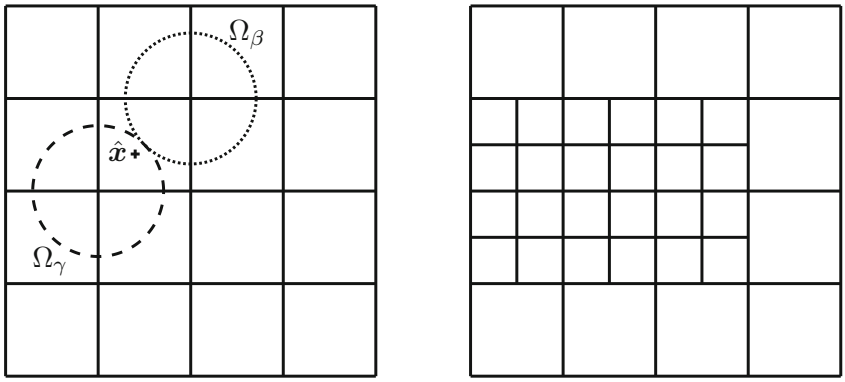

Fig. 5 A sample domain where we desire to refine the grid at a point $\hat{\boldsymbol{x}}$, which is marked by a cross. For the leaf box Ω_γ, the shortest distance to $\hat{\boldsymbol{x}}$ satisfies $d_\gamma < tl_\gamma$ for $t = \sqrt{2}$, so Ω_γ is refined. The maximum distance tl_γ is shown by the dashed circle, which is centered at the closest point from Ω_γ to $\hat{\boldsymbol{x}}$. For the leaf box Ω_β, the shortest distance to $\hat{\boldsymbol{x}}$ does not satisfy $d_\beta < tl_\beta$, so Ω_β is not refined. The maximum distance tl_β is shown by the dotted circle

"close" to $\hat{\boldsymbol{x}}$, which is represented by the cross. These 6 boxes are refined into smaller leaf boxes.

A leaf box Ω_τ is close to $\hat{\boldsymbol{x}}$ if the distance d_τ from $\hat{\boldsymbol{x}}$ to the box Ω_τ satisfies $d_\tau \leq tl_\tau$, where $t = \sqrt{2}$ and $2l_\tau$ is the length of one side of the leaf box Ω_τ. In Fig. 5 we show circles of size tl_γ and tl_β at the points in Ω_γ and Ω_β closest to $\hat{\boldsymbol{x}}$ (in this case the boxes are all the same size so $l_\gamma = l_\beta$). We see $\hat{\boldsymbol{x}}$ is "close" to Ω_γ, but not "close" to Ω_β. Just as in Sect. 5.1, we place a $p \times p$ tensor product grid of Chebyshev nodes on each new leaf and a set of q Gaussian ("Legendre") interpolation nodes on the edge of each leaf. The vector $\{\boldsymbol{y}_j\}_{j=1}^N$ holds the locations of all Gaussian nodes across all leaves in the domain.

6.2 Refined mesh

Notice that with the refined grid the nodes along common boundaries are no longer aligned. Figure 6 is an example of such a grid. This is a problem during the build stage of the method since the merge operation is performed by equating the Neumann data on the common boundary. We begin the discussion on how to address this problem by establishing some notation.

Define two boxes as *neighbors* if they are on the same level of the tree and they are adjacent. In the case that only one of two neighbors has been refined, such as Ω_α and Ω_β in Fig. 6, special attention needs to be paid to the nodes on the common boundary. In order to merge boxes with different number of Gaussian nodes on the common edge, interpolation operators will be required. The next section describes this process in detail.

Consider the nodes on the common boundary between the two leaf boxes Ω_α and Ω_β. Let q denote the number of Gaussian nodes on one side of each leaf. Let $\{J_{\alpha,i}\}_{i=1}^q$ denote the index vector for the common boundary nodes from Ω_α and $\{J_{\beta,i}\}_{i=1}^{2q}$ denote

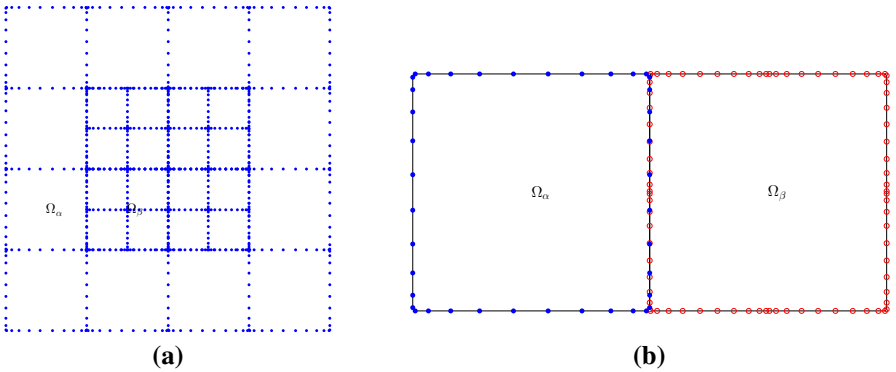

Fig. 6 **a** Grid with refinement at the center. **b** A close up of neighbors Ω_α and Ω_β. Since only one of the boxes is refined the exterior Gaussian nodes on the common boundary are not aligned

the index vector for the common boundary nodes from Ω_β. That is, recalling that y holds the locations of all Gaussian nodes in the domain, $\mathsf{y}(J_{\alpha,i})$ contains the q Gaussian nodes on the Eastern side of box Ω_α and $\mathsf{y}(\mathsf{J}_{\beta,i})$ contains the $2q$ nodes on the Western side of box Ω_β.

6.3 Modifications to build stage

Once the grid with the Gaussian and Chebyshev nodes is constructed, as described in Sect. 6.2, the build stage starts with the construction of all leaf operators as described in Sect. 3. Then, for simplicity of presentation, boxes are merged from the lowest level moving up the tree. After merging the children of a refined parent, such as Ω_β in Fig. 6 it is seen that the parent's exterior nodes do not align with the exterior nodes of any neighbor which has not been refined.

Recalling the index notation used in Sect. 6.2, we form the interpolation matrix $\mathsf{P}_{\text{up,W}}$ mapping data on $\mathsf{y}(J_{\alpha,i})$ to data on $\mathsf{y}(J_{\beta,i})$ and the interpolation matrix $\mathsf{P}_{\text{down,W}}$ mapping data on $\mathsf{y}(\mathsf{J}_{\beta,i})$ to data on $\mathsf{y}(J_{\alpha,i})$. Observe that when interpolating from two sets of q Gaussian nodes to a set of q Gaussian nodes, the interpolation must be done as two separate interpolations from q to $q/2$ nodes. The matrix $\mathsf{P}_{\text{down,W}}$ is a block diagonal matrix consisting of two $q/2 \times q$ matrices (assuming q is divisible by 2).

For a refined parent, such as Ω_β in Fig. 6, we form the operators T^β, S^β, and X^β and form the interpolation operators for every side of the parent, regardless of whether the exterior nodes align with the neighbor's exterior nodes. Observe that in the case of Ω_β in Fig. 6 the Eastern and Northern sides of Ω_β will have $\mathsf{P}_{\text{down}} = \mathsf{P}_{\text{up}} = \mathsf{I}$, where I is the identity matrix.

Then interpolation operators mapping the entire boundary data between fine and coarse grids are given by block diagonal matrices P_{up} and P_{down} whose diagonal blocks are the interpolation operators for each edge. The interpolation operators for Ω_β are

$$\mathsf{P}^\beta_{\text{up}} = \texttt{blkdiag}\left(\mathsf{P}^\beta_{\text{up,S}}, \mathsf{P}^\beta_{\text{up,E}}, \mathsf{P}^\beta_{\text{up,N}}, \mathsf{P}^\beta_{\text{up,W}}\right)$$

and

$$\mathsf{P}^{\beta}_{\text{down}} = \texttt{blkdiag}\left(\mathsf{P}^{\beta}_{\text{down,S}}, \mathsf{P}^{\beta}_{\text{down,E}}, \mathsf{P}^{\beta}_{\text{down,N}}, \mathsf{P}^{\beta}_{\text{down,W}}\right).$$

(the text `blkdiag` denotes the function that forms a block diagonal matrix from its arguments). Then we form the new operators $\mathsf{T}^{\beta}_{\text{new}}$ and $\mathsf{S}^{\beta}_{\text{new}}$ for the parent box Ω_β as follows

$$\mathsf{T}^{\beta}_{\text{new}} = \mathsf{P}^{\beta}_{\text{down}} \mathsf{T}^{\beta} \mathsf{P}^{\beta}_{\text{up}}$$

and

$$\mathsf{S}^{\beta}_{\text{new}} = \mathsf{S}^{\beta} \mathsf{P}^{\beta}_{\text{up}}.$$

Now $\mathsf{T}^{\beta}_{\text{new}}$ is a map defined on the same set of points as all of the neighbors of Ω_β and Neumann data can be equated on all sides.

Next, suppose a refined parent does not have a neighbor on one of its sides. Then on that side we use $\mathsf{P}_{\text{down}} = \mathsf{P}_{\text{up}} = \mathsf{I}$. This could happen if the parent is on the boundary of our domain Ω. For example, suppose Ω_α in Fig. 6 was also refined. Then Ω_α would not have a neighbor on its Western side. Additionally, if multiple levels of refinement are done then a refined parent could have no neighbors on one side. For example, suppose the Northwestern child of Ω_β in Fig. 6 was refined. Then the Northwestern child would not have a neighbor on its Western side since box Ω_α is on a different level of the tree.

Forming the interpolation operators for each side of Ω_β before we perform any following merge operations is the easiest approach. The alternative would be to form an interpolation operator every time two boxes are merged and the nodes do not align.

6.4 Modifications to solve stage

On the upwards pass of the solve stage, the fluxes for the particular solution must be calculated on the same nodes so the particular solution can be calculated on those nodes. This is easily achieved by applying the already computed interpolation operator P_{down} to obtain $\mathsf{h}^{\beta}_{\text{new}} = \mathsf{P}_{\text{down}} \mathsf{h}^{\beta}_{\text{old}}$.

In the downwards pass of the solve stage, the application of the solution operators results in the approximate solution at the coarse nodes on the Western and Southern sides of Ω_β. The solution operator $\mathsf{S}^{\beta}_{\text{new}}$ now maps the solution on the coarse nodes on the Western and Southern sides of Ω_β (and the dense nodes on the Eastern and Northern sides) to the solution on the interior of Ω_β. However, we also need the solution on the dense nodes on the Western and Southern sides of Ω_β. Let $\mathsf{u}_{ge,\text{new}}$ denote the solution on the boundary of Ω_β with the coarse nodes on the Western and Southern edges. Then the approximate solution on the dense nodes is given by $\mathsf{u}_{ge,\text{old}} = \mathsf{P}_{\text{up}} \mathsf{u}_{ge,\text{new}}$.

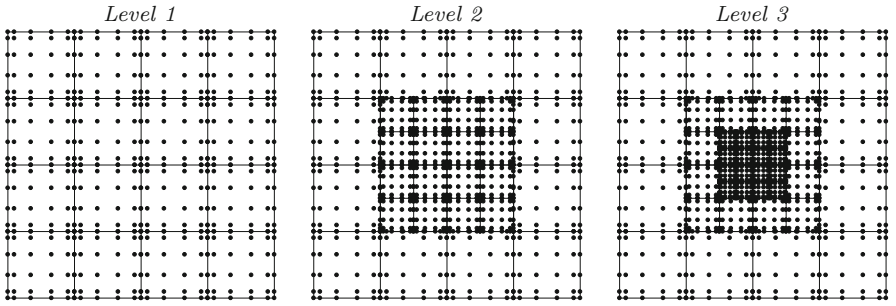

Fig. 7 Local refinement around a single point, as discussed in Sect. 6.5. In the figure, each leaf has a $q \times q$ local grid, for $q = 6$. At each level of refinement, the center 2×2 block of nodes is refined into 4×4 blocks of half the size

6.5 Impact of refinement on the asymptotic complexity

We demonstrated in Sect. 5.4 that the asymptotic complexity of the build stage is $O(N^{3/2})$ for the case of a uniform balanced tree. It turns out that such a tree is the most adversarial case from the point of view of the number of flops required, just as for nested dissection and multifrontal solvers. While we will not carry out a detailed analysis of every possible tree, it is perhaps illustrative to consider what happens in the extreme case where we start with a course level grid consisting of 4×4 nodes, each with a $q \times q$ tensor product grid. Then we recursively refine the middle 2×2 nodes, as shown in Fig. 7. In this case, each refinement adds roughly $4q^2$ nodes, so that with L levels in the tree, the total number of nodes satisfies $N \sim Lq^2$. During the build stage, the processing of each level in the tree requires dense operations to be carried out on matrices of size $O(q^2) \times O(q^2)$, at cost $O(q^6)$. This leads to a total cost of $T \sim L\, q^6 \sim N\, q^4$ for the case of local refinement around a single point. In the general case, asymptotic complexity between $O(N)$ and $O(N^{3/2})$, depending on the structure of the tree, will be observed.

7 Numerical experiments

In this section, we present the results of numerical experiments that illustrate the performance of the scheme proposed. Section 7.1 reports on the computational cost and memory requirements. Sections 7.2–7.5 report on the accuracy of the proposed solution technique for a variety of problems where local mesh refinement is required. Finally, Sect. 7.6 illustrates the use of the proposed method in the acceleration of an implicit time stepping scheme for solving a parabolic partial differential equation.

For each experiment, the error is calculated by comparing the approximate solution with a reference solution u_{ref} constructed using a highly over resolved grid. Errors are measured in ℓ^∞-norm, on all Chebyshev nodes on leaf boundaries.

In all of the experiments, each leaf is discretized using a $p \times p$ tensor product mesh of Chebyshev nodes. The number of Legendre nodes per leaf edge is set to $q = p - 1$. In all experiments except the one described in Sect. 7.5, the computational

domain is the square $\Omega = [0, 1]^2$ discretized into $n \times n$ leaf boxes, making the total number of degrees of freedom roughly $N \approx p^2 \times n^2$ (to be precise, $N = n^2 \times (p-1)^2 + 2n \times (p-1) + 1$).

The proposed method was implemented in Matlab and all experiments were run on a laptop computer with a 4 core Intel i7-3632QM CPU running at 2.20 GHz with 12 GB of RAM.

7.1 Computational speed

The experiments in this section illustrate the computational complexity and memory requirements of the direct solver. Recall that the asymptotic complexity of the method scales as nested dissection or multifrontal methods, with execution times scaling as $O(N^{3/2})$ and $O(N \log N)$ for the "build" and "solve" stages, respectively. The asymptotic memory requirement is $O(N \log N)$.

The computational complexity and memory requirements of the proposed method depend only on the domain and the computational mesh; the choice of PDE is irrelevant. In the experiments reported here, we used $\Omega = [0, 1]^2$ with a uniform mesh.

Figure 8 reports the time in seconds for the (a) build and (b) solve stages of the proposed solution technique when there is a body load (BL), when the leaf computation is done on the fly as described in Remark 3 (BL(econ)), and when there is no body load (NBL). Results for two different orders of discretization ($q = 8$ and 16) are

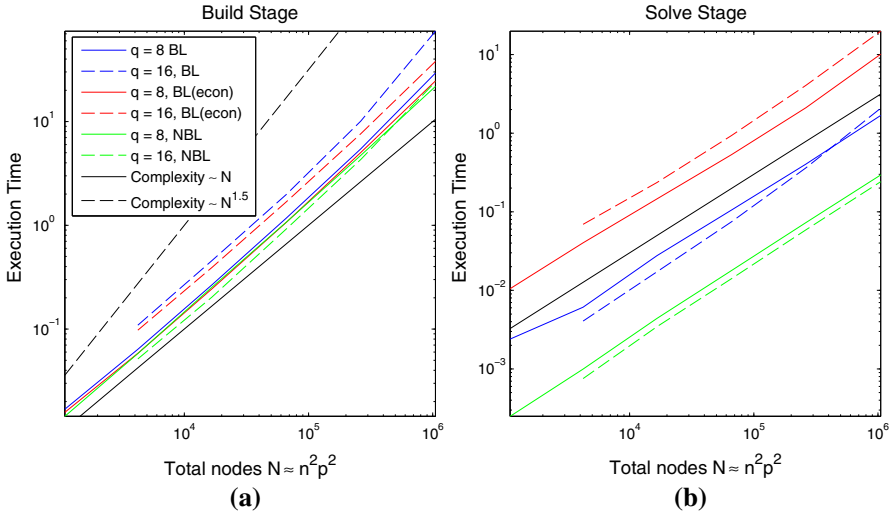

Fig. 8 **a** Time to execute build stage for the algorithm with and without a body load. These algorithms all have complexity $O(N^{1.5})$, and we see that the scaling factors depend strongly on the order of the method, but only weakly on whether body loads are included or not. **b** Time to execute the solve stage. Three cases are considered: NBL is the scheme for problems without a body load. BL is the scheme for problems with a body load. BL(econ) is a scheme that allows for body loads, but do not store the relevant solution operators at the leaves. p denotes the order in the local Chebyshev grids, and $q = p - 1$ is the number of Legendre nodes on the edge of each leaf

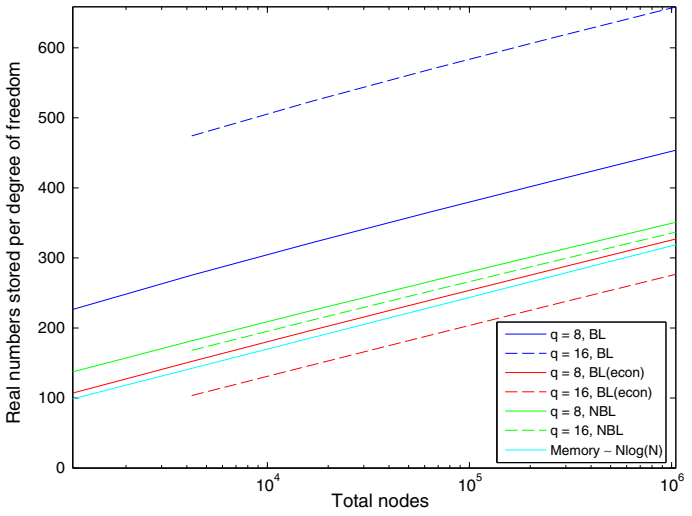

Fig. 9 Memory requirements. Notation is as in Fig. 8

shown. Notice that as expected the constant scaling factor for both stages is larger for the higher order discretization (it may be of interest that our numerical experiments in many cases demonstrate *linear* scaling for the build stage, despite its $O(N^{3/2})$ asymptotic algorithmic complexity. The reason is that even for $N \sim 10^6$, we have hardly yet entered the regime where the $O(N^{3/2})$ term starts to dominate).

Figure 9 reports on memory requirements. Letting R denote total memory used, we plot R/N versus the number of discretization points N, where R is measured in terms of number of floating point numbers. We see that storing the solution operators on the leaves is quite costly in terms of memory requirements. The trade-off to be considered here is whether the main priority is to conserve memory, or to maximize the speed of the solve stage, cf. Remark 3. As an illustration, we see that for a problem with $q = 8$ and $n = 128$, for a total of 10^6 unknowns, the solve stage takes 1.7 s with the solution operators stored versus 9.8 s for performing the local solves on the fly. In situations where the solution is only desired in prescribed local regions of the geometry, computing the operators on the fly is ideal.

Remark 4 (Complexity for a locally refined tree) We claim in Sect. 6.5 that for a locally refined tree, the complexity of the scheme is no worse (and sometimes better) than the $O(N^{3/2})$ complexity obtained for a balanced uniform tree. To substantiate this claim, we timed an experiment involving a locally refined tree, as shown in Fig. 7. The results are presented in Fig. 10, for two different orders of discretization ($q = 16$ and 32). We use larger numbers for q in this section so that we obtain a significant increase in the number of nodes. If we use $q = 8$ then each refinement only adds a few hundred nodes and it is hard to see any significant scaling without doing an unreasonable number of refinements. For reference, we include a comparison to the case of $q = 16$ with a body load and no refinement.

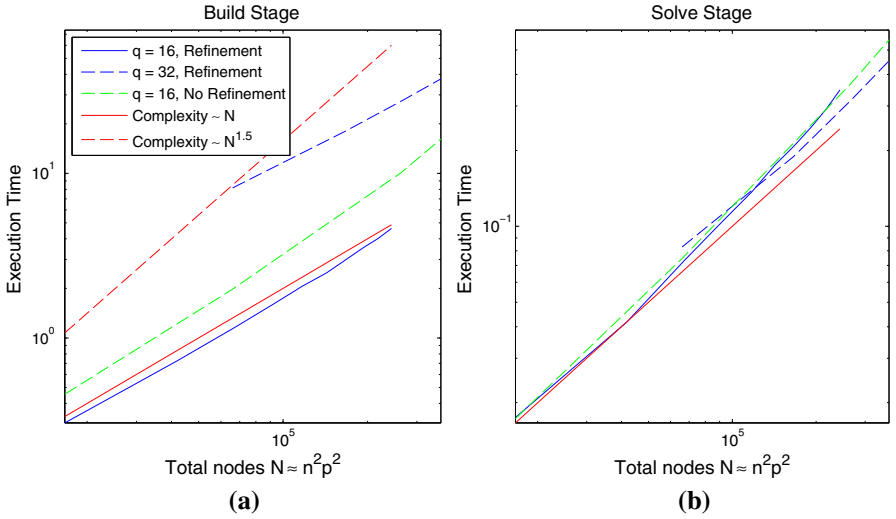

Fig. 10 **a** Time to execute build stage for the algorithm with refinement. **b** Time to execute the solve stage with refinement. In both plots we see that the refinement does not impact the time to build the operators

Remark 5 When the underlying BVP that is discretized involves a constant coefficient operator, many of the leaf solution operators are identical. This observation can be used to greatly reduce storage requirements while maintaining very high speed in the solve stage. This potential acceleration was *not* exploited in the numerical experiments reported.

7.2 Variable coefficients

In this section, the proposed scheme is applied to the variable coefficient Helmholtz problem

$$-\Delta u - \kappa^2(1 - c(\boldsymbol{x}))u = g, \quad \boldsymbol{x} \in \Omega,$$

where $\Omega = [0, 1]\times[0, 1]$ and where c is a "scattering potential." The body load is taken to be a Gaussian given by $g = \exp(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2)$ with $\alpha = 300$ and $\hat{\boldsymbol{x}} = [1/4, 3/4]$ while the variable coefficient is a sum of Gaussians $c(\boldsymbol{x}) = \frac{1}{2}\exp(-\alpha_2|\boldsymbol{x} - \hat{\boldsymbol{x}}_2|^2) + \frac{1}{2}\exp(-\alpha_3|\boldsymbol{x}-\hat{\boldsymbol{x}}_3|^2)$ with $\alpha_2 = \alpha_3 = 200, \hat{\boldsymbol{x}}_2 = [7/20, 6/10]$, and $\hat{\boldsymbol{x}}_3 = [6/10, 9/20]$ for the scattering potential. We set $\kappa = 40$, making the domain 6.4×6.4 wavelengths in size.

Figure 11 reports the l^∞ error versus the number of discretization points N. We get no accuracy for $q = 4$, but as q is increased, the errors rapidly decrease.

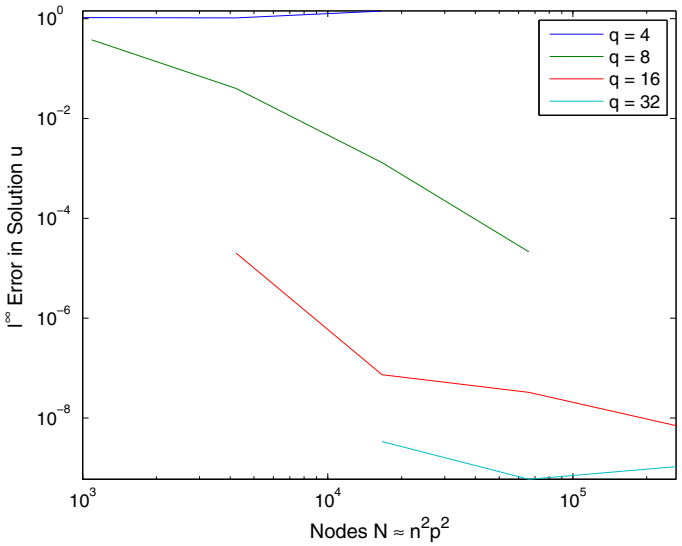

Fig. 11 The error for the variable coefficient problem described in Sect. 7.2. As before, q denotes the number of Legendre nodes along one side of a leaf

7.3 Concentrated body load

In this section, we consider a low frequency ($\kappa = 20$) Helmholtz boundary value problem

$$-\Delta u - \kappa^2 u = g, \qquad \boldsymbol{x} \in \Omega,$$

with $\Omega = [0, 1] \times [0, 1]$ and a very concentrated Gaussian for the body load, $g = \exp(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2)$ with $\alpha = 3000$. In this case, we chose the Dirichlet boundary data to equal the solution to the free space equation $-\Delta u - \kappa^2 u = g$ with a radiation condition at infinity. In other words, u is the convolution between g and the free space fundamental solution. We computed the boundary data and the reference solution by numerically evaluating this convolution to very high accuracy.

To test the refinement strategy, we build a tree first with a uniform grid, i.e. $n \times n$ leaf boxes then add n_{ref} levels of refinement around the point $\hat{\mathbf{x}}$. Figure 12 reports the l^∞ norm of the error versus n_{ref} for four choices of uniform starting discretization. When $n = 4$ one level of refinement (i.e. 28 leaf boxes) results in approximately the same accuracy as when $n = 8$ and no levels of refinement (i.e. 64 leaf boxes).

7.4 Discontinuous body load

In section, we consider a Poisson boundary value problem on $\Omega = [0, 1]^2$ with an indicator function body load g that has support $[1/4, 1/2] \times [1/4, 1/2]$. Observe that the lines of discontinuity of g coincide with edges of leaves in the discretization. Figure 13 reports the l^∞ error versus the number of discretization points N with uni-

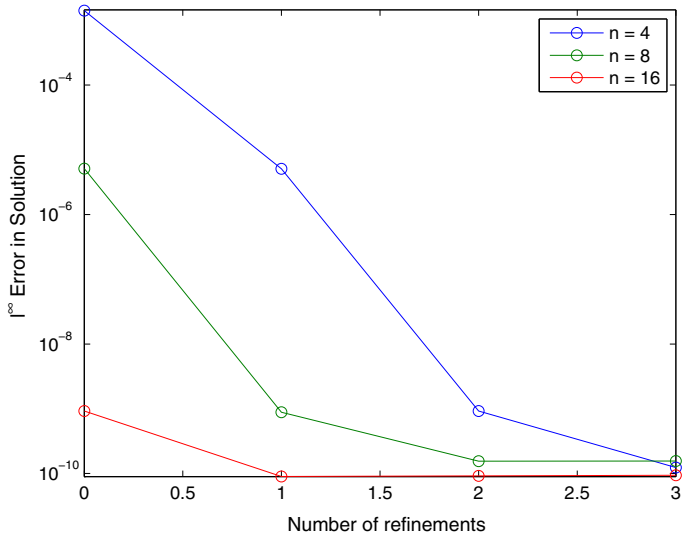

Fig. 12 Error for Helmholtz equation with $\kappa = 20$ and a very concentrated body load, demonstrating the ability to improve the solution with refinement. We use local Chebyshev grids with 17×17 Chebyshev nodes per leaf, and $n \times n$ leaves, before refinement. For a problem like this with a concentrated body load we can improve the error just as much by refining the discretization at the troublesome location as we can from doubling the number of leaves, which would give the same grid at the target location

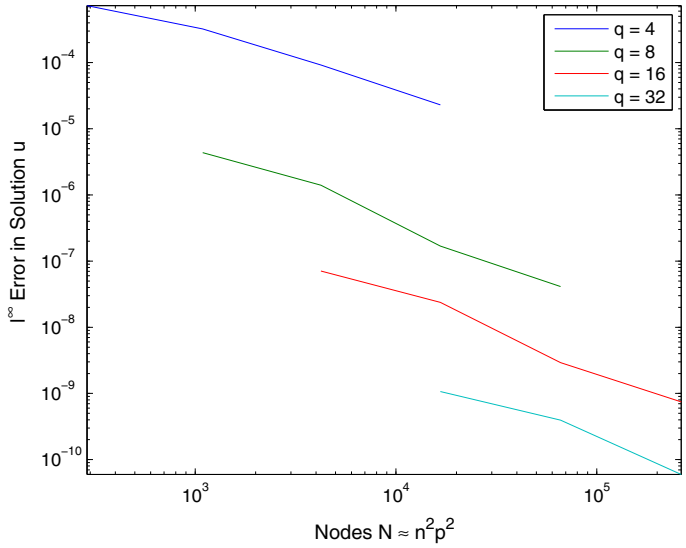

Fig. 13 The error for a problem with a discontinuous body load. The discontinuities align with the edges of the leaves so we still get 10 digits of accuracy. In the legend, q denotes the number of Legendre nodes along one side of a leaf

form refinement for four different orders of discretization. Note that the approximate solution and its first derivative are continuous through the boundaries of the leaves (even on the boundaries where the jump in the body load occurs) since the algorithm enforces them by derivation.

Remark 6 Applying the scheme to a problem where a discontinuity in the body load does *not* align with the leaf boundaries results in a very low accuracy approximation to the solution. For a problem analogous to the one described in this section, we observed slow convergence and attained no better than two or three digits of accuracy on the most finely resolved mesh.

7.5 Tunnel

This section reports on the performance of the solution technique when applied to the Helmholtz Dirichlet boundary value problem

$$\begin{aligned} -\Delta u - \kappa^2 u &= g, \qquad \boldsymbol{x} \in \Omega, \\ u &= f \qquad \boldsymbol{x} \in \partial\Omega \end{aligned}$$

with $\kappa = 60$ where the domain Ω is a "tunnel" as illustrated in Fig. 14. The body load is taken to be a Gaussian $g = \exp(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2)$ with $\alpha = 300$ and $\hat{\boldsymbol{x}} = [1, 3/4]$. The Dirichlet boundary data is given by

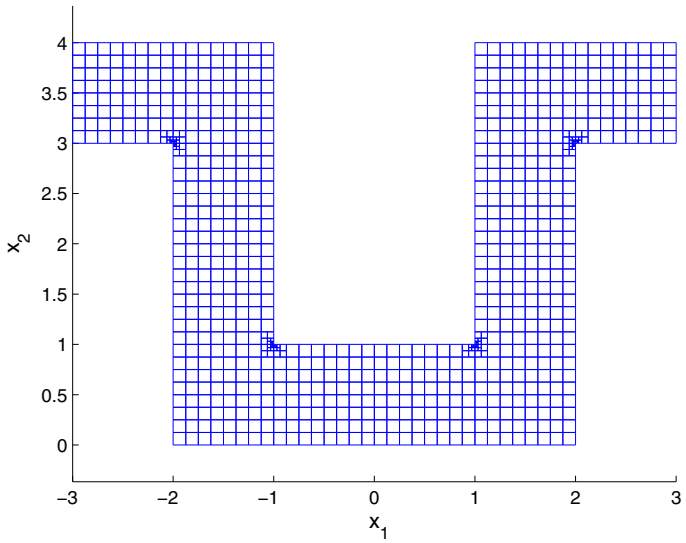

Fig. 14 The domain used for the tunnel problem. We solve Helmholtz equation with $\kappa = 60$, making the tunnel about 10λ wide and 115λ long. The end caps have fixed Dirichlet data and the sides of the tunnel have the Dirichlet data set to $u(\boldsymbol{x}) = 0$

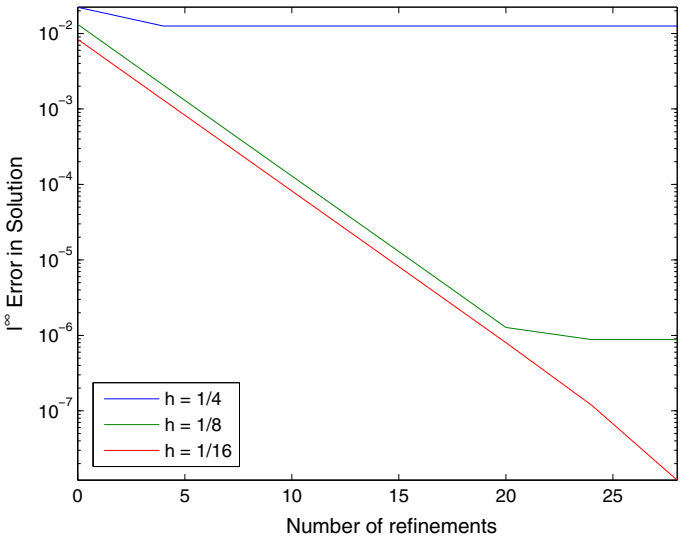

Fig. 15 Error for Helmholtz equation with $\kappa = 60$ on the tunnel. The end caps have fixed Dirichlet data and the sides of the tunnel have the Dirichlet data set to $f(\boldsymbol{x}) = 0$. A Gaussian $g = \exp(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2)$ with $\alpha = 300$ located at $\hat{\boldsymbol{x}} = [1, 3/4]$ is used for the body load

$$f(\boldsymbol{x}) = \begin{cases} 0 & \text{for } x_1 \neq \pm 3 \\ \frac{1}{100}\sin(2\pi(x_2 - 3)) & \text{for } x_1 = 3 \\ \frac{1}{100}\sin(\pi(x_2 - 3)) & \text{for } x_1 = -3. \end{cases}$$

Note that $f(\boldsymbol{x})$ is continuous on $\partial\Omega$ and with this choice of wave number κ the domain Ω is about 10 wavelengths wide and 115 wavelengths long. The presence of the re-entrant corners results in a solution that has strong singularities which require local refinement in order for the method to achieve high accuracy.

Figure 15 reports the l^∞ error versus the number of refinements into the corners with three choices of coarse grid. We use $q = 16$ for all examples and h gives the width and height of each leaf box. When $h = 1/4$, the discretization is only sufficient to resolve the Helmholtz equation with $\kappa = 60$ within 1% of the exact solution. When $h = 1/8$, the solution technique stalls at 5 digits of accuracy independent of the number of refinement levels.

Remark 7 (Symmetries) This problem is rich in symmetries that can be used to accelerate the build stage. In our implementation, we chose to exploit the fact that the tunnel is made up of four L-shaped pieces glued together. The DtN operator and corresponding solution operators were constructed for one L-shape. Then creating the solver for the entire geometry involved simply gluing the 4 L-shaped geometries together via *three* merge operations.

7.6 A parabolic problem

Our final numerical example involves a convection-diffusion initial value problem on $\Omega = [0, 1]^2$ given by

$$\left(\epsilon\Delta - \frac{\partial}{\partial x_1}\right) u(\boldsymbol{x}, t) = \frac{\partial u}{\partial t}, \qquad \boldsymbol{x} \in \Omega,\ t > 0$$
$$u(\boldsymbol{x}, 0) = \exp\left(-\alpha|\boldsymbol{x} - \hat{\boldsymbol{x}}|^2\right), \qquad \boldsymbol{x} \in \Omega.$$

We imposed zero Neumann boundary conditions on the south and north boundaries ($x_2 = 0, 1$) and periodic boundary conditions on the west and east boundaries ($x_1 = 0, 1$). These boundary conditions correspond to fluid flowing through a periodic channel where no fluid can exit the top or bottom of the channel. To have a convection dominated problem, we chose $\epsilon = 1/200$. Finally, the parameters in the body load were chosen to be $\alpha = 50$ and $\hat{\boldsymbol{x}} = [1/4, 1/4]$.

Applying the Crank–Nicolson time stepping scheme with a time step size k results in having to solve the following elliptic problem at each time step:

$$\left(\frac{1}{k}I - \frac{1}{2}A\right) u_{n+1} = \left(\frac{1}{k}I + \frac{1}{2}A\right) u_n, \tag{24}$$

where $A = \epsilon\Delta - \partial/\partial x_1$ is our partial differential operator.

Observe that the algorithm does not change for this problem. The build stage execution time and memory requirement are identical to those seen in Sect. 7.1. The execution time for the solve stage for each individual time step is nearly identical to the solve stage execution time shown in Sect. 7.1. The only new step in the solve stage is the need to evaluate $(I/k + A/2)u_n$ at each time step.

Figure 16 reports the l^∞ error vs. the time step size k at three different times $t = 0.025,\ 0.1$, and 0.5. We use 16 leaf boxes per side with $q = 16$, to ensure that the spatial resolution error is far smaller than the time-stepping error. Note that even with a low order time stepping scheme, a high accuracy can still be attained since our fast time stepper allows us to use a very short time step without incurring an unduly long solution time.

8 Concluding remarks

We have described an algorithm for solving non-homogeneous linear elliptic PDEs in two dimensions based on a multidomain spectral collocation discretization. The solver is designed explicitly for being combined with a nested dissection type direct solver. Its primary advantage over existing methods is that it enables the use of very high order local discretization without jeopardizing computational efficiency in the direct solver. The scheme is an evolution on previously published methods [5,6,11]. The novelty in this work is that the scheme has been extended to allow for problems involving body loads, and for local refinement.

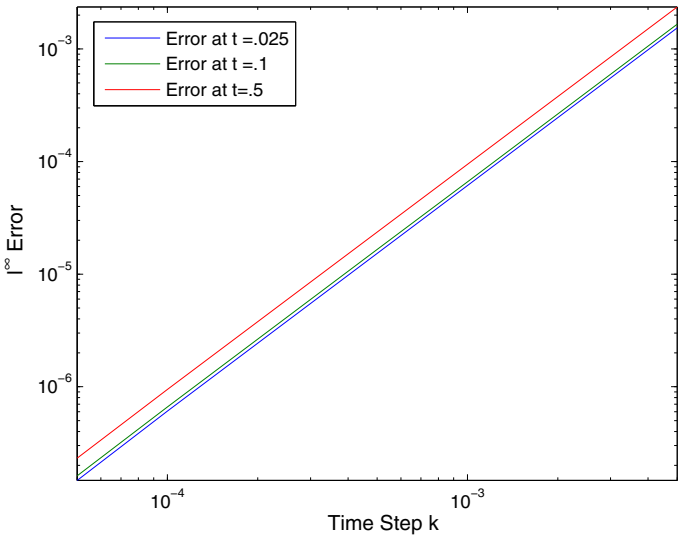

Fig. 16 Error for the convection-diffusion equation described in Sect. 7.6. The error is estimated by comparing against a highly over-resolved solution

The scheme is particularly well suited for executing the elliptic solve required solving parabolic problems using implicit time-stepping techniques in situations where the domain is fixed, so that the elliptic solve is the same in every time step. In this environment, the cost of computing an explicit solution operator is amortized over many time-steps, and can also be recycled when the same equation is solved for different initial conditions.

The fact that the method can with ease incorporate high order local discretizations, and allows for very efficient implicit time-stepping appears to make it particularly well suited for solving the Navier–Stokes equations at low Reynolds numbers. Such a solver is currently under development and will be reported in future publications. Other extensions currently under way includes the development of adaptive refinement criteria (as opposed to the supervised adaptivity used in this work), and the extension to problems in three dimensions, analogous to the work in [7] for homogeneous equations.

Acknowledgements The research reported was supported by DARPA, under the Contract N66001-13-1-4050, and by the NSF, under the Contracts DMS-1407340 and DMS-1620472.

References

1. Causley, M., Christlieb, A., Ong, B., Van Groningen, L.: Method of lines transpose: an implicit solution to the wave equation. Math. Comput. **83**, 2763–2786 (2014)

2. Davis, T.: Direct methods for sparse linear systems, vol. 2. Siam, Philadelphia (2006)
3. Duff, I.S., Erisman, A.M., Reid, J.K.: Direct methods for sparse matrices. Oxford University Press, Oxford (1989)
4. George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. **10**, 345–363 (1973)
5. Gillman, A., Barnett, A., Martinsson, P.G.: A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media. BIT Numer. Math. **55**(1), 141–170 (2015)
6. Gillman, A., Martinsson, P.: A direct solver with $O(N)$ complexity for variable coefficient elliptic pdes discretized via a high-order composite spectral collocation method. SIAM J. Sci. Comput. **36**(4), A2023–A2046 (2014). arXiv:1307.2665
7. Hao, S., Martinsson, P.G.: A direct solver for elliptic PDEs in three dimensions based on hierarchical merging of Poincaré–Steklov operators. J. Comput. Appl. Math. **308**, 419–434 (2016)
8. Khoromskij, B., Wittum, G.: Numerical solution of elliptic differential equations by reduction to the interface, vol. 36. Springer, Berlin (2004)
9. Kopriva, D.A.: A staggered-grid multidomain spectral method for the compressible Navier–Stokes equations. J. Comput. Phys. **143**(1), 125–158 (1998)
10. Martinsson, P.G.: A composite spectral scheme for variable coefficient Helmholtz problems, (2012). arXiv preprint arXiv:1206.4136
11. Martinsson, P.G.: A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method. J. Comput. Phys. **242**, 460–479 (2013)
12. Pfeiffer, H.P., Kidder, L.E., Scheel, M.A., Teukolsky, S.A.: A multidomain spectral method for solving elliptic equations. Comput. Phys. Commun. **152**(3), 253–273 (2003)
13. Trefethen, L.N.: Spectral methods in matlab. SIAM, Philadelphia (2000)
14. Yang, B., Hesthaven, J.S.: Multidomain pseudospectral computation of Maxwell's equations in 3-d general curvilinear coordinates. Appl. Numer. Math. **33**(1–4), 281–289 (2000)