Journal of Computational Physics 436 (2021) 110087

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

The ultraspherical spectral element method ☆

Daniel Fortunato [a,*], Nicholas Hale [b], Alex Townsend [c]

[a] School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, United States of America
[b] Department of Mathematical Sciences, Stellenbosch University, Stellenbosch, 7602, South Africa
[c] Department of Mathematics, Cornell University, Ithaca, NY 14853, United States of America

ARTICLE INFO

Article history:
Available online 24 December 2020

Keywords:
Spectral element method
Ultraspherical spectral method
Hierarchical Poincaré–Steklov method
hp-adaptivity

ABSTRACT

We introduce a novel spectral element method based on the ultraspherical spectral method and the hierarchical Poincaré–Steklov scheme for solving second-order linear partial differential equations on polygonal domains with unstructured quadrilateral or triangular meshes. Properties of the ultraspherical spectral method lead to almost banded linear systems, allowing the element method to be competitive in the high-polynomial regime ($p > 5$). The hierarchical Poincaré–Steklov scheme enables precomputed solution operators to be reused, allowing for fast elliptic solves in implicit and semi-implicit time-steppers. The resulting spectral element method achieves an overall computational complexity of $\mathcal{O}(p^4/h^3)$ for mesh size h and polynomial order p, enabling *hp*-adaptivity to be efficiently performed. We develop an open-source software system, `ultraSEM`, for flexible, user-friendly spectral element computations in MATLAB.

© 2020 Elsevier Inc. All rights reserved.

1. Introduction

Traditional approaches for solving partial differential equations (PDEs) on meshed geometries include finite element methods (FEMs) [26], discontinuous Galerkin (DG) methods [13], and spectral element methods (SEMs) [38]. Each approach typically represents the solution of the PDE as a piecewise polynomial, with continuity or jump conditions weakly or strongly imposed between elements. Convergence is achieved by either refining the mesh (h-refinement) or increasing the polynomial degree on the elements (p-refinement). In theory, super-algebraic convergence can be observed—even for solutions with singularities—by optimally selecting a refinement strategy (*hp*-adaptivity) [7]. However, *hp*-adaptivity theory can require high polynomial degrees, which are rarely used in practice as traditional methods can have prohibitive computational costs and numerical stability issues in this regime.

In particular, constructing efficient solvers for traditional high-order nodal element methods can be challenging. Direct solvers can become computationally intractable even for relatively small polynomial degrees as nodal discretizations result in dense linear algebra; in d dimensions, the computational complexity for a direct solver naïvely scales as $\mathcal{O}(p^{3d})$. Iterative solvers may require an increasing number of iterations as p increases because of the difficulties in designing robust preconditioners in the high p regime [37]. Because of these challenges, traditional element methods are typically restricted to low polynomial degrees, and h-refinement is generically preferred over p-refinement irrespective of local error estimators [47].

☆ **Funding:** The first author was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program. The second author was supported by the National Research Foundation (NRF) of South Africa (grant 109210). The third author was supported by NSF grant 1818757.

* Corresponding author.
E-mail addresses: dfortunato@g.harvard.edu (D. Fortunato), nickhale@sun.ac.za (N. Hale), townsend@cornell.edu (A. Townsend).

https://doi.org/10.1016/j.jcp.2020.110087
0021-9991/© 2020 Elsevier Inc. All rights reserved.

In practice, the physical considerations of the PDE—informed by *hp*-adaptivity theory—can take a back seat to the practical considerations of the numerical method.

Much work has gone toward reducing the computational costs associated with high-order element methods. For discretizations that possess tensor-product structure (e.g., standard nodal bases on quadrilateral elements or certain bases on triangular elements [43]), sum factorization [37] reduces the cost of operator assembly from $\mathcal{O}(p^{3d})$ to $\mathcal{O}(p^{2d+1})$, and matrix-free evaluation reduces the cost of matrix-vector multiplication from $\mathcal{O}(p^{2d})$ to $\mathcal{O}(p^{d+1})$ [3, Tab. 1]. Solvers for the resulting linear systems are often based on iterative methods coupled with sufficient preconditioning. Low-order FEM discretizations on a mesh constructed from the high-order SEM nodes can be shown to be spectrally equivalent to the SEM discretizations [12], and matrix-free preconditioners based on this equivalence can perform well when coupled with a multigrid method using specialized smoothers [39]. Multigrid methods applied to high-order DG discretizations can perform well if the discrete operators are coarsened according to the flux formulation of the PDE [17]. Spectral element multigrid methods have proven effective when applied to nodal discretizations of Poisson's equation in one dimension, though multigrid convergence factors can weakly depend on both h and p [29,41]. Modal discretizations for p-FEM based on integrated Jacobi polynomials can yield sparse stiffness matrices that contain an optimal number of nonzeros, but developing optimal solvers for such discretizations remains a challenge [9]. Many open-source software libraries exist for high-order element computation, including MFEM [3], Firedrake [40], Nektar++ [11,34], and Nek5000 [4].

Though solvers for element methods are commonly based on preconditioned iterative methods, fast direct solvers for high-order methods have become an active area of research in recent years. The hierarchical Poincaré–Steklov (HPS) scheme [5,22,23,30,31] is a multidomain spectral collocation method based on a recursive domain decomposition approach, which "glues" together solutions at interfaces between elements using Poincaré–Steklov operators (such as Dirichlet-to-Neumann operators). The accompanying direct solver is analogous to classical nested dissection. The formulation hierarchically merges Dirichlet-to-Neumann operators and results in an in-memory solution operator, which can be reapplied fast to multiple right-hand sides on static meshes. The ability to reuse computed solution operators allows for efficient implicit time-stepping for parabolic problems [6]. The method has been extended to handle mesh adaptivity [19], three-dimensional problems [25], and boundary integral equations [21]. The HPS scheme based on spectral collocation has an overall complexity of $\mathcal{O}(Np^4 + N^{3/2})$, where $N \approx (p/h)^2$ is the total number of degrees of freedom, p is the polynomial degree on each element, and h is the minimum mesh element size.

In this paper, we take advantage of recent advances in sparse spectral methods to propose an SEM in two dimensions with a computational complexity of

$$\underbrace{\frac{p^4}{h^2} + \frac{p^3}{h^3}}_{\text{build stage}} + \underbrace{\frac{p^3}{h^2} + \frac{p^2}{h^2}\log\frac{1}{h^2}}_{\text{solve stage}} \approx \frac{p^4}{h^2} + \frac{p^3}{h^3} \approx Np^2 + N^{3/2}.$$

Specifically, we propose a variant of the HPS scheme that employs the ultraspherical spectral method [36,44] instead of spectral collocation for element-wise discretization. The method retains sparsity in the high-p regime by carefully selecting bases to be specific families of orthogonal polynomials and employing sparse recurrence relations between them. The discretization is not nodal, but modal; that is, the unknowns are not values on a grid, but coefficients in a polynomial expansion.

In this work, we are interested in solving linear PDEs on two-dimensional meshed geometries with Dirichlet boundary conditions,[1] i.e.,

$$\begin{aligned} \mathcal{L}u(x,y) &= f(x,y) \quad \text{in } \Omega, \\ u(x,y) &= g(x,y) \quad \text{on } \partial\Omega. \end{aligned} \tag{1.1}$$

Here, Ω is a domain in $\mathbb{R}^2$, f and g are given functions defined on Ω and its boundary, and $\mathcal{L}$ is a variable-coefficient, second-order, elliptic partial differential operator (PDO) of the form

$$\mathcal{L}u = \nabla\cdot(A(x,y)\nabla u) + \nabla\cdot(b(x,y)u) + c(x,y)u, \tag{1.2}$$

with $A(x,y) \in \mathbb{C}^{2\times 2}$, $b(x,y) \in \mathbb{C}^2$, and $c(x,y) \in \mathbb{C}$.

The paper is structured as follows. In section 2, we review the ultraspherical spectral method, a sparse and spectrally-accurate method for solving linear ODEs and PDEs on rectangular domains, and discuss its application to quadrilateral and triangular domains. In section 3, we extend this spectral method to the non-overlapping domain decomposition setting, highlighting the differences from traditional collocation-based patching approaches. We describe how the hierarchical merging of Poincaré–Steklov operators efficiently performs domain decomposition on meshes with many elements. In section 4, we present an implementation of the ultraspherical SEM in the software package `ultraSEM`, and briefly describe its syntax and design. In section 5, we present numerical results and applications of the method.

[1] Robin boundary conditions can be converted to equivalent Dirichlet boundary conditions using the Dirichlet-to-Neumann operators constructed by the HPS scheme, and so we focus on Dirichlet boundary conditions throughout the paper.

2. Background material

2.1. The ultraspherical spectral method

First, we review the fundamental ideas in the ultraspherical spectral method [36], which in one dimension solves linear ordinary differential equations (ODEs) with variable coefficients of the form

$$\sum_{\lambda=0}^{M} a_\lambda(x)\frac{d^\lambda u}{dx^\lambda} = f(x), \qquad x \in [-1,1], \tag{2.1}$$

along with general linear boundary conditions $\mathcal{B}u = \mathbf{g} \in \mathbb{C}^M$ to ensure that there is a unique solution. For an integer p, the method seeks to approximate the first $p+1$ Chebyshev expansion coefficients $\{u_j\}_{j=0}^{p}$ of the solution u, where

$$u(x) = \sum_{j=0}^{\infty} u_j T_j(x), \qquad x \in [-1,1],$$

and $T_j(x) = \cos(j\cos^{-1}x)$ is the degree-j Chebyshev polynomial of the first kind.

Classical spectral methods represent differentiation as a dense operator [10,45], but the ultraspherical spectral method employs the "sparse" recurrence relations

$$\frac{d^\lambda T_j}{dx^\lambda} = \begin{cases} 2^{\lambda-1} j(\lambda-1)!\, C_{j-\lambda}^{(\lambda)}, & j \geq \lambda, \\ 0, & 0 \leq j \leq \lambda-1, \end{cases} \tag{2.2}$$

where $C_j^{(\lambda)}$ is the degree-j ultraspherical polynomial of parameter $\lambda > 0$ [35, Sec. 18.3]. This results in a sparse representation of differentiation operators. In particular, the differentiation operator for the λth derivative is given by

$$\mathcal{D}_\lambda = 2^{\lambda-1}(\lambda-1)! \begin{pmatrix} \overbrace{0 \;\cdots\; 0}^{\lambda \text{ times}} & \lambda & & & \\ & & \lambda+1 & & \\ & & & \lambda+2 & \\ & & & & \ddots \end{pmatrix}, \qquad \lambda \in \mathbb{N} \setminus \{0\}.$$

For $\lambda \geq 1$, the matrix $\mathcal{D}_\lambda$ maps a vector of Chebyshev coefficients to a vector of $C^{(\lambda)}$ coefficients of the λth derivative. For convenience, we use $\mathcal{D}_0$ to denote the identity operator.

Since $\mathcal{D}_\lambda$ returns a vector of ultraspherical coefficients for $\lambda \geq 1$, operators to convert between the Chebyshev and ultraspherical bases are required. Let $\mathcal{S}_0$ be the operator that converts a vector of Chebyshev coefficients to a vector of $C^{(1)}$ coefficients, and let $\mathcal{S}_\lambda$, for $\lambda \geq 1$, be the operator that converts a vector of $C^{(\lambda)}$ coefficients to a vector of $C^{(\lambda+1)}$ coefficients. Using the recurrence relations [35, (18.9.7) & (18.9.9)]

$$T_j = \begin{cases} \frac{1}{2}\left(C_j^{(1)} - C_{j-2}^{(1)}\right), & j \geq 2, \\ \frac{1}{2}C_1^{(1)}, & j = 1, \\ C_0^{(1)}, & j = 0, \end{cases} \qquad C_j^{(\lambda)} = \begin{cases} \frac{\lambda}{\lambda+j}\left(C_j^{(\lambda+1)} - C_{j-2}^{(\lambda+1)}\right), & j \geq 2, \\ \frac{\lambda}{\lambda+1}C_1^{(\lambda+1)}, & j = 1, \\ C_0^{(\lambda+1)}, & j = 0, \end{cases}$$

it can be shown that the conversion operators $\mathcal{S}_0$ and $\mathcal{S}_\lambda$ are sparse and given by [36]

$$\mathcal{S}_0 = \begin{pmatrix} 1 & 0 & -\frac{1}{2} & & \\ & \frac{1}{2} & 0 & -\frac{1}{2} & \\ & & \frac{1}{2} & 0 & \ddots \\ & & & \frac{1}{2} & \ddots \\ & & & & \ddots \end{pmatrix}, \quad \mathcal{S}_\lambda = \begin{pmatrix} 1 & 0 & -\frac{\lambda}{\lambda+2} & & \\ & \frac{\lambda}{\lambda+1} & 0 & -\frac{\lambda}{\lambda+3} & \\ & & \frac{\lambda}{\lambda+2} & 0 & \ddots \\ & & & \frac{\lambda}{\lambda+2} & \ddots \\ & & & & \ddots \end{pmatrix}, \quad \lambda \geq 1.$$

To represent multiplication by the variable coefficients $a_\lambda(x)$ in (2.1), multiplication operators $\mathcal{M}_\lambda[a_\lambda]$ for $C^{(\lambda)}$ coefficients[2] can be explicitly constructed. If $a_\lambda(x)$ is approximated by a degree-m_λ polynomial, then the operator $\mathcal{M}_\lambda[a_\lambda]$ is m_λ-banded [36].

[2] The multiplication operator for $\lambda = 0$, $\mathcal{M}_0[a_0]$, acts on a vector of Chebyshev coefficients.

Discretizing (2.1) using these operators to represent differentiation, conversion between bases, and multiplication by variable coefficients results in a banded $(p+1)\times(p+1)$ linear system given by

$$\left(\mathcal{M}_M[a_M]\mathcal{D}_M + \sum_{\lambda=0}^{M-1} \mathcal{S}_{M-1}\cdots\mathcal{S}_\lambda \mathcal{M}_\lambda[a_\lambda]\mathcal{D}_\lambda\right)\mathbf{u} = \mathcal{S}_{M-1}\cdots\mathcal{S}_0\mathbf{f}, \tag{2.3}$$

where $\mathbf{u}$ and $\mathbf{f}$ are vectors of Chebyshev coefficients of u and f, respectively. Note that since the order-M differential operator in (2.3) maps the vector of Chebyshev coefficients $\mathbf{u}$ to $C^{(M)}$ coefficients, the vector of Chebyshev coefficients $\mathbf{f}$ must also be converted to $C^{(M)}$ coefficients. The bandwidth of the linear system in (2.3) scales as $\mathcal{O}(\max_\lambda m_\lambda)$, independent of the polynomial order p. If the variable coefficients $a_\lambda(x)$ can be approximated by polynomials such that $m_\lambda \ll p$, then (2.3) is a sparse linear system.

To impose the boundary constraints given by $\mathcal{B}$, we must encode $\mathcal{B}$ in terms of its action on a vector of Chebyshev coefficients. For Dirichlet boundary conditions on $[-1,1]$, such action is given by

$$\mathcal{B} = \begin{pmatrix} T_0(-1) & T_1(-1) & \cdots & T_p(-1) \\ T_0(1) & T_1(1) & \cdots & T_p(1) \end{pmatrix} = \begin{pmatrix} 1 & -1 & \cdots & (-1)^p \\ 1 & 1 & \cdots & 1 \end{pmatrix}, \tag{2.4}$$

because $\mathcal{B}\mathbf{u} \approx (u(-1), u(1))^T$. Neumann, Robin, and more general boundary constraints can be similarly encoded. To impose the M boundary conditions $\mathcal{B}\mathbf{u} = \mathbf{g}$ on the linear system (2.3), the ultraspherical spectral method uses boundary bordering [10], wherein the last M rows of the linear system are replaced by dense rows that impose constraints on the Chebyshev coefficients of the solution (e.g., (2.4) for Dirichlet boundary conditions). The resulting $(p+1)\times(p+1)$ linear system has a distinctive almost banded[3] structure with bandwidth $m = \max_\lambda m_\lambda$ and can be solved in $\mathcal{O}(m^2 p)$ operations using the adaptive QR algorithm [36] or the Woodbury formula. Fig. 2.1 (left) shows the almost banded structure typical of the linear systems in the ultraspherical spectral method.

The ultraspherical spectral method can be extended to solve PDEs in two dimensions on rectangular domains [44]. For the PDE given in (1.1) and for a polynomial order p, the method computes modes $X \in \mathbb{C}^{(p+1)\times(p+1)}$ of the solution $u(x,y)$ in a bivariate tensor-product Chebyshev basis, such that

$$u(x,y) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} X_{ij} T_i(y) T_j(x), \qquad (x,y) \in [-1,1]^2.$$

Discretization of the PDE is based on separable models of linear partial differential operators. For example, the elliptic PDO $\mathcal{L}$ given by (1.2) can be decomposed into a sum of tensor products of one-dimensional differential operators

$$\mathcal{L} = \sum_{j=1}^{K} \left(\mathcal{L}_j^y \otimes \mathcal{L}_j^x\right), \tag{2.5}$$

where $\mathcal{L}_1^y, \ldots, \mathcal{L}_K^y$ are operators associated with ODEs in y, $\mathcal{L}_1^x, \ldots, \mathcal{L}_K^x$ are operators associated with ODEs in x. In (2.5), the tensor product operator '$\otimes$' is defined such that if $u(x,y) = v(y)w(x)$, then

$$\left(\mathcal{L}^y \otimes \mathcal{L}^x\right) u(x,y) = \left(\mathcal{L}^y v(y)\right)\left(\mathcal{L}^x w(x)\right)$$

for some operators $\mathcal{L}^y$ and $\mathcal{L}^x$. Such separable representations of PDOs can be automatically computed [44]. The univariate differential operators $\mathcal{L}_1^y, \ldots, \mathcal{L}_K^y, \mathcal{L}_1^x, \ldots, \mathcal{L}_K^x$ can each be discretized using the ultraspherical spectral method in one dimension, and boundary conditions in x and y can be imposed on the rows and columns of X, thus giving us a scheme for discretizing the PDE. The resulting linear system of size $(p+1)^2 \times (p+1)^2$ is almost block-banded with a bandwidth of $\mathcal{O}(mp)$ and $\mathcal{O}(mp)$ dense rows, where $m = \max\{m_1^y, \ldots, m_K^y, m_1^x, \ldots, m_K^x\}$ and $m_1^y, \ldots, m_K^y, m_1^x, \ldots, m_K^x$ are the bandwidths of the discretized operators $\mathcal{L}_1^y, \ldots, \mathcal{L}_K^y, \mathcal{L}_1^x, \ldots, \mathcal{L}_K^x$. This can be solved in $\mathcal{O}(p^4)$ operations. In special cases, e.g., where $K=1$ or $K=2$, further structure can be exploited to arrive at faster solvers [18,44].

2.2. Spectral methods on quadrilaterals and triangles

Global spectral methods defined on rectangles can be used on other polygons through coordinate transformation. Let $\mathcal{Q}_{\text{ref}} = [-1,1]^2$ be the reference square with vertices given by $(r_0,s_0) = (-1,-1)$, $(r_1,s_1) = (1,-1)$, $(r_2,s_2) = (1,1)$, $(r_3,s_3) = (-1,1)$. Denote by (r,s) the coordinates in reference space and by (x,y) the coordinates in real space, and suppose we have a mapping from reference space to real space, $(r,s) \mapsto (x,y)$. To apply a global spectral method on $\mathcal{Q}_{\text{ref}}$ to a PDE defined in real space, the differential operator $\mathcal{L}$ and right-hand side $f(x,y)$ are transformed into reference space. The

[3] A matrix is *almost banded* if it is banded except for a small number of columns or rows.

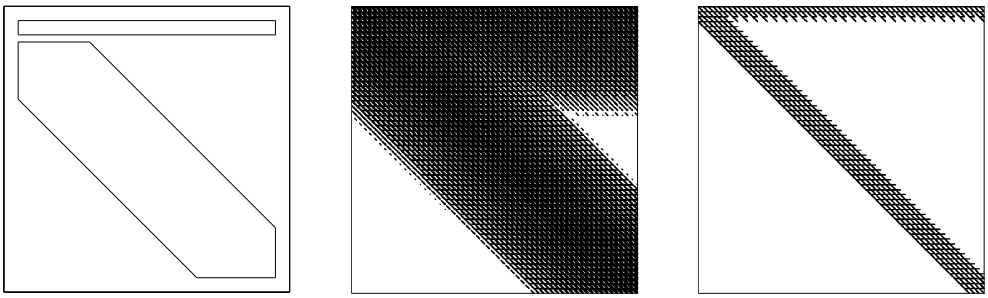


Fig. 2.1. (Left) Typical structure of the almost banded matrices constructed by the ultraspherical spectral method, i.e., banded matrices except for a small number of dense rows. In one dimension, ODEs are discretized as almost banded $(p+1)\times(p+1)$ linear systems with bandwidth m independent of p and $\mathcal{O}(1)$ dense rows; such systems can be solved in $\mathcal{O}(p)$ operations. In two dimensions on $[-1,1]^2$, PDEs are discretized as almost block-banded $(p+1)^2\times(p+1)^2$ linear systems, with a bandwidth of $\mathcal{O}(mp)$ and $\mathcal{O}(mp)$ dense rows; such systems can be solved in $\mathcal{O}(p^4)$ operations. (Center) In two dimensions on a quadrilateral, PDEs are transformed to $[-1,1]^2$ and then discretized. When Jacobian factors are kept as rational functions (see subsection 2.2), the discrete differential operator has a large bandwidth. (Right) By scaling the transformed PDE by a power of the determinant of the Jacobian, the discrete differential operator remains sparse. This improvement is most notable when the quadrilaterals involved have small angles.

coordinate transformation alters the differential operator via the chain rule. For a function $u(r,s)$ defined on $\mathcal{Q}_{\text{ref}}$, first- and second-order derivatives in x and y are given by

$$\begin{aligned}
u_x &= r_x u_r + s_x u_s,\\
u_y &= r_y u_r + s_y u_s,\\
u_{xx} &= (r_x)^2 u_{rr} + 2r_x s_x u_{rs} + (s_x)^2 u_{ss} + r_{xx} u_r + s_{xx} u_s,\\
u_{xy} &= r_x r_y u_{rr} + (r_x s_y + r_y s_x) u_{rs} + s_x s_y u_{ss} + r_{xy} u_r + s_{xy} u_s,\\
u_{yy} &= (r_y)^2 u_{rr} + 2r_y s_y u_{rs} + (s_y)^2 u_{ss} + r_{yy} u_r + s_{yy} u_s,
\end{aligned}$$

where the Jacobian factors $r_x, r_{xx}, \ldots$ depend on the coordinate mapping. In this paper, we are interested in mappings from $\mathcal{Q}_{\text{ref}}$ to quadrilaterals or triangles.

For a quadrilateral domain $\mathcal{Q}$ with vertices $(x_0,y_0),\ldots,(x_3,y_3)$, a bilinear mapping from $(r,s)\in\mathcal{Q}_{\text{ref}}$ to $(x,y)\in\mathcal{Q}$ is given by

$$\begin{bmatrix} r\\ s\end{bmatrix} \mapsto \begin{bmatrix} a_0^x + a_1^x r + a_2^x s + a_3^x rs\\ a_0^y + a_1^y r + a_2^y s + a_3^y rs\end{bmatrix} = \begin{bmatrix} x\\ y\end{bmatrix},$$

where the coefficients $a_0^x,\ldots,a_3^x$ and $a_0^y,\ldots,a_3^y$ satisfy the linear system

$$\begin{bmatrix} 1 & r_0 & s_0 & r_0 s_0\\ 1 & r_1 & s_1 & r_1 s_1\\ 1 & r_2 & s_2 & r_2 s_2\\ 1 & r_3 & s_3 & r_3 s_3\end{bmatrix}\begin{bmatrix} a_0^x & a_0^y\\ a_1^x & a_1^y\\ a_2^x & a_2^y\\ a_3^x & a_3^y\end{bmatrix} = \begin{bmatrix} x_0 & y_0\\ x_1 & y_1\\ x_2 & y_2\\ x_3 & y_3\end{bmatrix}.$$

While the mapping from (r,s) to (x,y) is bilinear, the mapping from (x,y) to (r,s) is more complicated and in particular is not polynomial, and so we would like to avoid directly computing the inverse maps $r(x,y)$ and $s(x,y)$. Therefore, to compute the first-order Jacobian factors r_x, s_x, r_y, and s_y, we apply the inverse function theorem to the Jacobian matrix $J_{rs} = \partial(r,s)/\partial(x,y)$, which states that $J_{rs} = \left(J_{xy}\right)^{-1}$ with $J_{xy} = \partial(x,y)/\partial(r,s)$. Writing out the Jacobians explicitly, we obtain the following formulae for the first-order factors r_x, s_x, r_y, and s_y:

$$\begin{bmatrix} r_x & r_y\\ s_x & s_y\end{bmatrix} = \begin{bmatrix} x_r & x_s\\ y_r & y_s\end{bmatrix}^{-1} = \frac{1}{\det(J_{xy})}\begin{bmatrix} y_s & -x_s\\ -y_r & x_r\end{bmatrix},$$

where $\det(J_{xy}) = x_r y_s - x_s y_r$. Applying the chain rule to these definitions yields formulae for the second-order factors r_{xx}, r_{xy}, r_{yy}, s_{xx}, s_{xy}, and s_{yy}.

However, note that the Jacobian factors are rational functions, due to factors of $\det(J_{xy})$, $\det(J_{xy})^2$, and $\det(J_{xy})^3$ in the denominators of the first- and second-order terms. Thus, the coordinate transformation from $\mathcal{Q}$ to $\mathcal{Q}_{\text{ref}}$ introduces rational variable coefficients into the differential operator, and the discretization of the transformed operator by the ultraspherical spectral method results in a linear system with large bandwidth (see Fig. 2.1 (center)). To recover sparsity, we scale the transformed differential operator $\mathcal{L}_{rs}$ and right-hand side $f(r,s)$ by the factor $\det(J_{xy})^3$ [48], and discretize the scaled PDE

$$\underbrace{\left(\det(J_{xy})^3 \mathcal{L}_{rs}\right)}_{\widehat{\mathcal{L}}_{rs}} u(r,s) = \underbrace{\det(J_{xy})^3 f(r,s)}_{\widehat{f}}.$$

As all Jacobian factors can be written with denominator $\det(J_{xy})^3$, this scaling turns the rational variable coefficients induced by the transformation into polynomial variable coefficients of degree ≤ 3 (see Fig. 2.1 (right)). Thus, PDEs on $\mathcal{Q}$ with degree-m variable coefficients are transformed into PDEs on $\mathcal{Q}_{\text{ref}}$ with degree-$(m+3)$ variable coefficients.

For a triangular domain $\mathcal{T}$, the Duffy transformation [15,42] may be used to define a mapping from $\mathcal{Q}_{\text{ref}}$ to $\mathcal{T}$ by collapsing one side of $\mathcal{Q}_{\text{ref}}$ to a point. Let $\mathcal{T}_{\text{ref}}$ be the reference triangle with vertices $(x_0, y_0) = (0,0)$, $(x_1, y_1) = (1,0)$, and $(x_2, y_2) = (0,1)$. A mapping from $(r,s) \in \mathcal{Q}_{\text{ref}}$ to $(x,y) \in \mathcal{T}_{\text{ref}}$ can be defined by

$$\begin{bmatrix} r \\ s \end{bmatrix} \mapsto \begin{bmatrix} \frac{1}{4}(1+r)(1-s) \\ \frac{1}{2}(1-s) \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix},$$

which maps the line segment between $(-1,1)$ and $(1,1)$ in $\mathcal{Q}_{\text{ref}}$ to the point $(0,1)$ in $\mathcal{T}_{\text{ref}}$. The inverse of this transformation, mapping from $(x,y) \in \mathcal{T}_{\text{ref}}$ to $(r,s) \in \mathcal{Q}_{\text{ref}}$, possesses a singularity at the point $(0,1)$, i.e.,

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} 2x/(1-y) - 1 \\ 2y - 1 \end{bmatrix} = \begin{bmatrix} r \\ s \end{bmatrix}.$$

If discretized directly, Jacobian factors based on this transformation introduce singular variable coefficients into the differential operator when the operator is transformed to $\mathcal{T}_{\text{ref}}$. However, the singularity induced by the Duffy transformation may be removed by scaling the PDE by powers of $1-y$. For a general triangular domain $\mathcal{T}$ with vertices $(x_0, y_0), \ldots, (x_2, y_2)$, the Duffy transformation may be composed with an affine transformation of the form

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} x_0 + (x_1 - x_0)\, x + (x_2 - x_0)\, y \\ y_0 + (y_1 - y_0)\, x + (y_2 - y_0)\, y \end{bmatrix}$$

to yield a mapping from $\mathcal{Q}_{\text{ref}}$ to $\mathcal{T}$.

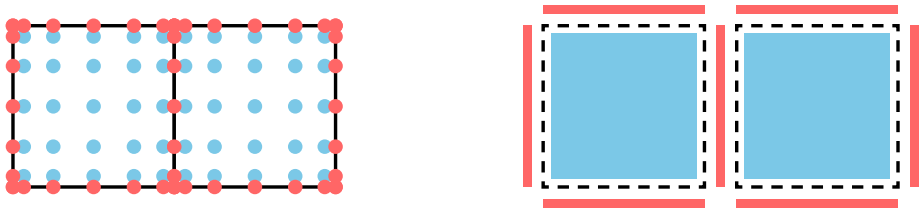
We focus our attention on straight-sided quadrilateral elements in the remainder of this work. However, the algorithms presented below can be applied to triangular elements through simple modifications. The `ultraSEM` software supports both triangular and quadrilateral elements.

3. The ultraspherical spectral element method

We now describe how to adapt the ultraspherical spectral method into an SEM, focusing on key implementation aspects. Our method is based on the hierarchical Poincaré–Steklov scheme, an efficient non-overlapping domain decomposition approach [5,22,23,30,31]. We employ a variant of the HPS scheme to handle irregular, non-tensor-product meshes (see subsection 3.4). Broadly, our method is the following:

1. The method takes as input a second-order elliptic PDO $\mathcal{L}$, a right-hand side f, Dirichlet data g, and a mesh with elements $\{\mathcal{E}_i\}_{i=1}^{n_{\text{elem}}}$.
2. On each element, two local operators are constructed: (i) a solution operator, which computes the local solution to the PDE on the element when given Dirichlet data, and (ii) a Dirichlet-to-Neumann operator, which computes the outward flux of the local solution when given Dirichlet data (see subsection 3.3.1).
3. Local elemental operators are merged pairwise in a hierarchical fashion, yielding solution operators and Dirichlet-to-Neumann operators, which act on the interfaces between elements or groups of elements. Merging continues until a single global solution operator is computed for the entire mesh (see subsection 3.3.1).
4. The given Dirichlet data g is passed in at the top level. Solution operators are applied down the tree, providing the solution at unknown interfaces between elements (see subsection 3.3.2).
5. Once the solution is known at all the interfaces, local solution operators are applied on each element to determine the interior solution over the entire mesh.

The method naturally lends itself to parallelization. Specifically, steps 2 and 5 can be performed independently on each element as the computations involved are entirely decoupled. Moreover, step 2 is often the bottleneck when p is large, and so significant speedups may be gained if parallelism is exploited (see subsection 3.4). The hierarchical steps 3 and 4 may also be parallelized, as the operations taking place on two branches in the hierarchy are decoupled until the two branches are merged. Thus, a careful, load-balanced strategy for parallelizing across branches in the hierarchy may lead to further speedups.



(a) Nodal discretization (b) Modal discretization

Fig. 3.1. Two interpretations of non-overlapping domain decomposition for nodal and modal discretizations, with interface data (red) and interior data (blue). (a) In a nodal discretization, neighboring elements communicate directly through degrees of freedom at nodes, which can be partitioned into shared interface nodes and local interior nodes. (b) In a modal discretization, neighboring elements communicate indirectly through unshared interface functions, allowing for coefficients in a modal discretization to be spatially separated. (For color versions of the figures, the reader is referred to the web version of this article.)

3.1. Domain decomposition for modal discretizations

Adapting a domain decomposition approach such as the HPS scheme—originally formulated around a spectral collocation method [30,31]—to a modal discretization such as the ultraspherical spectral method gives rise to a few subtleties. In the nodal setting, values along interfaces are inherently shared between elements, allowing for an intuitive way to separate the nodes in each element into "interior" and "interface" degrees of freedom and solve for them accordingly (see Fig. 3.1a). Cross point conditions (e.g., at a point where the corners of four quadrilaterals meet) can then be avoided by removing the degrees of freedom located at cross points [6]. In the modal setting, on the other hand, the coefficients in a bivariate Chebyshev expansion are not spatially localized, and therefore do not intuitively separate into such categories. To regain a decoupling for Chebyshev coefficients, it is helpful to think about bivariate functions on each element communicating not with each other directly, but with univariate functions on each interface (see Fig. 3.1b). Using a modal discretization for these bivariate interior functions and univariate interface functions then allows Chebyshev coefficients to be separated as before. Cross point conditions must then be imposed directly for the resulting linear systems to be nonsingular (see subsection 3.3.1).

An alternative remedy to localize modal discretizations is to use a basis that has intrinsic spatial separation between interior and interface, such as a basis consisting of bubble functions (functions that are zero on the edges of an element) and edge functions (functions that are nonzero on the edges of an element) [42]. However, such a basis may not yield a sparse discretization of the PDE. We choose to use the ultraspherical basis to obtain sparse linear algebra, which affords our method a lower computational complexity with respect to p.

3.2. Model problem: two "glued" squares

To begin, we consider the simple domain decomposition setting of two square-shaped elements that are "glued" together. That is, we wish to use the ultraspherical spectral method to solve the patching problem[4]

$$\begin{aligned}
\nabla^2 u_1 &= f_1 && \text{in } \mathcal{E}_1,\\
\nabla^2 u_2 &= f_2 && \text{in } \mathcal{E}_2,\\
u_1 &= g_1 && \text{on } \partial\mathcal{E}_1 \cap \partial\Omega,\\
u_2 &= g_2 && \text{on } \partial\mathcal{E}_2 \cap \partial\Omega,\\
u_1 &= u_2 && \text{on } \Gamma,\\
\tfrac{\partial u_1}{\partial \boldsymbol{n}_1} + \tfrac{\partial u_2}{\partial \boldsymbol{n}_2} &= 0 && \text{on } \Gamma,
\end{aligned} \tag{3.1}$$

where $\mathcal{E}$ is a mesh of the domain $\Omega = [-2,2] \times [-1,1]$ with elements $\mathcal{E}_1 = [-2,0] \times [-1,1]$ and $\mathcal{E}_2 = [0,2] \times [-1,1]$, Γ is the interface between the two elements, f and g are given functions, and $f_i = f|_{\mathcal{E}_i}$ for any function f. This model problem of a pairwise merge serves as a building block in the HPS scheme. The problem setup is depicted in Fig. 3.2.

The patching problem (3.1) couples two three-sided Dirichlet problems via continuity conditions across the interface Γ. Equivalently, (3.1) can be regarded as two decoupled, four-sided Dirichlet problems when given a suitable piece of Dirichlet data along Γ. That is, there exists an interface function φ such that (3.1) is equivalent to

[4] It is worth noting that this formulation is equivalent to the global problem $\nabla^2 u = f$ in Ω, $u = g$ on $\partial\Omega$, for any domain Ω. This holds for any second-order linear elliptic boundary value problem [12].

$$\begin{aligned} \nabla^2 u_1 &= f_1 \quad \text{in } \mathcal{E}_1, & \nabla^2 u_2 &= f_2 \quad \text{in } \mathcal{E}_2, \\ u_1 &= g_1 \quad \text{on } \partial\mathcal{E}_1 \cap \partial\Omega, & u_2 &= g_2 \quad \text{on } \partial\mathcal{E}_2 \cap \partial\Omega, \\ u_1 &= \varphi \quad \text{on } \Gamma, & u_2 &= \varphi \quad \text{on } \Gamma. \end{aligned} \tag{3.2}$$

To determine this unknown interface function φ, we aim to build a direct solver—an operator S_Γ such that $\varphi = S_\Gamma g$—using ingredients from local operators on each element. In particular, we construct local direct solvers on $\mathcal{E}_1$ and $\mathcal{E}_2$, and then use pieces of these operators to construct the interfacial solution operator S_Γ. Once the interface function φ is found, the two subproblems in (3.2) decouple and can be solved independently by applying local direct solvers on $\mathcal{E}_1$ and $\mathcal{E}_2$. By building a direct solver for the global interface problem based on direct solvers for the subproblems in (3.2), the generalization to multiple elements follows naturally.

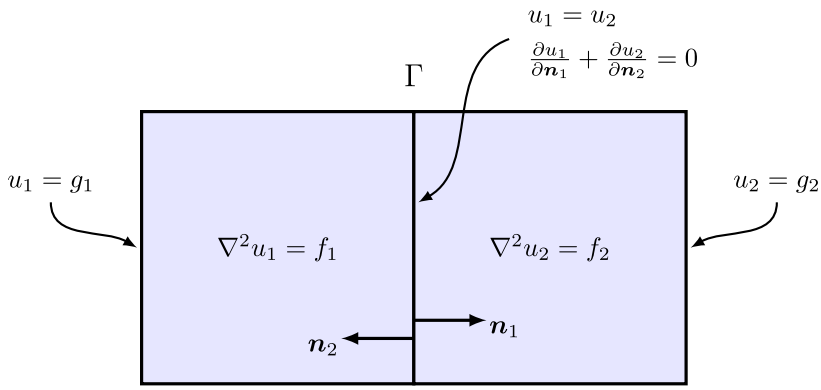


Fig. 3.2. The canonical problem setup for two "glued" squares.

3.2.1. *Constructing local operators*

To construct a direct solver for (3.2), we first build operators that encode how to solve the PDE locally on elements $\mathcal{E}_1$ and $\mathcal{E}_2$. Such operators, called solution operators, take in Dirichlet data and return the corresponding solution to the PDE on an element. For a quadrilateral domain, the solution operator takes in four univariate functions—representing four sides of Dirichlet data—and returns a bivariate function that satisfies the PDE (see Fig. 3.3a).

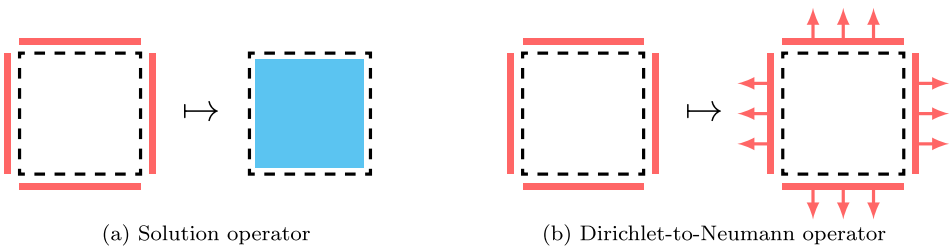


Fig. 3.3. A visualization of the local operators computed for each element. (a) The solution operator on a quadrilateral takes in four univariate functions of Dirichlet data and returns a bivariate function that solves the PDE using the given boundary conditions. (b) The Dirichlet-to-Neumann operator on a quadrilateral takes in four univariate functions of Dirichlet data and returns four univariate functions of Neumann data, representing the normal derivative of the solution to the PDE on the four sides.

We use the ultraspherical spectral method for solving PDEs on quadrilaterals (see subsection 2.2). If on each element we employ a $(p+1) \times (p+1)$ coefficient discretization for the solution so that the solution is at most a degree-(p, p) polynomial, then the solution operator $S_{\mathcal{E}} \in \mathbb{C}^{(p+1)^2 \times 4(p+1)+1}$ on element $\mathcal{E}$ is a dense matrix. For a column vector $\boldsymbol{c} \in \mathbb{C}^{4(p+1)}$ and scalar $\alpha \in \mathbb{C}$, the product $S_{\mathcal{E}} \left[\begin{smallmatrix} \boldsymbol{c} \\ \alpha \end{smallmatrix}\right] \in \mathbb{C}^{(p+1)^2}$ represents the $(p+1) \times (p+1)$ Chebyshev coefficients of the solution to the PDE on $\mathcal{E}$ with Dirichlet data $\boldsymbol{c}$ and right-hand side αf.[5] Here, $\boldsymbol{c} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4]^T$ represents the Chebyshev coefficients of four univariate functions of Dirichlet data on the left, right, bottom, and top of $\mathcal{E}$, respectively, each discretized with $p+1$ coefficients. For example, on the left side, the coefficients $\boldsymbol{c}_1 \in \mathbb{C}^{p+1}$ define the degree-p boundary function $h_1(y)$ as

[5] In practice, we always take $\alpha = 1$.

$$h_1(y) = \sum_{j=0}^{p} (\boldsymbol{c}_1)_j T_j(y).$$

Similarly, $\boldsymbol{c}_2$, $\boldsymbol{c}_3$, and $\boldsymbol{c}_4$ define functions on the other three sides.

The solution operator on $\mathcal{E}$ can be decomposed into four operators $S^1_{\mathcal{E}}, S^2_{\mathcal{E}}, S^3_{\mathcal{E}}, S^4_{\mathcal{E}} \in \mathbb{C}^{(p+1)^2 \times (p+1)}$ that account for the homogeneous part of the solution and one column vector $S^{\text{rhs}}_{\mathcal{E}} \in \mathbb{C}^{(p+1)^2 \times 1}$ that accounts for the particular part of the solution.[6] That is,

$$S_{\mathcal{E}} = \left[\begin{array}{cccc|c} S^1_{\mathcal{E}} & S^2_{\mathcal{E}} & S^3_{\mathcal{E}} & S^4_{\mathcal{E}} & S^{\text{rhs}}_{\mathcal{E}} \end{array}\right],$$

where the vector $S^{\text{rhs}}_{\mathcal{E}}$ is defined by

$$S^{\text{rhs}}_{\mathcal{E}} = \text{vec}(X), \qquad u_{\text{rhs}}(x, y) = \sum_{i=0}^{p} \sum_{j=0}^{p} X_{ij} T_i(y) T_j(x),$$

and $\text{vec}(\cdot)$ is the column-wise vectorization operator. Here, u_{rhs} satisfies the PDE on $\mathcal{E}$ with homogeneous boundary conditions, i.e.,

$$\nabla^2 u_{\text{rhs}} = f|_{\mathcal{E}}, \qquad u_{\text{rhs}}|_{\partial\mathcal{E}} = 0.$$

The products $S^i_{\mathcal{E}} \boldsymbol{c}_i$ represent the $(p+1) \times (p+1)$ Chebyshev coefficients of the approximate solution to the homogeneous problem (i.e., $f = 0$) with Dirichlet data on side i given by the Chebyshev coefficients $\boldsymbol{c}_i$ and zero Dirichlet data on the other three sides. Thus, the solution operator $S_{\mathcal{E}}$ depends on the domain $\mathcal{E}$, the PDO, and the right-hand side f, but not the Dirichlet data g. However, the solution operator can be efficiently updated if f is changed (see the discussion of `updateRHS` in section 4).

We construct the matrices $S^i_{\mathcal{E}}$ column-by-column. To construct the jth column of $S^i_{\mathcal{E}}$, we set the jth Dirichlet coefficient to one and the rest to zero, i.e.,

$$(\boldsymbol{c}_i)_k = \begin{cases} 1, & \text{if } k = j, \\ 0, & \text{otherwise,} \end{cases} \tag{3.3}$$

for $0 \le k \le p$. We wish to solve the PDE using this Dirichlet data for each $0 \le j \le p$ to obtain the $(p+1) \times (p+1)$ coefficients of the solution, which are reshaped and placed as a column into $S^i_{\mathcal{E}}$. That is, the jth column of the solution operator for the ith side of the element $\mathcal{E}$, i.e., $(S^i_{\mathcal{E}})_{:,j}$, is constructed as

$$(S^i_{\mathcal{E}})_{:,j} = \text{vec}(X), \qquad v_j(x, y) = \sum_{k=0}^{p} \sum_{\ell=0}^{p} X_{k\ell} T_k(y) T_\ell(x),$$

where v_j approximately satisfies the following homogeneous PDE:

$$\nabla^2 v_j = 0, \qquad v_j|_{\partial\mathcal{E}} = \begin{cases} T_j, & \text{on side } i, \\ 0, & \text{otherwise.} \end{cases}$$

Unfortunately, the Dirichlet data used in this construction process may have discontinuities at the corners of the domain, leading to incompatible boundary conditions. To ensure compatibility is satisfied, we orthogonally project each function $\boldsymbol{c}_i$ onto the space of functions that are continuous at the corners before solving the PDE. The compatibility conditions at the four corners of the quadrilateral can be encoded into a matrix $B \in \mathbb{C}^{4 \times 4(p+1)}$ given by

$$B = \begin{array}{c} \left[\begin{array}{cccc} B_{-1} & 0 & -B_{-1} & 0 \\ B_{+1} & 0 & 0 & -B_{-1} \\ 0 & B_{-1} & -B_{+1} & 0 \\ 0 & B_{+1} & 0 & -B_{+1} \end{array}\right] \end{array} \begin{array}{l} \}\text{ bottom left corner} \\ \}\text{ top left corner} \\ \}\text{ bottom right corner} \\ \}\text{ top right corner} \end{array}$$

with

$$B_{\pm 1} = \begin{bmatrix} T_0(\pm 1) & T_1(\pm 1) & \cdots & T_p(\pm 1) \end{bmatrix},$$

[6] Although including the particular solution in the solution operator is not typically standard in the HPS literature, we do this here because it avoids repeating a description of the linear algebra when constructing the particular solution separately and matches how `ultraSEM` is implemented.

where $T_j(\pm 1) = (\pm 1)^j$. The matrix $B_{\pm 1}$ is an evaluation operator at the endpoints of the interval $[-1, 1]$. So, for the functions h_i defined above, $B_{\pm 1}\boldsymbol{c}_i = h_i(\pm 1)$. A given piece of boundary data defined by the coefficients $\boldsymbol{c} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4]$ is compatible at the corners if and only if $B\boldsymbol{c} = 0$. To project the boundary data so that it satisfies compatibility, we build a basis for null(B), which is of rank $4(p+1) - 4$. Taking the singular value decomposition $B = U\Sigma V^*$ and letting $\tilde{V}$ be the last $4(p+1) - 4$ columns of V, we construct a projection matrix $P = \tilde{V}\tilde{V}^*$. Since this projection matrix depends only on p, it can be precomputed and stored. The product $\tilde{\boldsymbol{c}} = P\boldsymbol{c}$ orthogonally projects the functions defined by $\boldsymbol{c}_1$, $\boldsymbol{c}_2$, $\boldsymbol{c}_3$, and $\boldsymbol{c}_4$ onto the space of compatible boundary conditions, so that $\tilde{\boldsymbol{c}}_1$, $\tilde{\boldsymbol{c}}_2$, $\tilde{\boldsymbol{c}}_3$, and $\tilde{\boldsymbol{c}}_4$ are continuous at the four corners of the quadrilateral. We apply this projection during the construction process to the Dirichlet data $\boldsymbol{c}$ in (3.3) to obtain compatible Dirichlet data $\tilde{\boldsymbol{c}}$. It is this Dirichlet data that we use to construct the columns of the solution operator $S^i_{\mathcal{E}}$.

Continuity conditions between elements are communicated locally via the Dirichlet-to-Neumann operator, or Poincaré–Steklov operator. The Dirichlet-to-Neumann operator on an element $\mathcal{E}$, denoted by $\Sigma_{\mathcal{E}}$, maps Dirichlet data on each side of $\mathcal{E}$ to the outward flux of the local solution to the PDE on each side of $\mathcal{E}$ (see Fig. 3.3b). One may apply $\Sigma_{\mathcal{E}}$ by first computing the local solution to the PDE on $\mathcal{E}$ for the given Dirichlet data and then evaluating the outward flux of the solution on the boundary. Hence, the Dirichlet-to-Neumann operator can be written as a product of the normal derivative operator and the solution operator. That is, $\Sigma_{\mathcal{E}} = D_{\mathcal{E}} S_{\mathcal{E}}$, where $D_{\mathcal{E}}$ computes the outward flux of a bivariate function on each side of the element $\mathcal{E}$ when given its $(p+1)^2$ Chebyshev coefficients. On the reference square $[-1, 1]^2$, $D_{[-1,1]^2} \in \mathbb{C}^{4(p+1)\times(p+1)^2}$ is given by

$$D_{[-1,1]^2} = \begin{bmatrix} I \otimes D_{-1} \\ I \otimes D_{+1} \\ D_{-1} \otimes I \\ D_{+1} \otimes I \end{bmatrix} \begin{matrix} \}\,\text{left normal derivative} \\ \}\,\text{right normal derivative} \\ \}\,\text{bottom normal derivative} \\ \}\,\text{top normal derivative} \end{matrix}$$

where '$\otimes$' denotes the Kronecker product operator for matrices, I is the $(p+1)\times(p+1)$ identity matrix, and

$$D_{\pm 1} = \pm\begin{bmatrix} T_0'(\pm 1) & T_1'(\pm 1) & \cdots & T_p'(\pm 1) \end{bmatrix}, \qquad T_j'(\pm 1) = (\pm 1)^j j^2.$$

On quadrilaterals and triangles, the normal derivative operator is transformed according to the Jacobian factors described in subsection 2.2. Hence, the Dirichlet-to-Neumann operator $\Sigma_{\mathcal{E}} \in \mathbb{C}^{4(p+1)\times(4(p+1)+1)}$ is a dense matrix. The product $\Sigma_{\mathcal{E}}\left[\begin{smallmatrix}\boldsymbol{c}\\ \alpha\end{smallmatrix}\right] \in \mathbb{C}^{4(p+1)}$ represents the four normal derivatives of the solution to the PDE on the element $\mathcal{E}$ with Dirichlet data $\boldsymbol{c}$ and right-hand side αf,[7] each discretized with $p+1$ Chebyshev coefficients. In the context of the model problem (3.2), the Dirichlet-to-Neumann operators $\Sigma_{\mathcal{E}_1}$ and $\Sigma_{\mathcal{E}_2}$ on the elements $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively, are merged to make the interfacial solution operator S_Γ, allowing for the direct solution of the unknown interface function φ.

3.2.2. Merging two operators

With local operators constructed on each element $\mathcal{E}_1$ and $\mathcal{E}_2$, we now aim to build a global solution operator, S_Γ, from the local operators $S_{\mathcal{E}_1}$, $S_{\mathcal{E}_2}$, $\Sigma_{\mathcal{E}_1}$, and $\Sigma_{\mathcal{E}_2}$, to solve for the unknown interface function φ. Mathematically, this decomposition mimics the classical Schur complement method for domain decomposition, keeping the physical interpretation for modal discretizations from subsection 3.1 in mind.

For elements $\mathcal{E}_1$ and $\mathcal{E}_2$, let Γ_1 and Γ_2 denote the indices of the local Dirichlet data corresponding to the shared boundary Γ. For $\mathcal{E}_1$, since the shared interface Γ is on the right side and the boundary data $\boldsymbol{c} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4]$ is ordered as left, right, bottom, and top, the indices corresponding to the $p+1$ Chebyshev coefficients of the right-side Dirichlet data $\boldsymbol{c}_2$ are given by the set $\Gamma_1 = \{(p+1)+1, \ldots, 2(p+1)\}$. Similarly, since the interface Γ is on the left side of $\mathcal{E}_2$, the indices of the local Dirichlet data on the shared boundary of $\mathcal{E}_2$ are given by $\Gamma_2 = \{1, \ldots, p+1\}$. Finally, denote by L_1 and L_2 the sets containing the indices corresponding to the coefficients of the unshared Dirichlet data on each element, so that $\mathrm{L}_1 = \{1, \ldots, 4(p+1)\} \setminus \Gamma_1$ and $\mathrm{L}_2 = \{1, \ldots, 4(p+1)\} \setminus \Gamma_2$. For a matrix

With these indices defined for $\mathcal{E}_1$ and $\mathcal{E}_2$ based on interaction with the Dirichlet data on Γ, the rows and columns of the local operators $\Sigma_{\mathcal{E}_1}$ and $\Sigma_{\mathcal{E}_2}$ can be partitioned into "interior" and "interface" blocks. The pieces of $\Sigma_{\mathcal{E}_1}$ and $\Sigma_{\mathcal{E}_2}$ that affect the shared interface naturally separate, and a Schur complement may be performed to write down the following $(p+1)\times(p+1)$ linear system for the solution operator on the interface:

$$-\left(\Sigma_{\mathcal{E}_1}^{\Gamma_1,\Gamma_1} + \Sigma_{\mathcal{E}_2}^{\Gamma_2,\Gamma_2}\right) S_\Gamma = \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_1}^{\Gamma_1,\mathrm{L}_1} & \Sigma_{\mathcal{E}_2}^{\Gamma_2,\mathrm{L}_2} & \Sigma_{\mathcal{E}_1}^{\Gamma_1,\mathrm{end}} + \Sigma_{\mathcal{E}_2}^{\Gamma_2,\mathrm{end}} \end{array}\right], \tag{3.4}$$

where the last column of the right-hand side of (3.4) encodes the contribution from the particular solution. Here, superscripts denote row and column indices for slicing a matrix and "end" denotes the index of the last column of a matrix. The linear system in (3.4) has a clear interpretation: the matrix $\Sigma_{\mathcal{E}_1}^{\Gamma_1,\Gamma_1} + \Sigma_{\mathcal{E}_2}^{\Gamma_2,\Gamma_2}$ computes the jump in the normal derivative across the shared interface Γ and enforces this jump to be offset by the contributions from the unshared sides and particular solution, resulting in a discrete analogue of the original continuity condition in (3.1). As before, the merged solution operator $S_\Gamma \in \mathbb{C}^{(p+1)\times(6(p+1)+1)}$ is a dense matrix. For a column vector $\boldsymbol{c} \in \mathbb{C}^{6(p+1)}$ and scalar $\alpha \in \mathbb{C}$, the product $S_\Gamma\left[\begin{smallmatrix}\boldsymbol{c}\\ \alpha\end{smallmatrix}\right] \in \mathbb{C}^{p+1}$

[7] Again, note that our definition of the Dirichlet-to-Neumann operator includes the particular solution.

represents the $p+1$ Chebyshev coefficients of the solution to the PDE on Γ with Dirichlet data $\boldsymbol{c} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_6]^T$ and right-hand side αf, where now the Dirichlet data $\boldsymbol{c}$ is specified on the six sides of the merged domain Ω.

The Schur complement also allows us to write down the Dirichlet-to-Neumann operator for the merged domain. Using the new solution operator S_Γ, we can construct a new Dirichlet-to-Neumann operator on Ω as

$$\Sigma_\Omega = \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_1}^{\mathrm{L}_1,\mathrm{L}_1} & 0 & \Sigma_{\mathcal{E}_1}^{\mathrm{L}_1,\mathrm{end}} \\ 0 & \Sigma_{\mathcal{E}_2}^{\mathrm{L}_2,\mathrm{L}_2} & \Sigma_{\mathcal{E}_2}^{\mathrm{L}_2,\mathrm{end}} \end{array}\right] + \left[\begin{array}{c} \Sigma_{\mathcal{E}_1}^{\mathrm{L}_1,\Gamma_1} \\ \Sigma_{\mathcal{E}_2}^{\mathrm{L}_2,\Gamma_2} \end{array}\right] S_\Gamma, \tag{3.5}$$

where $\Sigma_\Omega \in \mathbb{C}^{6(p+1)\times(6(p+1)+1)}$. The vector $\Sigma_\Omega \left[\begin{smallmatrix} \boldsymbol{c} \\ \alpha \end{smallmatrix}\right]$ represents normal derivatives on the six sides of Ω of the solution to the PDE on Ω with Dirichlet data $\boldsymbol{c}$ and right-hand side αf, each discretized with $p+1$ Chebyshev coefficients.

3.2.3. Computing the solution

We now have all the ingredients we need to compute the solution to (3.1). We begin by converting the given boundary functions, g_1 and g_2, into Chebyshev coefficients. On each of the three sides of $\mathcal{E}_1$ and $\mathcal{E}_2$ where g_1 and g_2 are known, we construct the degree-p Chebyshev approximant to the boundary data and compile the coefficients into vectors $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$ of length $3(p+1)$. Next, to solve for the interface function φ that makes (3.2) equivalent to (3.1), we simply compute the matrix-vector product $S_\Gamma \left[\begin{smallmatrix} \boldsymbol{g} \\ 1 \end{smallmatrix}\right]$, where $\boldsymbol{g} = [\boldsymbol{g}_1, \boldsymbol{g}_2]^T$, which yields the $p+1$ Chebyshev coefficients of φ. With the Dirichlet data now known on all four sides of each of the elements $\mathcal{E}_1$ and $\mathcal{E}_2$, the local solution operators $S_{\mathcal{E}_1}$ and $S_{\mathcal{E}_2}$ can finally be applied. Defining vectors $\widehat{\boldsymbol{g}}_i$ such that $\widehat{\boldsymbol{g}}_i^{\Gamma_i} = \varphi$ and $\widehat{\boldsymbol{g}}_i^{\mathrm{L}_i} = \boldsymbol{g}_i$ for $i = 1, 2$, the matrix-vector products $S_{\mathcal{E}_1} \left[\begin{smallmatrix} \widehat{\boldsymbol{g}}_1 \\ 1 \end{smallmatrix}\right]$ and $S_{\mathcal{E}_2} \left[\begin{smallmatrix} \widehat{\boldsymbol{g}}_2 \\ 1 \end{smallmatrix}\right]$ contain the $(p+1)\times(p+1)$ coefficients of the solutions u_1 and u_2, respectively, satisfying (3.1).

3.3. The hierarchical scheme

At the end of merge process for the model problem of two "glued" squares, we are left with two operators acting on Ω: (1) a solution operator, S_Γ, to solve for the unknown interface inside Ω, and (2) a Dirichlet-to-Neumann operator, Σ_Ω, to map boundary data to outward fluxes on Ω. These operators encode everything we need to know to solve the PDE on Ω. In effect, Ω is now no different from the original elements $\mathcal{E}_1$ or $\mathcal{E}_2$, and so it can be treated as just another element, ready to be merged again with a new domain. After another merge, we are once again in the same situation, with access to local operators that allow us to treat the merged domain as a black box. This is the hierarchical Poincaré–Steklov scheme.

3.3.1. Build stage

For a mesh $\mathcal{E} = \{\mathcal{E}_i\}_{i=1}^{n_{\mathrm{elem}}}$ of a domain Ω, the scheme begins with a local build stage, wherein local solution operators $S_{\mathcal{E}_i}$ and Dirichlet-to-Neumann operators $\Sigma_{\mathcal{E}_i}$ are constructed on each element $\mathcal{E}_i$ according to subsection 3.2.1. The local build process is outlined in Algorithm 3.1a and is referred to as "initialization" in `ultraSEM`. As the operations performed in this stage are local to each element, Algorithm 3.1a can be parallelized across elements.

Algorithm 3.1a Local build stage: `initialize`$(\mathcal{E}, \mathcal{L}, f)$.

Input: Mesh $\mathcal{E} = \{\mathcal{E}_i\}_{i=1}^{n_{\mathrm{elem}}}$, partial differential operator $\mathcal{L}$, right-hand side f
Output: Solution and Dirichlet-to-Neumann operators for every element, $\{S_{\mathcal{E}_i}, \Sigma_{\mathcal{E}_i}\}_{i=1}^{n_{\mathrm{elem}}}$

```
for each element E_i in mesh do
    Transform (L, f) ↦ (L̂, f̂) into reference space (see subsection 2.2).
    Discretize L̂ and f̂ using the ultraspherical spectral method.
    Construct the solution operator S_{E_i} (see subsection 3.2.1).
    Construct the Dirichlet-to-Neumann operator Σ_{E_i} := D_{E_i} S_{E_i}
    (see subsection 3.2.1).
return {S_{E_i}, Σ_{E_i}}_{i=1}^{n_elem}
```

Once local operators have been computed for each element, the scheme enters the global build stage, where a hierarchy of merged operators is constructed in an upward pass. Given a set of indices $\mathcal{I}$ that define a sequence of pairwise merges between elements, operators are merged as in (3.4) and (3.5) in the order $\mathcal{I}$ until the entire mesh has been merged into one large conglomerate. Along the way, merged elements store their newly computed solution operators and Dirichlet-to-Neumann operators. The global build stage ends with a solution operator that acts on the entire mesh, taking in Dirichlet data on every boundary of Ω and returning the solution to the PDE along the penultimate merged interface. The global build stage is outlined in Algorithm 3.1b.

When the mesh contains cross points (i.e., points in the interior of the mesh where corners of multiple elements meet), the linear system defining the solution operator,

$$-\left(\Sigma_{\mathcal{E}_i}^{\Gamma_i,\Gamma_i} + \Sigma_{\mathcal{E}_j}^{\Gamma_j,\Gamma_j}\right) S_{\Gamma_{ij}} = \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_i}^{\Gamma_i,\mathrm{L}_i} & \Sigma_{\mathcal{E}_j}^{\Gamma_j,\mathrm{L}_j} & \Sigma_{\mathcal{E}_i}^{\Gamma_i,\mathrm{end}} + \Sigma_{\mathcal{E}_j}^{\Gamma_j,\mathrm{end}} \end{array}\right], \tag{3.6}$$

may be rank deficient, as a continuity condition on the sum of the normal fluxes around the cross point has not been imposed [12]. Rather than imposing this condition directly, we solve the rank-deficient system by projecting out the cross-point modes, which has the same effect as removing the degrees of freedom located at cross points in a collocation-based scheme [6]. The nullspace of (3.6) contains precisely the cross-point modes, and so we implement this projection step by performing a minimum-norm least-squares solve on (3.6). As this is a projection method, the resulting residual is guaranteed to be identically zero.

Algorithm 3.1b Global build stage (upward pass): `build`$(\mathcal{E}, \Sigma_{\mathcal{E}}, \mathcal{I})$.

Input: Mesh $\mathcal{E} = \{\mathcal{E}_i\}_{i=1}^{n_{\text{elem}}}$, local operators $\Sigma_{\mathcal{E}} = \{\Sigma_{\mathcal{E}_i}\}_{i=1}^{n_{\text{elem}}}$, merge indices $\mathcal{I}$
Output: Solution and Dirichlet-to-Neumann operators for every merge, $\{S_{\Gamma_{ij}}, \Sigma_{\mathcal{E}_{ij}}\}_{(i,j)\in\mathcal{I}}$

for each pair in $(i, j) \in \mathcal{I}$ **do**
  Define the merged domain $\mathcal{E}_{ij} := \mathcal{E}_i \cup \mathcal{E}_j$.
  Define the shared interface $\Gamma_{ij} := \mathcal{E}_i \cap \mathcal{E}_j$.
  Define indices Γ_i, Γ_j for the shared boundary Γ_{ij} on $\mathcal{E}_i$, $\mathcal{E}_j$.
  Define indices $\mathrm{L}_i := \overline{\Gamma_i}$, $\mathrm{L}_j := \overline{\Gamma_j}$ for the unshared boundaries on $\mathcal{E}_i$, $\mathcal{E}_j$.
  Solve the linear system

$$-\left(\Sigma_{\mathcal{E}_i}^{\Gamma_i,\Gamma_i} + \Sigma_{\mathcal{E}_j}^{\Gamma_j,\Gamma_j}\right) S_{\Gamma_{ij}} = \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_i}^{\Gamma_i,\mathrm{L}_i} & \Sigma_{\mathcal{E}_j}^{\Gamma_j,\mathrm{L}_j} & \Sigma_{\mathcal{E}_i}^{\Gamma_i,\text{end}} + \Sigma_{\mathcal{E}_j}^{\Gamma_j,\text{end}} \end{array}\right]$$

  for the merged solution operator $S_{\Gamma_{ij}}$.
  Define the merged Dirichlet-to-Neumann operator,

$$\Sigma_{\mathcal{E}_{ij}} := \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_i}^{\mathrm{L}_i,\mathrm{L}_i} & 0 & \Sigma_{\mathcal{E}_i}^{\mathrm{L}_i,\text{end}} \\ 0 & \Sigma_{\mathcal{E}_j}^{\mathrm{L}_j,\mathrm{L}_j} & \Sigma_{\mathcal{E}_j}^{\mathrm{L}_j,\text{end}} \end{array}\right] + \left[\begin{array}{c} \Sigma_{\mathcal{E}_i}^{\mathrm{L}_i,\Gamma_i} \\ \Sigma_{\mathcal{E}_j}^{\mathrm{L}_j,\Gamma_j} \end{array}\right] S_{\Gamma_{ij}}.$$

return $\{S_{\Gamma_{ij}}, \Sigma_{\mathcal{E}_{ij}}\}_{(i,j)\in\mathcal{I}}$

3.3.2. Solve stage

The final stage of the scheme is the solve stage, which uses the merged solution operators to recover the unknown interface data in a downward pass through the hierarchy. Beginning at the top of the hierarchy, the solution operator acting on the entire mesh is applied to the known Dirichlet data g, returning the Chebyshev coefficients of the solution on the top-level merged interface. These coefficients are then used as Dirichlet data on the next level, where solution operators are again applied to compute the unknown interface data on subdomains. Finally, at the bottom level of the hierarchy—where the solution is now known at each interface between elements—the local solution operators $S_{\mathcal{E}_i}$ are applied to compute the bivariate solution in the interior of each element $\mathcal{E}_i$. The solve stage is outlined in Algorithm 3.2.

The solve stage may be executed multiple times using different boundary data without recomputing the operators constructed in the build stage. The stored operators may also be efficiently updated to solve (1.1) with a different right-hand side f. Recall that the last column of every solution operator and Dirichlet-to-Neumann operator in the hierarchy corresponds to the contribution from the particular solution. Using a new right-hand side, an updated particular solution can be constructed on each element $\mathcal{E}_i$ as in subsection 3.2.1, replacing the last columns of $S_{\mathcal{E}_i}$ and $\Sigma_{\mathcal{E}_i}$. A modified build stage may then be executed, where the last column of each interfacial solution and Dirichlet-to-Neumann operator is updated by solving the linear system (3.6) in an upward pass.

Algorithm 3.2 Solve stage (downward pass): `solve`$(\mathcal{E}, g)$.

Input: Element (or merged element) $\mathcal{E}$, Dirichlet data g
Output: Solutions $\{u_i\}_{i=1}^{n_{\text{elem}}}$

if $\mathcal{E}$ is the entire domain **then**
  Get all boundary faces $(\partial\mathcal{E})_i$.
  Evaluate Dirichlet data $g_i := g((\partial\mathcal{E})_i)$ and convert to Chebyshev coefficients.
if $\mathcal{E}$ is a leaf **then**
  Compute the local solution $u := S_{\mathcal{E}}\left[\begin{smallmatrix} g \\ 1 \end{smallmatrix}\right]$.
  return u
else
  Look up the elements $\mathcal{E}_i$, $\mathcal{E}_j$ that were merged to make $\mathcal{E}$.
  Define the shared interface $\Gamma_{ij} := \mathcal{E}_i \cap \mathcal{E}_j$.
  Recover the missing interface data $\varphi := S_{\Gamma_{ij}}\left[\begin{smallmatrix} g \\ 1 \end{smallmatrix}\right]$.
  Define vectors $\widehat{\boldsymbol{g}}_i$, $\widehat{\boldsymbol{g}}_j$ such that $\widehat{\boldsymbol{g}}_i^{\Gamma_i} = \varphi$, $\widehat{\boldsymbol{g}}_i^{\mathrm{L}_i} = g_i$ and $\widehat{\boldsymbol{g}}_j^{\Gamma_j} = \varphi$, $\widehat{\boldsymbol{g}}_j^{\mathrm{L}_j} = g_j$.
  Compute the solution on $\mathcal{E}_i$, $\{u_i\} :=$ `solve`$(\mathcal{E}_i, \widehat{\boldsymbol{g}}_i)$.
  Compute the solution on $\mathcal{E}_j$, $\{u_j\} :=$ `solve`$(\mathcal{E}_j, \widehat{\boldsymbol{g}}_j)$.
  return $\{u_i\} \cup \{u_j\}$

3.4. Computational complexity

We now determine the computational complexity of the build and solve stages in terms of the number of degrees of freedom, $N \approx (p/h)^2$, where h is the minimum mesh size and p is the polynomial order. Here, we assume that the number of elements in the mesh, n_{elem}, scales as $\mathcal{O}(1/h^2)$, which is valid for a mesh that is approximately uniformly refined. For a mesh that is adaptively refined, the number of elements is typically much less than this estimate.

We begin with the local build stage. On each element $\mathcal{E}_i$, we approximate the solution as a degree-(p, p) polynomial using $(p+1)^2$ degrees of freedom. After transforming the PDE into the local coordinate system of the element, we discretize $\widehat{\mathcal{L}}$ and $\widehat{f}$ using the ultraspherical spectral method. The bivariate Chebyshev coefficients of $\widehat{f}$ can be computed in $\mathcal{O}(p^2 \log p)$ operations via a discrete cosine transform [46]. A separable representation of $\widehat{\mathcal{L}}$ can be computed in $\mathcal{O}(p^3)$ operations using the singular value decomposition, and differentiation, conversion, and multiplication matrices can be constructed for each separable piece in $\mathcal{O}(p)$ operations. The $(p+1)^2 \times (p+1)^2$ discrete PDO can then be assembled using Kronecker products in $\mathcal{O}(p^4)$ operations. The discrete PDO L is almost block-banded, with a bandwidth[8] of $\mathcal{O}(p)$ and $\mathcal{O}(p)$ dense rows. To compute the solution operator $S_{\mathcal{E}_i}$, we must solve a linear system with $\mathcal{O}(p)$ right-hand sides. That is, we must solve a system of the form $LX = B$, where L is $\mathcal{O}(p^2) \times \mathcal{O}(p^2)$ and B is $\mathcal{O}(p^2) \times \mathcal{O}(p)$. The almost-banded matrix L may be written as the sum of an $\mathcal{O}(p)$-banded matrix A and a rank-$\mathcal{O}(p)$ correction, $L = A + UCV^T$, where U and V are $\mathcal{O}(p^2) \times \mathcal{O}(p)$ and C is $\mathcal{O}(p) \times \mathcal{O}(p)$. Using the Woodbury formula, the solution to $LX = B$ becomes

$$X = L^{-1}B = \left(A + UCV^T\right)^{-1} B = \left(I \; - \; A^{-1}U\left(C^{-1} + V^T A^{-1} U\right)^{-1} V^T\right) A^{-1} B.$$

The banded matrix A can be inverted in $\mathcal{O}(p^3)$ operations and its inverse applied to $\mathcal{O}(p)$ right-hand sides in $\mathcal{O}(p^4)$ operations. The matrix $C^{-1} + V^T A^{-1} U$ is $\mathcal{O}(p) \times \mathcal{O}(p)$ and so its inverse can be applied to $\mathcal{O}(p)$ right-hand sides in $\mathcal{O}(p^3)$ operations. Therefore, the solution operator $S_{\mathcal{E}_i}$ on an element can be constructed in $\mathcal{O}(p^4)$ operations. The Dirichlet-to-Neumann operator $\Sigma_{\mathcal{E}_i}$ can be computed as a matrix product in $\mathcal{O}(p^4)$ operations. As these operators are computed once for each element, the overall cost of the local build stage scales as

$$\frac{p^4}{h^2} \approx Np^2.$$

The cost of the global build stage and solve stage depends on the merge scheme defined by the indices $\mathcal{I}$. If the mesh $\mathcal{E} = \{\mathcal{E}_i\}_{i=1}^{n_{\text{elem}}}$ is approximately tensor-product, the merge indices $\mathcal{I}$ can be defined so that the hierarchy is approximately a binary tree (i.e., a binary tree with $\mathcal{O}(1)$ additional merges). If the mesh is unstructured, a hierarchical partitioning of the mesh may be computed by conversion to a graph partitioning problem [27]. The partitioning should be as balanced as possible, so that the indices $\mathcal{I}$ define a balanced tree. If the user specifies merge indices that correspond to an unbalanced tree, then the tree may be automatically rebalanced. We assume that the merge indices $\mathcal{I}$ have been given so that the hierarchy in the build and solve stages approximately forms a binary tree with $\mathcal{O}(\log n_{\text{elem}})$ levels.

Let level $\ell = 0$ denote the bottom level of the hierarchy, where no elements have been merged. For a merge between $\mathcal{E}_i$ and $\mathcal{E}_j$ on level ℓ of the build stage, the solution operator $S_{\Gamma_{ij}}$ is computed by solving the linear system (3.6). The agglomerates $\mathcal{E}_i$ and $\mathcal{E}_j$ each contain $\mathcal{O}(2^\ell)$ mesh elements, with the interface between them, Γ_{ij}, containing $\mathcal{O}(2^{\ell/2})$ boundaries. Hence, the linear system in (3.6) is $\mathcal{O}(2^{\ell/2}p) \times \mathcal{O}(2^{\ell/2}p)$ and can be solved in $\mathcal{O}((2^{\ell/2}p)^3)$ operations. As level ℓ has $\mathcal{O}(2^{-\ell} n_{\text{elem}})$ elements, the cost of processing all merges on level ℓ scales as

$$\left(2^{-\ell} n_{\text{elem}}\right) \cdot \left(2^{\ell/2} p\right)^3 \; = \; n_{\text{elem}} 2^{\ell/2} p^3.$$

The total cost for the global build stage then scales as

$$p^3 n_{\text{elem}} \sum_{\ell=0}^{\mathcal{O}(\log n_{\text{elem}})} 2^{\ell/2} \; \approx \; p^3 (n_{\text{elem}})^{3/2} \; \approx \; \frac{p^3}{h^3} \; \approx \; N^{3/2}$$

as $N \to \infty$.

At level $\ell > 0$ of the solve stage, the unknown interface data is computed via a matrix-vector multiply with an $\mathcal{O}(2^{\ell/2}p) \times \mathcal{O}(2^{\ell/2}p)$ matrix. As level ℓ has $\mathcal{O}(2^{-\ell} n_{\text{elem}})$ elements, the cost of computing the solution on all interfaces scales as

$$\left(2^{-\ell} n_{\text{elem}}\right) \cdot \left(2^{\ell/2} p\right)^2 \; = \; p^2 n_{\text{elem}}.$$

[8] The bandwidth of the discrete PDO depends on the polynomial degree, m, used to approximate the variable coefficients. As in subsection 2.1, we assume that $m \ll p$ so that the discrete PDO is sparse.

The total cost for all levels $\ell > 0$ is then

$$\sum_{\ell=1}^{\mathcal{O}(\log n_{\text{elem}})} p^2 n_{\text{elem}} \approx p^2 n_{\text{elem}} \log n_{\text{elem}}.$$

At the bottom level, $\ell = 0$, the solution is computed on each element through matrix-vector multiplication with local solution operators of size $(p+1)^2 \times (4(p+1)+1)$, which requires $\mathcal{O}(p^3)$ operations. Therefore, the total cost for the solve stage scales as

$$p^2 n_{\text{elem}} \log n_{\text{elem}} + p^3 n_{\text{elem}} \approx \frac{p^2}{h^2} \log \frac{1}{h^2} + \frac{p^3}{h^2} \approx N \log \tfrac{1}{h^2} + Np.$$

The overall computational complexity of the method is therefore

$$\underbrace{Np^2 + N^{3/2}}_{\text{build stage}} + \underbrace{N \log \tfrac{1}{h^2} + Np}_{\text{solve stage}} \approx Np^2 + N^{3/2}.$$

As the method stores dense solution operators and Dirichlet-to-Neumann operators on every level of the hierarchy, the total storage cost is analogous to the computational cost of the solve stage. The amount of storage required by the method scales as

$$N \log \tfrac{1}{h^2} + Np.$$

The storage cost can become prohibitive when p is large, as the local solution operators on each element require $\mathcal{O}(p^3 n_{\text{elem}})$ storage. However, these operators need not be constructed and stored. In the build stage, local Dirichlet-to-Neumann operators can be constructed directly by locally solving the PDE, evaluating the outward flux, and then discarding the solution. In the solve stage, the solution on the interior of each element can be computed by locally solving the PDE on the fly. This reduces the storage cost to $\mathcal{O}(N \log \frac{1}{h^2})$ while increasing the computational cost of the solve stage to $\mathcal{O}(N \log \frac{1}{h^2} + Np^2)$ operations, but does not change the overall computational complexity of the method.

4. Software

We have implemented the ultraspherical SEM in an open-source software package, `ultraSEM`, written in MATLAB without parallelization [16]. An outline of the workflow is depicted in Fig. 4.1, and a simple example is shown in Fig. 4.2.

The user constructs each element as an `ultraSEM.Domain`, which encodes the coordinate transformations and merge indices local to each element. Convenient functions for constructing rectangles, quadrilaterals, triangles, and polygons are available via the commands `ultraSEM.rectangle`, `ultraSEM.quad`, `ultraSEM.triangle`, and `ultraSEM.polygon`, respectively (see Fig. 4.2 (left)), which automatically encode the suitable transformations and merge indices. Elements can be combined to form larger domains by merging them with the '&' operator; the merge indices $\mathcal{I}$ will then correspond to the order induced by the sequence of '&' operations. More general meshes can be constructed using the `refine(dom)` method (see Fig. 4.2 (center)), which performs uniform h-refinement on a given domain `dom`, or the `refinePoint(dom, [x,y])` method, which performs adaptive h-refinement on `dom` around the point (x, y) (see Fig. 5.3).

A PDO is specified by its coefficients for each derivative, in the form `{{uxx, uxy, uyy}, {ux, uy}, b}`, where each term `uxx`, `uxy`, ... can be a scalar (constant coefficient) or function handle (variable coefficient). The domain and PDO are then passed—along with a right-hand side and polynomial order—to construct an `ultraSEM` object (see Fig. 4.2 (right)). The `ultraSEM` constructor initializes the local operators on each element (see Algorithm 3.1a), which are represented as `ultraSEM.Leaf` objects in the hierarchy. The hierarchy of merged operators may then be built in an upward pass via the `build` command (see Algorithm 3.1b), which creates a tree of `ultraSEM.Parent` objects (if `build` is not explicitly called, the build stage is automatically performed when the user requests a solve to be executed). The solve stage is invoked via the `solve` command (or equivalently, the '\' operator), which computes the solution by applying the hierarchy of operators in a downward pass (see Algorithm 3.2). The solution is returned as an `ultraSEM.Sol` object, which overloads a host of functions for plotting (e.g., `plot`, `contour`) and evaluation (e.g., `feval`, `norm`).

An `ultraSEM` object that has been built can be repeatedly applied to new boundary conditions by invoking `solve` multiple times. The object can also be cheaply updated to solve with a new right-hand side by calling `updateRHS`, which alters the last column of each operator in the hierarchy to correspond to a new particular solution.

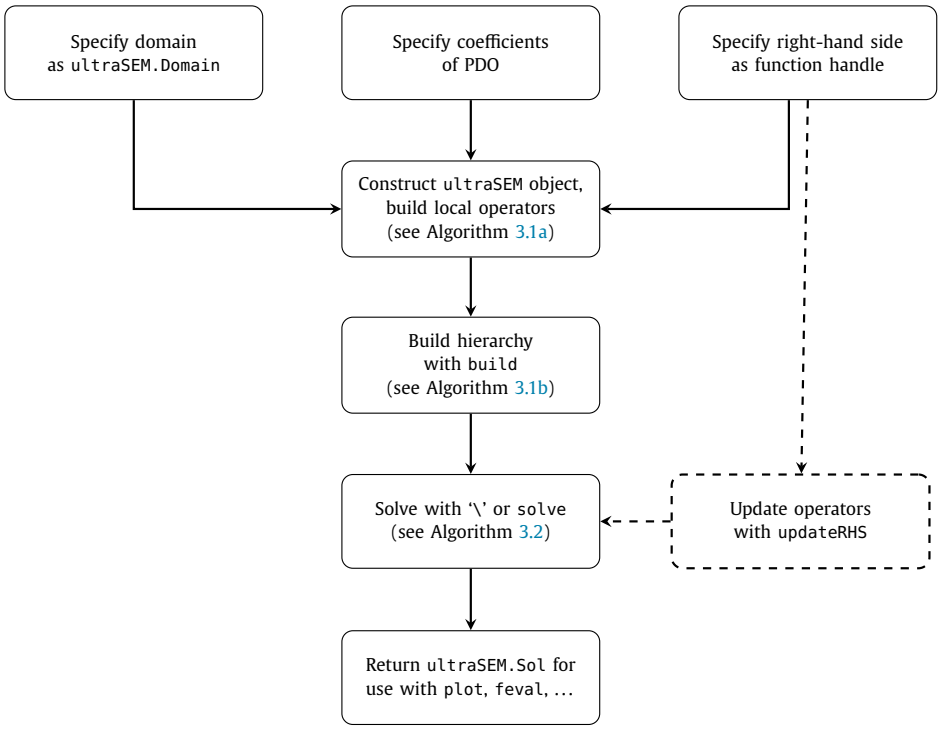
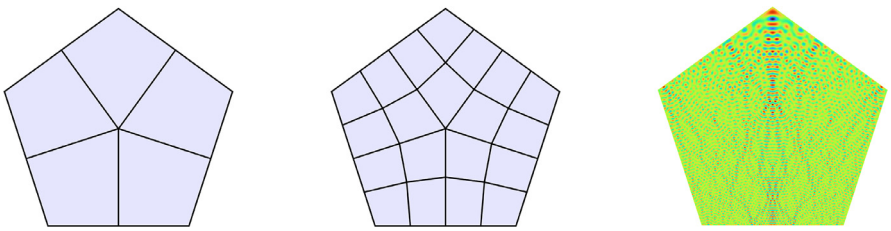


Fig. 4.1. A diagram of the code workflow in `ultraSEM`. The code is designed to mirror the steps of the hierarchical Poincaré–Steklov scheme.



```
dom = ultraSEM.polygon(5);
plot(dom)
```

```
dom = refine(dom);
plot(dom)
```

```
p = 60; rhs = -1; bc = 0;
pdo = {{1,0,1}, {0,0}, @(x,y) 50000*(1-y)};
S = ultraSEM(dom, pdo, rhs, p);
u = S \ bc; plot(u)
```

Fig. 4.2. A simple example of the syntax in `ultraSEM`. A pentagonal domain (with side length 1.2) is meshed into five quadrilaterals (left) and uniformly refined (center). The gravity Helmholtz equation $\nabla^2 u + 50000(1-y)u = -1$ with zero Dirichlet boundary conditions is then solved on the mesh using polynomials of degree 60 on each element (right).

5. Numerical results

5.1. Computational complexity

To illustrate the computational complexity of `ultraSEM`, we measure the execution times[9] of the build and solve stages of the method under uniform h- and p-refinement. Fig. 5.1 shows the recorded timings for solving the variable coefficient PDE $\nabla^2 u + \sin(xy)u = f$ on the domain $\Omega = [0,1]^2$ with a spatially varying right-hand and spatially varying Dirichlet boundary conditions.

In Fig. 5.1 (left), the polynomial order is fixed at $p = 4$ and a Cartesian mesh with $\mathcal{O}(1/h^2)$ elements is successively refined. Both the local and global build stages exhibit $\mathcal{O}(1/h^2)$ scaling as $h \to 0$, while the solve stage scales as $\mathcal{O}(1/h^2 \log(1/h^2))$. The timings for the build stage do not exhibit the expected $\mathcal{O}(1/h^3)$ scaling. This is likely due to the

[9] All numerical experiments were performed in MATLAB R2020a on a 40-core Intel Xeon E5-2630 workstation with 128 GB of RAM and no explicit parallelization.

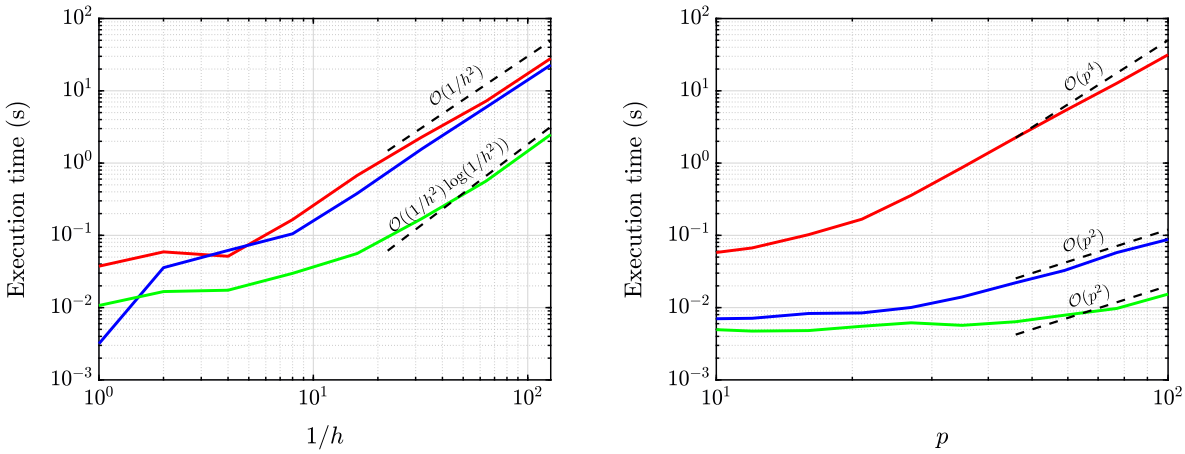


Fig. 5.1. Execution time (in seconds) for `ultraSEM` over a range of mesh sizes (left) and polynomial orders (right). Timings are depicted for the local build stage (red), global build stage (blue), and solve stage (green), when solving the PDE $\nabla^2 u + \sin(xy)u = f$ on the domain $\Omega = [0, 1]^2$ with spatially varying right-hand side and spatially varying Dirichlet boundary conditions. On the left, we successively refine a Cartesian mesh while keeping the polynomial order fixed at $p = 4$. On the right, we use a 4×4 Cartesian mesh while successively increasing the polynomial order.

fact that the build stage relies on dense linear algebra routines that have been heavily optimized for the relatively small $\mathcal{O}(1/h) \times \mathcal{O}(1/h)$ matrices tested here. The storage used by the finest mesh in Fig. 5.1 (left) is approximately 2 GB.

In Fig. 5.1 (right), the Cartesian mesh is fixed to have 4×4 elements and the polynomial order p is successively increased. The cost of the local build stage dominates, exhibiting close to the expected $\mathcal{O}(p^4)$ scaling as $p \to \infty$. The global build and solve stages perform better than expected, both exhibiting $\mathcal{O}(p^2)$ scaling. Again, this can likely be attributed to the performance of dense linear algebra routines in the regime of p tested. The storage used by the finest mesh in Fig. 5.1 (right) is approximately 25 GB.

5.2. Convergence and hp-adaptivity

We now investigate the convergence properties of `ultraSEM` with respect to the mesh size h and polynomial order p. As a test problem, we consider solving the Helmholtz equation,

$$\nabla^2 u + (\sqrt{2}\omega)^2 u = 0, \qquad u \in [-1, 1]^2, \tag{5.1}$$

with $\omega \in \mathbb{R}$ and Dirichlet boundary conditions given so that the exact solution is $u(x, y) = \cos(\omega x)\cos(\omega y)$. To measure convergence over a range of polynomial orders, we set $\omega = p$ so that the number of degrees of freedom per wavelength remains fixed independent of p. We then solve (5.1) under uniform h-refinement. Fig. 5.2 shows the relative error in the L^2 norm as $h \to 0$ for polynomial orders $p = 5$, $p = 10$, and $p = 30$. The convergence rate is observed to be $\mathcal{O}(h^{p-1})$. If error is measured in the H^1 or H^2 norm, where H^k denotes the Sobolev space of functions whose weak derivatives up to order

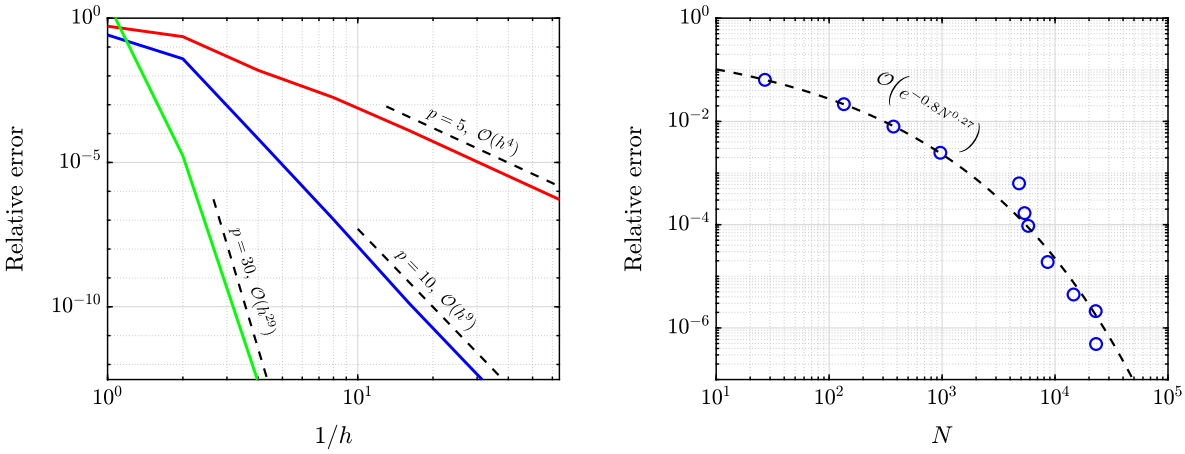


Fig. 5.2. Convergence of `ultraSEM` with respect to h and p. (Left) Relative error in the L^2 norm when solving (5.1) with $\omega = p$ under uniform h-refinement, for $p = 5$ (red), $p = 10$ (blue), and $p = 30$ (green). In each case, $\mathcal{O}(h^{p-1})$ convergence is observed. (Right) An *a priori* hp-adaptivity strategy is applied to the L-shape problem in (5.2). The relative error decays super-algebraically in the total number of degrees of freedom N.

Fig. 5.3. To avoid hanging nodes, `ultraSEM` performs h-refinement in a conforming way. (Left) A square is successively refined into a corner by subdividing into three children per refinement level. (Right) A square is successively refined around a point by subdividing into five children per refinement level. General quadrilaterals are refined in similar ways.

k are in L^2, then the convergence rate is similarly $\mathcal{O}(h^{p-1})$. Since our method is sparse with respect to p, the exact rate of convergence is not so important, as a degree-p discretization may easily be replaced by a degree-$(p+1)$ discretization with minimal increase in computational cost.

In general, the mesh size h and polynomial order p need not be the same on each element. Adaptive h-refinement can be performed on each element locally; however, subdividing an element may give rise to meshes with hanging nodes (i.e., nodes of the mesh which occur in the middle of an element's face). While hanging nodes may be handled in the hierarchical Poincaré–Steklov scheme through the use of interpolation operators [19], we choose to avoid them here. To avoid hanging nodes, `ultraSEM` performs h-refinement in a conforming way around specified corners or points, by subdividing a quadrilateral element into three or five children, respectively (see Fig. 5.3).

The ultraspherical spectral element method can naturally perform p-adaptivity by applying local interpolation and restriction operators to the elemental matrices involved in each merge operation. Since each unknown interface function is represented by a vector of Chebyshev coefficients, interpolation to and restriction from an interface can be performed simply by zero-padding or truncating the interface data. The polynomial order on an interface can be defined in a variety of ways. Popular choices include the minimum rule and maximum rule [14]; we employ the minimum rule here, which sets the polynomial order on an interface to be the minimum of the polynomial orders on the adjacent elements.

We now consider the application of an hp-adaptivity strategy to the classical L-shape domain problem [32],

$$\nabla^2 u = 0, \qquad u \in [-1,1]^2 \setminus [0,1] \times [-1,0], \tag{5.2}$$

with Dirichlet boundary conditions given so that the exact solution is $u(r,\theta) = r^{2/3}\sin(2\theta/3)$, where $r = \sqrt{x^2+y^2}$ and $\theta = \tan^{-1}(y/x)$. The reentrant corner of the domain induces a singularity in the solution so that $u \in H^{1+2/3}$ near the origin. Therefore, any strategy based on uniform h- or p-refinement is necessarily restricted to algebraic convergence[10] [8]. That is, for a numerical solution u_{hp} based on uniform refinement, the error can be bounded *a priori* by

$$\|u - u_{hp}\|_{L^2} \le \|u - u_{hp}\|_{H^1} \le C\left(\frac{h}{p}\right)^{2/3} \|u\|_{H^{1+2/3}},$$

for some constant $C > 0$. However, by employing a suitable hp-adaptivity strategy, super-algebraic convergence in the number of degrees of freedom N can be achieved [7], i.e.,

$$\|u - u_{hp}\|_{L^2} \le \|u - u_{hp}\|_{H^1} \le C_1 e^{-C_2 N^{1/3}},$$

for some constants $C_1, C_2 > 0$. Here we employ an *a priori* adaptivity strategy, where h-refinement is performed into the reentrant corner on elements adjacent to the origin and p-refinement is performed on all other elements [2]. Given a desired relative error tolerance and an initial coarse hp-mesh, an automatic hp-adaptivity loop is run that successively refines or coarsens each element in h or p based on an *a posteriori* error indicator [1,33]. Here, we compute the element-wise error from the exact solution as a surrogate for a true *a posteriori* error indicator. Fig. 5.2 (right) shows the relative error in the L^2 norm versus the total number of degrees of freedom N in the adaptive hp-mesh for a sequence of error tolerances. Super-algebraic convergence to the solution is observed as the number of degrees of freedom increases. A least-squares fit to the data gives an approximate convergence rate of $\mathcal{O}(e^{-0.8N^{0.27}})$. To illustrate the range of h and p used on a given mesh, for a relative error tolerance of 10^{-6} the final mesh contains 15 levels of corner h-refinement and polynomial orders ranging from 3 to 13.

As a practical example of hp-adaptivity, we consider using `ultraSEM` on a domain with small-scale geometric features along its boundary. The domain Ω is a snowflake shape created by a fractal-like Penrose tiling (see Fig. 5.4 (left)). We construct a mesh of 4,568 quadrilaterals over Ω using the meshing software Gmsh [20], with the element size constrained to be smaller near the boundary and larger in the interior. To specify a p-adaptive discretization, we define a function that varies smoothly from $p = 40$ in the center of Ω to $p = 7$ near the boundary, indicating that coarse elements in the interior of Ω employ a high-p discretization while fine elements close to the boundary of Ω employ a lower p. The total number of

[10] For this Laplace problem, alternative methods may provide higher accuracy per degree of freedom than element methods. For instance, root-exponential convergence in the supremum norm can be achieved by representing the solution as the real part of a rational function with poles exponentially clustered near each corner [24].

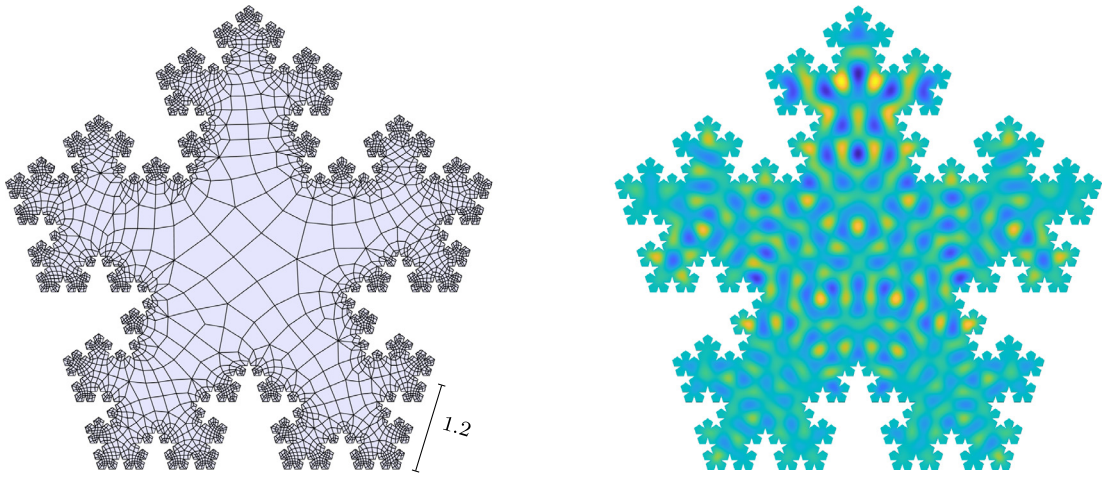


Fig. 5.4. (Left) A snowflake-shaped domain created by a Penrose tiling is adaptively meshed with 4,658 elements, with p varying from $p = 7$ on small elements to $p = 40$ on large elements. (Right) The gravity Helmholtz equation (5.3) is solved on this domain. The solution is represented by $N = 333{,}627$ degrees of freedom.

degrees of freedom for this hp-mesh is $N = 333{,}627$. We locate the domain Ω such that $y < 0$ for all $(x, y) \in \Omega$, and solve the gravity Helmholtz equation

$$\nabla^2 u + 100(1 - y)u = -1, \qquad u \in \Omega, \tag{5.3}$$

with zero Dirichlet boundary conditions. The computation in `ultraSEM` takes about 75 seconds (43 seconds in the local build stage, 31 seconds in the global build stage, and 1 second in the solve stage) and consumes approximately 10 GB of memory. The computed solution is shown in Fig. 5.4 (right), though is only accurate to about one digit when compared in relative infinity norm to an over-resolved solution. To obtain further accuracy, the use of impedance-to-impedance maps may be necessary to avoid artificial resonances when merging operators [21]. The h-adaptive nature of the discretization allows for the small-scale geometry of the domain boundary to be resolved without using a prohibitive number of elements, while the p-adaptive nature of the discretization allows for the high-degree approximation of smooth functions on coarse elements.

5.3. Implicit time-stepping for parabolic problems

The ability to reuse precomputed solution operators allows for efficient implicit time-stepping for parabolic problems. To demonstrate, we consider solving the variable-coefficient convection-diffusion equation on the domain $\Omega = [0, 10] \times [-1, 1]$ over the time span $[0, T]$,

$$\frac{\partial u}{\partial t} = \kappa \nabla^2 u - \nabla \cdot (\boldsymbol{b}(x, y)u), \qquad u \in \Omega \times [0, T], \tag{5.4}$$

for $T > 0$, with initial condition $u(x, y, 0) = e^{-4(x-1)^2 - 4y^2}$ and zero Dirichlet boundary conditions. This equation models the transport of a contaminant concentration in a flow. We define the diffusivity $\kappa = 0.01$ and the convective velocity $\boldsymbol{b}(x, y) = (1 - e^{\gamma x}\cos 2\pi y, \frac{\gamma}{2\pi} e^{\gamma x} \sin 2\pi y)$. Here, the velocity field $\boldsymbol{b}(x, y)$ is the analytical solution to the Kovasznay flow [28], where $\gamma = \mathrm{Re}/2 - \sqrt{\mathrm{Re}^2/4 - 4\pi^2}$ and $\mathrm{Re} = 100$ is the Reynolds number.

Define the time step $\Delta t = 0.1$ and time points $t_n = n\Delta t$ for integers $n \geq 0$, and let u^n denote the approximate solution to (5.4) at time t_n. Discretizing in time using the backward Euler method yields a steady-state PDE in u^{n+1},

$$u^{n+1} - \Delta t\, \kappa \nabla^2 u^{n+1} + \Delta t\, \nabla \cdot \left(\boldsymbol{b} u^{n+1}\right) = u^n, \tag{5.5}$$

which must be solved once per time step to compute u^{n+1} from u^n. We use `ultraSEM` to solve (5.5) on a 4×20 Cartesian mesh of Ω with polynomial order $p = 16$ on each element, which yields an infinity-norm relative error of 10^{-6} at time $t = 5$ when compared to an over-resolved solution. Fig. 5.5 (left) shows snapshots of the computed solution at times $t = 0$, $t = 1$, and $t = 5$. As the right-hand side of (5.5) depends on n, the operators in `ultraSEM` must be updated at each time step. If the operators are reconstructed from scratch at each time step, simulating to time $T = 5$ completes in roughly one minute (see Fig. 5.5 (right, red)). If instead only the particular solution is reconstructed using `updateRHS`, then the same simulation completes in roughly 6 seconds (see Fig. 5.5 (right, blue)). Fig. 5.5 (right) compares the execution times required to simulate (5.4) over the time span $[0, T]$ using these two methods. It is clear that when many time steps are taken, `updateRHS` should always be used.

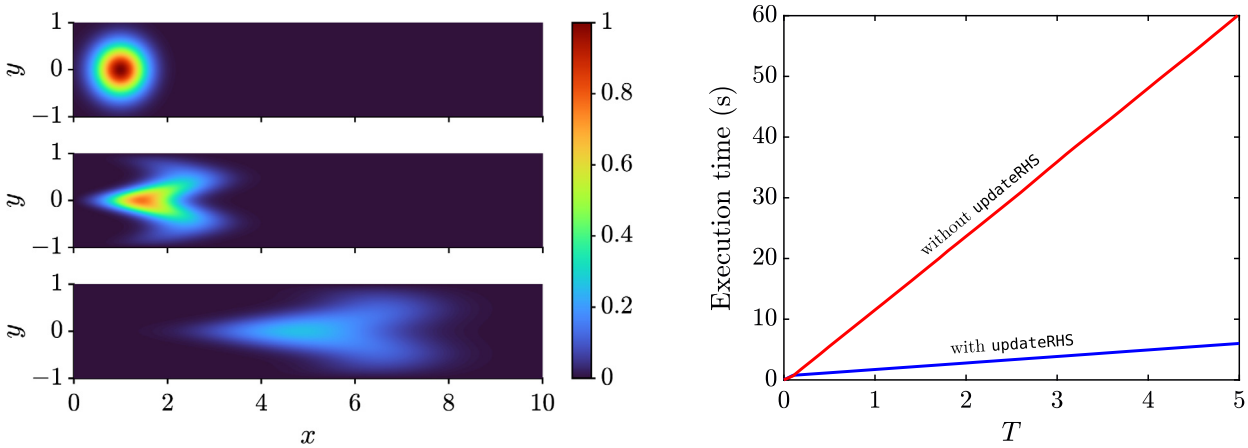


Fig. 5.5. (Left) Snapshots of the solution to the convection-diffusion equation (5.4) at times $t = 0$, $t = 1$, and $t = 5$, computed using `ultraSEM` in space and backward Euler in time. (Right) The execution time required to simulate (5.4) over the time span $[0, T]$, by either reconstructing the operators from scratch at each time step (red) or updating the particular solution using `updateRHS` (blue).

6. Future work

The development of fast direct solvers for three-dimensional problems is an active area of research [25], and it may be possible to generalize the ultraspherical spectral element method to three-dimensional meshed geometries. The collocation-based HPS scheme in three dimensions has a computational complexity of $\mathcal{O}(p^9)$ due to the inversion of dense $p^3 \times p^3$ matrices on each element in the local build stage. We believe that using the ultraspherical spectral method on each leaf would reduce this complexity to $\mathcal{O}(p^7)$ or even $\mathcal{O}(p^6)$ for certain problems. However, a careful analysis of the storage costs in three dimensions is necessary to determine if such a direct solver is practical.

CRediT authorship contribution statement

Daniel Fortunato: Conceptualization, Methodology, Software, Visualization, Writing – original draft. **Nicholas Hale:** Conceptualization, Software, Supervision, Writing – review & editing. **Alex Townsend:** Conceptualization, Funding acquisition, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like to thank Sheehan Olver, Keaton Burns, and Marc Gilles for their useful discussions on domain decomposition with ultraspherical polynomials, and Federico Fuentes for his expertise on *hp*-adaptivity theory. We are grateful to Heather Wilber, Sheehan Olver, Patrick Farrell, and Alex Barnett for their comments on a draft of this work.

References

[1] M. Ainsworth, J.T. Oden, A posteriori error estimation in finite element analysis, Comput. Methods Appl. Mech. Eng. 142 (1997) 1–88, https://doi.org/10.1016/S0045-7825(96)01107-3.
[2] M. Ainsworth, B. Senior, An adaptive refinement strategy for *hp*-finite element computations, Appl. Numer. Math. 26 (1998) 165–178, https://doi.org/10.1016/S0168-9274(97)00083-4.
[3] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny, V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, W. Pazner, M. Stowell, V. Tomov, J. Dahm, D. Medina, S. Zampini, MFEM: a modular finite element methods library, https://arxiv.org/abs/1911.09220, Nov 2019.
[4] Argonne National Laboratory, Nek5000, https://nek5000.mcs.anl.gov, 2020. (Accessed 24 March 2020), Version 19.0.
[5] T. Babb, A. Gillman, S. Hao, P.-G. Martinsson, An accelerated Poisson solver based on multidomain spectral discretization, BIT Numer. Math. 58 (2018) 851–879, https://doi.org/10.1007/s10543-018-0714-0.
[6] T. Babb, P.-G. Martinsson, D. Appelo, HPS accelerated spectral solvers for time dependent problems, https://arxiv.org/abs/1811.04555, Nov 2018.
[7] I. Babuška, B. Guo, The *h*-*p* version of the finite element method, Comput. Mech. 1 (1986) 21–41, https://doi.org/10.1007/BF00298636.
[8] I. Babuška, M. Suri, The *h*-*p* version of the finite element method with quasiuniform meshes, ESAIM: Math. Model. Numer. Anal. 21 (1987) 199–238, https://doi.org/10.1051/m2an/1987210201991.
[9] S. Beuchler, J. Schöberl, New shape functions for triangular *p*-FEM using integrated Jacobi polynomials, Numer. Math. 103 (2006) 339–366, https://doi.org/10.1007/s00211-006-0681-2.
[10] J. Boyd, Chebyshev and Fourier Spectral Methods, Dover, Mineola, 2001.

[11] C. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D.D. Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. Kirby, S. Sherwin, Nektar++: an open-source spectral/*hp* element framework, Comput. Phys. Commun. 192 (2015) 205–219, https://doi.org/10.1016/j.cpc.2015.02.008.
[12] C. Canuto, A. Quarteroni, M.Y. Hussaini, T.A. Zang, Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics, Scientific Computation, Springer, Berlin, 2007.
[13] B. Cockburn, G.E. Karniadakis, C.-W. Shu, Discontinuous Galerkin Methods: Theory, Computation and Applications, Springer, Berlin, 2000.
[14] L. Demkowicz, Computing with *hp*-Adaptive Finite Elements: Volume 1: One and Two Dimensional Elliptic and Maxwell Problems, CRC Press, 2006.
[15] M.G. Duffy, Quadrature over a pyramid or cube of integrands with a singularity at a vertex, SIAM J. Numer. Anal. 19 (1982) 1260–1262, https://doi.org/10.1137/0719090.
[16] D. Fortunato, N. Hale, A. Townsend, GitHub repository, https://github.com/danfortunato/ultraSEM, 2020.
[17] D. Fortunato, C.H. Rycroft, R. Saye, Efficient operator-coarsening multigrid schemes for local discontinuous Galerkin methods, SIAM J. Sci. Comput. 41 (2019) A3913–A3937, https://doi.org/10.1137/18M1206357.
[18] D. Fortunato, A. Townsend, Fast Poisson solvers for spectral methods, IMA J. Numer. Anal. (2019), https://doi.org/10.1093/imanum/drz034.
[19] P. Geldermans, A. Gillman, An adaptive high order direct solution technique for elliptic boundary value problems, SIAM J. Sci. Comput. 41 (2019) A292–A315, https://doi.org/10.1137/17M1156320.
[20] C. Geuzaine, J.-F. Remacle, Gmsh: a 3-D finite element mesh generator with built-in pre- and post-processing facilities, Int. J. Numer. Methods Eng. 79 (2009) 1309–1331, https://doi.org/10.1002/nme.2579.
[21] A. Gillman, A.H. Barnett, P.-G. Martinsson, A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media, BIT Numer. Math. 55 (2015) 141–170, https://doi.org/10.1007/s10543-014-0499-8.
[22] A. Gillman, 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 J. Sci. Comput. 36 (2014) A2023–A2046, https://doi.org/10.1137/130918988.
[23] A. Gillman, P.-G. Martinsson, An $O(N)$ algorithm for constructing the solution operator to 2D elliptic boundary value problems in the absence of body loads, Adv. Comput. Math. 40 (2014) 773–796, https://doi.org/10.1007/s10444-013-9326-z.
[24] A. Gopal, L.N. Trefethen, Solving Laplace problems with corner singularities via rational functions, SIAM J. Numer. Anal. 57 (2019) 2074–2094, https://doi.org/10.1137/19M125947X.
[25] S. Hao, P.-G. Martinsson, A direct solver for elliptic PDEs in three dimensions based on hierarchical merging of Poincaré–Steklov operators, J. Comput. Appl. Math. 308 (2016) 419–434, https://doi.org/10.1016/j.cam.2016.05.013.
[26] T.J. Hughes, The Finite Element Method: Linear Static and Dynamic Finite Element Analysis, Courier Corporation, 2012.
[27] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1998) 359–392, https://doi.org/10.1137/S1064827595287997.
[28] L.I.G. Kovasznay, Laminar flow behind a two-dimensional grid, Math. Proc. Camb. Philos. Soc. 44 (1948) 58–62, https://doi.org/10.1017/S0305004100023999.
[29] Y. Maday, R. Muñoz, Spectral element multigrid. II. Theoretical justification, J. Sci. Comput. 3 (1988) 323–353, https://doi.org/10.1007/BF01065177.
[30] P. Martinsson, A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method, J. Comput. Phys. 242 (2013) 460–479, https://doi.org/10.1016/j.jcp.2013.02.019.
[31] P.-G. Martinsson, A fast direct solver for a class of elliptic partial differential equations, J. Sci. Comput. 38 (2009) 316–330, https://doi.org/10.1007/s10915-008-9240-6.
[32] W.F. Mitchell, A collection of 2D elliptic problems for testing adaptive grid refinement algorithms, Appl. Math. Comput. 220 (2013) 350–364, https://doi.org/10.1016/j.amc.2013.05.068.
[33] W.F. Mitchell, M.A. McClain, A survey of *hp*-adaptive strategies for elliptic partial differential equations, in: T.E. Simos (Ed.), Recent Advances in Computational and Applied Mathematics, Springer, Dordrecht, 2011, pp. 227–258.
[34] D. Moxey, C.D. Cantwell, Y. Bao, A. Cassinelli, G. Castiglioni, S. Chun, E. Juda, E. Kazemi, K. Lackhove, J. Marcon, G. Mengaldo, D. Serson, M. Turner, H. Xu, J. Peiró, R.M. Kirby, S.J. Sherwin, Nektar++: enhancing the capability and application of high-fidelity spectral/*hp* element methods, Comput. Phys. Commun. 249 (2020), https://doi.org/10.1016/j.cpc.2019.107110.
[35] F.W.J. Olver, D.W. Lozier, R.F. Boisvert, C.W. Clark, NIST Handbook of Mathematical Functions, Cambridge University Press, New York, 2010.
[36] S. Olver, A. Townsend, A fast and well-conditioned spectral method, SIAM Rev. 55 (2013) 462–489, https://doi.org/10.1137/120865458.
[37] S.A. Orszag, Spectral methods for problems in complex geometries, J. Comput. Phys. 37 (1980) 70–92, https://doi.org/10.1016/0021-9991(80)90005-4.
[38] A.T. Patera, A spectral element method for fluid dynamics: laminar flow in a channel expansion, J. Comput. Phys. 54 (1984) 468–488, https://doi.org/10.1016/0021-9991(84)90128-1.
[39] W. Pazner, Efficient low-order refined preconditioners for high-order matrix-free continuous and discontinuous Galerkin methods, https://arxiv.org/abs/1908.07071, Aug 2019.
[40] F. Rathgeber, D.A. Ham, L. Mitchell, M. Lange, F. Luporini, A.T.T. Mcrae, G.-T. Bercea, G.R. Markall, P.H.J. Kelly, Firedrake: automating the finite element method by composing abstractions, ACM Trans. Math. Softw. 43 (2016), https://doi.org/10.1145/2998441.
[41] E.M. Rønquist, A.T. Patera, Spectral element multigrid. I. Formulation and numerical results, J. Sci. Comput. 2 (1987) 389–406, https://doi.org/10.1007/BF01061297.
[42] S. Sherwin, G. Karniadakis, Spectral/hp Element Methods for Computational Fluid Dynamics, Oxford University Press, Oxford, 2005.
[43] S.J. Sherwin, G.E. Karniadakis, A new triangular and tetrahedral basis for high-order (*hp*) finite element methods, Int. J. Numer. Methods Eng. 38 (1995) 3775–3802, https://doi.org/10.1002/nme.1620382204.
[44] A. Townsend, S. Olver, The automatic solution of partial differential equations using a global spectral method, J. Comput. Phys. 299 (2015) 106–123, https://doi.org/10.1016/j.jcp.2015.06.031.
[45] L.N. Trefethen, Spectral Methods in MATLAB, SIAM, Philadelphia, 2000.
[46] L.N. Trefethen, Approximation Theory and Approximation Practice, SIAM, Philadelphia, 2012.
[47] P.E. Vos, S.J. Sherwin, R.M. Kirby, From *h* to *p* efficiently: implementing finite and spectral/*hp* element methods to achieve optimal performance for low- and high-order discretisations, J. Comput. Phys. 229 (2010) 5161–5181, https://doi.org/10.1016/j.jcp.2010.03.031.
[48] A. Yeiser, A. Townsend, A spectral element method for meshes with skinny elements, SIAM Undergrad. Res. Online (2018) 421–437, https://doi.org/10.1137/18S017053.