A scalable elliptic solver with task-based parallelism for the SpECTRE numerical relativity code

Nils L. Vu [1, *] Harald P. Pfeiffer [1] Gabriel S. Bonilla [2] Nils Deppe [3] François Hébert [3]
Lawrence E. Kidder [4] Geoffrey Lovelace [2] Jordan Moxon [3] Mark A. Scheel [3] Saul
A. Teukolsky [3, 4] William Throwe [4] Nikolas A. Wittek [1] and Tom Włodarczyk [1]

[1] *Max Planck Institute for Gravitational Physics (Albert Einstein Institute), Am Mühlenberg 1, Potsdam 14476, Germany*
[2] *Nicholas and Lee Begovich Center for Gravitational-Wave Physics and Astronomy,
California State University, Fullerton, Fullerton, California 92831, USA*
[3] *Theoretical Astrophysics, Walter Burke Institute for Theoretical Physics,
California Institute of Technology, Pasadena, California 91125, USA*
[4] *Cornell Center for Astrophysics and Planetary Science, Cornell University, Ithaca, New York 14853, USA*
(Dated: April 19, 2022)

Elliptic partial differential equations must be solved numerically for many problems in numerical relativity, such as initial data for every simulation of merging black holes and neutron stars. Existing elliptic solvers can take multiple days to solve these problems at high resolution and when matter is involved, because they are either hard to parallelize or require a large amount of computational resources. Here we present a new solver for linear and nonlinear elliptic problems that is designed to scale with resolution and to parallelize on computing clusters. To achieve this we employ a discontinuous Galerkin discretization, an iterative multigrid-Schwarz preconditioned Newton-Krylov algorithm, and a task-based parallelism paradigm. To accelerate convergence of the elliptic solver we have developed novel subdomain-preconditioning techniques. We find that our multigrid-Schwarz preconditioned elliptic solves achieve iteration counts that are independent of resolution, and our task-based parallel programs scale over 200 million degrees of freedom to at least a few thousand cores. Our new code solves a classic initial data problem for binary black holes faster than the spectral code SpEC when distributed to only eight cores, and in a fraction of the time on more cores. It is publicly accessible in the next-generation SpECTRE numerical relativity code. Our results pave the way for highly parallel elliptic solves in numerical relativity and beyond.

I. INTRODUCTION

Solving elliptic partial differential equations (PDEs) numerically is important in many areas of science, including numerical relativity [1]. All numerical time evolutions begin with initial data that capture the physical scenario to be evolved, and the initial data must typically satisfy a set of constraint equations formulated as elliptic PDEs. Specifically, to construct initial data for general-relativistic simulations of black holes and neutron stars we must solve the Einstein constraint equations, which admit formulations as elliptic PDEs [2, 3]. Binary configurations of black holes and neutron stars enjoy particular prominence as primary sources for gravitational-wave detectors, and numerical simulations of these systems play an essential role in their observations [4–9].

To construct initial data for general-relativistic simulations, the numerical relativity (NR) community has put considerable effort towards developing numerical codes that solve elliptic problems. Most of the existing codes employ spectral methods to discretize the elliptic equations, such as LORENE [10, 11] and TwoPunctures [12, 13], Spells [14–18] that is part of SpEC [19], as well as SGRID [20, 21], KADATH [22, 23] and Elliptica [24]. The COCAL [25, 26] code employs finite-difference methods, and

NRPyElliptic [27] a hyperbolic relaxation scheme [28]. All of these codes vary significantly in the numerical methods employed to solve the discretized equations. For example, SpEC uses the PETSc library to perform an iterative matrix-free solve with a custom preconditioner [14], whereas KADATH and Elliptica construct explicit matrix representations and invert the matrices directly [22, 24].

While successful in constructing initial data for many general-relativistic scenarios, these codes can still take a significant amount of time or require excessive computational resources to solve the elliptic problems. For example, the SpEC and SGRID codes typically require a few hours to days to solve for initial data that involves orbiting neutron stars, at a resolution required for state-of-the-art simulations, using $\mathcal{O}(10)$ cores for the computation. KADATH, on the other hand, quote a few hours to solve for low-resolution initial data involving orbiting neutron stars on about 128 cores, and "a larger timescale" and more cores for higher resolutions, with high memory demands for the explicit matrix construction [22, 23], and assuming symmetry with respect to the orbital plane [23]. Elliptica also quote a few days to solve an initial data problem for a black hole–neutron star binary (BHNS) on 20 cores [24].

Despite the time required to solve the elliptic initial data problem, simulations of merging black holes and neutron stars are currently dominated by their time evolution, which can take weeks to months. However, significant efforts are underway to develop faster and more accurate

evolution codes for next-generation numerical relativity. The open-source SpECTRE [29, 30] code aims to evolve general-relativistic multiphysics scenarios on petascale and future exascale computers, and is the main focus of this article. Other recent developments include the CarpetX driver for the Einstein Toolkit [31] that is based on the AMReX framework [32], and the Dendro-GR [33], Nmesh [34], bamps [35], GRAthena++ [36], and ExaHyPE [37] codes.

To seed these next-generation evolutions of general-relativistic scenarios with initial data, we have developed a highly scalable elliptic solver based on discontinuous Galerkin methods, matrix-free iterative algorithms, and task-based parallelism. We focus strongly on parallelization to take advantage of the increasing number of cores in high-performance computing (HPC) systems. These systems have at least $\mathcal{O}(10)$, but often closer to 50–100, physical cores per node, often with many thousand interconnected nodes. Therefore, even routine compute jobs that request only a few nodes on contemporary HPC clusters, and hence spend little to no time waiting in a queue, have tens to hundreds of cores at their disposal. Larger compute jobs with thousands of cores and more are also readily available, and the amount of available computational resources is expected to increase rapidly in the future.

The SpECTRE code embraces parallelism as a core design principle [29, 30]. It employs a task-based parallelism paradigm instead of the conventional message passing interface (MPI) protocol. MPI-parallelized programs typically alternate between computation and communication at global synchronization points, meaning that all threads must reach a globally agreed-upon state before the program proceeds. Global synchronization points can limit the effective use of the available cores when some threads reach the synchronization later than others, thus holding up the program. The effect becomes more pronounced with increasing core count, often limiting the number of cores that MPI-parallelized programs can efficiently scale to. Task-based parallel programs, on the other hand, aim to avoid global synchronization points as much as possible. They partition the computational work into interdependent tasks and distribute them among the available cores. Tasks can migrate to undersubscribed cores while the program is running to balance the computational load. SpECTRE builds upon the Charm++ [38] task-based parallelism and CPU-abstraction library. Reference [30] describes SpECTRE's task-based parallism paradigm in more detail.

Our new elliptic solver in the SpECTRE code is based on the prototype presented in Ref. [39] and employs the discontinuous Galerkin discretization for generic elliptic equations developed in Ref. [40], which makes it applicable to a wide range of elliptic problems in numerical relativity and beyond. In this article we present the task-based iterative algorithms that we have developed to parallelize the elliptic solver effectively on computing clusters, including novel subdomain-preconditioning techniques. We

demonstrate that our new elliptic solver can solve a classic initial data problem for binary black holes faster than SpEC when running on as few as eight cores, and in a fraction of the time on a computing cluster. In particular, the number of iterations that our new elliptic solver requires to converge remains constant with increasing resolution. The additional computational work needed to solve high-resolution problems manifests in subproblems that become either more numerous or more expensive, but that can be solved in parallel to offset the increase in runtime.

This article is structured as follows. Section II summarizes the discontinuous Galerkin scheme that was presented in Ref. [40] and that we employ to discretize all elliptic equations in this article. Section III details the stack of task-based algorithms that constitutes the elliptic solver, and that we have implemented in the SpECTRE code. In Sec. IV we assess the performance and parallel efficiency of our new elliptic solver by applying it to a set of test problems. We conclude in Sec. V.

II. DISCONTINUOUS GALERKIN DISCRETIZATION

We employ the discontinuous Galerkin (DG) scheme developed in Ref. [40] to discretize all elliptic problems in this article and summarize it in this section.

Schematically, the discretization procedure translates a linear elliptic problem to a matrix equation, such as

$$-\partial_i\partial_i\varphi(\boldsymbol{x}) = 4\pi\rho(\boldsymbol{x}) \quad \xrightarrow[\text{Ref. [40]}]{} \quad \mathcal{A}\underline{u} = \underline{b}, \quad (1)$$

where $\underline{u} = (u_1, \ldots, u_{N_{\text{DOF}}})$ is a discrete representation of all variables on the computational grid, $\underline{b} = (b_1, \ldots, b_{N_{\text{DOF}}})$ is a discrete representation of the fixed sources in the PDEs, and $\mathcal{A}$ is an $N_{\text{DOF}} \times N_{\text{DOF}}$ matrix that represents the discrete Laplacian operator in this example. Equation (1) represents the Maxwell constraint equation for the electric potential $\varphi(\boldsymbol{x})$ in Coulomb gauge, written here in Cartesian coordinates, where $\rho(\boldsymbol{x})$ is the electric charge density sourcing the field. We employ the Einstein sum convention to sum over repeated indices.

The subject of this section is to define the matrix equation (1) for a wide range of elliptic problems, as detailed in Ref. [40]. Then, the remainder of this article is concerned with solving the matrix equation numerically for $\underline{u}$, and doing so iteratively, in parallel on computing clusters, and without ever explicitly constructing the full matrix $\mathcal{A}$. Instead, we only need to define the matrix-vector product $\mathcal{A}\underline{u}$. We solve nonlinear problems $\mathcal{A}(\underline{u}) = \underline{b}$ by repeatedly solving their linearization.

The discontinuous Galerkin scheme detailed in Ref. [40] applies to a wide range of elliptic problems. Specifically, it applies to any set of elliptic PDEs that admits a formulation in first-order flux form

$$-\partial_i \mathcal{F}_\alpha{}^i[u_A, v_A; \boldsymbol{x}] + \mathcal{S}_\alpha[u_A, v_A; \boldsymbol{x}] = f_\alpha(\boldsymbol{x}), \quad (2)$$

where the fluxes $\mathcal{F}_\alpha{}^i$ and the sources $\mathcal{S}_\alpha$ are functionals of a set of *primal* variables $u_A(\boldsymbol{x})$ and *auxiliary* variables $v_A(\boldsymbol{x})$, and the fixed sources $f_\alpha(\boldsymbol{x})$ are functions of coordinates. The index α enumerates both primal and auxiliary equations. The primal variables can be scalars, such as the electric potential $\varphi(\boldsymbol{x})$ in the Maxwell constraint (1), higher-rank tensor fields such as the displacement vector in an elasticity problem, or combinations thereof such as in Eq. (4) below. The auxiliary variables are typically gradients of the primal variables, such as $v_i = \partial_i \varphi(\boldsymbol{x})$ for the Maxwell constraint. For example, the Maxwell constraint (1) can be formulated with the fluxes and sources

$$\mathcal{F}_v{}^i{}_j = \varphi\,\delta^i_j, \quad \mathcal{S}_{v\,j} = v_j, \quad f_{v\,j} = 0, \tag{3a}$$
$$\mathcal{F}_\varphi{}^i = v_i, \qquad \mathcal{S}_\varphi = 0, \qquad f_\varphi = 4\pi\rho(\boldsymbol{x}), \tag{3b}$$

where δ^i_j denotes the Kronecker delta. Note that Eq. (3a) is the definition of the auxiliary variable, and Eq. (3b) is the Maxwell constraint (1).

In particular, the flux form (2) also encompasses the extended conformal thin-sandwich (XCTS) formulation of the Einstein constraint equations [3],

$$\bar\nabla^2 \psi = \frac{1}{8}\psi\bar{R} + \frac{1}{12}\psi^5 K^2 \tag{4a}$$
$$- \frac{1}{8}\psi^{-7}\bar{A}_{ij}\bar{A}^{ij} - 2\pi\psi^5\rho$$

$$\bar\nabla^2(\alpha\psi) = \alpha\psi\left(\frac{7}{8}\psi^{-8}\bar{A}_{ij}\bar{A}^{ij} + \frac{5}{12}\psi^4 K^2 + \frac{1}{8}\bar{R}\right.$$
$$\left. + 2\pi\psi^4(\rho + 2S)\right) - \psi^5\partial_t K + \psi^5\beta^i\bar\nabla_i K \tag{4b}$$

$$\bar\nabla_i(\bar{L}\beta)^{ij} = (\bar{L}\beta)^{ij}\bar\nabla_i \ln(\bar\alpha) + \bar\alpha\bar\nabla_i\left(\bar\alpha^{-1}\bar{u}^{ij}\right) \tag{4c}$$
$$+ \frac{4}{3}\bar\alpha\psi^6\bar\nabla^j K + 16\pi\bar\alpha\psi^{10} S^j$$

with $\bar\nabla^2 = \bar\gamma^{ij}\bar\nabla_i\bar\nabla_j$, $\bar{A}^{ij} = \frac{1}{2\bar\alpha}\left((\bar{L}\beta)^{ij} - \bar{u}^{ij}\right)$ and $\bar\alpha = \alpha\psi^{-6}$. The XCTS equations are a set of coupled nonlinear elliptic PDEs that the spacetime metric of general relativity must satisfy at all times.[1] They are solved for the conformal factor ψ, the product of lapse and conformal factor $\alpha\psi$, and the shift vector β^j. The remaining quantities in the equations, i.e., the conformal metric $\bar\gamma_{ij}$, the trace of the extrinsic curvature K, their respective time derivatives $\bar{u}_{ij}$ and $\partial_t K$, the energy density ρ, the stress-energy trace S and the momentum density S^i, are freely-specifiable fields that define the scenario at hand. In particular, the conformal metric $\bar\gamma_{ij}$ defines the background geometry of the elliptic problem, which determines the covariant derivative $\bar\nabla$, the Ricci scalar $\bar{R}$ and the
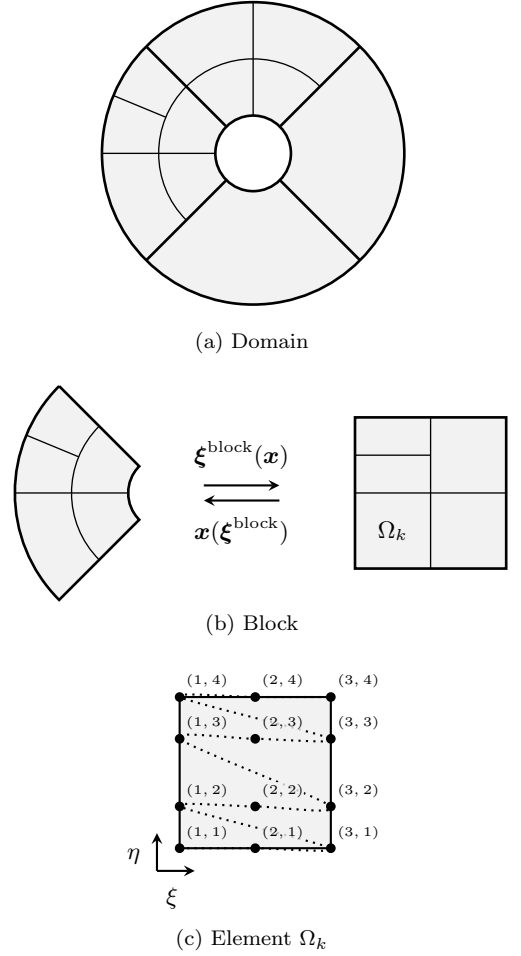
---

[1] See, e.g., Ref. [1] for an introduction to the XCTS equations, in particular Box 3.3.

(a) Domain

$$\boldsymbol{\xi}^{\text{block}}(\boldsymbol{x})$$
$$\boldsymbol{x}(\boldsymbol{\xi}^{\text{block}})$$

Ω_k

(b) Block

(c) Element Ω_k

FIG. 1. *Top:* Geometry of a two-dimensional computational domain composed of four wedge-shaped blocks. *Middle:* The coordinate transformation $\boldsymbol{\xi}^{\text{block}}(\boldsymbol{x})$ maps a block to a reference cube in block-logical coordinates $[-1, 1]^2$. A block is split into elements Ω_k along its logical coordinates axes. *Bottom:* The element Ω_k in element-logical coordinates $\boldsymbol{\xi} = (\xi, \eta)$ with its grid of Legendre-Gauss-Lobatto collocation points. In this example we chose $N_{k,\xi} = 3$ and $N_{k,\eta} = 4$. Each grid point is labeled with its index (p_ξ, p_η). The dotted line connects points in the order they are enumerated in by the index p.

longitudinal operator

$$\left(\bar{L}\beta\right)^{ij} = \bar\nabla^i\beta^j + \bar\nabla^j\beta^i - \frac{2}{3}\bar\gamma^{ij}\bar\nabla_k\beta^k. \tag{5}$$

Reference [40] lists fluxes and sources for the XCTS equations, and for a selection of other elliptic systems.

Once we have formulated the equations, we choose a computational domain on which to discretize them. We decompose the d-dimensional computational domain $\Omega \subset \mathbb{R}^d$ into a set of *blocks* shaped like deformed cubes, as illustrated in Fig. 1a. Blocks do not overlap, but they share boundaries. Each face of a block is either shared with precisely one other block, or is external. For example, the domain depicted in Fig. 1a has four wedge-shaped blocks. Each block $B \subset \Omega$ carries a map from the coordinates $\boldsymbol{x} \in B$, in which the elliptic equations (2) are

formulated, to *block-logical* coordinates $\boldsymbol{\xi}^{\text{block}} \in [-1,1]^d$ representing the d-dimensional reference cube, as illustrated in Fig. 1b.

Blocks decompose into *elements* $\Omega_k \subset \Omega$, by recursively splitting in half along any of their logical coordinate axes (h refinement). We limit the h refinement of our computational domain such that an element shares its boundary with at most two neighbors per dimension in every direction ("two-to-one balance"), both within a block and across block boundaries. Each element defines element-logical coordinates $\boldsymbol{\xi} \in [-1,1]^d$ by an affine transformation of the block-logical coordinates. The resulting coordinate map to the reference cube of the element is characterized by the Jacobian

$$\mathrm{J}^i_j = \frac{\partial x^i}{\partial \xi^j} \qquad (6)$$

with determinant J and inverse $(\mathrm{J}^{-1})^j_i = \partial \xi^j / \partial x^i$.

On the reference cube of the element we choose a regular grid of collocation points along the logical coordinate axes, as illustrated in Fig. 1c (p refinement). Specifically, we choose $N_{k,i}$ Legendre-Gauss-Lobatto (LGL) collocation points, ξ_{p_i}, in each dimension i, where the index $p_i \in \{1, \ldots, N_{k,i}\}$ identifies the grid point along dimension i. We also enumerate all $N_k = \prod_i N_{k,i}$ d-dimensional grid points $\boldsymbol{\xi}_p = (\xi_{p_1}, \ldots, \xi_{p_d})$ in an element with a single index $p \in \{1, \ldots, N_k\}$, as illustrated in Fig. 1c.

Then, fields are represented numerically by their values at the collocation points. We denote the set of discrete field values within an element Ω_k as $\underline{u}^{(k)} = (u_1, \ldots, u_{N_k})$, and the collection of discrete field values over *all* elements as $\underline{u}$. The field values at the collocation points within an element define a d-dimensional Lagrange interpolation,

$$u^{(k)}(\boldsymbol{x}) := \sum_{p=1}^{N_k} u_p \psi_p(\boldsymbol{\xi}(\boldsymbol{x})) \quad \text{with} \quad \boldsymbol{x} \in \Omega_k, \qquad (7)$$

where the basis functions $\psi_p(\boldsymbol{\xi})$ are products of Lagrange polynomials,

$$\psi_p(\boldsymbol{\xi}) := \prod_{i=1}^d \ell_{p_i}(\xi^i) \quad \text{with} \quad \boldsymbol{\xi} \in [-1,1]^d. \qquad (8)$$

based on the collocation points in dimension i of the element. Since Eqs. (7) and (8) are local to each element, fields over the entire domain are discontinuous across element boundaries.

Finally, we employ the strong discontinuous Galerkin scheme developed in Ref. [40] to discretize the equations in first-order flux form, Eq. (2). To compute the matrix-vector product in Eq. (1) we first compute the auxiliary variables v_A, given the primal variables u_A, as

$$v_A = D_i \cdot \mathcal{F}_{v_A}{}^i + L \cdot ((n_i \mathcal{F}_{v_A}{}^i)^* - n_i \mathcal{F}_{v_A}{}^i) - \tilde{\mathcal{S}}_{v_A}, \quad (9a)$$

where we assume the auxiliary sources can be written in the form $\mathcal{S}_{v_A} = v_A + \tilde{\mathcal{S}}_{v_A}[u_A; \boldsymbol{x}]$ such that Eq. (9a)

depends only on the primal variables. We also assume $f_{v_A} = 0$ for convenience. All elliptic equations that we consider in this article fulfill these assumptions. In a second step, we use the computed auxiliary variables v_A, as well as the primal variables u_A, to compute the DG residuals

$$-M D_i \cdot \mathcal{F}_{u_A}{}^i - M L \cdot ((n_i \mathcal{F}_{u_A}{}^i)^* - n_i \mathcal{F}_{u_A}{}^i)$$
$$+ M \cdot \mathcal{S}_{u_A} = M \cdot f_{u_A}. \qquad (9b)$$

The operation $\cdot$ in Eq. (9) denotes a matrix multiplication with the field values over the computational grid of an element. We make use of the mass matrix

$$M_{pq} = \int_{[-1,1]^d} \psi_p(\boldsymbol{\xi}) \psi_q(\boldsymbol{\xi}) \sqrt{g} \, \mathrm{J} \, \mathrm{d}^d \xi, \qquad (10)$$

the stiffness matrix

$$M D_{i,pq} = \int_{[-1,1]^d} \psi_p(\boldsymbol{\xi}) \frac{\partial \psi_q}{\partial \xi^j}(\boldsymbol{\xi}) (\mathrm{J}^{-1})^j_i \sqrt{g} \, \mathrm{J} \, \mathrm{d}^d \xi, \quad (11)$$

and the lifting operator

$$M L_{pq} = \int_{[-1,1]^{d-1}} \psi_p(\boldsymbol{\xi}) \psi_q(\boldsymbol{\xi}) \sqrt{g^\Sigma} \, \mathrm{J}^\Sigma \, \mathrm{d}^{d-1} \xi \qquad (12)$$

on the element Ω_k, as well as the associated "massless" operators $D_i := M^{-1} M D_i$ and $L := M^{-1} M L$. Here, $\sqrt{g}$ denotes the determinant of the metric on which the elliptic equations are formulated, such as the conformal metric $\bar{\gamma}_{ij}$ in the XCTS equations (4). The integral in Eq. (12) is over the boundary of the element, $\partial \Omega_k$, where n_i is the outward-pointing unit normal one-form, g^Σ is the surface metric determinant induced by the background metric, and J^Σ is the determinant of the surface Jacobian.

The quantities $(n_i \mathcal{F}_{v_A}{}^i)^*$ and $(n_i \mathcal{F}_{u_A}{}^i)^*$ in Eq. (9) denote a numerical flux that couples grid points across nearest-neighbor element boundaries. We employ the generalized internal-penalty numerical flux developed in Ref. [40],

$$(n_i \mathcal{F}_{v_A}{}^i)^* = \frac{1}{2} \Big[n_i^{\text{int}} \mathcal{F}_{v_A}{}^i(u_A^{\text{int}}) - n_i^{\text{ext}} \mathcal{F}_{v_A}{}^i(u_A^{\text{ext}}) \Big], \quad (13a)$$

$$(n_i \mathcal{F}_{u_A}{}^i)^* = \frac{1}{2} \Big[n_i^{\text{int}} \mathcal{F}_{u_A}{}^i \big(\partial_j \mathcal{F}_{v_A}{}^j(u_A^{\text{int}}) - \tilde{\mathcal{S}}_{v_A}(u_A^{\text{int}}) \big)$$
$$- n_i^{\text{ext}} \mathcal{F}_{u_A}{}^i \big(\partial_j \mathcal{F}_{v_A}{}^j(u_A^{\text{ext}}) - \tilde{\mathcal{S}}_{v_A}(u_A^{\text{ext}}) \big) \Big]$$
$$- \sigma \Big[n_i^{\text{int}} \mathcal{F}_{u_A}{}^i \big(n_j^{\text{int}} \mathcal{F}_{v_A}{}^j(u_A^{\text{int}}) \big)$$
$$- n_i^{\text{ext}} \mathcal{F}_{u_A}{}^i \big(n_j^{\text{ext}} \mathcal{F}_{v_A}{}^j(u_A^{\text{ext}}) \big) \Big] \qquad (13b)$$

with the penalty function

$$\sigma = C \frac{\big(\max(p^{\text{int}}, p^{\text{ext}}) + 1 \big)^2}{\min(h^{\text{int}}, h^{\text{ext}})}. \qquad (14)$$

Here, u_A^{int} denotes the primal variables on the *interior* side of an element's shared boundary with a neighbor,

and u_A^{ext} denotes the primal variables on the neighbor's side, i.e., the *exterior*. Note that $n_i^{\text{ext}} = -n_i^{\text{int}}$ for the purpose of this article, since we only consider equations formulated on a fixed background metric, but the scheme does not rely on this assumption. For Eq. (14) we also make use of the polynomial degree p, and a measure of the element size, h, orthogonal to the element boundary on either side of the interface, as detailed in Ref. [40].

We impose boundary conditions through fluxes, i.e., by a choice of exterior quantities in the numerical flux, Eq. (13). Specifically, on external boundaries we set

$$(n_i \mathcal{F}_\alpha{}^i)^{\text{ext}} = (n_i \mathcal{F}_\alpha{}^i)^{\text{int}} - 2(n_i \mathcal{F}_\alpha{}^i)^{\text{b}}, \qquad (15)$$

where we choose the boundary fluxes $(n_i \mathcal{F}_\alpha{}^i)^{\text{b}}$ depending on the boundary conditions we intend to impose. For *Neumann-type* boundary conditions we choose the primal boundary fluxes $(n_i \mathcal{F}_{u_A}{}^i)^{\text{b}}$ directly, e.g., $(n_i \mathcal{F}_\varphi{}^i)^{\text{b}} = n_i \partial_i \varphi|_{\text{b}}$ for the Maxwell constraint (1), and set the auxiliary boundary fluxes to their interior values, $(n_i \mathcal{F}_{v_A}{}^i)^{\text{b}} = (n_i \mathcal{F}_{v_A}{}^i)^{\text{int}}$. For *Dirichlet-type* boundary conditions we choose the primal boundary fields u_A^{b}, e.g., $\varphi|_{\text{b}}$ for the Maxwell constraint (1), to compute the auxiliary boundary fluxes $(n_i \mathcal{F}_{v_A}{}^i)^{\text{b}} = n_i^{\text{b}} \mathcal{F}_{v_A}{}^i(u_A^{\text{b}})$, and set the primal boundary fluxes to their interior values, $(n_i \mathcal{F}_{u_A}{}^i)^{\text{b}} = (n_i \mathcal{F}_{u_A}{}^i)^{\text{int}}$.

In summary, the DG residuals (9) are algebraic equations for the discrete primal field values $\underline{u}_A$ on all elements and grid points in the computational domain. For linear PDEs, the left-hand side of Eq. (9b) defines a matrix-vector product with a set of primal field values on the computational domain. The right-hand side of Eq. (9b) is a set of fixed values on the computational domain. Therefore, Eq. (9b) has the form of Eq. (1).

III. TASK-BASED ITERATIVE ALGORITHMS

Once the elliptic problem is discretized, it is the responsibility of the elliptic solver to invert the matrix equation (1) numerically, in order to obtain the solution vector $\underline{u}$ over the computational grid. For large problems on high-resolution grids it is typically unfeasible to invert the matrix $\mathcal{A}$ in Eq. (1) directly, or even to explicitly construct and store it.[2] Instead, we employ iterative algorithms that require only the matrix-vector product $\mathcal{A}\underline{u}$ be defined, and that parallelize to computing clusters.

The discontinuous Galerkin (DG) matrix-vector product $\mathcal{A}\underline{u}$ is well suited for parallelization. As Sec. II summarized, it decomposes into a set of operations local to
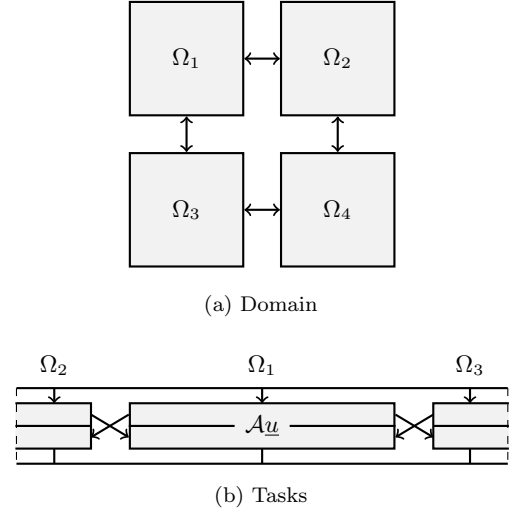
———————

[2] The KADATH [22] code explicitly constructs, stores and inverts the matrix $\mathcal{A}$, which it obtains by a spectral discretization, by distributing its columns over the available cores. References [22, 23] quote the high memory demand of storing the explicit matrix and list iterative approaches to solve the linear system as a possible resolution. Such iterative approaches, and their parallelization, are the main focus of this article.

(a) Domain

(b) Tasks

FIG. 2. Parallelization structure of the matrix-vector product $\mathcal{A}\underline{u}$. *Top:* Decomposition of a two-dimensional rectangular domain into four elements. Arrows illustrate the dependence between nearest-neighbor elements to compute the matrix-vector product $\mathcal{A}\underline{u}$. *Bottom:* Tasks involved to compute the matrix-vector product $\mathcal{A}\underline{u}$. Each element performs a task that prepares and sends data to its neighbors (upper half of the rectangle), and another that receives data from its neighbors and performs the computation (lower half of the rectangle). The arrows between elements are the same as in the top panel.

the elements that make up the computational domain. Figure 2 illustrates a computational domain composed of elements, as well as the dependence of the elements on each other for computing the matrix-vector product. The matrix-vector product requires only data local to each element and on both sides of the boundary that the element shares with its nearest neighbors. Therefore, it can be computed in parallel, and requires only a single communication between each pair of nearest-neighbor elements to exchange data on their shared boundary. The matrix-vector product acts as a "soft" global synchronization point, meaning that it requires *all* elements have sent data to their neighbors before *all* elements can proceed with the algorithm, but individual elements can already proceed once they receive data from their nearest neighbors.

The decomposition of the domain into elements also admits a strategy to distribute computation across the processors of the computer system. We distribute elements among the available cores in a way that, ideally, minimizes the number of internode communications and assigns an equal amount of work to each core. In this article we employ a Morton ("z-order") space-filling curve [41] to traverse the elements within a block of the computational domain and fill up the available cores. We weight the elements by their number of grid points to approximately balance the amount of work assigned to each core. With this strategy, neighboring elements tend to lie on the same node, though more effective element-distribution and load-balancing strategies based on, for
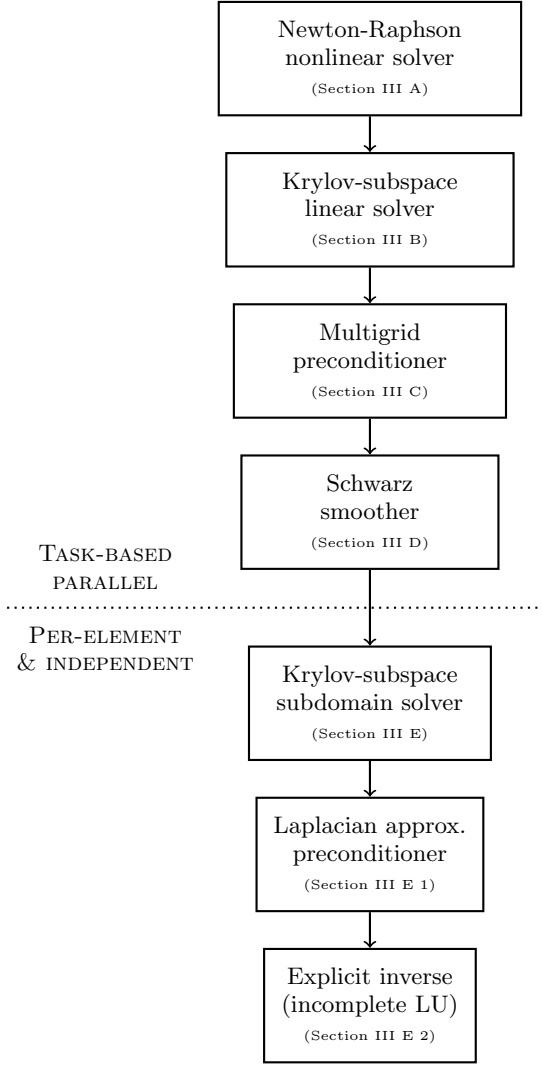
FIG. 3. Overview of the technology stack we employ to solve the discretized elliptic problem (1). All algorithms above the dotted line follow SpECTRE's task-based parallelism paradigm. The algorithms below the dotted line run within a task, and on all elements independently.

example, Hilbert space-filling curves [42] are a subject of future work.

Once elements have been assigned to the available cores, each element traverses the list of tasks in the algorithm. When it encounters a task whose dependencies are not yet fulfilled, e.g., when neighbors have not yet sent the data on shared boundaries needed for the DG matrix-vector product, the element relinquishes control of the core to another whose dependencies are fulfilled. Reference [30] describes SpECTRE's task-based parallel runtime system based on the Charm++ [38] framework in more detail.

Figure 3 provides an overview of the algorithms that we employ to iteratively solve the discretized elliptic problem (1), with details given in subsequent sections: nonlinear equations are linearized with a Newton-Raphson scheme with a line-search globalization (Sec. III A). The

resulting linear subproblems are solved with an iterative Krylov-subspace method (Sec. III B), preconditioned with a multigrid solver (Sec. III C). On every level of the multigrid hierarchy we run a few iterations of an additive Schwarz smoother, which solves the problem approximately on independent, overlapping, element-centered subdomains (Sec. III D). Each subdomain problem is solved by another Krylov-type method, which carries a Laplacian-approximation preconditioner with an incomplete LU explicit-inversion scheme to accelerate the solve (Sec. III E). All algorithms are implemented in the open-source SpECTRE code and take advantage of its task-based parallel infrastructure [29, 30].

A. Newton-Raphson nonlinear solver

The Newton-Raphson scheme iteratively refines an initial guess $\underline{u}_0$ for a nonlinear problem $\mathcal{A}(\underline{u}) = \underline{b}$ by repeatedly solving the linearized problem

$$\frac{\delta\mathcal{A}}{\delta u}(\underline{u})\,\Delta\underline{u} = \underline{b} - \mathcal{A}(\underline{u}) \tag{16}$$

for the correction $\Delta\underline{u}$, and then updating the solution as $\underline{u} \to \underline{u} + \Delta\underline{u}$ [43, 44].

The Newton-Raphson method converges quadratically once it reaches a basin of attraction, but can fail to converge when the initial guess is too far from the solution. We employ a line-search globalization strategy to recover convergence in such cases, following Alg. 6.1.3 in Ref. [44]. It iteratively reduces the step length λ until the corrected residual $\|\underline{b} - \mathcal{A}(\underline{u} + \lambda\Delta\underline{u})\|_2$ has sufficiently decreased, meaning it has decreased by a fraction of the predicted decrease if the problem was linear. This fraction is the *sufficient-decrease parameter* controlling the line search. The line search typically starts at $\lambda = 1$ in every Newton-Raphson iteration, but the initial step length can be decreased to dampen the nonlinear solver. Although the line-search globalization has proven effective for the cases we have encountered so far, alternative globalization strategies such as a trust-region method or more sophisticated nonlinear preconditioning techniques can be investigated in the future.[3]

Figure 4 illustrates our task-based implementation of the Newton-Raphson algorithm. The sufficient-decrease condition, and the necessity to check the global residual magnitude against convergence criteria, introduce a synchronization point in the form of a global reduction to assemble the residual magnitude $\|\underline{b} - \mathcal{A}(\underline{u} + \lambda\Delta\underline{u})\|_2$. The algorithm requires one nonlinear operator application $\mathcal{A}(\underline{u} + \lambda\Delta\underline{u})$ per iteration, plus one additional nonlinear operator application for every globalization step that reduces the step length. Since a typical nonlinear

---

[3] See, e.g., Ref. [45] for an overview of nonlinear preconditioning techniques in the context of the PETSc [46] library.
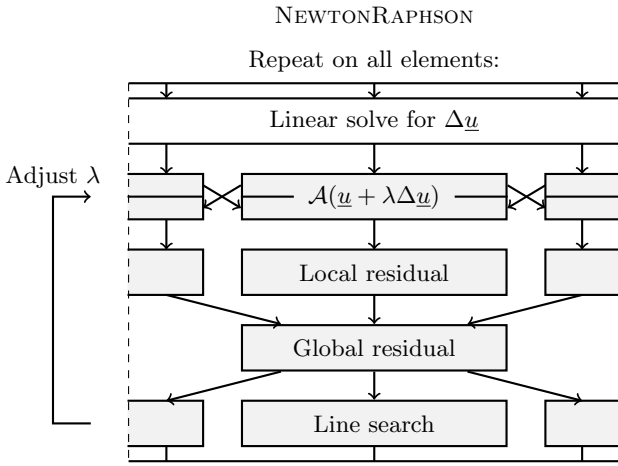
NEWTONRAPHSON

Repeat on all elements:

Linear solve for $\Delta \underline{u}$

Adjust λ

$\mathcal{A}(\underline{u} + \lambda \Delta \underline{u})$

Local residual

Global residual

Line search

FIG. 4. Parallelization structure of the task-based Newton-Raphson nonlinear solver (Sec. III A).

GMRES

Repeat on all elements:

Precondition $\underline{z}$

k times

$\mathcal{A}\underline{z}$

Local inner product

Global inner product

Orthogonalize

Update $\underline{u}$

FIG. 5. Parallelization structure of the task-based GMRES Krylov-subspace linear solver (Sec. III B).

elliptic solve requires $\lesssim 10$ Newton-Raphson iterations, the parallelization properties of this algorithm are not particularly important for the overall performance.

Exactly once per iteration the Newton-Raphson algorithm dispatches a linear solve of Eq. (16) for the correction $\Delta \underline{u}$. This iterative linear-solver algorithm is the subject of the following section.

B. Krylov-subspace linear solver

We solve the linearized problem (16) with an iterative Krylov-subspace algorithm. We generally employ a GMRES algorithm, but have also developed a conjugate gradients algorithm for discretized problems that are symmetric positive definite [47, 48]. These algorithms solve a linear problem $\mathcal{A}\underline{u} = \underline{b}$ iteratively by building up a basis of the Krylov subspace $K_k = \{\underline{b}, \mathcal{A}\underline{b}, \mathcal{A}^2\underline{b}, \ldots, \mathcal{A}^{k-1}\underline{b}\}$. Krylov-subspace algorithms are guaranteed to find a solution in at most N_{DOF} iterations, where N_{DOF} is the size of the matrix $\mathcal{A}$.

Figure 5 illustrates our task-based GMRES algorithm, which is based on Alg. 9.6 in Ref. [48]. It requires one application of the linear operator $\mathcal{A}$ per iteration. Then, the algorithm is characterized by an Arnoldi orthogonalization procedure to construct a new basis vector $\underline{z}$ of the Krylov subspace that is orthogonal to all previously constructed basis vectors. The orthogonalization procedure requires a global reduction to assemble the inner product of the new basis vector with every existing basis vector, meaning the GMRES algorithm needs to perform k reductions in the kth iteration. Every reduction constitutes a global synchronization point, since it requires that all elements send data to a single core on the computer system and wait for a broadcast from that core back to all elements. A conjugate gradients algorithm also requires a global reduction per iteration, but avoids the additional reductions from the orthogonalization procedure.
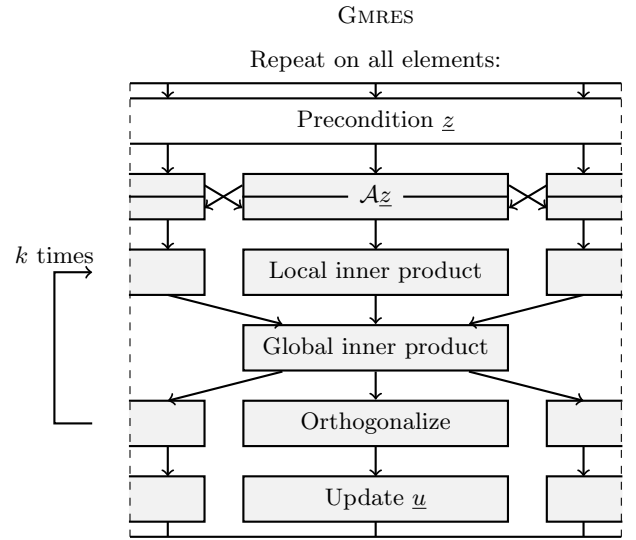
Due to the global synchronization points involved in every iteration of the Krylov-subspace solver, it is essential to keep the number of iterations to a minimum in order to achieve good parallel performance. To this end, we invoke a *preconditioner* in every iteration of the Krylov-subspace algorithm and place particular focus on its parallelization properties. The preconditioner is responsible for solving the linear problem approximately to accelerate the convergence of the Krylov-subspace algorithm.[4] Effective parallel preconditioning techniques for our DG-discretized elliptic problems are the main focus of this article. Since we employ the *flexible* variant of the GMRES algorithm, the preconditioner may change between iterations [48]. While the flexible GMRES algorithm with a variable preconditioner is not mathematically guaranteed to converge in at most N_{DOF} iterations anymore, in practice, it converges in much fewer than N_{DOF} iterations (see test problems in Sec. IV, in particular Figs. 12 and 15).[5]

Typically, the number of iterations needed by an unpreconditioned Krylov-subspace algorithm increases with the size of the problem. The convergence behavior is often connected to the condition number of the linear operator,

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}, \quad (17)$$

where $\lambda_{\max}$ and $\lambda_{\min}$ denote the largest and smallest eigenvalue of the matrix, respectively. However, note that rigorous convergence bounds for the GMRES algorithm in terms of the condition number exist only when the matrix is *normal* [48]. Nevertheless, the condition number

---

[4] See, e.g., Ref. [48] for an introduction to iterative linear solvers and preconditioning techniques.
[5] See also Sec. 9.4.1 in Ref. [48] for a discussion of the flexible GMRES algorithm.
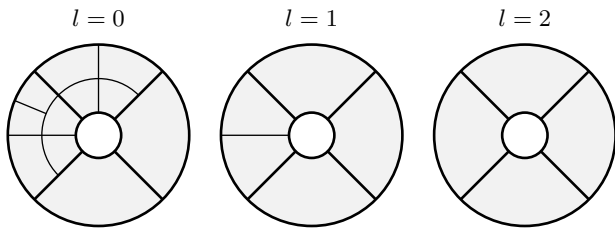
FIG. 6.   Multigrid hierarchy based on the domain depicted in Fig. 1a.

can provide an indication for the expected rate of convergence. For the discontinuous Galerkin discretization we employ in this article, the condition number scales as $\kappa \propto p^2/h$, where p denotes a typical polynomial degree of the elements and h denotes a typical element size [39, 40, 49, 50]. This scaling is related to the decrease of the minimum spacing between Legendre-Gauss-Lobatto collocation points, which scales quadratically with p near element boundaries and linearly with the element size.

More specifically, Krylov-subspace methods struggle to solve large-scale modes in the solution. The algorithm solves modes on the scale of the grid-point spacing or the size of elements in just a few iterations, but it needs a lot more iterations to solve modes spanning the full domain. Such large-scale modes carry, for example, information from boundary conditions that must traverse the entire domain. The test problem presented in Fig. 11 below illustrates this effect. Therefore, we precondition the Krylov-subspace solver with a multigrid algorithm that uses information from coarser grids, where the large-scale modes from finer grids become small scale.

C.   Multigrid preconditioner

We employ a geometric V-cycle multigrid algorithm, as prototyped in Ref. [39].[6] Our multigrid solver can be used standalone, or to precondition a Krylov-type linear solver as described in Sec. III B ("Krylov-accelerated multigrid").

1.   Grid hierarchy

The geometric multigrid algorithm relies on a strategy to coarsen the computational grid. We primarily h-coarsen the domain, meaning that we create multigrid levels $l > 0$ by successively combining two elements into one along every dimension of the grid, as illustrated in Fig. 6. We only p-coarsen the grid in the sense that we choose the smaller of the two polynomial degrees when combining elements along an axis. This strategy follows

Ref. [39] and ensures that coarse-grid field approximations always have an exact polynomial representation on finer grids.

The coarsest possible grid that our domain decomposition can achieve has a single element per block that make up the domain. For example, the two-dimensional shell depicted in Fig. 6 has four wedge-shaped blocks, each of which is a deformed cube. Our multigrid algorithm works best when the domain is composed of as few blocks as possible.

2.   Intermesh operators

To project data between grids we use the standard L_2-projections (or *Galerkin* projections) detailed in Ref. [40].[7] Fields on coarser grids are projected to finer grids with the *prolongation operator*

$$P_{\tilde{p}p}^{l+1 \to l} = \prod_{i=1}^{d} \ell_{p_i}(\tilde{\xi}_{\tilde{p}_i}), \tag{18}$$

where p enumerates grid points on the coarser grid, $\tilde{p}$ enumerates grid points on the finer grid, and $\tilde{\xi}_{\tilde{p}_i}$ are the coarse-grid logical coordinates of the fine-grid collocation points. For fine-grid (child) elements that cover the full coarse-grid (parent) element in dimension i the coarse-grid logical coordinates are just the fine-grid collocation points, $\tilde{\xi}_{\tilde{p}_i} = \xi_{\tilde{p}_i}$. For child elements that cover the lower or upper logical half of the parent element in dimension i they are $\tilde{\xi}_{\tilde{p}_i} = (\xi_{\tilde{p}_i} - 1)/2$ or $\tilde{\xi}_{\tilde{p}_i} = (\xi_{\tilde{p}_i} + 1)/2$, respectively. Note that the prolongation operator (18) is just a Lagrange interpolation from the coarser to the finer grid. The interpolation retains the accuracy of the polynomial approximation because the finer grid always has sufficient resolution.

To project data from finer to coarser grids we employ the *restriction operator*

$$R^{l \to l+1} = (P^{l+1 \to l})^T, \tag{19}$$

which is the transpose of the prolongation operator (18). Contrary to the restriction operator listed in Ref. [40] the multigrid restriction involves no mass matrices because it applies to DG residuals, Eq. (9b), which already include mass matrices.

3.   Algorithm

Figure 7 illustrates our task-based implementation of the multigrid V-cycle algorithm to solve linear problems $\mathcal{A}\underline{u} = \underline{b}$. On every grid l we approximately solve the linear problem

$$\mathcal{A}_l \, \underline{u}^{(l)} = \underline{b}^{(l)}, \tag{20}$$

---

[6] See, e.g., Ref. [51] for an introduction to multigrid methods.

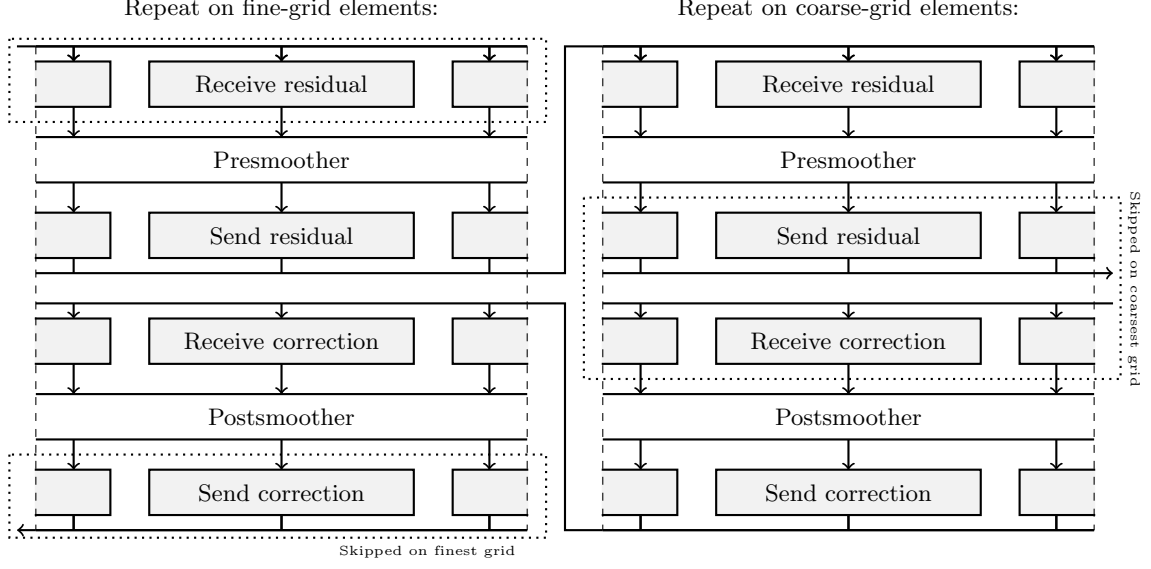[7] See also Ref. [52] for details on the intermesh operators.

FIG. 7. Parallelization structure of the task-based multigrid algorithm (Sec. III C). Elements on all grids perform the same set of tasks, with some tasks skipped on the finest grid and other tasks skipped on the coarsest grid.

where the operator $\mathcal{A}_l$ is the discretization of the elliptic PDEs on the grid l. At the beginning of a V-cycle, on the finest grid $l = 0$, we select $\underline{u}^{(0)} = \underline{u}$ and $\underline{b}^{(0)} = \underline{b}$; hence, approximately solving the original linear problem ("presmoothing"). Then, the remaining residual

$$\underline{r}^{(l)} = \underline{b}^{(l)} - \mathcal{A}_l\,\underline{u}^{(l)} \qquad (21)$$

is restricted to source the linear problem (20) on the next-coarser grid,

$$\underline{b}^{(l+1)} = R^{l \to l+1}\,\underline{r}^{(l)}. \qquad (22)$$

Once presmoothing is complete on the coarsest grid (the "tip" of the V-cycle), we approximately solve Eq. (20) again ("postsmoothing"). The solution of the postsmoothing step is prolongated to the next-finer grid as a correction,

$$\underline{u}^{(l)} \quad \leftarrow \quad \underline{u}^{(l)} + P^{l+1 \to l}\,\underline{u}^{(l+1)}. \qquad (23)$$

Prolongation, correction, and postsmoothing proceed until we have returned to the finest grid, where the correction and postsmoothing apply to the original linear problem. Our choice of presmoother and postsmoother to approximately solve Eq. (20) is detailed in Sec. III D below. Note that on the coarsest level we apply both presmoothing and postsmoothing.

The restriction of residuals to the next-coarser grid and the prolongation of corrections to the next-finer grid incur soft synchronization points. Specifically, only once *all* elements on the finer grid have restricted their residuals to the coarser grid can *all* elements on the coarser grid proceed, though individual coarse-grid (parent) elements

can already proceed once only their corresponding fine-grid (child) elements have sent the restricted residuals. The same applies to the prolongation of corrections from the parent elements back to their children. Therefore, parent elements on coarser grids ideally follow their children when distributed among the available cores, initially and at load-balancing operations.

In contrast to Krylov-subspace algorithms, the multigrid algorithm involves no global synchronization points. In particular, for diagnostic output we perform a reduction to compute the global residual norm $\|\underline{r}^{(l)}\|$ on every grid, but do so asynchronously in order to avoid a synchronization that is not algorithmically necessary. Therefore, we do not use the global residual norm as a convergence criterion for the multigrid solver. Instead, we run a fixed number of multigrid V-cycles, and typically only a single one to precondition a Krylov-subspace solver.

D. Schwarz smoother

On every level of the grid hierarchy the multigrid solver relies on a *smoother* that approximately solves Eq. (20). In principle, the smoother can be any linear solver, including a Krylov-subspace solver as detailed in Sec. III B. However, to achieve good parallel performance we have developed a highly asynchronous additive Schwarz smoother [39, 53, 54], that we employ for presmoothing and postsmoothing on every level of the multigrid hierarchy. Note that we also apply it on the coarsest grid, where some authors apply a dedicated *bottom smoother* instead [55, 56]. Since our coarsest grids are rarely reduced to a single element (see Sec. III C 1), we have, so
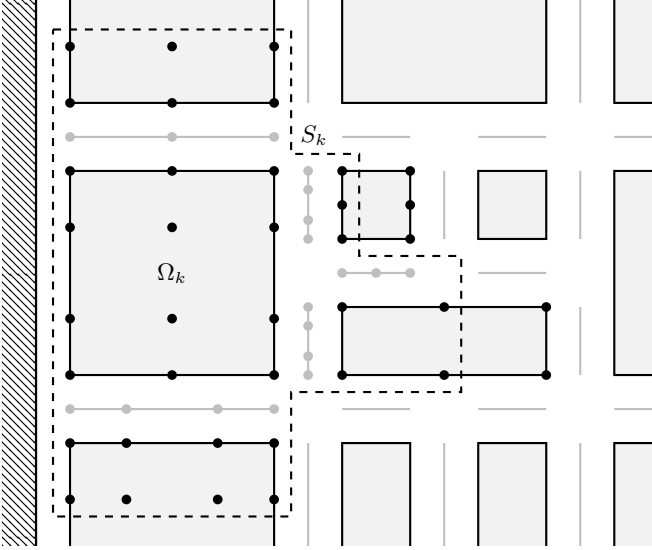
FIG. 8. An element-centered subdomain S_k with $N_{\text{overlap}} = 2$ (dashed line) associated with the element Ω_k in a two-dimensional computational domain. The domain is composed of elements (black rectangles) with their mesh of grid points (black dots) and depicted here in block-logical coordinates. The diagonally-shaded region to the left illustrates an external domain boundary. The light gray lines between neighboring element faces illustrate mortar meshes, which are relevant for the subdomain operator in a DG context but play no role in the Schwarz algorithm [40]. Note that the empty space between the elements in this visualization is not part of the computational domain.

far, preferred the asynchronous Schwarz smoother over direct bottom smoothers. Our Schwarz smoother can be used standalone, as a preconditioner for a Krylov-type linear solver, or as a smoother for the multigrid solver (which may, in turn, precondition a Krylov-type solver).

The additive Schwarz method works by solving many subproblems in parallel and combining their solutions as a weighted sum to converge towards the global solution. The decomposition into independent subproblems makes this linear solver very parallelizable. The Schwarz solver is based on our prototype in Ref. [39], with variations to the subdomain geometry and weighting to take better advantage of our task-based parallelism, and with novel subdomain preconditioning techniques.

1. Subdomain geometry

We partition the computational domain into overlapping, element-centered subdomains $S_k \subset \Omega$, which have a one-to-one association with the DG elements Ω_k. Each subdomain S_k is centered on the DG element Ω_k. It extends by N_{overlap} collocation points into neighboring elements across every face of Ω_k, up to, but excluding, the collocation points on the face of the neighbor pointing

away from the subdomain. Fig. 8 illustrates the geometry of our element-centered subdomains.

The subdomain does not extend into corner or edge neighbors, which is a choice different to both Ref. [54] and Ref. [39]. We avoid diagonal couplings because in a DG context information only propagates across faces, as already noted in Ref. [54]. Elimination of the corner and edge neighbors reduces the complexity of the subdomain geometry, the number of communications necessary to exchange data between elements in the subdomain, and hence the connectivity of the dependency graph between tasks. This element-centered subdomain geometry based solely on face neighbors has proven viable for the test problems presented below, and for our task-based parallel architecture.

The one-to-one association between elements and subdomains allows to store all quantities that define the subdomain geometry local to the central element, i.e., on the same core. The same applies to all data on the grid points of the subdomain. Therefore, operations local to the subdomain require no communication, but communication between overlapping elements is necessary to assemble data on the subdomains, and to make data on subdomains available to overlapped elements.

2. Subdomain restriction

To restrict quantities defined on the full computational domain Ω to a subdomain $S \subset \Omega$ the Schwarz solver employs a *restriction operator* R_S. Since our subdomains are subsets of the grid points in the full computational domain, our restriction operator simply discards all nodal data on grid points outside the subdomain. Similarly, the transpose of the restriction operator, R_S^T, extends subdomain data with zeros on all grid points outside the subdomain.[8]

The Schwarz solver also relies on a restriction of the global linear operator $\mathcal{A}$ to the subdomains. The subdomain operator $\mathcal{A}_S$ on a subdomain S is formally defined as $\mathcal{A}_S = R_S \mathcal{A} R_S^T$. In practice, it evaluates the same DG matrix-vector product as the full operator $\mathcal{A}$, i.e., the left-hand side of Eq. (9b), but assumes that all data outside the subdomain is zero. It performs all interelement operations of the full DG operator, but computes them entirely with data local to the subdomain. Therefore, it requires no communication between cores, as opposed to the global linear operator $\mathcal{A}$ that must communicate data between nearest neighbors for every operator application.

———————

[8] See also Sec. 3.1 in Ref. [54] for details on the subdomain restriction operation.
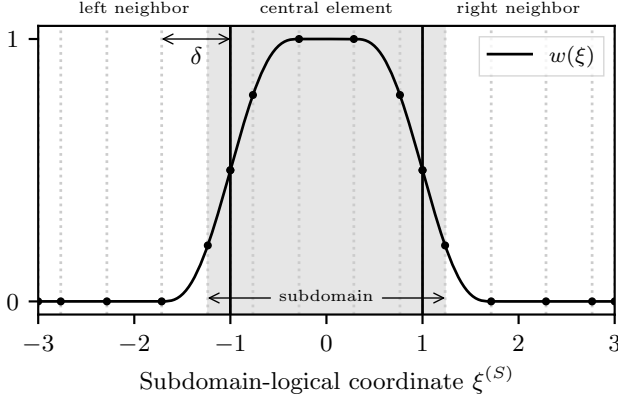
FIG. 9. The one-dimensional weight function $w(\xi)$ for the Schwarz solver. Depicted is an element-centered subdomain in one dimension with $N_{\text{overlap}} = 2$. Every element has $N_k = 6$ LGL collocation points, which includes grid points on the shared element boundaries (black vertical lines). The overlap width δ is the logical coordinate distance to the first point outside the subdomain, where the weight becomes zero.

3. Subdomain problems

On every subdomain S we solve the restricted problem

$$\mathcal{A}_S \,\Delta\underline{u}^{(S)} = \underline{r}^{(S)} \tag{24}$$

for the subdomain correction $\Delta\underline{u}^{(S)}$. Here, $\mathcal{A}_S$ is the subdomain operator and $\underline{r}^{(S)} = R_S \,\underline{r}$ is the global residual $\underline{r} = \underline{b} - \mathcal{A}\underline{u}$ restricted to the subdomain.

The subdomain problems (24) are solved by means of a *subdomain solver*, detailed in Sec. III E. The choice of subdomain solver affects only the performance of the Schwarz algorithm, not its convergence or parallelization properties, assuming the solutions to the subdomain problems (24) are sufficiently precise.

4. Weighting

Once we have obtained the subdomain correction $\Delta\underline{u}^{(S)}$ on every subdomain S, we combine them as a weighted sum to correct the solution,

$$\underline{u} \;\leftarrow\; \underline{u} + \sum_S R_S^T \left(\underline{w}^{(S)} \,\Delta\underline{u}^{(S)} \right), \tag{25}$$

where $\underline{w}^{(S)}$ is a weight at every grid point of the subdomain. This is the *additive* approach of the algorithm, which has the advantage over *multiplicative* Schwarz methods that all subdomain problems decouple and can be solved in parallel.[9] The weighted sum, Eq. (25), is never

---

[9] See also Sec. 3.1 in Ref. [54] for details on multiplicative variants of Schwarz algorithms.

assembled globally. Instead, every element adds the contribution from its locally centered subdomain and from all overlapping subdomains to the components of $\underline{u}$ that resides on the element.

The weights $\underline{w}^{(S)}$ represent a scalar field on every subdomain, which must be conserved as

$$\sum_S R_S^T \,\underline{w}^{(S)} = \underline{1}. \tag{26}$$

We follow Refs. [39, 54] in constructing the weights as quintic smoothstep polynomials, but must account for the missing weight from corner and edge neighbors. Specifically, we compute

$$w_p^{(S)} = W(\boldsymbol{\xi}_p^{(S)}) \tag{27}$$

by evaluating the scalar weight function $W(\boldsymbol{\xi})$ at the logical coordinates $\boldsymbol{\xi}_p^{(S)}$ of the grid points in the subdomain. These subdomain-logical coordinates coincide with the element-logical coordinates of the central element, and extend outside the central element such that $\xi^{(S)} = \pm 3$ coincides with the sides of the overlapped neighbors that face away from the subdomain (see abscissa of Fig. 9). The scalar weight function

$$W(\boldsymbol{\xi}) = \prod_{i=0}^{d} w(\xi^i) \tag{28}$$

is a product of one-dimensional weight functions,

$$w(\xi) = \frac{1}{2}\left(\phi\left(\frac{\xi+1}{\delta}\right) - \phi\left(\frac{\xi-1}{\delta}\right) \right) \tag{29}$$

$$\text{with} \quad \phi(\xi) = \begin{cases} \frac{1}{8}\left(15\xi - 10\xi^3 + 3\xi^5\right) & \xi \in [-1,1] \\ \text{sign}(\xi) & |\xi| > 1, \end{cases} \tag{30}$$

where $\phi(\xi)$ is a second-order smoothstep function, i.e., a quintic polynomial, and $\delta \in (0,2]$ is the *overlap width*. The overlap width is the logical coordinate distance from the boundary of the central element to the first collocation point *outside* the overlap region (see Fig. 9). With this definition the overlap width is nonzero even when the overlap extends only to a single LGL point in the neighbor, which coincides with the element boundary. Furthermore, the weight is always zero at subdomain-logical coordinates $\xi^{(S)} = \pm 3$, even for $\delta = 2$ when the overlap region covers the full neighbor in width. This is the reason we never include the collocation points on the side of the neighbor facing away from the subdomain (see Sec. III D 1). Figure 9 illustrates the shape of the weight function.

To account for the missing weight from corner and edge neighbors, we could add it to the central element, to the overlap data from face neighbors, or split it between the two. We choose to add it to the face neighbors that share a corner or edge, since in a DG context that is where the information from those regions propagates through.
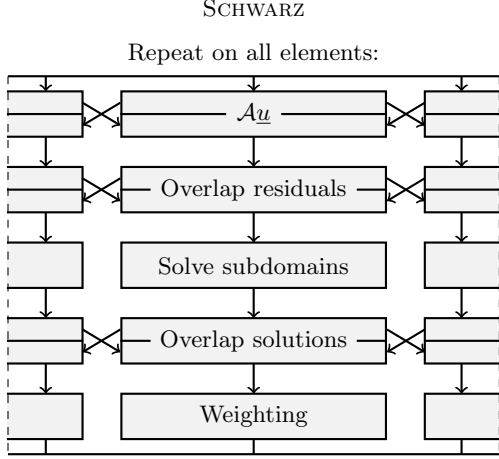
Schwarz

Repeat on all elements:

FIG. 10. Parallelization structure of the task-based Schwarz smoother (Sec. III D).

5. Algorithm

Fig. 10 illustrates our task-based implementation of the additive Schwarz solver. In each iteration, the algorithm computes the residual $\underline{r} = \underline{b} - \mathcal{A}\underline{u}$, restricts it to all subdomains as $\underline{r}^{(S)} = R_S\,\underline{r}$, and exchanges it on overlap regions with neighboring elements. Once an element has received all residual data on its subdomain, it solves the subdomain problem, Eq. (24), for the correction $\Delta\underline{u}^{(S)}$. Since all elements perform such a subdomain solve, we end up with a subdomain solution $\Delta\underline{u}^{(S)}$ on every element-centered subdomain, and the solutions overlap. Therefore, the algorithm exchanges the subdomain solutions on overlap regions with neighboring elements and adds them to the solution field $\underline{u}$ as the weighted sum, Eq. (25).

In order to compute the residual $\underline{r}$ that is restricted to the subdomains to serve as source for the subdomain solves, we must apply the global linear operator $\mathcal{A}$ to the solution field $\underline{u}$ once per Schwarz iteration. This operator application, as well as the steps to communicate the residuals and the solutions on overlaps, incur soft synchronization points through nearest-neighbor couplings. However, once the residuals on overlaps are communicated, all subdomain solves are independent of each other. This constitutes the main source of parallelization in the elliptic solver.

The subdomain solves not only run in parallel, but also scale with the problem size. Increasing the number of grid points in the elements (p refinement) makes the subdomain solves more expensive, but the effectiveness of a Schwarz iteration ideally remains the same. Increasing the number of elements (h refinement) leads to more subdomain solves that can run in parallel. The Schwarz solver does not resolve large-scale modes, so Krylov-type solvers still rely on the multigrid algorithm to scale with h refinement.

E. Subdomain solver

Once overlapping subdomains have exchanged data we can solve all subdomain problems (24) in parallel with data local to each subdomain. Since the subdomain operator $\mathcal{A}_S$ is defined as a matrix-vector product, we solve Eq. (24) with a preconditioned GMRES algorithm, or with conjugate gradients for symmetric positive definite problems. The algorithm is the same as detailed in Sec. III B for our task-based parallel Krylov-subspace linear solver, but implemented separately as a serial algorithm. In future work, we may opt to parallelize the subdomain solver over a few threads with shared memory, but currently we prefer to employ the available cores to solve multiple subdomain problems in parallel. In particular, on coarse multigrid levels where the number of elements can be smaller than the number of available cores, parallelizing the subdomain solves over otherwise idle cores may increase performance.

The iterative Krylov subdomain solves constitute the majority of the total computational expenses, so a suitable preconditioner for them can speed up the elliptic solve significantly. To our knowledge, preconditioners for Schwarz subdomain solvers have gotten little attention in the literature so far. In some cases, the discretization scheme allows to construct a matrix representation for the subdomain operator explicitly, making it possible to invert it directly with little effort [54]. In other cases, the subdomain operator is small enough to build the matrix representation column-by-column (see Sec. III E 2), e.g., when solving the Poisson equation. However, when solving sets of coupled elliptic equations the subdomain operator can easily become too large to construct explicitly. For example, the subdomain operator for the XCTS equations (4) (five variables) on a three-dimensional grid with 8^3 grid points per element and $N_{\mathrm{overlap}} = 2$ is a matrix of size 6400×6400. Stored densely, it requires over $300\,\mathrm{MB}$ of memory per element, so typical contemporary computing clusters with a few GB of memory per core could only hold a few elements per core. Sparse storage reduces the memory cost significantly, but still requires 6400 subdomain-operator applications to construct the matrix representation and a nonnegligible cost to invert and to apply it. With an iterative Krylov-subspace algorithm and a suitable preconditioner we can solve the subdomain problems on an element with significantly lower cost and memory requirements. For example, test problem IV B completes about an order of magnitude faster with the subdomain preconditioner laid out in this section, than with an unpreconditioned GMRES subdomain solver.

1. Laplacian-approximation preconditioner

We support the iterative subdomain solver with a Laplacian-approximation preconditioner. It approximates the linearized elliptic PDEs with a Poisson equation for every variable. A similar preconditioning strategy has

proven successful for the `SpEC` code [14], but in the context of a spectral discretization scheme and a very different linear-solver stack. Specifically, we approximate the subdomain problem, Eq. (24), as a set of independent Poisson subdomain problems

$$\mathcal{A}_S^{\mathrm{Poisson}} \, \Delta \underline{u}_A^{(S)} = \underline{r}_A^{(S)}, \qquad (31)$$

where the index A iterates over all primal variables (see Sec. II). Here, $\mathcal{A}_S^{\mathrm{Poisson}}$ is the DG-discretization of the negative Laplacian $-g^{ij}\nabla_i \nabla_j$ on the subdomain according to Sec. II. For example, a three-dimensional XCTS problem has five variables, so Eq. (31) approximates the linearization (16) of the five equations (4) as

$$\bar{\nabla}^2 \delta\psi = 0, \quad \bar{\nabla}^2 \delta(\alpha\psi) = 0 \quad \text{and} \quad \bar{\nabla}^2 \delta\beta^i = 0. \qquad (32)$$

Depending on the elliptic system at hand we either choose a flat background metric $g_{ij} = \delta_{ij}$, or the background metric of the elliptic system, such as the conformal metric $g_{ij} = \bar{\gamma}_{ij}$ for an XCTS system. A curved background metric reduces the sparsity of the Poisson operator but approximates the elliptic equations better. In practice, we have found little difference in runtime between the flat-space and curved-space Laplacian approximations.

We choose homogeneous Dirichlet or Neumann boundary conditions for $\mathcal{A}_S^{\mathrm{Poisson}}$. For variables and element faces where the original boundary conditions are of Dirichlet type we choose homogeneous Dirichlet boundary conditions, and for those where the original boundary conditions are of Neumann type we choose homogeneous Neumann boundary conditions. This may lead to more than one distinct Poisson operator on subdomains with external boundaries, one per unique combination of element face and boundary-condition type among the variables. Subdomains that have exclusively internal boundaries only ever have a single Poisson operator, which applies to all variables. Note that the choice of homogeneous boundary conditions for the Poisson subdomain problems is compatible with inhomogeneous boundary value problems, because the inhomogeneity in the boundary conditions is absorbed in the fixed sources when the equations are linearized [40].

To solve the Poisson subdomain problems (31), one per variable, we can (again) employ any choice of linear solver, such as a (preconditioned) Krylov-subspace algorithm.[10] However, at this point we have reduced the full elliptic problem down to a single Poisson problem limited to a subdomain that is solved for all variables, or a few Poisson problems on subdomains with external boundaries. Therefore, it becomes feasible, and indeed worthwhile, to construct the Poisson subdomain-operator matrix explicitly and to invert it directly. In particular, the approximate Poisson subdomain-operator matrix remains valid throughout the full nonlinear elliptic solve,

as long as the grid, the background metric, and the type of boundary conditions remain unchanged, so that its construction cost is amortized over many applications.

2. Explicit-inverse solver

We solve the Poisson subproblems of the Laplacian-approximation preconditioner, Eq. (31), with an explicit-inverse solver. It constructs the matrix representation of a linear subdomain operator $\mathcal{A}_S$ column-by-column, and then inverts it directly by means of an LU decomposition. Once the inverse $\mathcal{A}_S^{-1}$ has been constructed, each subdomain problem $\mathcal{A}_S \, \Delta \underline{u}^{(S)} = \underline{r}^{(S)}$ is solved by a single application of the inverse matrix,

$$\Delta \underline{u}^{(S)} = \mathcal{A}_S^{-1} \, \underline{r}^{(S)}. \qquad (33)$$

This means that subdomains have a large initialization cost, but fast repeated solves.

When the explicit-inverse solver is employed as a preconditioner, e.g., to solve the individual Poisson problems of the Laplacian-approximation preconditioner (Sec. III E 1), the inverse does not need to be exact. Therefore, we construct an *incomplete* LU decomposition with a configurable fill-in and store it in sparse format. Then, each subdomain problem reduces to two sparse triangular matrix solves. We use the `Eigen` [57] sparse linear algebra library for the incomplete LU decomposition, which uses the `ILUT` algorithm [58]. The Poisson subdomain-operator matrices $\mathcal{A}_S^{\mathrm{Poisson}}$ have a sparsity of about $90\,\%$, which translates to a sparsity of about $90\,\%$ for the incomplete LU decomposition as well, since we use a fill-in factor of one. The sparsity of the inverse reduces the computational cost for applying it to every subdomain problem, as well as the memory required to store the inverse.

Note that the explicit matrix must be reconstructed when the linear operator changes. The Poisson operators of the Laplacian-approximation preconditioner do not typically change, which makes the explicit-inverse solver very effective (see Sec. III E 1). However, in case we apply the explicit-inverse solver to the full subdomain problem directly, the linearized operator typically changes between every outer nonlinear solver iteration. In such cases, we can choose to skip the reconstruction of the explicit matrix to avoid the computational expense, at the cost of losing accuracy of the solver. When the reconstruction is skipped, the cached matrix only approximates the subdomain operator, but can still provide effective preconditioning.

IV. TEST PROBLEMS

The following numerical tests demonstrate the accuracy, scalability, and parallel efficiency of the elliptic solver on a variety of linear and nonlinear elliptic problems.

All computations were performed on our local computing cluster `Minerva`. It is composed of 16-core nodes, each

---

[10] The absurdity of adding a *third* layer of nested preconditioned linear solvers was not lost on the authors.

(a) Initial error $\underline{u}_0 - \underline{u}_{\mathrm{analytic}}$

(b) Error after 6 unpreconditioned GMRES iterations

(c) Error after 6 Schwarz-smoothing iterations
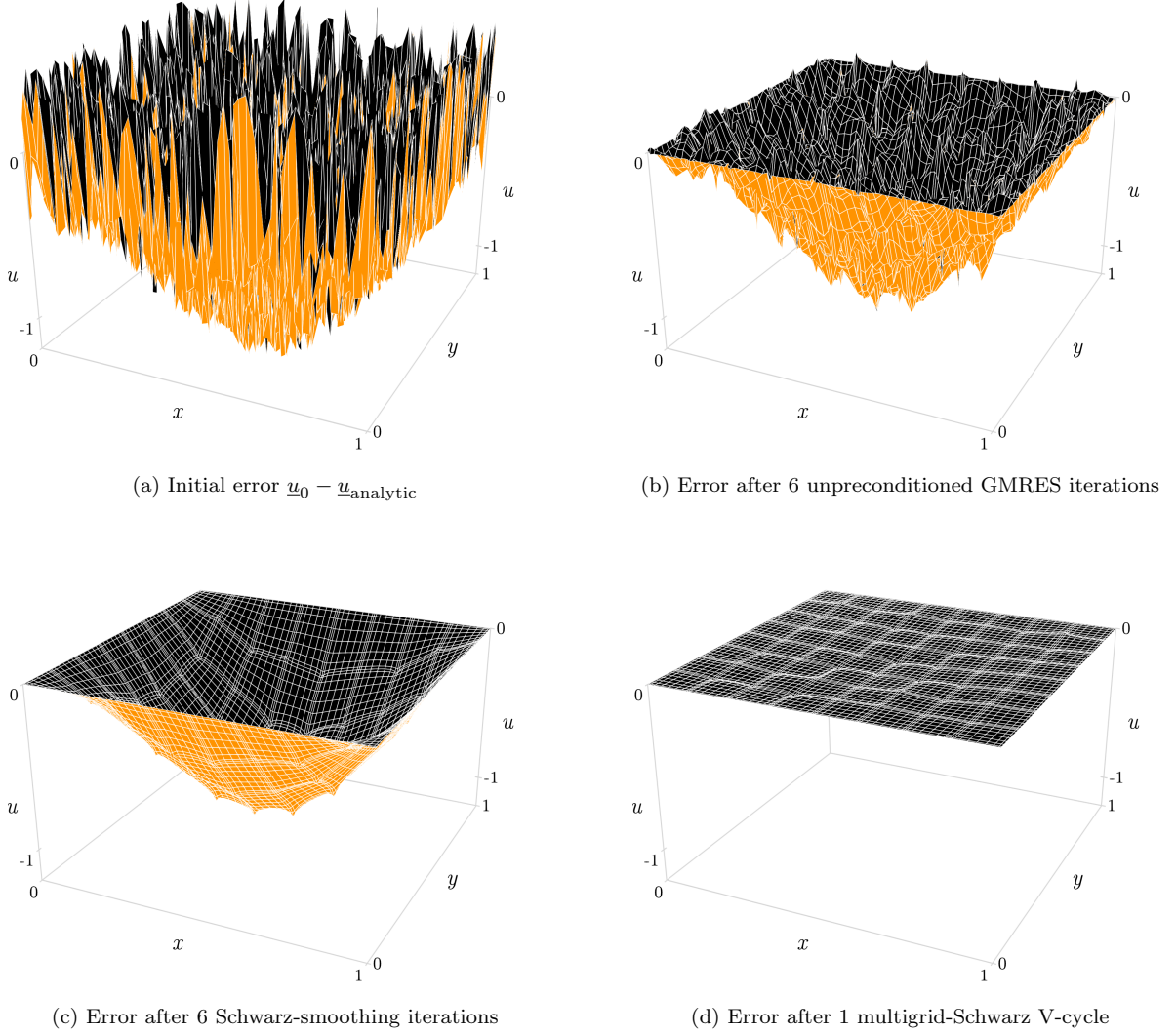
(d) Error after 1 multigrid-Schwarz V-cycle

FIG. 11.   Intermediate errors of the 2D sinusoidal Poisson problem (Sec. IV A) with different components of the elliptic solver. Panel (a) shows the error of the random initial guess. Panels (b)–(d) show the error after six applications of the linear operator.

with two eight-core Intel Haswell E5-2630v3 processors clocked at $2.40\,\mathrm{GHz}$ and $64\,\mathrm{GB}$ of memory, connected with an Intel Omni-Path network. We distribute elements evenly among cores following the strategy detailed in Sec. III, leaving one core per node free to perform communications.

A.   A Poisson problem

As a first test we solve the flat-space Poisson equation in two dimensions,

$$-\partial_i \partial_i u(\boldsymbol{x}) = f(\boldsymbol{x}), \tag{34}$$

for the solution

$$u_{\mathrm{analytic}}(\boldsymbol{x}) = \sin(\pi x)\sin(\pi y) \tag{35}$$

on a rectilinear domain $\Omega = [0,1]^2$ with Dirichlet boundary conditions. This problem is also studied in the context of multigrid-Schwarz methods, with slight variations, in Refs. [53, 54]. To obtain the solution (35) numerically we choose the fixed source $f(\boldsymbol{x}) = 2\pi^2 \sin(\pi x)\sin(\pi y)$, select homogeneous Dirichlet boundary conditions $u^{\mathrm{b}} = 0$, and solve the DG-discretized problem (9) with penalty parameter $C = 1$. We have evaluated the properties of the DG-discretized operator for this problem in Ref. [40]. To assess the convergence behavior of the elliptic solver for this test problem we choose an initial guess $\underline{u}_0$, where each value is uniformly sampled from $[-0.5, 0.5]$.

Figure 11 illustrates the effectiveness of our algorithm in resolving small-scale and large-scale modes in the solution. Plotted is the error to the analytic solution, $\underline{u} - \underline{u}_{\mathrm{analytic}}$. Figure 11a depicts the initial error on a computational domain that is partitioned into 8×8 quadratic elements with 9×9 grid points each. Figures 11b to 11d present the
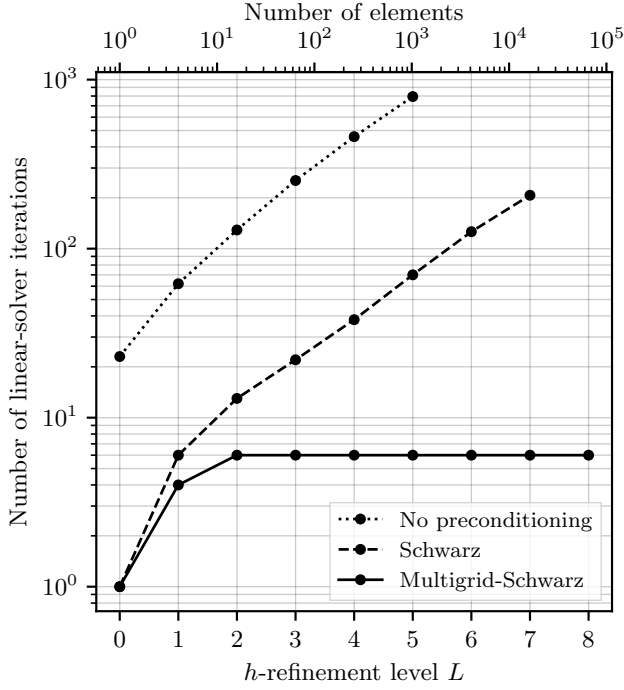
FIG. 12. Number of linear-solver iterations for the Poisson problem (Sec. IV A). The multigrid-Schwarz preconditioner achieves scale independence.

error after six applications of the linear operator, but with different components of the elliptic solver enabled. Figure 11b employs no preconditioning at all, thus reaches six operator applications after six iterations of the GMRES algorithm. It resolves some of the random fluctuations, but retains the large-scale sinusoidal error. Figure 11c preconditions every GMRES iteration with six Schwarz-smoothing steps, thus reaches six operator applications after a single GMRES iteration. The Schwarz smoother uses $N_{\text{overlap}} = 2$. It resolves most of the random fluctuations, but retains the large-scale error. Note that the six operator applications do not accurately reflect the computational expense to arrive at Fig. 11c, because a significant amount of work is spent on the subdomain solves. We employ the explicit-inverse subdomain solver directly here to solve subdomain problems because the Laplacian-approximation preconditioner is redundant for a pure Poisson problem (see Sec. III E), but note that the subdomain solver has no effect on the results depicted in Fig. 11 as long as it is sufficiently precise. Finally, Fig. 11d preconditions every GMRES iteration with a single four-level multigrid-Schwarz V-cycle. The V-cycle employs three Schwarz presmoothing and postsmoothing steps on every level, thus reaches six operator applications on the finest grid after a single GMRES iteration. Again, the number of operator applications is not entirely representative of the computational expense because it disregards the work done on coarser levels. The V-cycle successfully resolves the large-scale error.

Figure 12 presents the number of GMRES iterations that the elliptic solver needs to reduce the magnitude of the residual by a factor of 10^{10}, for a series of h-refined domains. We construct h-refinement levels L by repeatedly splitting all elements in the rectangular domain in two along both dimensions. All elements have 6×6 grid points. Shown in Fig. 12 are an unpreconditioned GMRES algorithm, a GMRES algorithm preconditioned with three Schwarz-smoothing steps per iteration, and a GMRES algorithm preconditioned with one multigrid-Schwarz V-cycle per iteration. The number of multigrid levels is equal to the number L of refinement levels, so that the coarsest level always covers the entire domain with a single element. Every level runs three presmoothing and postsmoothing steps, and subdomains have $N_{\text{overlap}} = 2$. The Schwarz preconditioning alone reduces the number of iterations by a factor of ~ 10, but does not affect the scaling with element size. However, the multigrid-Schwarz preconditioning removes the scaling entirely, meaning the number of GMRES iterations remains constant even when the domain is partitioned into more and smaller elements. The multigrid algorithm achieves this scale independence because it supports the iterative solve with information from coarser grids, including large-scale modes in the solution that span the entire domain. Each preconditioned iteration is typically more computationally expensive than an unpreconditioned iteration, but the preconditioner reduces the number of iterations such that the solve completes faster overall.[11] We find that even for the simple Poisson problem the unpreconditioned algorithm becomes prohibitively slow around $\mathcal{O}(10^3)$ elements ($L = 5$), approaching an hour of runtime and the memory capacity of our ten compute nodes. In contrast, the Schwarz preconditioner reduces the runtime to solve the same problem to below one minute, and the multigrid-Schwarz preconditioner reduces the runtime to only three seconds. Crucial to achieving these runtimes at high resolution are the parallelization properties of the algorithms. In particular, the additional computational expense that the preconditioner spends Schwarz-smoothing all multigrid levels is parallelizable within each level. The following test problem IV B explores the parallelization in greater detail.

Figure 13 gives a detailed insight into the convergence behavior of the elliptic solver for the $L = 1$ configuration. Presented is both the linear-solver residual magnitude $\|\underline{b} - \mathcal{A}\underline{u}\|_2$, and the error to the analytic solution, $\|\underline{u} - \underline{u}_{\text{analytic}}\|_2$. The linear-solver residual (solid line) is being reduced by a factor of 10^{10} by the GMRES algorithm, equipped with the three different preconditioning

———

[11] Note that the cost of unpreconditioned GMRES iterations is eventually dominated by the orthogonalization procedure (see Sec. III B), which slows down the unpreconditioned solve significantly at large iteration counts. This effect can be remedied by *restarting* GMRES variants, but at the cost of possible stagnation. See Sec. 6.5.5 in Ref. [48] for a discussion. Conjugate gradient algorithms avoid this issue for symmetric linear operators.
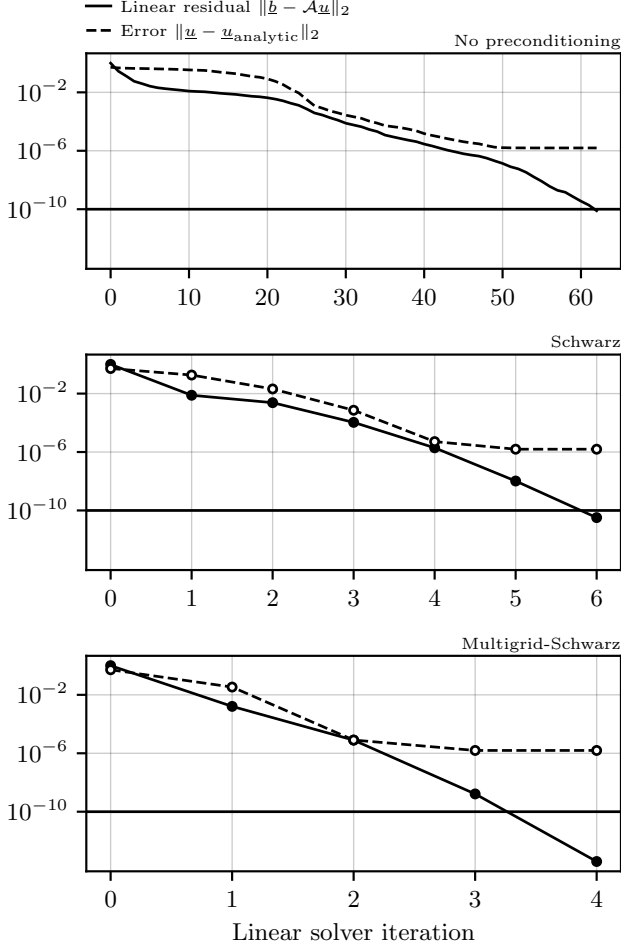
FIG. 13. Convergence of the elliptic solver for the linear Poisson problem with h-refinement level $L = 1$. The solid line shows the relative linear-solver residual magnitude $\|\underline{b} - \mathcal{A}\underline{u}\|_2$, and the dashed line shows the error to the analytic solution, $\|\underline{u} - \underline{u}_{\text{analytic}}\|_2$ as a root mean square over grid points, which approaches the DG discretization error.

configurations explored in Fig. 12. With no preconditioning, the convergence stagnates until large-scale modes in the solution are resolved (see also Fig. 11). The Schwarz preconditioner reduces the number of iterations by about an order of magnitude, and the multigrid-Schwarz preconditioner achieves clean exponential convergence. The error to the analytic solution (dashed line) follows the convergence of the residual magnitude. Once the discrete problem $\mathcal{A}\underline{u} = \underline{b}$, Eq. (1), is solved to sufficient precision, the remaining error $\underline{u} - \underline{u}_{\text{analytic}}$ is the DG discretization error. It is independent of the computational technique used to solve the discrete problem, and determined entirely by the discretization scheme on the computational grid, as summarized in Sec. II and detailed in Ref. [40].[12]

---

[12] For a study of the DG discretization error for this problem see

B. A black hole in general relativity

Next, we solve a general-relativistic problem involving a black hole. Specifically, we solve the Einstein constraint equations in the XCTS formulation, Eq. (4), for a Schwarzschild black hole in Kerr-Schild coordinates. To this end we set the conformal metric and the trace of the extrinsic curvature to their respective Kerr-Schild quantities,

$$\bar{\gamma}_{ij} = \delta_{ij} + \frac{2M}{r} l_i l_j \tag{36a}$$

and

$$K = \frac{2M\alpha^3}{r^2}\left(1 + \frac{3M}{r}\right), \tag{36b}$$

where M is the mass parameter, $r = \sqrt{x^2 + y^2 + z^2}$ is the Euclidean coordinate distance, and $l^i = l_i = x^i/r$.[13] The time-derivative quantities $\bar{u}_{ij}$ and $\partial_t K$ in the XCTS equations (4) vanish, as do the matter sources ρ, S, and S^i. With these background quantities specified, the solution to the XCTS equations is

$$\psi = 1, \tag{37a}$$

$$\alpha = \left(1 + \frac{2M}{r}\right)^{-1/2}, \tag{37b}$$

$$\beta^i = \frac{2M}{r}\alpha^2 l^i. \tag{37c}$$

Note that we have chosen a conformal decomposition with $\psi = 1$ here, but other choices of ψ and $\bar{\gamma}_{ij}$ that keep the spatial metric $\gamma_{ij} = \psi^4 \bar{\gamma}_{ij}$ invariant are equally admissable.

We solve the XCTS equations numerically for the conformal factor ψ, the product $\alpha\psi$, and the shift β^i. The conformal metric $\bar{\gamma}_{ij}$ and the trace of the extrinsic curvature, K, are background quantities that remain fixed throughout the solve. Note that for this test problem the conformal metric $\bar{\gamma}_{ij}$ is not flat, resulting in a problem formulated on a curved manifold.

We employ the DG scheme (9) with penalty parameter $C = 1$ to discretize the XCTS equations (4) on a three-dimensional spherical-shell domain, as illustrated in Fig. 14. The domain envelops an excised sphere that represents the black hole, so it has an inner and an outer external boundary that require boundary conditions. To obtain the Schwarzschild solution in Kerr-Schild coordinates we impose Eqs. (37a) to (37c) as Dirichlet conditions at the outer boundary of the spherical shell at $r = 10M$. We place the inner radius of the spherical shell at $r = 2M$ and impose mixed Dirichlet and Neumann conditions at

---

Fig. 7 in Ref. [40], where the configuration solved in Fig. 13 is circled.
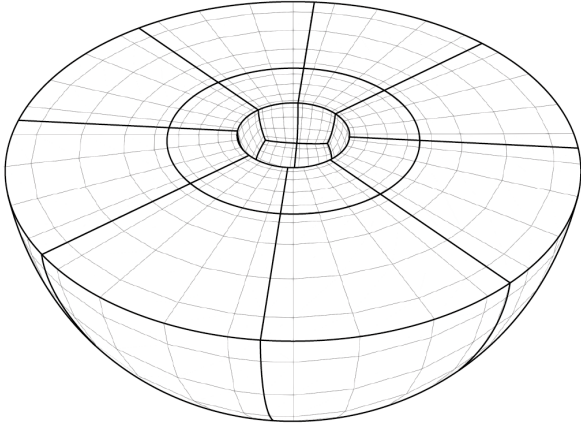
[13] See Table 2.1 in Ref. [1].

FIG. 14. A cut through the uniformly-refined spherical-shell domain used in the black hole problem (Sec. IV B). The domain consists of six wedges with a logarithmic radial coordinate map enveloping an excised sphere. In this example each wedge is isotropically h-refined once, i.e., split once in all three dimensions, resulting in a total of 48 elements. Note the elements are split in half along their logical axes, so the element size scales logarithmically in radial direction just like the distribution of grid points within the elements. Each element has six grid point per dimension, so fields are represented as polynomials of degree five.
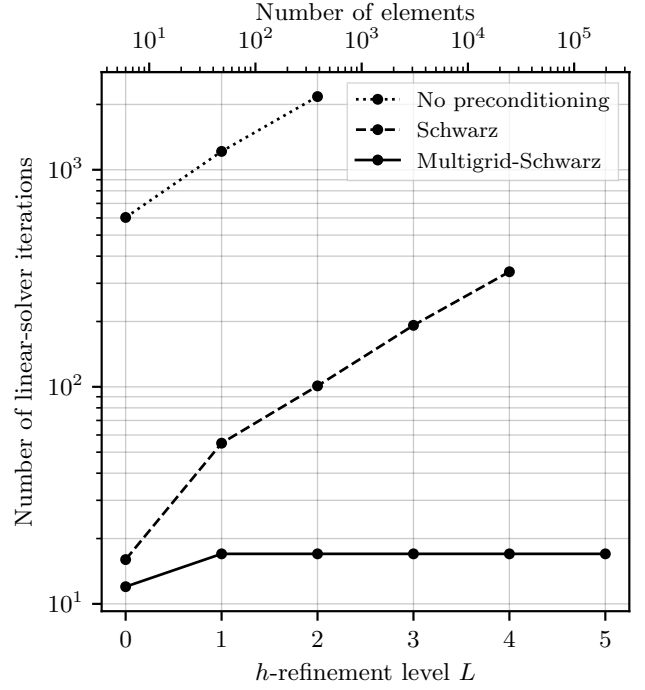

FIG. 15. Number of linear-solver iterations for the black hole problem (Sec. IV B). The multigrid-Schwarz preconditioner achieves scale independence. The $L = 1$ configuration (48 elements) is pictured in Fig. 14.

the inner boundary. Specifically, we impose the Neumann condition $n^i \partial_i \psi = 0$ on the conformal factor, and Eqs. (37b) to (37c) as Dirichlet conditions on the lapse and shift. The reason for this choice is to mimic apparent-horizon boundary conditions, as employed in the following test problem (Sec. IV C). Choosing apparent-horizon boundary conditions for the Kerr-Schild problem is also possible, but requires either an initial guess close to the solution to converge, or a conformal decomposition different from $\psi = 1$. The reason is the strong nonlinearity in the apparent-horizon boundary conditions that takes the solution away from $\psi_0 = 1$ initially. We have confirmed this behavior of the XCTS equations with the SpEC code, and have presented the convergence of the DG discretization error with apparent-horizon boundary conditions for the Kerr-Schild problem in Ref. [40]. With the simpler Dirichlet and Neumann boundary condition we can seed the elliptic solver with a flat initial guess, i.e., $\psi_0 = 1$, $\alpha_0 = 1$ and $\beta_0^i = 0$, which allows for better control of the test problem.

To assess the convergence behavior of the elliptic solver for this problem we successively h-refine the wedges of the spherical-shell domain into more and smaller elements, each with six grid points per dimension. We iterate the Newton-Raphson algorithm until the magnitude of the nonlinear residual has decreased by a factor of 10^{10}. In all configurations we have tested, the nonlinear solver needs five steps and no line-search globalization to reach the target residual. The linear solver is configured to solve the linearized problem, Eq. (16), by reducing its residual magnitude by a factor of 10^4. Schwarz subdomains have

$N_{\text{overlap}} = 2$, and we run three Schwarz presmoothing and postsmoothing iterations on every multigrid level, including the coarsest. Figure 15 presents the total number of linear-solver iterations accumulated over the five nonlinear solver steps. Just like we found for the simple Poisson problem in Fig. 12, the multigrid-Schwarz preconditioner achieves scale-independent iteration counts under h refinement.

Figure 16 presents the convergence behavior of the elliptic solver for the $L = 1$ configuration (pictured in Fig. 14) in detail. The convergence of the nonlinear residual magnitude (dotted line) is independent of the preconditioner chosen for the linear solver in each iteration (solid lines), since the linearized problems are solved to sufficient accuracy (10^{-4}). Similar to the Poisson problem in Fig. 13, the multigrid-Schwarz preconditioning achieves clean exponential convergence, reducing the linear residual by an order of magnitude per iteration. The nonlinear residual magnitude decreases slowly at first, when the fields $\underline{u}$ are still far from the solution, and begins to converge quadratically, following the linear-solver residual, once the fields are closer to the solution and hence the linearization is more accurate (see Sec. III A). The error to the analytic solution (dashed line) approaches the DG discretization error, as detailed in Sec. IV A.[14]

————

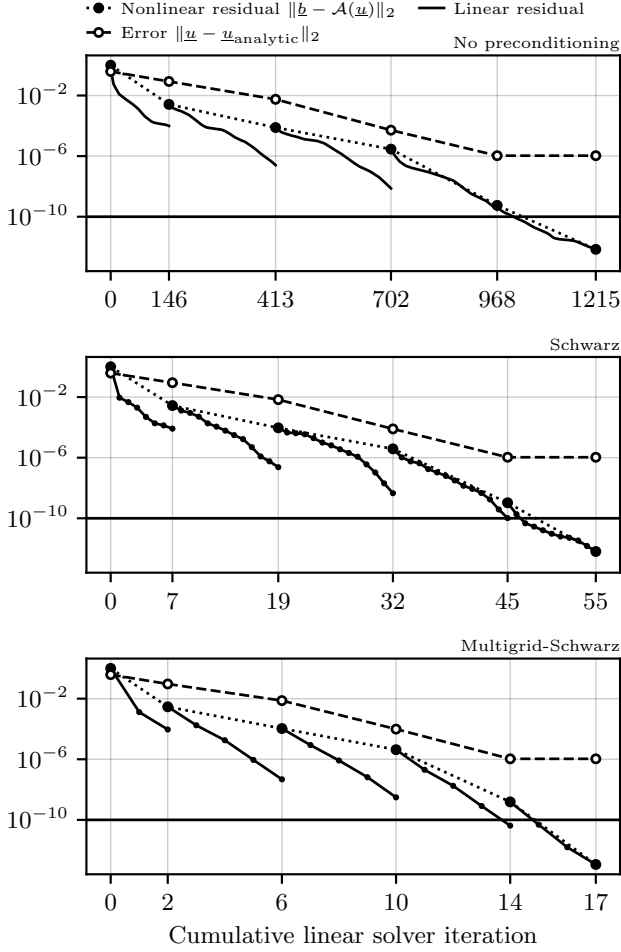[14] For a study of the DG discretization error for this problem see

FIG. 16. Convergence of the Newton-Krylov elliptic solver for the black hole problem with h-refinement level $L = 1$. The dotted line shows the relative residual magnitude of the nonlinear solver, $\|\underline{b} - \mathcal{A}(\underline{u})\|_2$, which is driven by a linear solve in every iteration (solid lines, see Eq. (16)). The dashed line shows the error to the analytic solution, $\|\underline{u} - \underline{u}_{\text{analytic}}\|_2$ as a root mean square over all five variables of the XCTS equations, $\{\psi, \alpha\psi, \beta^i\}$, and over grid points. It approaches the DG discretization error.

To solve subdomain problems here we equip the GM-RES subdomain solver with the Laplacian-approximation preconditioner, and solve the five Poisson subproblems on every subdomain with the incomplete LU explicit-inverse solver (see Sec. III E). Figure 17 illustrates the importance of matching the boundary conditions of the approximate Laplacian to the original problem. When we approximate all five tensor components of the original XCTS problem with a Dirichlet-Laplacian, ignoring that we impose Neumann-type boundary condition on ψ at the inner boundary, some subdomains require a significantly

———

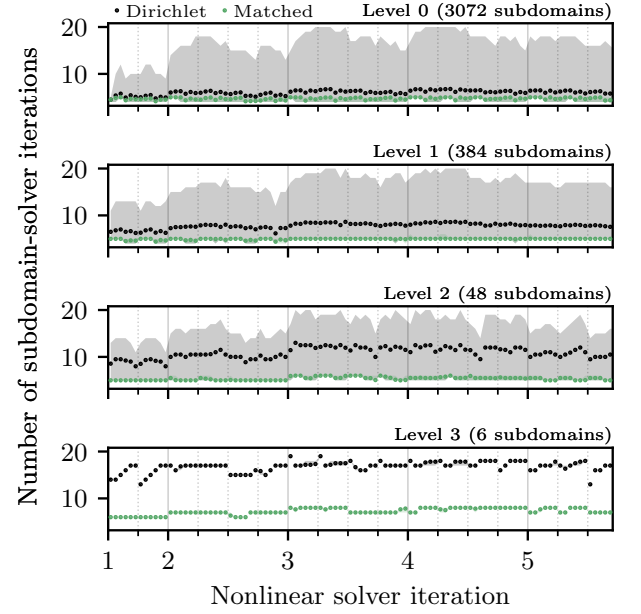Fig. 11 in Ref. [40], where the configuration solved in Fig. 16 is circled.

FIG. 17. Number of subdomain-solver iterations for the black hole problem (Sec. IV B) with the Laplacian-approximation preconditioner. The domain is isotropically h-refined thrice, so the solve involves four multigrid levels. Dots illustrate the average across all subdomains on the level, and shaded regions the smallest and largest number of iterations. When approximating all equations with a Dirichlet-Laplacian (black), subdomains facing the inner excision boundary (see Fig. 14) require more iterations than the average. Matching the Laplacian boundary conditions to the problem (green) reduces the iteration count and resolves the load imbalance.

larger number of subdomain-solver iterations than others. We have confirmed that these subdomains face the inner boundary of the spherical shell. When we use a Laplacian approximation with matching boundary-condition type for these subdomains, they need no more subdomain-solver iterations than interior subdomains. Specifically, the sub-domain preconditioner constructs a Poisson operator matrix with homogeneous Neumann boundary conditions to apply to the conformal-factor component of the equations, and another with homogeneous Dirichlet boundary conditions to apply to the remaining four tensor components. Therefore, subdomains that face the inner boundary of the spherical shell domain build and cache two inverse matrices, and all other subdomains build and cache a single inverse matrix, in the form of an incomplete LU decomposition. Furthermore, when the Laplacian-approximation preconditioner takes the type of boundary conditions into account, we find that it is sufficiently precise so we can limit the number of subdomain-solver iterations to a fixed number. This further balances the load between elements, decreasing runtime significantly in our tests. Therefore, in the following we always limit the number of subdomain-solver iterations to three. With this strategy we find a reduction in runtime of about 50 % compared to the naive
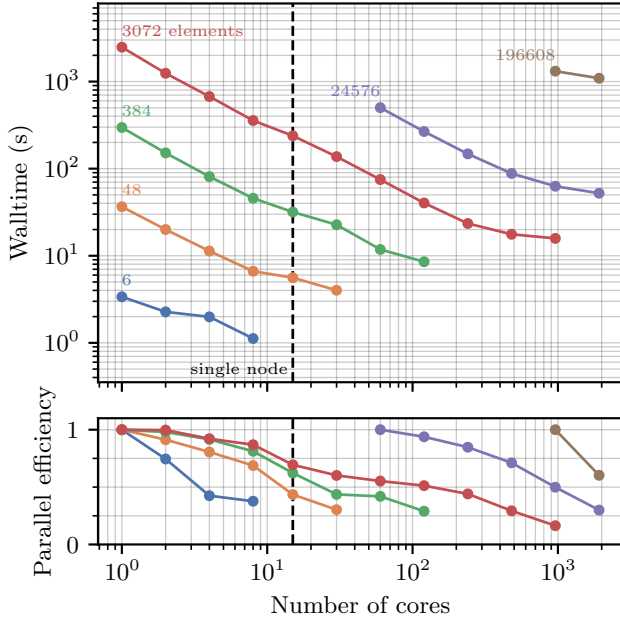
FIG. 18. Parallel scaling of the black hole problem (Sec. IV B).

Dirichlet-Laplacian approximation.

Figure 18 presents the wall time and parallel efficiency of the elliptic solves for the black hole problem on up to 2048 cores, which approaches the capacity of our local computing cluster. We split the domain into more and smaller elements, keeping the number of grid points in each element constant at six per dimension, and solve each configuration on a variable number of cores. These configurations are increasingly expensive to solve, involving up to 42 million grid points, or ~200 million degrees of freedom. They all complete in at most a few minutes of wall time by scaling up to a few thousand cores, until they reach the capacity of our cluster. We compute their parallel efficiency as

$$\text{Parallel efficiency} = \frac{t_{\text{serial}}}{t_{\text{CPU}}}, \qquad (38)$$

where $t_{\text{CPU}} = N_{\text{cores}}\, t_{\text{wall}}$ is the CPU time of a run, i.e., the product of wall time and the number of cores, and t_{serial} is the wall time of the configuration runing on a single core. Since configurations with 24576 elements and more did not complete on a single core in the allotted time of two hours, we approximate t_{serial} with the CPU time of the run with the lowest number of cores for these configurations, meaning that they begin at a fiducial parallel efficiency of one. The parallel efficiency decreases when the number of elements per core becomes small and falls below 25 % once each core holds only a few elements.

Figure 18 also shows that the parallel efficiency decreases more steeply when filling up a single node, than it does when we begin to allocate multiple nodes. We take this behavior as an indication that shared hardware

resources on a node currently limit our parallel efficiency, which is an issue also found in Ref. [55]. We have confirmed this hypothesis by running a selection of configurations on the same number of cores, but distributed over more nodes, so each node is only partially subscribed, and found that runs speed up significantly. We intend to address this issue in future optimizations of the elliptic solver. Possible resolutions include better use of CPU caches, e.g., through a contiguous layout of data on subdomains, or a shared-memory OpenMP parallelization of subdomain solves, so the cores of a node are working on a smaller amount of data at any given time. The parallel efficiency also decreases once we reach the capacity of our cluster, at which point we expect that communications spanning the full cluster dominate the computational expense. We intend to test the parallel scaling on larger clusters with more cores per node in the future. We also plan to investigate the effect of hyperthreading on the parallel efficiency of the elliptic solver.

C. A black hole binary

Finally, we solve a classic black hole binary (BBH) initial data problem, which stands at the beginning of every BBH simulation performed with the SpEC code. Again, we solve the full Einstein constraint system in the XCTS formulation, Eq. (4), but now we choose background quantities and boundary conditions that represent two black holes in orbit. Following the formalism for *superposed Kerr-Schild* initial data, e.g., laid out in Ref. [59, 60], we set the conformal metric and the trace of the extrinsic curvature to the superpositions

$$\bar{\gamma}_{ij} = \delta_{ij} + \sum_{n=1}^{2} e^{-r_n^2/w_n^2} \left(\gamma_{ij}^{(n)} - \delta_{ij} \right) \qquad (39a)$$

and

$$K = \sum_{n=1}^{2} e^{-r_n^2/w_n^2} K^{(n)}, \qquad (39b)$$

where $\gamma_{ij}^{(n)}$ and $K^{(n)}$ are the conformal metric and extrinsic-curvature trace of two isolated Schwarzschild black holes in Kerr-Schild coordinates as given in Eq. (36). They have mass parameters M_n and are centered at coordinates $\boldsymbol{C}_n$, with r_n being the Euclidean coordinate distance from either center. The superpositions are modulated by two Gaussians with widths w_n. The time-derivative quantities $\bar{u}_{ij}$ and $\partial_t K$ in the XCTS equations (4) vanish, as do the matter sources ρ, S and S^i.

To handle orbital motion we split the shift in a *background* and an *excess* contribution [14],
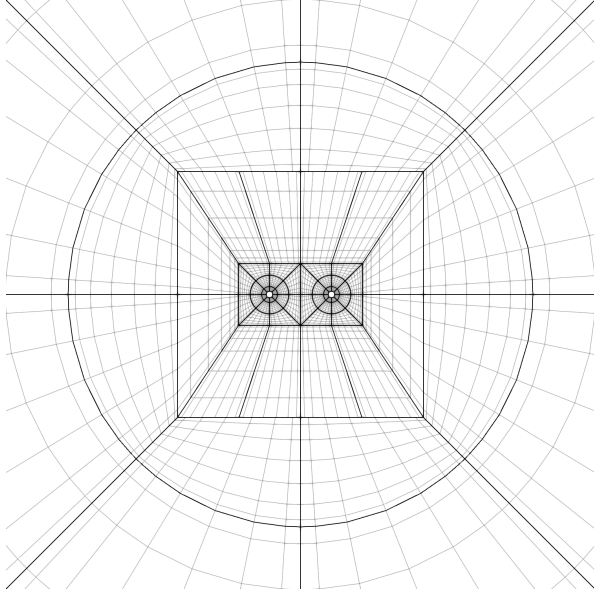
$$\beta^i = \beta^i_{\text{background}} + \beta^i_{\text{excess}}, \qquad (40)$$

and choose the background shift

$$\beta^i_{\text{background}} = (\boldsymbol{\Omega}_0 \times \boldsymbol{x})^i, \qquad (41)$$
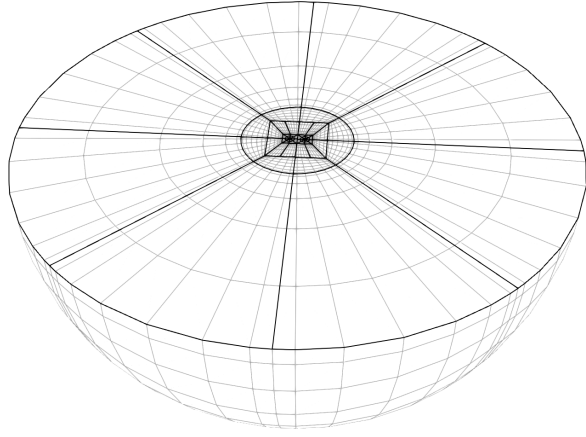
(a) Black hole binary domain

(b) Close-up

FIG. 19.   A cut through the three-dimensional black hole binary domain used in Sec. IV C. It involves two excised spheres centered at $\boldsymbol{C}_n$ along the x-axis and extends to a spherical outer surface at radius R. The domain is h-refined such that spherical wedges have equal angular size, so the cube-to-sphere boundary is nonconforming. All elements in this picture have eight angular grid points, and $\{7, 8, 8, 9, 11, 11\}$ radial grid points in the layers ordered from outermost to innermost.

where $\boldsymbol{\Omega}_0$ is the orbital angular velocity. We insert Eq. (40) in the XCTS equations (4) and henceforth solve them for β^i_{excess}, instead of β^i.

We solve the XCTS equations on the domain depicted in Fig. 19. It has two excised spheres with radius $2M_n$ that are centered at $\boldsymbol{C}_n$, and correspond to the two black holes, and an outer spherical boundary at finite radius R. We impose boundary conditions on these three boundaries as follows. At the outer spherical boundary of the domain we impose asymptotic flatness,

$$\psi = 1, \quad \alpha\psi = 1, \quad \beta^i_{\text{excess}} = 0. \qquad (42)$$

Since the outer boundary is at a finite radius, the solution will only be approximately asymptotically flat. On the two excision boundaries we impose (nonspinning) quasiequilibrium apparent-horizon boundary conditions [61]

$$\bar{s}^k \partial_k \psi = -\frac{\psi^3}{8\alpha} \bar{s}_i \bar{s}_j \left((\bar{L}\beta)^{ij} - \bar{u}^{ij} \right)$$
$$- \frac{\psi}{4} \bar{m}^{ij} \bar{\nabla}_i \bar{s}_j + \frac{1}{6} K \psi^3, \qquad (43a)$$

$$\beta^i = \frac{\alpha}{\psi^2} \bar{s}^i, \qquad (43b)$$

where $\bar{m}^{ij} = \bar{\gamma}^{ij} - \bar{s}^i \bar{s}^j$. Here, $\bar{s}_i = -n_i = \psi^{-2} s_i$ is the conformal surface normal to the apparent horizon, which is opposite the normal to the domain boundary since both are normalized with the conformal metric. For the lapse we choose to impose the isolated solution, Eq. (37b), as Dirichlet conditions at both excision surfaces. Note that this choice differs slightly from Ref. [60], where the *superposed* isolated solutions are imposed on the lapse at both excision surfaces.

To assess the accuracy and parallel performance of the elliptic solver for the BBH initial data problem we solve the same scenario with the SpEC [14, 19] code. In SpEC we successively increment the resolution from Lev0 to Lev6, which correspond to domain configurations determined with an adaptive mesh-refinement (AMR) algorithm. In SpECTRE we simply increment the number of grid points in all dimensions of all elements by one from each resolution to the next, based on the domain depicted in Fig. 19. To compare the solution between the two codes, we interpolate all five fields $u_A = \{\psi, \alpha\psi, \beta^i_{\text{excess}}\}$ to a set of sample points $\boldsymbol{x}_m$. We do the same for a very high-resolution run with SpECTRE that we use as reference, $u_{A,\text{ref}}$, for which we have split every element in the domain in two along all three dimensions. Then, we compute the discretization error for all SpEC and SpECTRE solutions as an L_2-norm of the difference to the high-resolution reference run over all fields and sample points,

$$\|u - u_{\text{ref}}\| := \left(\sum_{A,m} \left(u_A(\boldsymbol{x}_m) - u_{A,\text{ref}}(\boldsymbol{x}_m) \right)^2 \right)^{1/2}. \qquad (44)$$

We have chosen $M_n = 0.4229$, $\boldsymbol{C}_n = (\pm 8, 0, 0)$, $\Omega_0 = 0.0144$, $w_n = 4.8$, $R = 300$, and sample points along the x-axis at $x_1 = 8.846$ (near horizon), $x_2 = 0$ (origin) and $x_3 = 100$ (far field) here. This configuration coincides with our convergence study in Ref. [40], where we list the reference values $u_{A,\text{ref}}(\boldsymbol{x}_m)$ at the interpolation points explicitly.

Figure 20 compares the performance of the BBH initial data problem with the SpEC code. Both SpEC and SpECTRE converge exponentially with resolution, since SpEC employs a spectral scheme and SpECTRE a DG scheme under p refinement. SpECTRE currently needs about 30 % more grid points per dimension to achieve the same accuracy as SpEC. To an extent this is to be expected, since we split the domain into more elements than
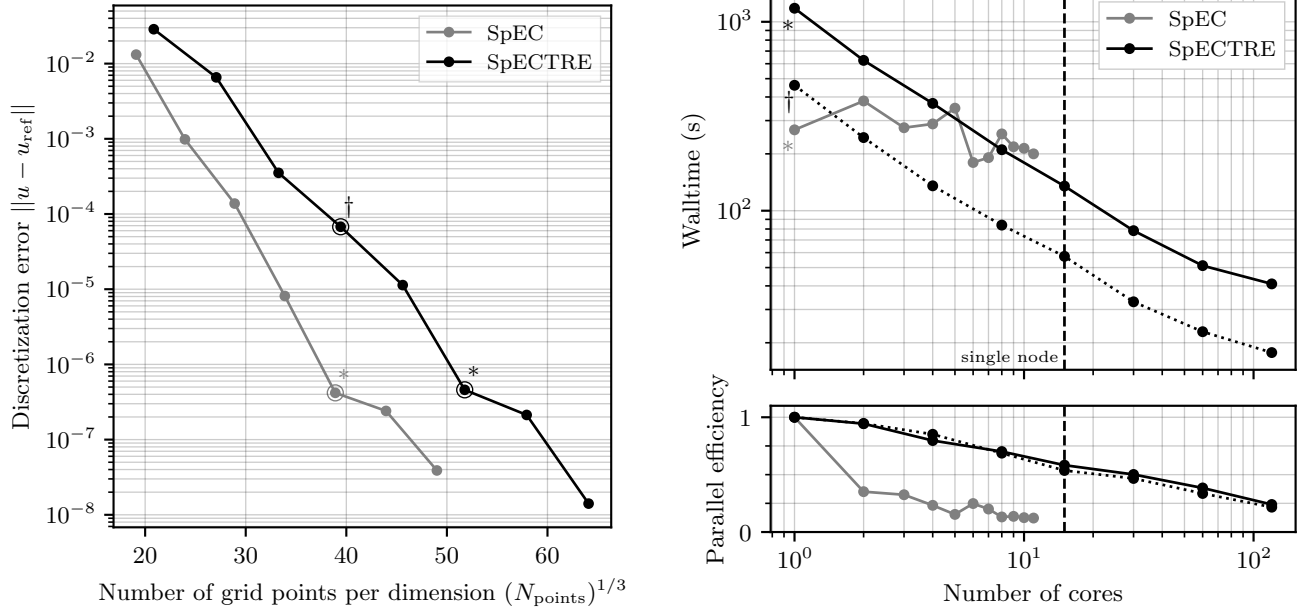
FIG. 20. Comparison of the BBH initial data problem (Sec. IV C) solved with our new elliptic solver in SpECTRE (black), and with the SpEC elliptic solver (gray). *Left:* Both codes converge exponentially with resolution. SpECTRE needs about 30 % more grid points per dimension than SpEC to reach the same accuracy. *Right:* Parallel scaling of both codes. The SpEC elliptic solver scales to at most eleven cores and reaches a speedup of at most a factor of two compared to the single-core runtime. Our new elliptic solver in SpECTRE is faster than SpEC on eight cores, and scales the problem reliably to 120 cores, at which point it is seven times faster than SpEC's single-core runtime. The dotted line corresponds to a configuration with the same number of grid points as SpEC (but lower accuracy), which is faster than SpEC on only two cores.

SpEC does and hence have more shared element boundaries with duplicate points. In particular, SpEC employs shells with spherical basis functions that avoid duplicate points in angular directions altogether. While the SpEC elliptic solver always decomposes the domain into eleven subdomains, each with up to 34 grid points per dimension in our test, our domain in SpECTRE has 232 elements with up to 13 grid points per dimension. However, neither is our BBH domain in SpECTRE optimized as well as SpEC's yet, nor have we refined it with an AMR algorithm. We are planning to do both in future work. Furthermore, SpEC's initial data domain involves overlapping patches to enable matching conditions for its spectral scheme, which our DG scheme in SpECTRE does not need. Therefore, we expect to achieve domain configurations that come closer to SpEC in their number of grid points with future optimizations.

The right panel in Fig. 20 demonstrates the superior parallel performance that our new elliptic solver achieves over SpEC's. We choose the runs marked with * in the left panel because they solve the BBH problem to comparable accuracy. We scale these runs to an increasing number of cores and measure the wall time for the elliptic solves to complete. Since the SpEC initial data domain is composed of eleven subdomains, it can parallelize to at most eleven cores. The runtime decreases by a factor of about 1.5

to 2 when the solve is distributed to multiple cores, but shows little reliable scaling. Some SpEC configurations at higher resolutions have shown slightly better parallel performance, but none that exceeded a factor of about two in speedup compared to the single-core runtime. Our new elliptic solver in SpECTRE, on the other hand, scales reliably to 120 cores, at which point each core holds no more than two elements. On a single core it needs 1176 s where SpEC, with fewer grid points, needs only 268 s, but it overtakes SpEC on eight cores and completes in only 37 s on 120 cores. For reference we have also included a scaling test with SpECTRE that uses the same number of grid points as SpEC but does not yet reach the same accuracy (marked with †). It overtakes SpEC on two cores and completes in only 14 s on 120 cores. The configuration represents a potential improvement in the domain decomposition with future optimizations. We find the parallel efficiency for the BBH configurations behaves similarly to the single black hole configurations we investigated in Sec. IV B.

V. CONCLUSION AND FUTURE WORK

We have presented a new solver for elliptic partial differential equations that is designed to parallelize on

---

computing clusters. It is based on discontinuous Galerkin (DG) methods and task-based parallel iterative algorithms. We have shown that our solver is capable of parallelizing elliptic problems with ∼200 million degrees of freedom to at least a few thousand cores. It solves a classic black hole binary (BBH) initial data problem faster than the veteran code SpEC [19] on only eight cores, and in a fraction of the time when distributed to more cores on a computing cluster. The elliptic solver is implemented in the open-source SpECTRE [29] numerical relativity code, and the results in this article are reproducible with the supplemental input-file configurations.[15]

So far we can solve Poisson, elasticity, puncture and XCTS problems, including BBH initial data in the superposed Kerr-Schild formalism with unequal masses, spins and negative-expansion boundary conditions [60] (in this article we only explored an equal-mass and nonspinning BBH). In the short term we are planning to add the capability to solve for binary neutron star (BNS) and black hole–neutron star (BHNS) initial data, which involve the XCTS equations coupled to the equations of hydrostatic equilibrium.

A notable strength of our new elliptic solver is the multigrid-Schwarz preconditioner, which achieves iteration counts independent of the number of elements in the computational domain. Therefore, we expect our solver to scale to problems that benefit from h refinement, e.g., to resolve different length scales or to adapt the domain to features in the solution. Such problems include initial data involving neutron stars with steep gradients near the surface, equations of state with phase transitions, or simulating thermal noise in thin mirror coatings for gravitational-wave detectors [62, 63].

Our solver splits the computational domain into more elements than the spectral code SpEC to achieve superior parallelization properties. However, the larger number of elements with shared boundaries also means that we need more grid points than SpEC to reach the same accuracy for a BBH initial data problem. Variations of the DG scheme, such as a hybridizable DG method, can provide a possible resolution to this effect [64–66]. Even without changing the DG scheme, we expect that optimizations of our binary compact object domain can significantly reduce the number of grid points required to reach a certain accuracy. Possible domain optimizations include combining the enveloping cube and the cube-to-sphere transition into a single layer of blocks, equalizing the angular size of the enveloping wedges in a manner similar to Ref. [24], or more drastic changes that involve cylindrical or bipolar coordinate maps, such as the domain presented in Ref. [67]. To retain the effectiveness of the multigrid solver it is important to keep the number of blocks in the domain to a minimum under these optimizations. We have shown that our new elliptic solver reaches comparative single-core performance to SpEC when using the same number

of grid points, with the added benefit of parallel scaling. Since every contemporary computer has multiple cores, we prioritize parallelization over single-core performance.

To put the grid points of the computational domain to most effective use, adaptive mesh-refinement (AMR) techniques will be essential. All components of the elliptic solver, including the DG discretization, the multigrid algorithm, and the Schwarz subdomains, already support hp-refined domains. The refinement can be anisotropic, meaning elements can be split or differ in their polynomial degree along each dimension independently. A major subject of future work will be the development of an AMR scheme that adjusts the refinement during the elliptic solve automatically based on a local error estimate, distributing resolution to regions in the domain where it is most needed.

Along with AMR, we expect load balancing to become increasingly important. We currently approximately load balance the elliptic solver at the beginning of the program based on the number of grid points in each element. Charm++, and hence SpECTRE, also support dynamic load-balancing operations that migrate elements between cores periodically, or at specific points in the algorithm. Charm++ provides a variety of load-balancing algorithms that may take metrics such as runtime measurements, communication cost and the network topology into account. When the computational load on elements changes due to p-AMR, or when elements get created and destroyed due to h-AMR, we intend to invoke load balancing to improve the parallel performance of the elliptic solves.

The elliptic solver algorithms can be improved in many ways. The multigrid solver may benefit from an additive variant of the algorithm, which smooths every level independently and combines the solutions [55]. An additive multigrid algorithm has better parallelization properties than the multiplicative algorithm that we employ in this article, since coarse grids do not need to wait for fine grids to send data before the coarse-grid smoothing can proceed. However, the additive multigrid algorithm typically requires more iterations to converge than the multiplicative. Furthermore, multigrid patterns other than the standard V-cycle may accelerate convergence, such as a W-cycle or F-cycle pattern [51].

Schwarz solvers also come in many variations, e.g., involving face-centered subdomains, that we have not explored in this article. Our element-centered subdomains that eliminate corner and edge neighbors have served well for our DG-discretized problems so far, and we have focused on accelerating the subdomain solves with suitable preconditioners. Faster explicit-construction and approximate-inversion techniques for the subproblems in the Laplacian-approximation preconditioner have the greatest potential to speed up the elliptic solver. Possibilities include constructing matrix representations analytically, either from the DG scheme or from an approximate finite-difference scheme, and faster methods to solve the subproblems than the incomplete LU technique we cur-

---

rently employ.

A possible avenue for a more drastic improvement of the elliptic solver algorithm is to replace the multigrid-Schwarz preconditioner, or parts of it, altogether. For example, recent developments in the field of physics-informed neural networks (PINNs) suggest that hybrid strategies, combining a traditional linear solver with a PINN, can be very effective [68, 69]. Hence, an intriguing prospect for accelerating elliptic solves in numerical relativity is to combine our Newton-Krylov algorithm with a PINN preconditioner, use the PINN as a smoother on multigrids, or use it to precondition Schwarz subdomain solves.

Looking ahead, fast, scalable and highly-parallel elliptic solves in numerical relativity not only have the potential to accelerate initial data construction to seed high-resolution simulations of general-relativistic scenarios, and at extreme physical parameters, but they may also support evolutions. For example, some gauge constraints can be formulated as elliptic equations, and solving them alongside an evolution can allow the choice of beneficial coordinates, such as maximal slicing [1]. The apparent-horizon condition is also an elliptic equation, though current NR codes typically find apparent horizons with a parabolic relaxation method [70]. Some NR codes employ a constrained-evolution scheme, which evolves the system in time through a series of elliptic solves, or employ implicit-explicit (IMEX) evolution schemes [71]. Lastly, Einstein-Vlasov systems for collisionless matter involve elliptic equations, as do simulations that involve solving a Poisson equation alongside an evolution, such

as simulations of self-gravitating protoplanetary disks [72–74]. Currently, elliptic solvers are rarely applied to solve any of these problems alongside an evolution because they are too costly. Fast elliptic solves have the potential to enable these applications.

[1] T. W. Baumgarte and S. L. Shapiro, *Numerical Relativity: Solving Einsteins Equations on the Computer* (Cambridge University Press, Cambridge, England, 2010).

[2] G. B. Cook, Initial data for numerical relativity, Living Rev. Relativity **3**, 10.12942/lrr-2000-5 (2000), arXiv:gr-qc/0007085.

[3] H. P. Pfeiffer, The initial value problem in numerical relativity, J. Hyperbolic Differ. Equ. **2**, 497 (2005), arXiv:gr-qc/0412002.

[4] B. P. Abbott *et al.* (LIGO Scientific, Virgo Collaborations), Observation of gravitational waves from a binary black hole merger, Phys. Rev. Lett. **116**, 061102 (2016), arXiv:1602.03837 [gr-qc].

[5] B. P. Abbott *et al.* (LIGO Scientific, Virgo Collaborations), GW170817: Observation of gravitational waves from a binary neutron star inspiral, Phys. Rev. Lett. **119**, 161101 (2017), arXiv:1710.05832 [gr-qc].

[6] R. Abbott *et al.* (LIGO Scientific, VIRGO, KAGRA Collaborations), Observation of gravitational waves from two neutron star–black hole coalescences, Astrophys. J. Lett. **915**, L5 (2021), arXiv:2106.15163 [astro-ph.HE].

[7] B. P. Abbott *et al.* (LIGO Scientific, Virgo Collaborations), GWTC-1: A gravitational-wave transient catalog of compact binary mergers observed by LIGO and Virgo during the first and second observing runs, Phys. Rev. X **9**, 031040 (2019), arXiv:1811.12907 [astro-ph.HE].

[8] R. Abbott *et al.* (LIGO Scientific, Virgo Collaborations), GWTC-2: Compact binary coalescences observed by LIGO and Virgo during the first half of the third observing run, Phys. Rev. X **11**, 021053 (2021), arXiv:2010.14527 [gr-qc].

[9] R. Abbott *et al.* (LIGO Scientific, VIRGO, KAGRA Collaborations), GWTC-3: Compact binary coalescences observed by LIGO and Virgo during the second part of the third observing run, arXiv:2111.03606 [gr-qc] (2021).

[10] E. Gourgoulhon, P. Grandclément, J.-A. Marck, J. Novak, and K. Taniguchi, LORENE, http://www.lorene.obspm.fr.

[11] P. Grandclement, Accurate and realistic initial data for black hole-neutron star binaries, Phys. Rev. D **74**, 124002 (2006), arXiv:gr-qc/0609044.

[12] M. Ansorg, B. Bruegmann, and W. Tichy, A single-domain spectral method for black hole puncture data, Phys. Rev. D **70**, 064011 (2004), arXiv:gr-qc/0404056.

[13] M. Ansorg, A double-domain spectral method for black hole excision data, Phys. Rev. D **72**, 024018 (2005), arXiv:gr-qc/0505059.

[14] H. P. Pfeiffer, L. E. Kidder, M. A. Scheel, and S. A. Teukolsky, A multidomain spectral method for solving elliptic equations, Comput. Phys. Commun. **152**, 253 (2003).

[15] S. Ossokine, F. Foucart, H. P. Pfeiffer, M. Boyle, and

B. Szilágyi, Improvements to the construction of binary black hole initial data, Class. Quantum Gravity **32**, 245010 (2015), arXiv:1506.01689 [gr-qc].

[16] F. Foucart, L. E. Kidder, H. P. Pfeiffer, and S. A. Teukolsky, Initial data for black hole-neutron star binaries: A flexible, high-accuracy spectral method, Phys. Rev. D **77**, 124051 (2008), arXiv:0804.3787 [gr-qc].

[17] N. Tacik *et al.*, Binary neutron stars with arbitrary spins in numerical relativity, Phys. Rev. D **92**, 124012 (2015), arXiv:1508.06986 [gr-qc].

[18] N. Tacik, F. Foucart, H. P. Pfeiffer, C. Muhlberger, L. E. Kidder, M. A. Scheel, and B. Szilágyi, Initial data for black hole–neutron star binaries, with rotating stars, Class. Quantum Gravity **33**, 225012 (2016), arXiv:1607.07962 [gr-qc].

[19] L. E. Kidder, H. P. Pfeiffer, M. A. Scheel, *et al.*, Spectral Einstein Code (SpEC), `black-holes.org/code/SpEC`.

[20] T. Dietrich, N. Moldenhauer, N. K. Johnson-McDaniel, S. Bernuzzi, C. M. Markakis, B. Brügmann, and W. Tichy, Binary neutron stars with generic spin, eccentricity, mass ratio, and compactness - quasi-equilibrium sequences and first evolutions, Phys. Rev. D **92**, 124007 (2015), arXiv:1507.07100 [gr-qc].

[21] W. Tichy, A. Rashti, T. Dietrich, R. Dudi, and B. Brügmann, Constructing binary neutron star initial data with high spins, high compactnesses, and high mass ratios, Phys. Rev. D **100**, 124046 (2019), arXiv:1910.09690 [gr-qc].

[22] P. Grandclément, KADATH: A spectral solver for theoretical physics, J. Comput. Phys. **229**, 3334 (2010).

[23] L. J. Papenfort, S. D. Tootle, P. Grandclément, E. R. Most, and L. Rezzolla, New public code for initial data of unequal-mass, spinning compact-object binaries, Phys. Rev. D **104**, 024057 (2021), arXiv:2103.09911 [gr-qc].

[24] A. Rashti, F. M. Fabbri, B. Brügmann, S. V. Chaurasia, T. Dietrich, M. Ujevic, and W. Tichy, Elliptica: a new pseudo-spectral code for the construction of initial data, arXiv:2109.14511 [gr-qc] (2021).

[25] K. Uryū and A. Tsokaros, New code for equilibriums and quasiequilibrium initial data of compact objects, Phys. Rev. D **85**, 064014 (2012), arXiv:1108.3065 [gr-qc].

[26] A. Tsokaros, K. Uryū, and L. Rezzolla, New code for quasiequilibrium initial data of binary neutron stars: Corotating, irrotational, and slowly spinning systems, Phys. Rev. D **91**, 104030 (2015), arXiv:1502.05674 [gr-qc].

[27] T. Assumpcao, L. R. Werneck, T. P. Jacques, *et al.*, NRPyElliptic: A fast hyperbolic relaxation elliptic solver for numerical relativity, I: Conformally flat, binary puncture initial data, arXiv:2111.02424 [gr-qc] (2021).

[28] H. R. Rüter, D. Hilditch, M. Bugner, and B. Brügmann, Hyperbolic relaxation method for elliptic equations, Phys. Rev. D **98**, 10.1103/PhysRevD.98.084044 (2018), arXiv:1708.07358 [gr-qc].

[29] N. Deppe, W. Throwe, L. E. Kidder, N. L. Vu, F. Hébert, J. Moxon, C. Armaza, G. S. Bonilla, P. Kumar, G. Lovelace, E. O'Shea, H. P. Pfeiffer, M. A. Scheel, S. A. Teukolsky, *et al.*, `SpECTRE v2022.02.17`, 10.5281/zenodo.6127519 (2022).

[30] L. E. Kidder *et al.*, SpECTRE: A task-based discontinuous Galerkin code for relativistic astrophysics, J. Comput. Phys. **335**, 84 (2017), arXiv:1609.00098 [astro-ph.HE].

[31] E. Schnetter, CarpetX, 10.5281/zenodo.6131528 (2022).

[32] W. Zhang *et al.*, AMReX: a framework for block-structured adaptive mesh refinement, J. Open Source Softw. **4**, 1370 (2019).

[33] M. Fernando, D. Neilsen, H. Lim, E. Hirschmann, and H. Sundar, Massively parallel simulations of binary black hole intermediate-mass-ratio inspirals, SIAM J. Sci. Comput. **41**, C97–C138 (2019), arXiv:1807.06128 [gr-qc].

[34] W. Tichy, A. Adhikari, and L. Ji, Numerical relativity with the new Nmesh code, Bull. Am. Phys. Soc. **65** (2020).

[35] M. Bugner, T. Dietrich, S. Bernuzzi, A. Weyhausen, and B. Brügmann, Solving 3D relativistic hydrodynamical problems with weighted essentially nonoscillatory discontinuous Galerkin methods, Phys. Rev. D **94**, 084004 (2016), arXiv:1508.07147 [gr-qc].

[36] B. Daszuta, F. Zappa, W. Cook, D. Radice, S. Bernuzzi, and V. Morozova, GRAthena++: Puncture evolutions on vertex-centered oct-tree adaptive mesh refinement, Astrophys. J. Suppl. **257**, 25 (2021), arXiv:2101.08289 [gr-qc].

[37] A. Reinarz, D. E. Charrier, M. Bader, L. Bovard, M. Dumbser, K. Duru, F. Fambri, A.-A. Gabriel, J.-M. Gallard, S. Köppel, L. Krenz, L. Rannabauer, L. Rezzolla, P. Samfass, M. Tavelli, and T. Weinzierl, ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems, Comput. Phys. Commun. **254**, 107251 (2020).

[38] L. Kale *et al.*, The Charm++ parallel programming system, `https://charm.cs.illinois.edu` (2019).

[39] T. Vincent, H. P. Pfeiffer, and N. L. Fischer, hp-adaptive discontinuous Galerkin solver for elliptic equations in numerical relativity, Phys. Rev. D **100**, 084052 (2019), arXiv:1907.01572 [physics.comp-ph].

[40] N. L. Fischer and H. P. Pfeiffer, Unified discontinuous Galerkin scheme for a large class of elliptic equations, Phys. Rev. D **105**, 024034 (2022), arXiv:2108.05826 [math.NA].

[41] H. Sagan, *Space-Filling Curves* (Springer, New York, NY, 1994).

[42] R. Borrell, J. C. Cajas, D. Mira, A. Taha, S. Koric, M. Vázquez, and G. Houzeaux, Parallel mesh partitioning based on space filling curves, Comput. Fluids **173**, 264 (2018).

[43] W. H. Press, W. T. Vetterling, S. A. Teukolsky, and B. P. Flannery, *Numerical Recipes*, 3rd ed. (Cambridge University Press, Cambridge, England, 2007).

[44] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* (SIAM, Philadelphia, 1996).

[45] P. R. Brune, M. G. Knepley, B. F. Smith, and X. Tu, Composing scalable nonlinear algebraic solvers, SIAM Rev. **57**, 535–565 (2015), arXiv:1607.04254 [math].

[46] S. Balay *et al.*, PETSc, `https://www.mcs.anl.gov/petsc` (2021).

[47] Y. Saad and M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. **7**, 856 (1986).

[48] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. (Society for Industrial and Applied Mathematics, 2003).

[49] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods* (Springer, New York, 2008).

[50] K. Shahbazi, An explicit expression for the penalty parameter of the interior penalty method, J. Comput. Phys. **205**, 401 (2005).

[51] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed. (SIAM, Philadelphia, 2000).

[52] D. Fortunato, C. H. Rycroft, and R. Saye, Efficient operator-coarsening multigrid schemes for local discontinuous Galerkin methods, SIAM J. Sci. Comput. **41**, A3913 (2019).

[53] J. W. Lottes and P. F. Fischer, Hybrid Multigrid/Schwarz algorithms for the spectral element method, J. Sci. Comput. **24**, 45 (2005).

[54] J. Stiller, Robust multigrid for high-order discontinuous Galerkin methods: A fast Poisson solver suitable for high-aspect ratio Cartesian grids, J. Comput. Phys. **327**, 10.1016/j.jcp.2016.09.041 (2016), arXiv:1603.02524 [cs.CE].

[55] A. AlOnazi, G. S. Markomanolis, and D. Keyes, Asynchronous task-based parallelization of algebraic multigrid, in *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '17 (Association for Computing Machinery, New York, NY, USA, 2017).

[56] K. Kang, Scalable implementation of the parallel multigrid method on massively parallel computers, Comput. Math. Appl. **70**, 2701 (2015).

[57] G. Guennebaud, B. Jacob, *et al.*, Eigen, `http://eigen.tuxfamily.org` (2021), version 3.4.

[58] Y. Saad, ILUT: A dual threshold incomplete LU factorization, Numer. Linear Algebra Appl. **1**, 387 (1994).

[59] G. Lovelace, R. Owen, H. P. Pfeiffer, and T. Chu, Binary-black-hole initial data with nearly-extremal spins, Phys. Rev. D **78**, 084017 (2008), arXiv:0805.4192 [gr-qc].

[60] V. Varma, M. A. Scheel, and H. P. Pfeiffer, Comparison of binary black hole initial data sets, Phys. Rev. D **98**, 104011 (2018), arXiv:1808.08228 [gr-qc].

[61] G. B. Cook and H. P. Pfeiffer, Excision boundary conditions for black hole initial data, Phys. Rev. D **70**, 104016 (2004), arXiv:gr-qc/0407078.

[62] G. Lovelace, N. Demos, and H. Khan, Numerically modeling Brownian thermal noise in amorphous and crystalline thin coatings, Class. Quantum Gravity **35**, 025017 (2018), arXiv:1707.07774 [gr-qc].

[63] N. L. Vu, S. Rodriguez, T. Włodarczyk, G. Lovelace, H. P. Pfeiffer, G. S. Bonilla, N. Deppe, F. Hébert, L. E. Kidder, J. Moxon, and W. Throwe, High-accuracy numerical models of brownian thermal noise in thin mirror coatings, arXiv:2111.06893 [astro-ph.IM] (2021).

[64] B. Cockburn, J. Gopalakrishnan, and R. D. Lazarov, Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems, SIAM J. Numer. Anal. **47**, 1319 (2009).

[65] M. Giacomini, R. Sevilla, and A. Huerta, HDGlab: An open-source implementation of the hybridisable discontinuous Galerkin method in MATLAB, Arch. Comput. Methods Eng. **28**, 1941 (2021).

[66] S. Muralikrishnan, T. Bui-Thanh, and J. N. Shadid, A multilevel approach for trace system in HDG discretizations, J. Comput. Phys. **407**, 109240 (2020).

[67] L. T. Buchman, H. P. Pfeiffer, M. A. Scheel, and B. Szilágyi, Simulations of unequal-mass black hole binaries with spectral methods, Phys. Rev. D **86**, 084033 (2012).

[68] S. Markidis, The old and the new: Can physics-informed deep-learning replace traditional linear solvers?, Front. big data **4**, 10.3389/fdata.2021.669097 (2021), arXiv:2103.09655 [math.NA].

[69] V. Guidetti, F. Muia, Y. Welling, and A. Westphal, dNNsolve: an efficient NN-based PDE solver, arXiv:2103.08662 [cs.LG] (2021).

[70] C. Gundlach, Pseudospectral apparent horizon finders: An efficient new algorithm, Phys. Rev. D **57**, 863 (1998), arXiv:gr-qc/9707050.

[71] S. R. Lau, G. Lovelace, and H. P. Pfeiffer, Implicit-explicit (IMEX) evolution of single black holes, Phys. Rev. D **84**, 084023 (2011), arXiv:1105.3922 [gr-qc].

[72] H. Deng, L. Mayer, and H. Latter, Global simulations of self-gravitating magnetized protoplanetary disks, Astrophys. J. **891**, 154 (2020).

[73] P. F. Hopkins, A new class of accurate, mesh-free hydrodynamic simulation methods, Mon. Not. Roy. Astron. Soc. **450**, 53 (2015), arXiv:1409.7395 [astro-ph.CO].

[74] G. L. Bryan *et al.* (The Enzo Collaboration), Enzo: An adaptive mesh refinement code for astrophysics, Astrophys. J., Suppl. Ser. **211**, 19 (2014).

[75] N. L. Vu, `dgpy v0.1`, 10.5281/zenodo.5086181 (2021).

[76] J. D. Hunter, Matplotlib: A 2d graphics environment, Comput. Sci. Eng, **9**, 90 (2007).

[77] T. A. Caswell *et al.*, `matplotlib v3.3.3`, 10.5281/zenodo.4268928 (2020).

[78] T. Tantau, `pgf - a portable graphic format for TeX`, `github:pgf-tikz/pgf` (2021).

[79] J. Ahrens, B. Geveci, and C. Law, `ParaView`: An end-user tool for large-data visualization, in *Visualization Handbook* (Butterworth-Heinemann, Burlington, 2005).