Adaptive hp refinement for spectral elements in numerical relativity

Sarah Renkhoff,[1] Daniela Cors,[1] David Hilditch,[2] and Bernd Brügmann[1]
[1]Theoretical Physics Institute, University of Jena, 07743 Jena, Germany
[2]CENTRA, Departamento de Física, Instituto Superior Técnico IST, Universidade de Lisboa UL, Avenida Rovisco Pais 1, 1049 Lisboa, Portugal

(Received 8 February 2023; accepted 20 April 2023; published 18 May 2023)

When a numerical simulation has to handle a physics problem with a wide range of time-dependent length scales, dynamically adaptive discretizations can be the method of choice. We present a major upgrade to the numerical relativity code BAMPS in the form of fully adaptive, physics-agnostic hp refinement. We describe the foundations of mesh refinement in the context of spectral element methods, the precise algorithm used to perform refinement in BAMPS, as well as several indicator functions used to drive it. Finally, we test the performance, scaling, and the accuracy of the code in treating several 1D and 2D example problems, showing clear improvements over static mesh configurations. In particular, we consider a simple nonlinear wave equation, the evolution of a real scalar field minimally coupled to gravity, as well as nonlinear gravitational waves.

DOI: 10.1103/PhysRevD.107.104043

I. INTRODUCTION

At the heart of many numerical methods for differential equations lies discretization, the transfer of a problem posed on a continuum to a finite set of values. For spatial discretization especially, many approaches exist, from equidistant Cartesian sampling for use with finite-difference methods, to frequency-space decompositions on irregularly shaped subdomains used with finite-element methods.

The choice of discretization has direct impacts on the results of any given simulation, since the discretization determines the solution space itself. A poor choice of sample points might fail to resolve high-frequency components of the solution, or it might cause unphysical numerical noise to accumulate. This poses an inherent challenge to physics applications seeking to resolve *a priori* unknown solutions, which may contain features at different scales.

Using a variable resolution offers a solution to this problem. High resolution can be used in areas where it is required to resolve the physics to a given accuracy—in particular, to resolve features on short length scales, high-frequency modes, steep gradients, or small features requiring high precision, while a lower resolution is used elsewhere, saving resources. If the areas of interest are known beforehand—for example, the center of the domain—fixed mesh refinement (FMR) can be applied "by hand." In a more general setting, however, automatically detecting features of interest is desirable. Allowing such a heuristic detection to determine the discretization used is then referred to as adaptive mesh refinement (AMR). Although originating in finite-element methods, which are naturally suited to combining elements of different shapes and sizes, AMR is used in other types of methods as well.

Developments in numerical relativity were strongly influenced by Ref. [1], which describes a framework for using a flexible AMR scheme for finite-difference methods using rectangular boxes, recursively overlapping each other. Of particular note is the PAMR/AMRD toolset used by Choptuik to pioneer the use of AMR in numerical relativity [2,3] for the study of scalar field collapse in spherical symmetry, which motivated the first application of AMR in $3+1$ dimensions for black holes [4,5]. Box-based AMR approaches are used in GRChombo [6,7] and HAD [8], and nested-box AMR is the basis of many numerical relativity codes to this day, such as BAM [9], AMSS-NCKU [10], CACTUS [11], and the Einstein Toolkit [12]. A recent implementation of block-structured AMR for the Einstein Toolkit called GRaM-X is based on CarpetX and AMReX; see Ref. [13]. See also Ref. [14] for a discussion of the challenges involved in Berger-Oliger-type AMR.

As an alternative to overlapping grids, one can subdivide a given domain into nonoverlapping grids, leading to various types of finite-element methods; see, for example, Ref. [15] for time-dependent partial differential equations (PDEs). Examples of this type of refinement for numerical relativity include SpECTRE [16], dendro-GR [17], GR-Athena++ [18], and Nmesh [19].

Spectral element (SE) methods choose a set of basis functions for each element, so that the approximate solution is given by an expansion in a finite set of basis functions. Examples for SE methods include pseudospectral methods and Galerkin methods. When applying AMR to spectral elements, an additional distinction must be made.

Refinement in that case is possible both in terms of element size (h refinement), or in terms of the order of the spectral series (p refinement). Both techniques can be, and frequently are, used at the same time. Different approaches to dynamical meshes include specially constructed meshes coinciding with physical features, as used, for example, in SpEC [20].

Focusing on applications of AMR in numerical relativity, experience has shown that the use of AMR of some kind is crucial in order to tackle various problems of current interest, such as black hole or neutron star binaries including mergers, and the study of critical collapse. Specifically in the latter case, it is known that solutions near criticality contain features of ever-decreasing scale, which would be inaccessible without the use of progressive mesh refinement (or in some special cases, adapted coordinate systems).

In this paper, we present a major technical upgrade to the numerical relativity code BAMPS, adding hp refinement to its pseudospectral method. While simulations of critical collapse are the science driver for these developments, the theoretical considerations and technical insights are applicable to a much wider class of problems involving time-dependent PDEs. We describe the refinement algorithm in detail, as well as the heuristics used to drive it. The new version of BAMPS enables us to explore certain critical collapse spacetimes with unprecedented efficiency and accuracy; see Ref. [21] for results on the collapse of gravitational waves.

The paper is organized as follows: Section II introduces basic theoretical considerations of convergence and efficiency for hp refinement. In Sec. III, we describe the code used, including the overall structure in Sec. III A, and the AMR algorithm used in Sec. III B. The scaling behavior of the code is described in Sec. III C. Sections IV, V, and VI describe the application of the code to solving a nonlinear wave equation, the collapse of a real scalar field, and the collapse of gravitational waves, respectively.

II. BASICS OF HP REFINEMENT

In this section, we collect general statements about hp refinement regarding error estimates, convergence, and efficiency in the context of high-order spectral element methods. High-order pseudospectral methods are discussed, for example, in Refs. [22–24], but not with a focus on hp refinement, while, for example, Ref. [25] considers hp refinement, but not for high-order elements. Concretely, our focus is on pseudospectral methods with a polynomial order of around 10 or higher, which are applied to predominantly smooth solutions.

The goal of AMR and hp refinement is to optimize efficiency by adjusting the numerical method locally in space (and possibly also in time, which we would call hpt refinement). To prepare for the discussion of hp refinement, we show in Fig. 1 an elementary example

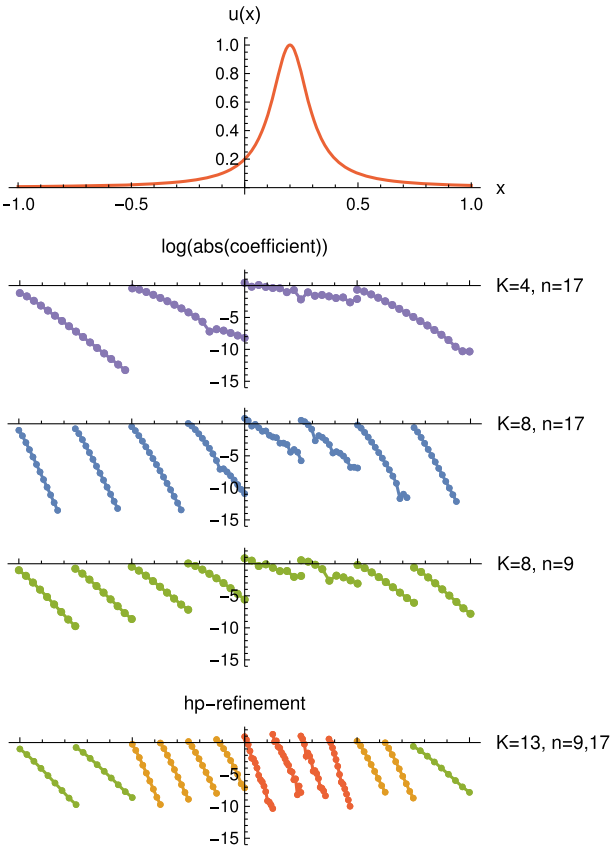

FIG. 1. Example of hp refinement with spectral elements. A function $u(x)$ is discretized on K elements with n collocation points, where n can vary across elements. The top panel shows the function $u(x)$ along the x axis, while the lower panels show the falloff of the coefficients of the polynomial expansion in each element on a logarithmic scale. The panels are aligned such that the horizontal axes for x and for the falloff with increasing index match for each element. In the bottom panel, the result of a specific hp refinement is shown for a target error of 10^{-7}.

of the discretization of a function $u(x)$, here $u(x) = 1/[1 + 100(x - \frac{1}{5})^2]$, on spectral elements. There are varying numbers of K elements, and each element may be discretized by varying numbers of n collocation points. For $K = 4$, $n = 17$, high-order convergence is visible, but the element near the peak in $u(x)$ shows rather slow but exponential convergence. For $K = 8$, $n = 17$, which is one step of h refinement compared to the previous panel, errors become smaller, but the outer region could be considered over-resolved, while the region near the peak is still somewhat inaccurate. For $K = 8$, $n = 9$, there is systematic convergence, but errors are comparatively large. In the bottom panel, $K = 13$ elements with varying h levels and p levels are used for an efficient representation with a target absolute error of around 10^{-7}.

For hp refinement, we have to define refinement criteria, which requires defining a measure of the local efficiency, so that we can optimize the global efficiency. The efficiency of

a numerical discretization can be defined in different ways: for example, (a) accuracy per number of grid points, (b) accuracy per number of floating point operations, (c) accuracy per memory usage, or (d) accuracy per run time. We introduce several of these theoretical considerations here before turning to a practical implementation in Sec. III.

A. Errors and convergence in a hp method

For simplicity, consider scalar functions on the real line. Consider a numerical approximation $u(x)$ to a sufficiently smooth function $u_0(x)$ that converges at order p in the grid spacing h, $u = u_0 + O(h^p)$. More explicitly,

$$u_{h,p}(x) = u_0(x) + c_p(x)h^p + O(h^{p+1}), \tag{1}$$

where the coefficient $c_p(x)$ is a function independent of h (as suggested by a Taylor expansion), and

$$\epsilon(x; h, p) = c_p(x)h^p \tag{2}$$

is the leading-order error term. Here, we introduce h as a small parameter that characterizes the discretization of the domain, the derivatives, and/or the PDE equation, as opposed to just a displacement in x.

For concreteness, we consider a discretization where $h = L/K$, L is the length of a 1D interval, and K is the number of cells or elements in the interval, such that each cell has size h. For finite difference (FD) methods, h is the "grid spacing" with one point per cell, and p is, in particular, the order of the FD approximation of derivatives. For a spectral element method [pseudospectral (PS), but also discontinuous Galerkin (DG)], each cell of size h can be discretized by $n = N + 1$ collocation points x_i. Polynomials of polynomial order up to N are used, so that $p = n$ is the typical order of convergence. (In this notation, a linear function has $N = 1$, and the term beyond linear is at $n = 2$.)

For example, a second-order FD method with error $O(h^2)$ becomes 4 times as accurate with 2 times the number of grid points, since $h \sim \frac{1}{K}$ and error $\epsilon \sim \frac{1}{K^2}$. For a SE method with error $O(\epsilon)$ for p points per cell, 2 times the number of collocation points implies an error of $O(\epsilon^2)$, since $\epsilon(p) \sim h^p$, and $\epsilon(2p) \sim h^{2p} \sim \epsilon(p)^2$.

Consider now an hp refinement method using spectral elements, where both h and p can be varied freely. A key question is under which conditions it is more efficient to decrease h or to increase p in order to reduce the error.

In terms of the computational degrees of freedom, or simply in terms of the number of grid points, there are a total of $N_{\text{total}} = Kn$ grid points for the interval of size L, assuming K equal-sized cells with n points each. Given $N_{\text{total}} = Kn$, we ask the question of how the error changes if we double the degrees of freedom by using either $2K$ cells with n points or K cells with $2n$ points for the same total of $2Kn$ points overall. Since $h = L/K$ and $p = n$, this corresponds to the calculation of $\epsilon(h, p)$ for different h and p. For an example, see Fig. 1.

The short answer is that spectral methods always win, or that p refinement is always more efficient than h refinement, *assuming certain conditions are met*. In particular, the functions we consider (i) are smooth, $u_0 \in C^\infty$; (ii) fit the convergent regime, where $\epsilon \ll 1$; and (iii) have infinite numerical precision.

Let us first assume that conditions (i)–(iii) are satisfied and consider the error estimate given in Eq. (1). For the comparison of different methods, we have to include the coefficients $c_p(x)$ in the calculation. Restricted to a single location x, we have

$$\epsilon(h, p) = c_p h^p, \tag{3}$$

$$\epsilon\left(\frac{h}{2}, p\right) = c_p\left(\frac{h}{2}\right)^p = \frac{1}{2^p}\epsilon(h, p), \tag{4}$$

$$\epsilon(h, 2p) = c_{2p}h^{2p} = \frac{c_{2p}}{c_p}h^p\epsilon(h, p). \tag{5}$$

This implies that for the relative changes

$$\frac{\epsilon(\frac{h}{2}, p)}{\epsilon(h, p)} = \frac{1}{2^p}, \qquad \frac{\epsilon(h, 2p)}{\epsilon(h, p)} = \frac{c_{2p}}{c_p}h^p, \tag{6}$$

and for the comparison of h and p refinement

$$\frac{\epsilon(h, 2p)}{\epsilon(\frac{h}{2}, p)} = \frac{c_{2p}}{c_p}2^p h^p, \tag{7}$$

p refinement is more efficient than h refinement if the last expression is less than 1. In particular, this is the case if we are in the "convergent regime" (assuming $\frac{c_{2p}}{c_p} < 1$) and if $2h < 1$. Furthermore, for decreasing h, the condition on the coefficients, which are derived from the function u_0, becomes less restrictive. For exponential convergence, we in fact expect $\frac{c_{2p}}{c_p} \ll 1$. As an aside, doubling p is a rather large step for spectral methods, but similar relations hold for increasing the order p in increments of $+1$ or $+2$, etc. We consider doubling here to use the same number of degrees of freedom in both cases.

With regard to condition (i), for nonsmooth functions $u_0 \in C^k$, a spectral method may lead to algebraic convergence of order (for example) $k + 2$ [22]. Other algebraic orders are possible, including half-integer powers like $k + \frac{3}{2}$. Sufficient smoothness of u_0 is an obvious criterion for the applicability of spectral methods, while nonsmoothness does not automatically rule out spectral methods, because they still converge. A large class of problems deals with shocks and conservation laws, which is beyond our discussion here (see, for example, Ref. [26]).

With regard to condition (ii), while for large K and n the exponential convergence is stronger than any polynomial factor, for small K and n there may be a regime where h refinement and p refinement offer comparable gains. In other words, spectral methods win beyond some minimal number of grid points. This minimum tends to be rather small, but it depends on the function u_0.

Let us assume that the convergent regime for some functions u_0 starts only beyond a specific n_{conv}. For example, u_0 could be constructed from just high-order components in a polynomial basis. Only when $p = n$ is sufficiently large, $n \geq n_{\mathrm{conv}}$, can the error start to decrease exponentially. A special example would be $u_0(x) = \sin(kx)$, which is only captured accurately if a Fourier series includes sufficiently high frequencies. For a single Fourier mode, h refinement might help, since on a cell of size $h/2$ fewer cycles have to be resolved, and relative to the smaller cell, short wavelengths have become longer wavelengths. Another special case is a well-localized wave packet. Suppose this wave packet is well resolved for n points in a cell of size h. If the same packet is placed in a single cell with $h' = 10h$, then $n' > n$ points are probably needed to resolve the packet. The localized wave packet suggests hp refinements that vary with position. A function u_0 may be optimally approximated by a pair (h, p) in some region, where it varies slowly on the scale of (h, p), while in another region u_0 may exhibit high-frequency features that require smaller h and/or larger p for optimal efficiency.

With regard to condition (iii), spectral element methods are usually implemented with finite numerical accuracy. Depending on the calculations required to, for example, find the approximate solution u of a PDE, round-off errors may be the dominant, limiting factor for the accuracy of the final result. This leads to the typical result that a spectral method may show an exponential drop in the error as n is increased—say, down to $\epsilon \approx 10^{-12}$ for $n \approx 20$—but increasing n further does not decrease the error further; instead, the error $\epsilon(n)$ levels off and may even increase for increasing n. From the perspective of hp-refinement criteria, if the round-off floor has been reached by increasing n for p refinement, it should be more efficient to switch to h refinement. While the overall accuracy may be the same, computations on two cells of size $h/2$ with n points can be expected to be more computationally efficient than for a single cell of size h with $2n$ points. See the discussion of computational efficiency that follows.

For FD methods, it may be hard to reach this level of round-off error, while for SE methods, reaching round-off may be straightforward, but a major design objective is to, say, lower the round-off floor from 10^{-5} to 10^{-12} by an improvement of the spectral method. A concrete example is given by the optimized spectral methods for certain elliptic problems by Ansorg *et al.* [27], which achieve this in part by a clever choice of coordinates.

In conclusion, theoretical estimates for errors and the convergence of SE methods are available. Their applicability, however, depends on the smoothness of u_0, and on application in the convergent regime, which may require reaching some minimal resolution, as well as avoiding the numerical round-off for high-order schemes.

B. Operation count for hp refinement

In a spectral element method for the numerical solution of a PDE, the most expensive part of the calculation is often the computation of the numerical derivatives. For a 1D problem with n grid points, the algebraic (nonderivative) part of the right-hand-side computation requires typically on the order of $O(n)$ floating point operations, while computing derivatives can require $O(n^2)$ operations for direct matrix methods. In special cases, this may be $O(n \log n)$—say, for FFT-Chebychev methods—but since n in many examples is comparatively small ($n < 50$), direct matrix methods for correspondingly small n are more efficient—for example, Refs. [22,28]. Hence, we will restrict the discussion to the case $O(n^2)$.

In d dimensions, consider a cube with the same number of collocation points in each direction and

$$V = n^d \tag{8}$$

points in total. We define the vector of function values u_i with a linear index $i = 0, \ldots, V-1$. Multiplication of a vector with V elements by a square $V \times V$ matrix is in general an $O(V^2)$ operation. However, let us consider spectral methods for first-order PDEs that involve only the standard partial derivatives ∂_j in each direction. For $d = 1$, the $n \times n$ derivative matrix D is dense (full). For $d = 2$, we define sparse derivative matrices $D_1 = I \times D$ and $D_2 = D \times I$, which are $n^2 \times n^2$ matrices defined by the Kronecker product of D with the $n \times n$ identity matrix. Assuming that the sparsity is utilized in the computation, computing derivatives is not an $O(V^2)$ operation, but one of order

$$n_{\mathrm{ops}}(n) = O(nV) = O(n^{d+1}). \tag{9}$$

Using an estimate for the number of floating point operations in a spectral method, we can define efficiency as "accuracy per work," or inefficiency as "work per accuracy"; or, since accuracy is the inverse of error (the smaller the error, the higher the accuracy), inefficiency is error times work. For the hp method as described above, with K cells in each dimension, error times work is

$$\alpha_{\mathrm{ineff}} = O(h^p) n_{\mathrm{ops}}(n) = O(K^{-n}) O(n^{d+1}). \tag{10}$$

Comparing SE methods in terms of α_{ineff} takes the work in terms of the operation count into account.

For p refinement, with K and d constant, α_{ineff} is the product of an exponential in $-n$ and a polynomial with leading order n^{d+1}. Therefore, assuming that we consider the regime of exponential convergence for the spectral method, the exponential reduction of the error outweighs the polynomial increase of the operation count. Incidentally, since d is constant, this would also hold if the operation count for derivatives were $O(n^{2d})$ instead of $O(n^{d+1})$.

For h refinement, the work for 1D derivatives per element remains constant, while for p refinement, the work for 1D derivatives increases, which may be compensated by faster convergence. For h refinement with a factor 2, the operation count is $2^d n_{\text{ops}}(n)$ when ignoring overhead at cell interfaces, while the error decreases by a factor of $1/2^n$. The overall gain in efficiency (reduction in inefficiency) is $1/2^{n-d}$. For p refinement by a factor of 2, the operation count increases by a factor of order 2^{d+1}, while the error decreases by a factor of $\frac{c_{2p}}{c_p} h^p$.

For hp refinement, h refinement is cheaper in terms of additional operations, but overall, p refinement is still favored in the regime defined by conditions (i)–(iii) in the previous section. In practical applications, some measure of the work should be included, and the balance between h and p refinement can then be based on the actual values of n, K, and the work estimate.

Similar considerations hold for specific time-stepping algorithms. On the one hand, the clustering of points on spectral elements may require smaller time steps—for example, for explicit Runge-Kutta time-stepping. On the other hand, this is rewarded with smaller errors in the time discretization, which in turn affects the work and accuracy balance of hp refinement.

C. Memory usage of hp refinement

Another aspect of efficiency is memory usage. FD and SE methods often have comparable memory usage of order $O(V)$. In particular, the additional storage for differentiation matrices is often small compared to the storage of $O(n_{var}V)$ function values for n_{var} variables. We can ask which method requires the least resources to achieve a fixed error bound—say, a maximum pointwise error of $\epsilon = 10^{-9}$. Considering hp refinement for such an error bound, in principle we can also balance h refinement and p refinement to minimize V.

However, in many of our applications, memory (RAM) size limitations of hardware are not an issue. With BAMPS, we rarely perform time evolutions that use the maximum memory available per node; rather, we utilize more nodes than required for memory to gain access to more CPUs for faster execution. Employing more nodes also implies a higher memory bandwidth for accessing the same total memory.

D. Run-time of hp refinement

Having discussed convergence with h and p, operation counts for typical hp methods, and memory constraints, we turn to another important metric for performance: How quickly does the code run? In particular, for large simulations on supercomputers, the bottom line may be how many CPU hours are required.

While the number of floating-point operations required to complete a simulation is a relevant metric, different methods implemented by different codes and run on different hardware typically show run times that are not trivially correlated with flops (floating-point operations per second). This is not that surprising, because in a complicated code like BAM or BAMPS, solving complicated problems (Einstein equations), there are many nontrivial issues like maintaining an optimal load of floating-point units (including vector units like AVX, etc.), parallelization, and memory access.

As a key example, even when considering just the right-hand-side calculation (and not complications of AMR or parallelization) of the Baumgarte-Shapiro-Shibata-Nakamura (BSSN) or generalized harmonic gauge (GHG) formulations, BAM and BAMPS seem to be significantly bound by memory access speed, rather than by the flops achievable by the hardware. Typical $(3+1)$-dimensional simulation codes are often memory-bound rather than compute-bound (in significant parts of the calculation). Consider arithmetic intensity—that is, the number of floating operations performed on each byte read from RAM into the CPU—or work per memory traffic. For current CPU/RAM platforms, and for the large number of 3D variables in a typical BSSN or GHG calculation, the arithmetic intensity is often comparatively low, so that the memory channels are saturated, while the CPUs/FPUs are not. Note that this is the case even when using differentiation matrices in our spectral methods. SE methods tend to be more compute intensive than FD methods, but in typical examples even the BAMPS code is in part memory-bound, and not compute-bound. See Ref. [28] on BAMPS, where there are indications that the strong performance gain on a GPU compared to a CPU can be attributed mostly to the much faster memory interface of the GPU, rather than the increased flops for the GPU.

In conclusion, for dynamic AMR with hp refinement, it is worthwhile to include either offline or live benchmarks that measure the speed of execution for different parts of the code. While the theoretical considerations above can be a good guideline for some aspects of performance, the optimal balance of h and p refinement to reach a certain error criterion should also consider run-time benchmarks. Furthermore, benchmarks of this type can be helpful for load balancing of parallel hp refinement.

III. THE BAMPS CODE

This section first gives a high-level overview before describing specific parts of the algorithm in detail, such as refinement indicators and load balancing.

A. Grid setup

In BAMPS, the numerical domain is organized in a hierarchical structure. The full total of the domain is divided into up to 13 different "patches," with each patch being defined by a patch type (cube, spherical shell, or transitional patch) and a direction (positive X, negative Y, etc.), corresponding to its position and orientation on the overall domain. Each patch is constructed as a cube in patch-local coordinates (u, v, w). Depending on the patch type, specific coordinate transformations are used to map the patch-local coordinates to a global set of Cartesian coordinates (x, y, z). These transformations are constructed such that they match at the boundaries between patches. Finally, the boundaries of patches with different orientations are connected so as to form an overall spherical domain, referred to as a "cubed sphere" [29]. The particular construction used in BAMPS is described in more detail in Ref. [30].

Each patch contains any number of "grids," which are self-contained spectral elements. On each grid, every spatial dimension is discretized using a nodal spectral grid of points, using either Chebyshev-Gauss-Lobatto or Legendre-Gauss-Lobatto collocation points in the (u, v, w) coordinates. Data are transmitted between these grids using a penalty method [24,31,32], which aims to ensure energy conservation. This treatment of grid boundaries is not unique; see Refs. [33–35] for discussion and examples of other approaches to data transmission.

For the present work, we consider parallelization with the MPI (message-passing interface). In a normal parallelized operation, a variable list of grids is distributed across a fixed number of MPI processes. Each MPI process contains roughly the same number of grids (see Sec. III B 4 on how this is achieved). Each process stores its grids and their metadata in its own copy of a singleton data structure, which also contains global properties of the domain. While some metadata are used to track grids not local to a particular process, no part of the state vector is duplicated between different processes.

Since the publication of Ref. [30], several changes to the grid structure have been made: most notably, that symmetry boundaries—that is, those boundaries of the domain on which symmetry conditions are enforced—are located in between grids, rather than being implemented using "half grids" overlapping the symmetry plane.

This necessitates more care to preserve the parity of the solution, since without extra steps, the filtering mechanism used to prevent the growth of unphysical high-frequency modes will lead to parity violations, which in some cases seems to lead to unstable behavior at the outer boundaries. We partially counteract this by providing such boundaries with virtual neighbor grids, containing data that exactly fulfill the parity conditions. We can then apply the same penalty method used between all internal grids as a boundary condition. Additionally, parity conditions on derivatives are generally enforced explicitly by setting the derivatives of fields with even parity to zero at the boundary.

B. Adaptive mesh refinement

The general grid structure described above serves as the base mesh, onto which AMR is then applied. Using the algorithm described in Sec. III B 2, we recursively generate new grids in order to supply additional resolution where needed to adequately represent the solution, we and consolidate these fine grids back into fewer coarser ones once the additional resolution is no longer required (h refinement). The per-grid resolution is also adjusted according to a separate refinement indicator (p refinement).

To determine which areas of the domain require refinement or coarsening, we employ different indicator functions, as described in Sec. III B 3, which are evaluated periodically during the evolution, typically every 100 time steps.

1. Data structures

A popular data structure for h refinement in three dimensions uses oct-trees (or octrees)—for example, Refs. [16–18,36]—which are natural for recursive, local domain decomposition. A given grid, the "parent" grid, is subdivided in each spatial direction by a factor of 2, resulting in eight (in 3D), four (in 2D), or two (in 1D) "child" grids. Given a set of root grids, which in BAMPS corresponds to an initial grid configuration based on coordinate patches, each child grid has a unique parent grid, and the data structure also keeps track of neighborhood relations between grids.

In essence, BAMPS implements a set of distributed, parallelized trees of grids, which is similar, for example, to the forest of octrees in Ref. [37]. Incidentally, some versions of the numerical relativity code BAM starting with Ref. [38] were internally based on octrees as well, which provided experience with a prototype for a MPI-parallel octree implementation. However, fully local mesh refinement was rarely used; rather, the octree was configured for the nested, moving box algorithm for compact binaries as in Ref. [9], and later replaced by a more efficient box-based algorithm for large, nested boxes [39].

For BAMPS, we decided to explore a nonstandard implementation of octrees, where the data structures do represent a virtual tree, but the actual implementation is directly based on lists and local list operations. Assuming familiarity with elementary data structures like lists and linked lists, a tree is a specific graph of nodes with links

between parent and child nodes. A binary tree is an efficient way to store and retrieve data in an ordered list of nodes. For AMR, the ordering is given by the geometry of the domain decomposition in 3D (or 2D, 1D). Since for PDEs, a key operation is the exchange of information between neighboring nodes across grid interfaces, for convenience and efficiency, an implementation may also store links (pointers) between nodes and their neighbors ("siblings" and "cousins"); see, for example, the "fully threaded trees" in Ref. [36], even though some of this information is redundant and can be deduced from the parent/child links. An alternative to linked lists and linked trees is based on hash-based node identification, which can be combined efficiently with the concept of space-filling curves [40,41].

In BAMPS, we do not implement a general-purpose octree, but guided by the actual requirements of parallelized, adaptive hp refinement, we arrived at the following—in some aspects simpler—model of a list of grids. The construction is based on the following observations and application-specific simplifications.

First, the physical domain is covered by a collection of elements (or grids), which for the purpose of parallelization is organized as a global, ordered list corresponding to a space-filling curve; see the discussion in Sec. III B 4. By ordering the elements in this way, each can be uniquely identified by its position, or index, in the list. This makes it possible to encode information about neighborhood relations between elements simply by storing the list indices of neighboring elements. We choose to maintain this ordered list of grids directly in all AMR operations.

Second, notice that AMR operations like refining and coarsening correspond to local list operations, assuming a well-formed octree and a z-ordered space-filling curve. In particular, the creation of child grids corresponds to replacing a single node in the list with several nodes, which by construction of the z-ordered curve for an octree places the new elements next to each other in the list. Coarsening means replacing several child grids with a single (parent) grid, which again is a local operation in the z curve—in particular, since only children without children of their own can be removed. This allows us to maintain the congruency of all stored list indices by adding and subtracting precomputed offsets to and from stored indices, based on the refinement and coarsening operations of each element and its neighbors.

Third, given a list of n elements stored contiguously in memory, inserting and removing n elements is potentially an order-$O(n^2)$ operation. For the h refinement considered here, however, we build the list of refinement flags ahead of time, and the construction of the new list can then be performed by a single sweep of $O(n^1)$ operations. In practice, two sweeps are required—one to determine new element indices, and one to assemble the new array—but overall, this is still an $O(n)$ operation.

While this method of a global list, implementing "a tree without a tree," works efficiently (see Sec. III C on the performance and scaling of BAMPS), we leave it to future work to investigate whether there are significant differences in performance and/or simplicity compared to other octree implementations.

2. Algorithm

The AMR procedure consists of several steps:

(1) Evaluate the h-refinement indicator function on each grid.
(2) Generate a set of initial h-refinement flags.
(3) Modify the h-refinement flags to satisfy refinement constraints.
(4) Apply h-refinement operations.
(5) Perform load balancing to consolidate grids that are marked to be coarsened together.
(6) Apply h-coarsening operations.
(7) Evaluate the p-refinement indicator function on each grid.
(8) Apply p-refinement and p-coarsening operations.
(9) Apply final load balancing.

We choose to use refinement indicators that are purely grid-local functions, and as such can be evaluated by each MPI process on all local grids without the need for interprocess communication. This precludes indicators based on large-scale feature detection, or preemptive refinement based on indicator values of neighboring grids.

The generated indicator function values are then compared to an interval of values deemed acceptable. This interval is set by the user as an external parameter, and appropriate values depend strongly on the chosen indicator function. For example, the interval $[10^{-12}, 10^{-9}]$ has been found to give good results for the truncation error estimator (Sec. III B 3 a). Suitable estimator bounds for the smoothness heuristic (Sec. III B 3 b) depend strongly on the equations and quantities being evolved. See Table I for examples.

If the indicator value is above the allowed maximum, the grid is flagged for refinement. Vice versa, if the value is below the set minimum, it is flagged for coarsening. Typically, the smoothness heuristic is chosen as the

TABLE I. Examples of refinement indicator settings that are known to generate useful amounts of refinement for evolving linear and nonlinear wave equations, as well as the generalized harmonic gauge (GHG) formulation of GR.

	Smoothness	Truncation error
Wave eq. (linear)	[0.005, 0.05]	$[10^{-12}, 10^{-9}]$
Wave eq. (nonlinear)	[0.001, 0.01]	$[10^{-12}, 10^{-10}]$
GHG (Kerr)	[0.001, 0.01]	$[10^{-12}, 10^{-9}]$
GHG (Brill wave)	[0.001, 0.005]	$[10^{-15}, 10^{-9}]$
GHG + scalar field	[0.001, 0.0025]	$[10^{-12}, 10^{-9}]$

h-refinement indicator, to make sure each grid represents a sufficiently small part of the solution as to be smooth. If this cannot be achieved, at least the use of this indicator should contain any nonsmooth parts of the solution in as small a region as possible, and help to preserve the overall quality of the solution. Our notion of "smoothness" here is not related to the smoothness of the continuum solution, but rather to the quality of its numerical approximation. The above procedure will assign each grid a "target h level" that falls within ± 1 of its current h-refinement level. Note that while each application of the AMR algorithm will only raise or lower the refinement level of a given grid by 1, it can be applied iteratively until all indicator bounds are satisfied on all grids.

As the next step requires information about the flags of neighboring grids, which might be stored on different MPI processes, these initial h-refinement flags are then synchronized across all MPI processes. The flags generated by this procedure are then modified to ensure that the end state satisfies the constraints for a "legal" BAMPS grid. As illustrated in Fig. 2, we impose a 1:2 condition on the grid structure, meaning that in crossing any boundary between grids, the h-refinement level may only change by 0 or ± 1:

Algorithm 1. Satisfying the 2:1 condition for h refinement.

```
l ← l_max
while l ≥ 0 do
  for each grid do
    for each neighbor of grid do
      if neighbor.level < l − 1 then
        neighbor.level ← l − 1
      end if
    end for
  end for
  l ← l − 1
end while
```

This constraint makes the structure of possible grids conform to the leaves of an octree. Since grids do not overlap, only the leaves of that tree actually exist as grids. However, coarsening operations require knowledge of which groups of grids correspond to "siblings" in the virtual grid tree. For this reason, each grid is assigned a unique sequence of numbers that mark its position in the tree: for example, a grid marked with the sequence $\{0, 3, 2\}$ is the second child of the third child of a parent grid with the ID 0.

At the same time, refinement is given precedence over coarsening and nonrefinement. In combination, these principles lead to potentially propagating refinement into grid regions that were not originally flagged as needing refinement. Since coarsening is given the least precedence, it will only be applied if an entire group of sibling grids—meaning grids that share a parent node in the virtual grid octree—is flagged for coarsening. Otherwise, the coarsening flags are disregarded.

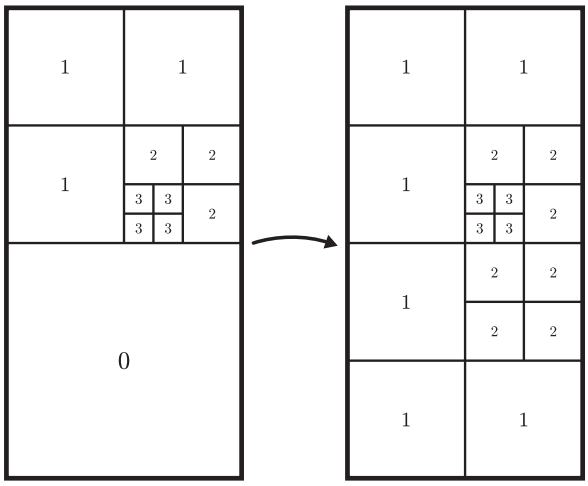

FIG. 2. A grid structure that conforms to an octree, but is not compatible with the 2:1 refinement condition, and a modified structure which is compatible, demonstrating the resulting refinement propagation, both with labels denoting the refinement level.

After the refinement flags have been modified to satisfy the constraints, all h-refinement operations can take place.

To perform h refinement on a grid, 2^d new grids are created, with d being the dimension of the domain, to fill the region currently represented by their parent grid. The solution is then interpolated from the parent grid onto its children, using Lagrange interpolation with barycentric weights [42]. Finally, the metadata about the parent's neighbor grids is transferred to the generated children. This includes their position in the global grid list, their level, and the patch they belong to, as well as information about the size and orientation of neighboring grids. The new metadata depend on both the current refinement level and the refinement flags of any neighboring grids. A set of nested lookup tables is used to generate the metadata that correspond to the grid state after all refinement and coarsening has taken place.

Because refining a grid into several children is always a process-local operation, and it generates new grids that must be taken into account and potentially moved during load balancing, all refinement operations can and must happen before load balancing.

Similarly, because coarsening a group of grids requires all grids involved to be local to a single process, coarsening operations must happen after a pass of load balancing, during which grids flagged for coarsening are shifted to consolidate all sibling grids on the same MPI process. To facilitate this, the grid-weighting system of the load-balancing procedure is used. Within a group of grids flagged for coarsening, all but one are assigned a weight of 0, since they will cease to exist. The other grids are assigned a weight depending on their individual resolution

(see Sec. III B 4). This results in such sibling grids always being assigned to the same grid list segment.

The coarsening procedure functions like the refinement procedure in reverse; one new grid is created, and data from the old grids are interpolated onto it. Like in the refinement step, all metadata from the old grids must be consolidated and modified to reflect the end state of the AMR operation. Here, too, a set of nested lookup tables is used to generate the correct metadata.

Once all h-refinement operations are complete, the refinement indicator selected for p refinement is evaluated on the resulting grids, and refinements are performed wherever indicated. In contrast to h refinement, p refinement can be performed as a fully grid-local operation, without the need for intermittent communication between processes. Only once all refinement and coarsening operations have been completed are the resulting refinement levels communicated between neighboring grids to ensure the proper allocation of boundary data buffers.

Finally, a second load-balancing step is performed, as the relative computational load of grids may have changed when their resolution was changed.

3. Refinement indicators

To determine whether a grid should be refined, coarsened, or kept at its current level, one or more indicator functions are used.

Each indicator fulfills the following criteria:

(1) It is grid-local, meaning it requires only data from a single grid to be evaluated.
(2) It evaluates to a single real number for each grid, which can be compared against bounds set by the user. Values higher than a set threshold trigger refinement, while values lower than a set threshold allow for coarsening.
(3) It returns a dimensionless value, so it can be used in a problem-agnostic way.

Indicators can be used interchangeably to drive either h or p refinement, or both. Both h and p refinement are assigned their own indicator function with their own bounds for the returned value. See Figs. 3 and 4 for examples of different refinement indicator functions applied to a Gaussian at different h-refinement levels.

Truncation error estimate.—One tool to gauge the quality of data representation on spectral grids is to consider the decay of the coefficients of the spectral series. For a smooth function, when using an appropriate polynomial basis, these coefficients will decrease exponentially as the order increases, given enough resolution to capture high orders [22]. The magnitude of the highest-order coefficient can thus be used to estimate the truncation error of the series. We use this to construct a refinement indicator similar to that described in Ref. [43], which aims to keep the estimated truncation error below a specified value everywhere on the domain.

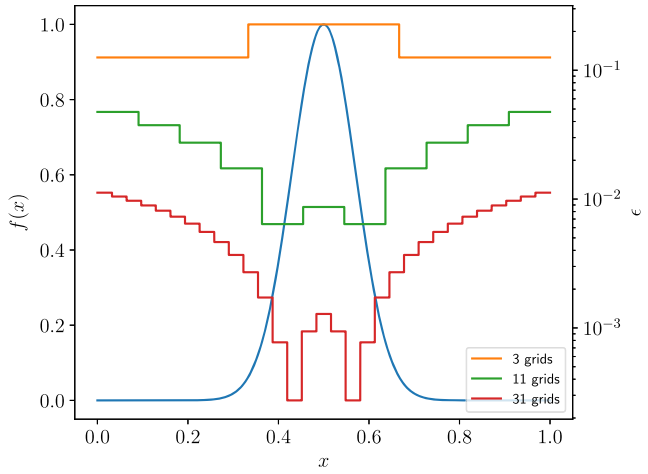

FIG. 3. Values returned by the smoothness-estimation-based refinement indicator evaluating a Gaussian (blue), at different grid sizes.

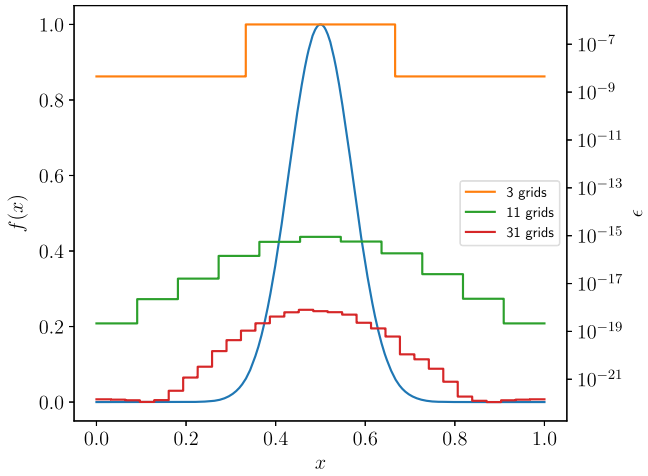

FIG. 4. Values returned by the truncation-error-based refinement indicator evaluating a Gaussian (blue), at different grid sizes.

To compute this indicator, we extract the spectral coefficients c_i from the nodal representation. For a given function basis $\{\varphi_i(x)\}$ and nodes $\{x_i\}$,

$$u(x) = \sum_j c_j \varphi_j(x), \qquad u_i = u(x_i), \qquad V_{ij} = \varphi_j(x_i), \tag{11}$$

$$u_i = V_{ij} c_j, \qquad c_i = V^{-1}_{ij} u_j. \tag{12}$$

Figure 1 shows an example of the c_i. For the indicator, we first compute the spectral coefficients c_i along each line of grid points. By default, these coefficients are then normalized with respect to c_0, which results in a measure of

relative truncation error. This can optionally be disabled to estimate the absolute error instead. Some of the higher modes are effectively eliminated due to filtering, and their coefficients are therefore removed from consideration. For the remaining coefficients, we construct a single sequence

$$\tilde{c}_i = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(c_i^k)^2} \qquad i = 1, 2, \ldots, \tilde{n} \tag{13}$$

of the root mean square of the ith coefficient, where $\tilde{n}$ is the highest nonfiltered mode, and k enumerates all N lines of grid points, so $N = d \cdot n^{d-1}$ for a d-dimensional grid of n^d points.

We then fit a simple model of exponential decay to the $\tilde{c}_i$, and we evaluate the resulting function at the highest nonfiltered order to obtain the final indicator value,

$$\varepsilon = 10^{a\tilde{n}+b}, \tag{14}$$

where a and b are the slope and offset as obtained by a linear least-squares fit on the logarithm of the $\tilde{c}_i$. This ε is returned as the indicator value.

Because the exponential decay of coefficients only sets in at sufficiently high order, and accumulated round-off errors due to finite machine precision prevent the accurate computation of coefficients for very high orders, resulting in a "round-off plateau," this method overall tends to underestimate the slope of the decay, thus overestimating the total truncation error.

This type of indicator is naturally suited for driving p refinement, since it directly corresponds to the success or failure of a series of particular polynomial order to represent the data.

Smoothness estimate.—A well-tested heuristic for determining the need for mesh refinement in an area is an estimate of the form

$$\epsilon = \sqrt{\frac{1}{N}\sum_{i}^{N}\frac{\sum_{k,l}\left(\frac{\partial^2 u}{\partial x_k \partial x_l}\Big|_{x_i}\right)^2}{\sum_{k,l}\left(\frac{|\frac{\partial u}{\partial x_k}|_L + |\frac{\partial u}{\partial x_k}|_R}{\Delta_l} + \varepsilon\left|\frac{\partial^2}{\partial x_k, \partial x_l}||u||_{x_i}\right|\right)^2}}, \tag{15}$$

similar to the indicator originally described in Ref. [44], and adapted to spectral grids.

Here, $|\frac{\partial u}{\partial x_k}|_L$ and $|\frac{\partial u}{\partial x_k}|_R$ refer to the first derivatives of the solution at the left and right boundaries of the spectral element, respectively, and Δ_l is the size of the element along the lth dimension. The term following ε is computed by taking the absolute value of the derivative matrix $D_{kl} = D_k \cdot D_l$ elementwise, and applying it to the piece of the state vector containing the variable u, also taking the absolute value elementwise. Effectively, we compute a normalized version of the second derivatives on the grid, where the normalization is based on an upper bound on first derivatives. If the first derivatives are small, then the term proportional to ε provides an alternative normalization. It acts as a filter to prevent small high-frequency "ripples" from triggering unwanted, and potentially cascading, refinement.

This type of indicator originates in finite element methods using linear elements, where the magnitude of second derivatives is justified as an error estimate. Because a spectral element of order $n > 2$ can still represent second-order polynomials exactly, it is less obvious why the indicator would give meaningful results. In practice, however, using it as a heuristic leads to refinement in exactly those regions that are "nonsmooth," as well as the regions where the solution shows the strongest features.

It is a natural choice as the indicator used by the h-refinement portion of the algorithm, as it pushes the algorithm to subdivide grids until each represents an approximately linear piece of the solution.

Static indicators.—Instead of using the data on a grid to determine its refinement status, it is also possible to construct indicators that result in a static, yet heterogeneous grid structure—for example, a domain with high resolution near one or more defined centers, and progressively lower resolution further away from them. Such a scheme can be described in terms of a target level l which depends on the distance d from the closest center—for example,

$$l = \left\lfloor \log_2\left(\frac{a}{d}\right)\right\rfloor, \tag{16}$$

where a determines the size of the refined region. Schemes such as this, which directly result in a target level, can easily be made to fit the above paradigm of returning a real number to be compared to a set interval by returning, for example, $+1$ if the current level is below the target level, -1 if the current level is above it, and 0 if the current level matches the target level. Setting the accepted interval to $[-0.5, 0.5]$ then results in the desired refinement operations being applied (see Fig. 5 for an example of the resulting grid structure). Applying this type of refinement indicator in combination with, for example, a center-of-mass detection will result in an AMR scheme that guarantees that regions of physical interest, such as orbiting compact objects, are always covered by highly resolved mesh regions that dynamically follow them.

4. Load balancing

In order to ensure an even distribution of computational work across the available CPU cores, grids are shifted between processes during AMR operations. Since each boundary shared by two or more processes necessitates

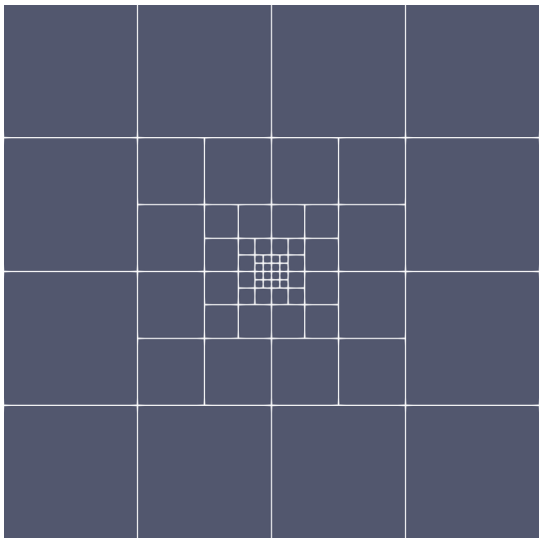

FIG. 5. A grid structure resulting from the use of a static refinement indicator such as Eq. (16).

data exchange between these processes, we also seek to minimize the amount of such boundaries, or maximize data locality.

This is achieved by arranging all grids into a list in the order of their intersection with a space-filling z-order curve, also known as a Morton curve [45], and then partitioning this list into as many sections as processes are used (see Fig. 6). Similar partitioning schemes are used in other codes; for example, Ref. [18] also uses a Morton curve for domain partitioning. See also Ref. [46] for a comprehensive treatment of space-filling curves and their numerical applications. We specifically use a z-order curve over the more common Hilbert curve in order to simplify the application of grid index offsets for neighbor tracking

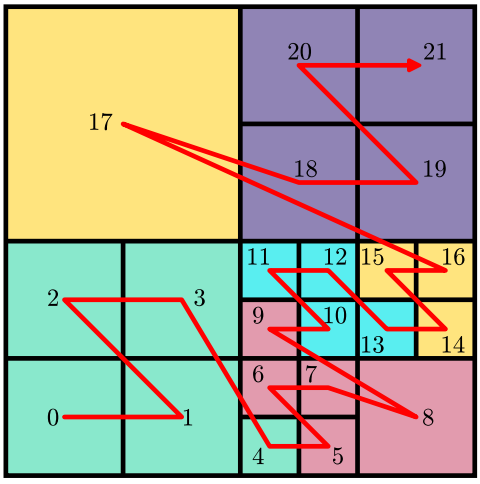

FIG. 6. An example of a grid configuration with the z-order curve determining the internal ordering of grids. Different colors show a possible division of 22 grids among 5 processes, demonstrating approximate data locality.

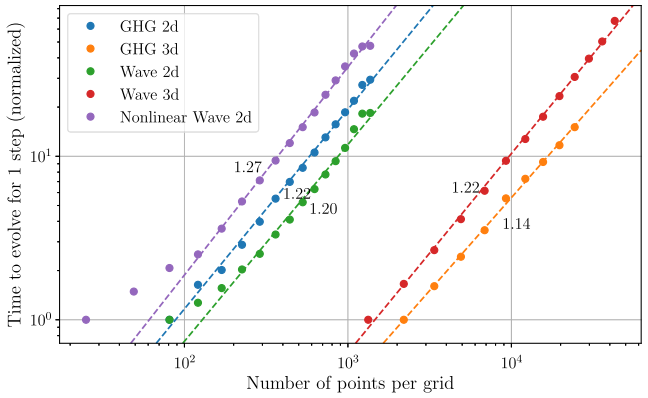

FIG. 7. Scaling of the workload associated with evolving a grid for different numbers of points per grid.

during refinement operations. We find that this simple mechanism fulfills both of the goals set out for load balancing.

As mesh refinement operations modify the grid structure (see Sec. III B), the ordering of grids in this list is maintained such that it always corresponds to a traversal of the domain along the z-order curve.

In numerical experiments, we empirically find that the runtime necessary to evolve a grid scales as

$$t = n^w, \tag{17}$$

where n is the total number of grid points, with powers w between 1.14 and 1.27, depending on the equations being evolved, as well as the number of dimensions (see Fig. 7).

Consequently, each grid is assigned a weight, according to

$$\rho_i = (n_i)^w, \tag{18}$$

where n_i is the total number of grid points on the ith grid, and w is chosen appropriately for the equations being solved. The partitioning of the grid list is then done such that each segment contains grids with approximately the same total weight. It should be noted that this represents a simplified model which combines both the amount of pointwise arithmetic, which depends on the particular system being evolved, and independent per-grid overhead (for example, computation of derivatives), depending on the particular equations being solved, into a single empirical parameter, w.

5. Boundary data exchange

The pseudospectral method used by BAMPS requires the exchange of data on the internal boundaries between grids. The optimal way to accomplish this can and will depend on the particular MPI implementation used. We use the following algorithm, applied in synchronous phases across all MPI processes:

(1) Determine how many boundaries the MPI process shares with each other MPI process.

(2) For each MPI process with shared boundaries, generate a list of identifiers t, from which the details of the necessary MPI message can be uniquely determined:

$$t = 24 \cdot n_g + 4 \cdot n_{\text{dir}} + n_{ne}, \tag{19}$$

where n_g is the index of the sending grid, n_{dir} is a number between 0 and 5 which encodes a direction, and n_{ne} is a number between 0 and 3 which specifies which of up to four neighboring grids information is sent to. This creates a pair of identifiers for each shared boundary: one for sending the local data, and one for receiving data from a remote MPI process. These identifier pairs are stored together as a single entry in the list of messages, which can be sorted by either one.

(3) Sort all lists of communications with MPI processes of higher rank by the send identifier, and sort the other lists by the receive identifier. This ensures that MPI processes sharing several boundaries will have identical lists of communication identifiers, in the same order.

(4) Sort the list of lists by length, to ensure that the shortest ones are handled first. We find that this significantly reduces the time other processes spend waiting on communications to be initiated, in some cases by up to 40%.

(5) For each list of communications, initiate both sending and receiving operations as asynchronous MPI calls, using the list index as the MPI message tag.

(6) While MPI communications are ongoing, perform all boundary data exchanges that are entirely local to each MPI process.

(7) Wait for all MPI communications to finish.

Once all boundary information has been transferred to the neighboring grids, the data are interpolated to match the resolution of the receiving grid. Note that performing the interpolation at the receiver is an arbitrary design decision, and it could equivalently be performed at the sender before any communication occurs. As this process results in data on coincident grid points, we are able to use the same penalty method as was used in Ref. [30] for equally sized grids.

For n_p different grid resolutions accessible by p refinement, there are n_p^2 possible combinations between equally sized grids, each requiring a different interpolation matrix. Since grids may also share boundaries with other grids either half or twice their size, which may overlap in either the upper or lower halves of their respective extents, this number is further multiplied to give a total of $5n_p^2$ possible cases. In practice, only a small subset of these cases will be reached during any given simulation. Therefore, we do not compute every possible interpolation matrix in advance, and instead generate them on demand. We then utilize memoization (caching) to minimize the duplication of work—that is, each MPI process keeps a cache of previously required interpolation matrices, to be reused if the same case appears again later. Each interpolation matrix that is required is thus only generated once on each process where it is needed, and unneeded matrices are never generated.

C. Performance and scaling

To evaluate the performance and scaling behavior of BAMPS, a series of benchmark runs was performed on a varying number of CPU cores. For these runs, static grid configurations were chosen in order to control the amount of work per CPU core. Each configuration was run three times, and the measured times were averaged. The times themselves were measured using built-in timers, capable of profiling specific sections of the code, including the runtime of the entire `main` routine. With this, both strong and weak scaling tests were performed. We use an axisymmetric subcritical Brill wave collapse simulation as our test case; see Sec. VI for details on the evolution system.

Strong scaling refers to the performance increase, as measured by the lower runtime (speedup), when distributing the same amount of work over more CPU cores. Ideal strong scaling would be achieved if a doubling of the number of CPU cores resulted in halving the necessary time for the same simulation. In practice, most programs have a nonparallelizable part, which leads to their speedup following Amdahl's law, with diminishing returns for increasing numbers of CPU cores.

Weak scaling refers to the ability to solve larger problems efficiently when provided with more CPU cores. It is measured by increasing both the problem size and the number of CPU cores by the same factor, and observing the change in computation time. Ideal weak scaling would be achieved if the required time remained the same under this change. Weak scaling is often measured by the weak scaling efficiency, determined by

$$e = \frac{t_0}{t} \cdot \frac{n}{n_0}, \tag{20}$$

where t is the total computation time summed over all CPU cores, and n is the number of utilized CPU cores. t_0 and n_0 are those quantities for a selected (small) reference run.

In Fig. 8, both the strong and weak scaling of BAMPS are shown. The solid lines represent series of runs showing strong scaling, as the computation time required decreases at the same rate at which the number of CPU cores is increased. The dotted lines show a series of runs demonstrating weak scaling, as the amount of work per CPU core remains constant along them, and the amount of computation time required also remains almost constant.

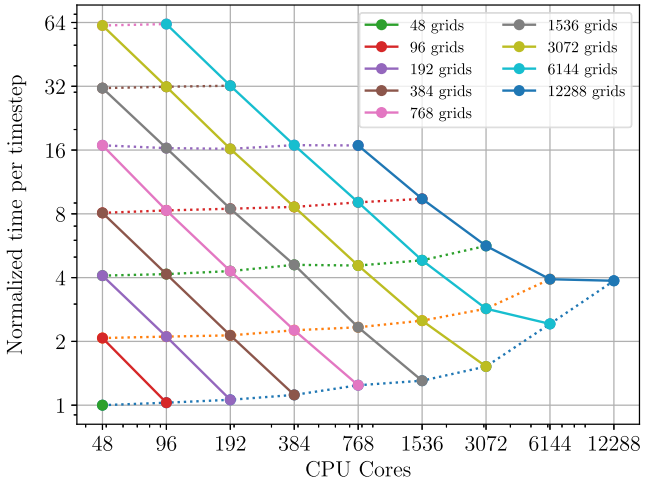

FIG. 8. Strong and weak scaling of BAMPS for different static grid configurations, based on benchmarks performed on SuperMUC-NG, evolving the GHG system in 2D. Solid lines represent a constant total number of grids, and dotted lines represent a constant number of grids per CPU core.

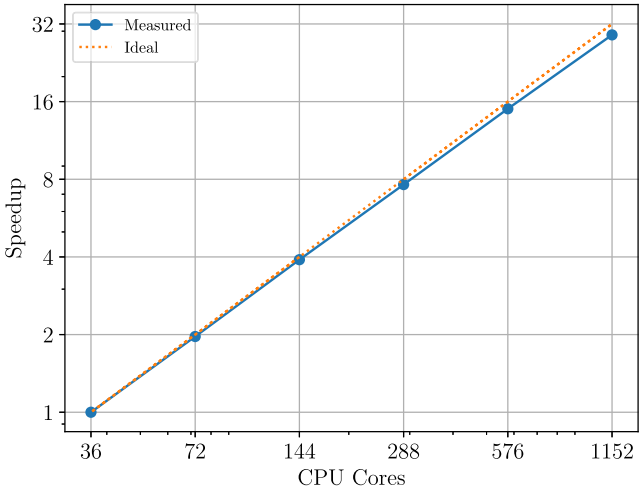

FIG. 9. Strong scaling of BAMPS for a static grid configuration of 9216 grids.

The strong scaling behavior of BAMPS is also shown separately in Fig. 9, and Fig. 10 shows the weak scaling efficiency. Provided the workload of a single CPU core is chosen to be sufficiently large (eight grids per CPU core for the GHG simulations studied here), we consistently observe a weak scaling efficiency above 90%.

With AMR enabled, the scaling behavior is expected to be slightly worse, since the refinement algorithm involves global communication between all MPI processes during the load-balancing procedure. However, because the AMR algorithm is only invoked every 100 time steps (or even more rarely), we do not expect it to have a noticeable impact on the overall scaling behavior.

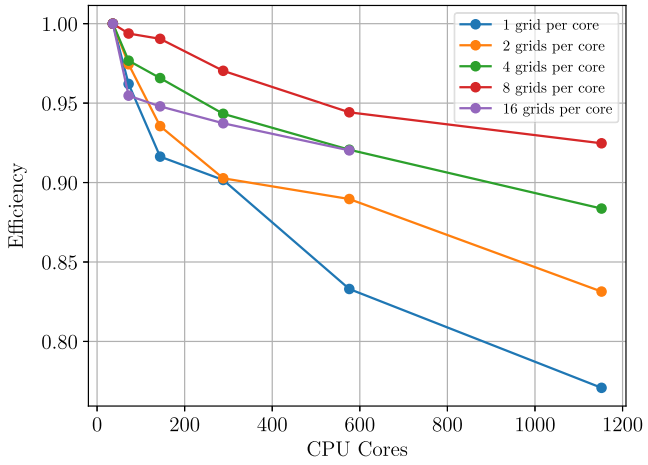

FIG. 10. Weak scaling efficiency of BAMPS for a static grid configuration.

Given the complexity of hp refinement even for fixed refinements, in this particular example, the scaling of BAMPS is excellent up to 1000 CPU cores and satisfactory up to 6000 CPU cores. These numbers change when considering runs with more grids, and in particular when moving from 2D to 3D spatial grids.

Overall, the strategy to consider MPI parallelization with single grids defining data granularity is successful for a wide range of configurations, down to even a few 2D grids per process. In part, this is a feature of the complexity of the Einstein equations, since the amount of data and work per grid tends to be large compared to the parallelization overhead. In general, this is also expected for spectral element methods, if only face-local data is exchanged at element interfaces. More fine-grained algorithms are available—for example, task-based parallelism as employed in Refs. [16,18]—but for the present applications of BAMPS, the single-grid granularity performs well.

IV. NONLINEAR WAVE EQUATION MODEL

To evaluate the capability of the AMR system in resolving strongly varying data during an evolution, we consider the nonlinear wave equation

$$\Box\psi + A_1 \nabla_a \nabla^a \psi = 0, \tag{21}$$

which corresponds to "model 1" in Ref. [47], choosing $A_1 = 1$. We apply a first-order reduction using the reduction variables $\Pi = \partial_t \psi$ and $\phi_i = \partial_i \psi$, resulting in the system

$$\partial_t \psi = -\Pi, \tag{22}$$

$$\partial_t \phi_i = -\partial_i \Pi + \gamma_2 \partial_i \psi - \gamma_2 \phi_i, \tag{23}$$

$$\partial_t \Pi = -\partial_i \phi^i - A_1(\phi_i \phi^i - \Pi^2). \tag{24}$$

Solutions to this equation can be built analytically, by first constructing a solution to the linear equation $\Box\psi = 0$ from partial waves as

$$\varphi = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \varphi_{lm}(t, r) Y_l^m(\theta^A), \tag{25}$$

where (r, θ^A) are the usual spherical coordinates, and $Y_{lm}(\theta^A)$ are the spherical harmonics. Applying the deformation function

$$D(\varphi) = A_1^{-1} \log(1 + A_1 \varphi) \tag{26}$$

then yields the solution $\psi = D(\varphi)$.

For this comparison, we evolve data built from a pure $(l = 2, m = 0)$ wave, such that

$$\varphi(t, r, \vartheta) = \frac{1}{4}\sqrt{\frac{5}{\pi}}\left(\frac{3}{r^3}[F(t_-) - F(t_+)] + \frac{3}{r^2}[F'(t_-) + F'(t_+)] + \frac{1}{r}[F''(t_-) - F''(t_+)]\right)(3\cos^2\vartheta - 1), \tag{27}$$

where $t_- = t - r$ and $t_+ = t + r$ are the retarded and advanced time, respectively (see Ref. [48] for details on this construction).

For the seed function $F(t)$, we choose an offset Gaussian

$$F(t) = A e^{-(t+1)^2}. \tag{28}$$

For sufficiently high amplitudes, this solution is known to "blow up" after finite time [47], while for smaller amplitudes, the solution continues to exist. We refer to these cases as supercritical and subcritical, respectively. In order to obtain strong features in the solution, we choose an amplitude of $A = 1.6784366869120966$, which we find to be barely subcritical when evolved with the highest resolution used here.

For a systematic convergence test, we employ the concept of a refinement schedule. A run is performed for a specific choice of hp-refinement parameters (typically for the lowest feasible resolution), and the time-dependent sequence of h refinements is recorded. Subsequent runs can use the same "h-refinement schedule" while varying the p refinement. Using this technique, we find the expected exponential convergence of the numerical solution as grid points are added, which corresponds to adding terms to the spectral series (see Fig. 11).

To determine the impact of AMR on the accuracy of the time evolution, we evolve this system using

(i) AMR, as well as static grid configurations corresponding to,
(ii) the lowest resolution accessible to AMR,
(iii) the highest resolution accessible to AMR, and
(iv) the average resource use of the AMR configuration.

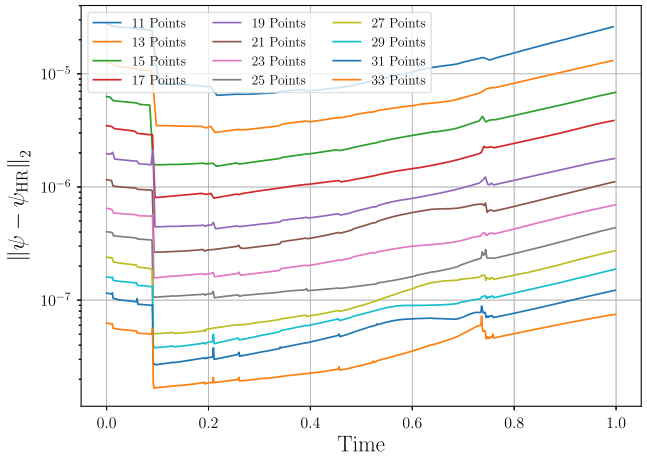

FIG. 11. Nonlinear wave model, convergence of the numerical solution at different per-grid resolutions, as compared to a reference simulation with 35×35 points per grid. For the sake of comparison, the same h-refinement schedule was applied to all runs.

To determine an appropriate average resolution for configuration (iv), the grid structure generated by AMR was analyzed *post hoc*, and the total number of points used at any given time step was averaged over the whole evolution. A static, approximately uniform grid configuration using a similar amount of points was then found by a brute force search over all possible grid configurations.

We find that the use of AMR results in a lowering of the total numerical error by up to 3 orders of magnitude, compared to using a similar number of grid points spread uniformly over the domain. Perhaps surprisingly, the evolution using configuration (iii) shows by far the largest numerical error after the solution forms strong features, several orders of magnitude higher than even configuration (ii). This is likely due to large amounts of round-off error piling up due to both a very large number of time steps necessary to satisfy the Courant-Friedrichs-Lewy condition [49], and gridwise operations such as derivative computation requiring multiplication with very large matrices. This effect is amplified in regions with large absolute values of the solution, decreasing the absolute numerical precision. We also observe the formation of nonphysical features caused by large amounts of noise at the boundaries between grids. This suggests that spacetime configurations with strong, but highly localized features, as often encountered close to criticality, are only accessible using AMR, as neither low nor high static resolutions are capable of resolving them accurately.

Apart from the final numerical error resulting from an evolution, we also consider the amount of computational work necessary to evolve a given grid configuration. Here, we use the workload [Eq. (17)] to compute the total work necessary to evolve data up to a given time step as

TABLE II. Runtime scaling behavior based on the number of grid points.

Type	w
Wave eq. (2D)	1.20
Wave eq. (3D)	1.22
Nonlinear wave eq. (2D)	1.27
GHG (2D)	1.22
GHG (3D)	1.14

$$W(k) = \sum_{j=1}^{k} \sum_{i=1}^{N_j} (n_i)^w, \tag{29}$$

where i enumerates all N_j grids present at a given step i, n_i is the number of points on each particular grid, and w is the appropriate weighting power as shown in Table II.

The product of numerical error and the work required to reach it can then be used as a measure of numerical efficiency (where a lower value corresponds to a higher efficiency). This measure is shown in Fig. 12. The additional overhead of the AMR mechanism initially makes the evolution less efficient than a comparable but homogeneous resolution, but once strong features form in the solution, the additional accuracy gained via AMR leads to the full evolution being not only more accurate by several orders of magnitude, but also more efficient in terms of work expended to obtain this accuracy.

V. REAL SCALAR FIELD

After testing AMR with a nontrivial toy model, we test how it performs in physical scenarios of interest. First, we consider a real massless scalar field minimally coupled to the Einstein field equations, in spherical symmetry.

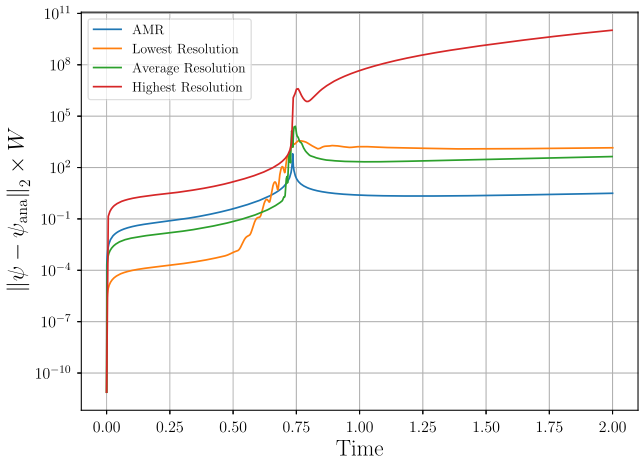

FIG. 12. Efficiency for the nonlinear wave model. Shown is the numerical error during the evolution of a nonlinear wave equation for several static resolutions, as well as using AMR, adjusted for the amount of work W necessary (lower values are better).

Denoting g_{ab} as the 4D metric and R_{ab} as the Ricci tensor, the Einstein equations read as

$$R_{ab} = 8\pi\left(T_{ab} - \frac{1}{2} g_{ab} T\right), \tag{30}$$

where $T = g^{ab} T_{ab}$, and

$$T_{ab} = \nabla_a \varphi \nabla_b \varphi - \frac{1}{2} g_{ab} (\nabla^c \varphi \nabla_c \varphi + m^2 \varphi^2) \tag{31}$$

is the energy-momentum tensor corresponding to a scalar field φ.

We evolve initial data in time according to the $3+1$ decomposition

$$ds^2 = -\alpha^2 dt^2 + \gamma_{ij} (\beta^i dt + dx^i)(\beta^j dt + dx^j), \tag{32}$$

where γ_{ij} is the 3D spatial metric, α is the lapse, and β^i is the shift. The normal unit vector is then $n^a = \alpha^{-1}(1, -\beta^i)$. We write 4D component indices with latin letters starting from a and 3D spatial indices with letters starting from i.

The evolution equations for the matter part of the Einstein field equations follow the first-order Einstein-Klein-Gordon system, which for $m = 0$ reads as

$$\partial_t \varphi = \alpha \pi + \beta^i \chi_i, \tag{33}$$

$$\partial_t \pi = \beta^i \partial_i \pi + \gamma^{ij} (\chi_j \partial_i \alpha + \alpha \partial_i \chi_j - \alpha \Gamma^k{}_{ij} \chi_k) + \alpha \pi K + \sigma \beta^i S_i, \tag{34}$$

$$\partial_t \chi_i = \pi \partial_i \alpha + \alpha \partial_i \pi + \chi_j \partial_i \beta^j + \beta^j \partial_j \chi_i + \sigma \alpha S_i, \tag{35}$$

where π is the time reduction variable $+n^a \partial_a \varphi$, χ_i is the spatial reduction variable associated with the reduction constraint $S_i := \partial_i \varphi - \chi_i$, and σ is a damping term. Similarly, the metric is evolved following the generalized harmonic gauge formalism of the Einstein field equations,

$$\partial_t g_{ab} = \beta^i \partial_i g_{ab} - \alpha \Pi_{ab} + \gamma_1 \beta^i C_{iab}, \tag{36}$$

$$\begin{aligned} \partial_t \Pi_{ab} = {} & \beta^i \partial_i \Pi_{ab} - \alpha \gamma^{ij} \partial_i \Phi_{jab} + \gamma_1 \gamma_2 \beta^i C_{iab} \\ & + 2\alpha g^{cd} (\gamma^{ij} \Phi_{ica} \Phi_{jdb} - \Pi_{ca} \Pi_{db} - g^{ef} \Gamma_{ace} \Gamma_{bdf}) \\ & - 2\alpha \left(\nabla_{(a} H_{b)} + \gamma_4 \Gamma^c{}_{ab} C_c - \frac{1}{2} \gamma_5 g_{ab} \Gamma^c C_c \right) \\ & - \frac{1}{2} \alpha n^c n^d \Pi_{cd} \Pi_{ab} - \alpha n^c \gamma^{ij} \Pi_{ci} \Phi_{jab} \\ & + \alpha \gamma_0 (2\delta^c{}_{(a} n_{b)} - g_{ab} n^c) C_c \\ & - 16\pi\alpha \left(T_{ab} - \frac{1}{2} g_{ab} T^c{}_c \right), \end{aligned} \tag{37}$$

$$\partial_t \Phi_{iab} = \beta^j \partial_j \Phi_{iab} - \alpha \partial_i \Pi_{ab} + \gamma_2 \alpha C_{iab} + \frac{1}{2} \alpha n^c n^d \Phi_{icd} \Pi_{ab} + \alpha \gamma^{jk} n^c \Phi_{ijc} \Phi_{kab}, \quad (38)$$

where the evolved variables are the metric g_{ab}, the time reduction variable Π_{ab} corresponding to $+n^d \partial_d g_{ab}$, and the spatial reduction variable Φ_{iab} associated with the reduction constraint $C_{iab} = \partial_i g_{ab} - \Phi_{iab}$. The constraint damping parameters are $\gamma_1 = -1$, $\gamma_0 = \gamma_2 = 2$, and $\gamma_4 = \gamma_5 = 0.5$. $C_a = H_a + \Gamma_a$ is the harmonic constraint, where H_a is a gauge source function.

The type of scalar field initial data we use for these simulations has a Gaussian profile of the form

$$\varphi = A(e^{-(r-R_0)^2} + e^{-(r+R_0)^2}), \quad (39)$$

with $R_0 = 3$ and with a vanishing gradient along the normal vector n^a,

$$n^a \nabla_a \varphi = 0. \quad (40)$$

A conformal decomposition of the metric allows us to solve the ADM constraints via the extended conformal thin sandwich (XCTS) equations [50,51]. We consider a flat conformal spatial metric $\bar{\gamma}_{ij} = \delta_{ij}$, with a vanishing time derivative $\partial_t \bar{\gamma}_{ij} = 0$, as well as maximal slicing $K = 0$, $\partial_t K = 0$. With these choices, the XCTS equations constitute a set of coupled elliptic PDEs for the conformal factor ψ and the gauge variables β^i and α. The latter are simply solved by $\beta^i = 0$ and $\alpha = 1$, thanks to the choice in Eq. (40). The remaining XCTS equation to solve for ψ is

$$0 = \delta^{ij} \partial_i \partial_j \psi + \pi \psi \delta^{ij} \partial_i \varphi \partial_j \varphi. \quad (41)$$

We use Robin outer boundary conditions compatible with a $1/r$ decay. The XCTS equation (41) is then solved by means of the hyperbolic relaxation method [52] provided by BAMPS.

Similarly to Sec. IV, we evolve this initial data using configurations (i)–(iv), where again configuration (iv) is determined by analyzing the results of (i), in order to find a setup using a comparable total amount of work to evolve. As no analytic solution is known, we use the integral of the constraint monitor C_{mon}, which aggregates both physical and reduction constraint violations, as a proxy for numerical error.

We find that the constraint violation during early times is larger by around 3 orders of magnitude when using AMR, compared to the static "average" resolution. However, once strong features develop in the solution, constraint violations on the low resolution and average resolution static meshes increase significantly. The constraint violations of the adaptive grid, while increasing as well, stay between 5 and 9 orders of magnitude below those on the static meshes. Figure 13 shows the total constraint violation over time,

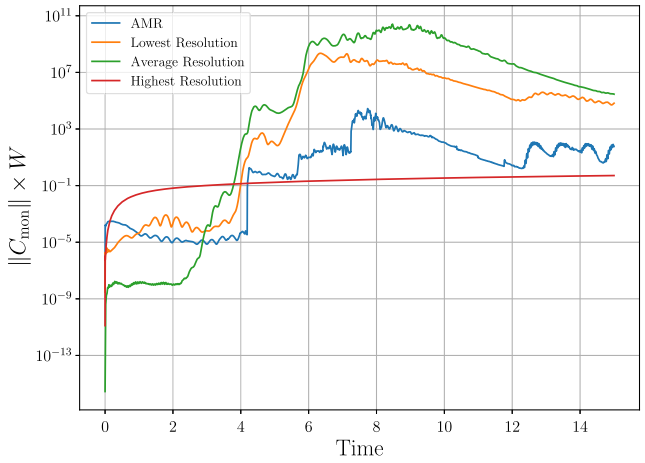

FIG. 13. Integral of the internal constraint violation monitor, multiplied by the cumulative workload W, during the time evolution of a real scalar field using AMR, as compared to the base resolution used by the AMR, a static resolution with comparable total workload, and the highest resolution accessible to the AMR system.

adjusted for cumulative necessary work. Notably, the highest-resolution static mesh shows the lowest constraint violations after the formation of strong features, staying at a constant level determined by finite floating-point precision. This suggests that in 1D simulations, using a very high resolution is still feasible, due to the much lower amount of arithmetic involved in operations on one-dimensional grids, avoiding the accumulation of large round-off errors that we observe in the two-dimensional examples. After adjusting the constraint violation for the computational workload, the high-resolution run retains the best efficiency, as the decreased cost of running with AMR (in this example, the AMR configuration required less than 0.6% of the work needed for the high-resolution run) is not sufficient to make up for the decrease in overall accuracy.

For this example as well, we observe exponential convergence as the polynomial order of each element is increased (see Fig. 14), up to the point that saturation is reached and numerical round-off errors dominate.

VI. BRILL WAVES

Finally, we consider the case of vacuum gravitational collapse in axisymmetry. We choose initial data analogous to those used in Ref. [21], specifically picking an off-center prolate Brill wave which is known to be subcritical. We evolve this data using the GHG evolution system [Eq. (36)], this time with $\alpha\gamma_0 = \gamma_2 = 1$ and using the cartoon method [53] to suppress the angular dimension corresponding to the symmetry.

To gauge the efficiency of the AMR, we choose a representative simulation of an off-center Brill wave, with initial data parameters of $\rho_0 = 5$ and $A = 0.06410$. This configuration is known to disperse in finite time [21].

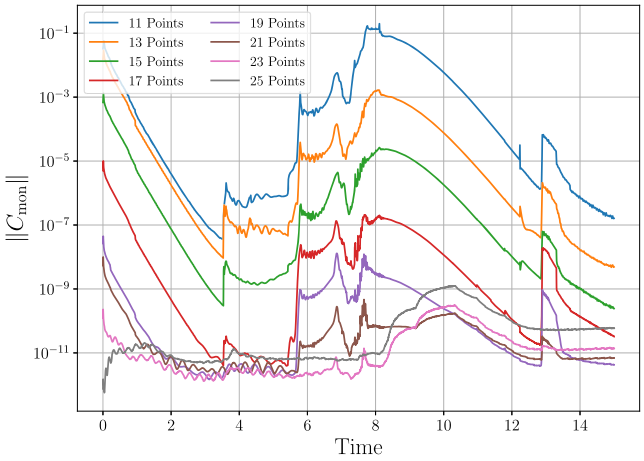

FIG. 14. Integral of the internal constraint violation monitor during the time evolution of a real scalar field using a fixed h-refinement schedule, with differing static per-grid resolutions.

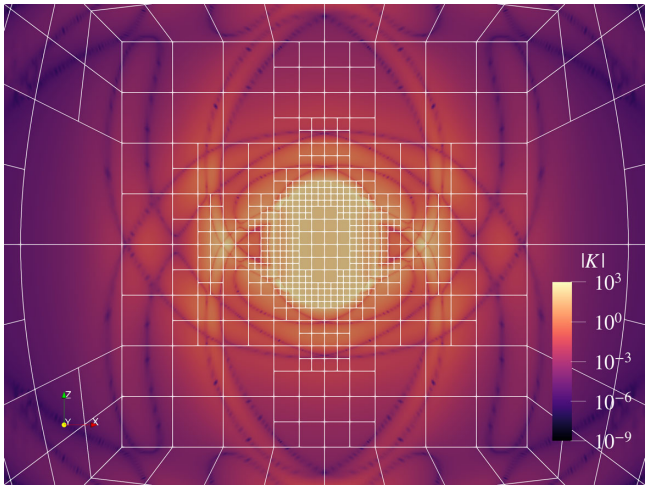

FIG. 15. Kretschmann scalar K and the grid structure generated by AMR during the time evolution of a Brill wave, based on the "smoothness" heuristic.

Figure 15 shows a snapshot of the grid structure as well as the Kretschmann scalar during the time evolution of these data with AMR enabled. We perform the same comparison as in Secs. IV and V, first evolving this initial data with AMR [configuration (i)], and then determining a configuration of static grids that takes a comparable total workload to evolve [configuration (iv)]. To evolve configuration (ii), the per-grid resolution had to be raised from 21×21 to 23×23 points, otherwise the evolution would become unstable after only a short time. For this comparison, it was not feasible to also evolve configuration (iii), as this configuration also quickly developed instabilities. This may be caused by excessive round-off error accumulation, similar to the results shown in Fig. 12.

We again use the constraint monitor variable C_{mon} as a proxy for numerical error. In this example, too, the adaptive

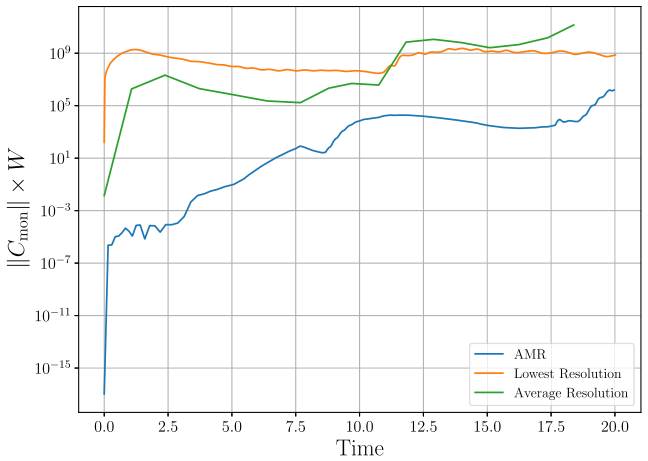

FIG. 16. Integral of the internal constraint violation monitor, multiplied by the cumulative workload W, during the time evolution of a Brill wave using AMR, as compared to the base resolution used by the AMR and a static resolution with comparable total workload.

mesh vastly outperforms the static configuration, showing more than 6 orders of magnitude smaller constraint violations. The total constraint violation adjusted by necessary work is shown in Fig. 16, showing the higher efficiency of AMR even as higher overall accuracy is achieved.

To verify the convergence of the method, we evolve identical initial data on several different per-grid resolutions. To make the results comparable, the h refinement is in each case driven by a predetermined refinement schedule, generated by a reference run using 21×21 points per grid. We again use the integral of the constraint monitor C_{mon} as a proxy for numerical error. Figure 17 shows the

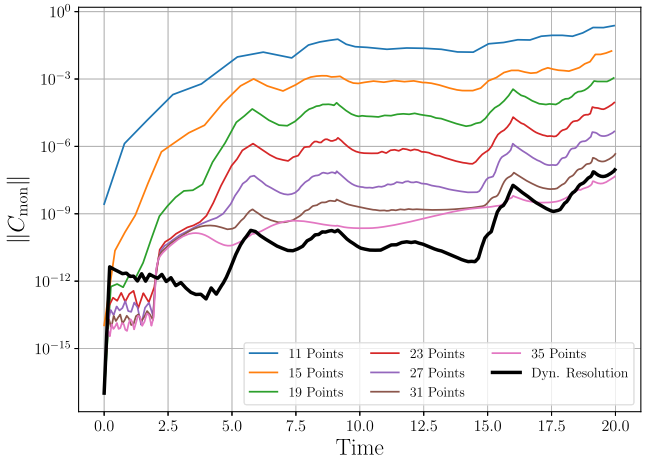

FIG. 17. Integral of the internal constraint violation monitor during the time evolution of a Brill wave using a fixed h-refinement schedule, with differing static per-grid resolutions, as well as a dynamic resolution determined by the p-refinement scheme.

integrated constraint violations for a variety of resolutions, showing exponential convergence of the solution.

Also in Fig. 17, we show the constraint violations for a run with dynamic, local p refinement. For this, we use the same fixed h-refinement schedule as for the static resolutions, and additionally enable p refinement, using the truncation error estimate, evaluated on all metric components, as well as C_{mon} itself. This results in overall constraint violations that are at times lower than the highest static resolution of 35×35 points per grid, despite this also being the highest resolution available to the AMR system. By the end of the evolution, the constraint violations are comparable between high-resolution static and dynamic resolutions. However, evolving the system with p refinement enabled requires only roughly 22% of the computational work required by evolving with static 35×35 points per grid, demonstrating the much higher efficiency obtained by dynamic p refinement.

VII. CONCLUSIONS

We have successfully implemented fully adaptive hp refinement into the BAMPS code and demonstrated the spectral convergence of simulations on the grid structures generated by adaptive h refinement, as well as the gains in accuracy and efficiency obtained through the use of p refinement. Using this new system, numerical evolutions for several physical systems show improvements in accuracy by several orders of magnitude when compared to evolutions using static resolutions requiring similar amounts of work.

In the case of two-dimensional evolutions, the adaptive resolutions show increased numerical efficiency—that is, less work is needed to reach a specific accuracy, even while greater total accuracy is achieved. This does not hold true, however, for the one-dimensional example we studied. Here, while using an adaptive resolution is computationally cheaper by several orders of magnitude, these savings do not make up for the decrease in overall accuracy.

It must be noted that the proxy used to measure numerical error, the integral of overall constraint violations, is an imperfect indicator of overall accuracy. We also observe that using very high resolutions is not feasible in some cases, as the accumulated round-off errors due to large amounts of arithmetic performed in the course of matrix multiplications destroys the accuracy of the simulation.

We find that BAMPS shows near-perfect scaling up to 1000 CPU cores, and satisfactory scaling up to 6000 cores. Current production runs using BAMPS in 1D and 2D do not exceed 1000 CPU cores. While we focused here on 1D and 2D tests, the design of BAMPS is aimed at fully three-dimensional simulations. In 3D, the ratio of overhead to work is more favorable, in particular for scaling, which we have confirmed in preliminary tests.

A previous version of BAMPS only supporting h refinement has already been used successfully to further the study of the critical collapse of gravitational waves [21], and the full hp-refinement algorithm is currently being applied in simulations of the critical collapse of real scalar fields, as well as the time evolution of complex scalar fields. In particular, the evolution of boson stars represents an ideal use case of the methods presented here. These sample applications focus on smooth fields. AMR is also a de facto necessary feature to effectively study problems involving general relativistic hydrodynamics (GRHD). However, more work is necessary to fully manage emerging shocks and other discontinuities in conjunction with AMR (see Ref. [54] for an exploration of GRHD in BAMPS using nonadaptive meshes). A technique that remains to be implemented is that of local time stepping to obtain hpt refinement, where elements of different resolutions are advanced in time at different rates, which offers great potential for increased efficiency. Spectral element methods are well suited to local time stepping due to the discontinuous coupling of elements. Local time-stepping schemes generally require significant changes to the underlying time evolution infrastructure, which are currently underway.

The technical upgrade of BAMPS will benefit most, if not all, future projects using the code. Furthermore, the methods developed for this purpose and the insights gained here can serve as a case study with a wide range of applicability.

ACKNOWLEDGMENTS

We are grateful to F. Atteneder, H. R. Rüter, and I. Suárez Fernández for helpful discussions and for collaboration on other aspects of BAMPS. This work was partially supported by the FCT (Portugal) IF Program No. IF/00577/2015, Projects No. UIDB/00099/2020 and No. PTDC/MAT-APL/30043/2017, and in part by the Deutsche Forschungsgemeinschaft (DFG) under Grant No. 406116891 within RTG 2522/1 and DFG Grant BR No. 2176/7-1.

[1] M. J. Berger and J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comput. Phys. **53**, 484 (1984).

[2] PAMR & RNPL Website, http://laplace.physics.ubc.ca/Group/Software.html.

[3] M. W. Choptuik, Universality and Scaling in Gravitational Collapse of Massless Scalar Field, Phys. Rev. Lett. **70**, 9 (1993).

[4] B. Brügmann, Adaptive mesh and geodesically sliced Schwarzschild spacetime in $3+1$ dimensions, Phys. Rev. D **54**, 7361 (1996).

[5] B. Brügmann, Binary black hole mergers in 3D numerical relativity, Int. J. Mod. Phys. D **08**, 85 (1999).

[6] K. Clough, P. Figueras, H. Finkel, M. Kunesch, E. A. Lim, and S. Tunyasuvunakool, GRChombo: Numerical relativity with adaptive mesh refinement, Classical Quantum Gravity **32**, 245011 (2015).

[7] T. Andrade *et al.*, GRChombo: An adaptable numerical relativity code for fundamental physics, J. Open Source Softwaare **6**, 3703 (2021).

[8] S. L. Liebling, The singularity threshold of the nonlinear sigma model using 3D adaptive mesh refinement, Phys. Rev. D **66**, 041703(R) (2002).

[9] B. Brügmann, J. A. González, M. Hannam, S. Husa, U. Sperhake, and W. Tichy, Calibration of moving puncture simulations, Phys. Rev. D **77**, 024027 (2008).

[10] Z.-j. Cao, H.-J. Yo, and J.-P. Yu, A reinvestigation of moving punctured black holes with a new code, Phys. Rev. D **78**, 124011 (2008).

[11] Cactus Website, Cactus Computational Toolkit, http://www.cactuscode.org.

[12] F. Löffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, R. Haas, I. Hinder, B. C. Mundim, C. D. Ott, E. Schnetter, G. A. Allen, M. Campanelli, and P. Laguna, The Einstein Toolkit: A community computational infrastructure for relativistic astrophysics, Classical Quantum Gravity **29**, 115001 (2012).

[13] S. Shankar, P. Mösta, S. R. Brandt, R. Haas, E. Schnetter, and Y. de Graaf, GRaM-X: A new GPU-accelerated dynamical spacetime GRMHD code for Exascale computing with the Einstein Toolkit, arXiv:2210.17509.

[14] M. Radia, U. Sperhake, A. Drew, K. Clough, P. Figueras, E. A. Lim, J. L. Ripley, J. C. Aurrekoetxea, T. França, and T. Helfer, Lessons for adaptive mesh refinement in numerical relativity, Classical Quantum Gravity **39**, 135006 (2022).

[15] H. O. Kreiss and G. Scherer, Finite element and finite difference methods for hyperbolic partial differential equations, in *Mathematical Aspects of Finite Elements in Partial Differential Equations*, edited by C. D. Boor (Academica Press, New York, 1974).

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

[17] M. Fernando, D. Neilsen, E. W. Hirschmann, and H. Sundar, A scalable framework for adaptive computational general relativity on heterogeneous clusters, in *Proceedings of the ACM International Conference on Supercomputing*, ICS '19 (Association for Computing Machinery, New York, NY, USA, 2019), p. 1–12.

[18] B. Daszuta, F. Zappa, W. Cook, D. Radice, S. Bernuzzi, and V. Morozova, GR-Athena++: Puncture evolutions on vertex-centered oct-tree adaptive mesh refinement, Astrophys. J. Suppl. Ser. **257**, 25 (2021).

[19] W. Tichy, L. Ji, A. Adhikari, A. Rashti, and M. Pirog, The new discontinuous Galerkin methods based numerical relativity program Nmesh, Classical Quantum Gravity **40**, 025004 (2023).

[20] B. Szilagyi, L. Lindblom, and M. A. Scheel, Simulations of binary black hole mergers using spectral methods, Phys. Rev. D **80**, 124010 (2009).

[21] I. Suárez Fernández, S. Renkhoff, D. Cors Agulló, B. Brügmann, and D. Hilditch, Evolution of Brill waves with an adaptive pseudospectral method, Phys. Rev. D **106**, 024036 (2022).

[22] J. P. Boyd, *Chebyshev and Fourier Spectral Methods (Second Edition, Revised)* (Dover Publications, New York, 2001).

[23] D. A. Kopriva, Metric identities and the discontinuous spectral element method on curvilinear meshes, J. Sci. Comput. **26**, 301 (2006).

[24] J. S. Hesthaven, S. Gottlieb, and D. Gottlieb, *Spectral Methods for Time-Dependent Problems* (Cambridge University Press, Cambridge, England, 2007).

[25] G. Karniadakis and S. Sherwin, *Spectral/hp Element Methods for Computational Fluid Dynamics* (Oxford University Press, Oxford, 2005).

[26] J. S. Hesthaven, *Numerical Methods for Conservation Laws: From Analysis to Algorithms* (SIAM, Philadelphia, 2018).

[27] R. Meinel, M. Ansorg, A. Kleinwächter, G. Neugebauer, and D. Petroff, *Relativistic Figures of Equilibrium* (Cambridge University Press, Cambridge, England, 2008).

[28] B. Brügmann, A pseudospectral matrix method for time-dependent tensor fields on a spherical shell, J. Comput. Phys. **235**, 216 (2013).

[29] C. Ronchi, R. Iacono, and P. Paolucci, The "cubed sphere": A new method for the solution of partial differential equations in spherical geometry, J. Comput. Phys. **124**, 93 (1996).

[30] D. Hilditch, A. Weyhausen, and B. Brügmann, Pseudospectral method for gravitational wave collapse, Phys. Rev. D **93**, 063006 (2016).

[31] J. S. Hesthaven, Spectral penalty methods, Appl. Numer. Math. **33**, 23 (2000).

[32] N. W. Taylor, L. E. Kidder, and S. A. Teukolsky, Spectral methods for the wave equation in second-order form, Phys. Rev. D **82**, 024037 (2010).

[33] D. A. Kopriva, A spectral multidomain method for the solution of hyperbolic systems, Appl. Numer. Math. **2**, 221 (1986), special Issue in Honor of Milt Rose's Sixtieth Birthday.

[34] D. A. Kopriva, Computation of hyperbolic equations on complicated domains with patched and overset Chebyshev grids, SIAM J. Sci. Stat. Comput. **10**, 120 (1989).

[35] D. A. Kopriva, S. L. Woodruff, and M. Y. Hussaini, Computation of electromagnetic scattering with a non-conforming discontinuous spectral element method, Int. J. Numer. Methods Eng. **53**, 105 (2002).

[36] A. M. Khokhlov, Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations, J. Comput. Phys. **143**, 519 (1998).
[37] C. Burstedde, L. C. Wilcox, and O. Ghattas, P4EST: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, SIAM J. Sci. Comput. **33**, 1103 (2011).
[38] B. Brügmann, W. Tichy, and N. Jansen, Numerical Simulation of Orbiting Black Holes, Phys. Rev. Lett. **92**, 211101 (2004).
[39] M. Thierfelder, S. Bernuzzi, and B. Brügmann, Numerical relativity simulations of binary neutron stars, Phys. Rev. D **84**, 044012 (2011).
[40] M. Griebel and G. Zumbusch, Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves, Parallel Comput. **25**, 827 (1999).
[41] G. Zumbusch, *Parallel Multilevel Methods: Adaptive Mesh Refinement and Loadbalancing*, 2nd ed. (Springer Science & Business Media, Berlin, 2012).
[42] J.-P. Berrut and L. N. Trefethen, Barycentric Lagrange interpolation, SIAM Rev. **46**, 501 (2004).
[43] B. Szilágyi, Key elements of robustness in binary black hole evolutions using spectral methods, Int. J. Mod. Phys. D **23**, 1430014 (2014).
[44] R. Löhner, An adaptive finite element scheme for transient problems in CFD, Comput. Methods Appl. Mech. Eng. **61**, 323 (1987).
[45] G. M. Morton, *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing* (International Business Machines Company, Ottawa, 1966).
[46] M. Bader, *Space-Filling Curves: An Introduction with Applications in Scientific Computing*, Vol. 9 (Springer Science & Business Media, Berlin, 2012).
[47] I. Suárez Fernández, R. Vicente, and D. Hilditch, Semilinear wave model for critical collapse, Phys. Rev. D **103**, 044016 (2021).
[48] C. Gundlach, R. Price, and J. Pullin, Late-time behaviour of stellar collapse and explosions: I. Linearized perturbations, Phys. Rev. D **49**, 883 (1994).
[49] R. Courant, K. O. Friedrichs, and H. Lewy, Über die partiellen Differenzengleichungen der mathematischen Physik, Math. Ann. **100**, 32 (1928).
[50] T. W. Baumgarte and S. L. Shapiro, *Numerical Relativity: Solving Einstein's Equations on the Computer* (Cambridge University Press, Cambridge, England, 2010).
[51] W. Tichy, The initial value problem as it relates to numerical relativity, Rep. Prog. Phys. **80**, 026901 (2017).
[52] H. R. Rüter, D. Hilditch, M. Bugner, and B. Brügmann, Hyperbolic relaxation method for elliptic equations, Phys. Rev. D **98**, 084044 (2018).
[53] M. Alcubierre, S. R. Brandt, B. Brügmann, D. Holz, E. Seidel, R. Takahashi, and J. Thornburg, Symmetry without symmetry: Numerical simulation of axisymmetric systems using Cartesian grids, Int. J. Mod. Phys. D **10**, 273 (2001).
[54] M. Bugner, T. Dietrich, S. Bernuzzi, A. Weyhausen, and B. Brügmann, Solving 3D relativistic hydrodynamical problems with WENO discontinuous Galerkin methods, Phys. Rev. D **94**, 084004 (2016).