ARTICLE IN PRESS

Computer Physics Communications xxx (xxxx) xxx

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems☆

Anne Reinarz [a,*], Dominic E. Charrier [b], Michael Bader [a], Luke Bovard [d], Michael Dumbser [c], Kenneth Duru [e,g], Francesco Fambri [f,c], Alice-Agnes Gabriel [g], Jean-Matthieu Gallard [a], Sven Köppel [d], Lukas Krenz [a], Leonhard Rannabauer [a], Luciano Rezzolla [d], Philipp Samfass [a], Maurizio Tavelli [c], Tobias Weinzierl [b]

[a] Department of Informatics, Technical University of Munich, Boltzmannstr. 3, 85748 Garching, Germany
[b] Department of Computer Science, Durham University, South Road, Durham DH1 3LE, UK
[c] Laboratory of Applied Mathematics, University of Trento, Via Messiano 77, I-38123 Trento, Italy
[d] Institute for Theoretical Physics, Goethe University, Max-von-Laue-Str. 1, 60438 Frankfurt am Main, Germany
[e] Mathematical Sciences Institute, Australian National University Canberra, Australia
[f] Max-Planck-Institute for Plasma Physics, Boltzmannstr. 2, 85748 Garching, Germany
[g] Department of Earth and Environmental Sciences, LMU Munich, Theresienstr. 41, 80333 Munich, Germany

ARTICLE INFO

Article history:
Received 20 May 2019
Received in revised form 19 December 2019
Accepted 25 February 2020
Available online xxxx

Keywords:
Hyperbolic
PDE
ADER-DG
Finite volumes
AMR
MPI
TBB
MPI+X

ABSTRACT

ExaHyPE ("An Exascale Hyperbolic PDE Engine") is a software engine for solving systems of first-order hyperbolic partial differential equations (PDEs). Hyperbolic PDEs are typically derived from the conservation laws of physics and are useful in a wide range of application areas. Applications powered by ExaHyPE can be run on a student's laptop, but are also able to exploit thousands of processor cores on state-of-the-art supercomputers. The engine is able to dynamically increase the accuracy of the simulation using adaptive mesh refinement where required. Due to the robustness and shock capturing abilities of ExaHyPE's numerical methods, users of the engine can simulate linear and non-linear hyperbolic PDEs with very high accuracy. Users can tailor the engine to their particular PDE by specifying evolved quantities, fluxes, and source terms. A complete simulation code for a new hyperbolic PDE can often be realised within a few hours — a task that, traditionally, can take weeks, months, often years for researchers starting from scratch. In this paper, we showcase ExaHyPE's workflow and capabilities through real-world scenarios from our two main application areas: seismology and astrophysics.

Program summary

Program title: ExaHyPE-Engine
Program Files doi: http://dx.doi.org/10.17632/6sz8h6hnpz.1
Licensing provisions: BSD 3-clause
Programming languages: C++, Python, Fortran
Nature of Problem: The ExaHyPE PDE engine offers robust algorithms to solve linear and non-linear hyperbolic systems of PDEs written in first order form. The systems may contain both conservative and non-conservative terms.
Solution method: ExaHyPE employs the discontinuous Galerkin (DG) method combined with explicit one-step ADER (arbitrary high-order derivative) time-stepping. An a-posteriori limiting approach is applied to the ADER-DG solution, whereby spurious solutions are discarded and recomputed with a robust, patch-based finite volume scheme. ExaHyPE uses dynamical adaptive mesh refinement to enhance the accuracy of the solution around shock waves, complex geometries, and interesting features.

1. Introduction

The study of waves has always been an important subject of research. It is not difficult to see why: Earthquakes and tsunamis have a direct and serious impact on the daily lives of millions

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: reinarz@in.tum.de (A. Reinarz).

https://doi.org/10.1016/j.cpc.2020.107251

Please cite this article as: A. Reinarz, D.E. Charrier, M. Bader et al., ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems, Computer Physics Communications (2020) 107251, https://doi.org/10.1016/j.cpc.2020.107251.

of people. A better understanding of electromagnetic waves has enabled ever faster wireless communication. The study of gravitational waves has allowed new insight into the composition and history of our Universe. Those physical phenomena, despite arising in different fields of physics and engineering, can be modelled in a similar way from a mathematical perspective: as a system of hyperbolic partial differential equations (PDE). The consortium behind the ExaHyPE project ("An Exascale Hyperbolic PDE Engine") translated this structural similarity into a software engine for modelling and simulating a wide range of hyperbolic PDE systems.

ExaHyPE is intended as an engine, i.e. it allows only a limited number of numerical schemes on a fixed mesh infrastructure, but provides high flexibility in terms of the PDE system to be solved. The consortium focuses on two challenging scenarios, long-range seismic risk assessment, see e.g. [1,2]; and the search for gravitational waves emitted by binary neutron stars, see e.g [3–6].

A user with a given application only needs to implement the PDE system and problem-specific well-posed initial and boundary conditions. To exploit dynamic mesh refinement the user only needs to implement suitable criteria for mesh refinement and admissibility of solutions. From a user's perspective, complicated PDE systems can be implemented without considering the complex issues of designing a performance-oriented high-order solver on a parallel compute cluster. Thus, the Engine enables medium-sized interdisciplinary research teams to quickly realise extreme-scale simulations of grand challenges modelled by hyperbolic conservation laws.

ExaHyPE implements a high-order discontinuous Galerkin (DG) approach. DG schemes were first introduced by Reed et al. [7] for the neutron transport equation and subsequently extended to general hyperbolic systems in a series of papers by Cockburn et al. [8–12]. Within our DG framework, higher-order accuracy in time and space is achieved using the *Arbitrary high-order DERivative* (ADER) approach first introduced for purely linear constant-coefficient equations in [13] and then extended to non-linear systems in [14]. In the original ADER approach, high-order accuracy in time is achieved by using the Cauchy–Kowalevsky procedure to replace time derivatives with spatial derivatives. This procedure is very efficient for linear problems [15], but becomes cumbersome for non-linear problems. Moreover, this procedure cannot deal with stiff source terms. To tackle these shortcomings, an alternative ADER approach was introduced by Dumbser et al. [16]. In this approach, the Cauchy–Kowalevsky procedure is replaced by an implicit solution of a cell-local space–time weak formulation of the PDE. This removes the problem dependency of the approach and allows the handling of stiff source terms. ExaHyPE provides an implementation of both ADER-DG variants.

In non-linear hyperbolic PDE systems, discontinuities and steep gradients can arise even from smooth initial conditions. High-order methods may then produce spurious oscillation which decrease the approximation quality and may even render the computed solution unphysical. To remedy this issue, a wide range of limiters for DG schemes have been proposed. Notably, these include approaches based on artificial viscosity [17,18], filtering [19] and WENO- or HWENO-based reconstruction [20,21]. The approach taken within the ExaHyPE engine is multi-dimensional optimal-order-detection (MOOD). This approach was initially applied to finite volume schemes [22–24] and has recently been extended to DG schemes [25]. In this approach, the solution is checked a-posteriori for certain admissibility or plausibility criteria and is marked troubled if it does not meet them. Troubled cells are then recalculated with a more robust finite volume scheme. The MOOD approach bypasses limitations of a-priori detection of troubled zones. We identify problematic regions after each time step and roll back to a more robust scheme on demand. This means that we do not need to reliably know all areas with issues a-priori. This approach allows for good resolution of shocks and other discontinuities [26].

The restriction to an adaptive cartesian mesh represents one of ExaHyPE's fundamental design choices. Problems are discretised on tree-structured fully adaptive Cartesian meshes provided by the Peano framework [27,28]. Dynamical adaptive mesh refinement (AMR) enhances the shock-capturing abilities of the ADER-DG scheme further [29] and allows good resolution of local features. ExaHyPE allows for complex geometry, either handled by a curvilinear mesh transformation, or by a diffuse interface approach. While the former approach is linear operation it is restricted to geometries that can be mapped smoothly to a cube. The latter approach extends a given PDE system by a parameter representing the volume fraction of material in the cell and thus determines the physical boundary through a diffuse interface instead of boundary-fitted unstructured meshes. Complex geometries can thus be readily represented and extensions to moving geometries are also possible [2,30,31].

ExaHyPE comprises the following key features:

- High-order ADER discontinuous Galerkin (ADER-DG) with a-posteriori subcell limiting and finite volume (FV) schemes;
- Dynamic mesh refinement on Cartesian grids in two and three dimensions;
- A simple API that allows users to quickly realise complex applications;
- User-provided code can be written in Fortran or C++;
- Automatically generated architecture- and application-specific optimised ADER-DG routines;
- Shared memory parallelisation through Intel's Threading Building Blocks (TBB);
- Distributed memory parallelisation with MPI.

Furthermore, ExaHyPE offers a wide range of post-processing and plotting facilities such as support for the output formats vtk or tecplot. The software can be compiled with Intel and GNU compilers and provides a switch for choosing different compilation modes: The release mode aggressively optimises the application, while the assertion and debug compilation modes activate the assertions within the code and print additional output. Users and developers can write log filters to filter output relevant to them. For continuous testing a Jenkins server [32] verifies that the code compiles with all parallelisation features in all different compilation modes including its ability to resolve complex geometries.

Our paper starts with an overview of the problem setting using several examples. We then briefly sketch the solution methods used in ExaHyPE, focusing on the ADER-DG algorithm. Next we use a simple example to demonstrate the workflow when using ExaHyPE and describe the engine architecture. We conclude with a sequence of numerical examples from various application areas to demonstrate the capabilities of the engine.

2. Problem formulation

We consider hyperbolic systems of balance laws that may contain both conservative and non-conservative terms. They have to be given in the following first-order form:

$$\frac{\partial}{\partial t}Q + \nabla \cdot F(Q, \nabla Q) + B(Q) \cdot \nabla Q = S(Q) + \sum_{i=1}^{n_{\text{ps}}} \delta_i, \tag{1}$$

where $Q : \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}^\nu$ is the state vector of the ν conserved variables, $\Omega \subset \mathbb{R}^d$ is the computational domain, $F(Q)$ is the flux

Please cite this article as: A. Reinarz, D.E. Charrier, M. Bader et al., ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems, Computer Physics Communications (2020) 107251, https://doi.org/10.1016/j.cpc.2020.107251.

tensor that may also depend on the gradient of Q in order to model viscous effects, and $B(Q)$ represents its non-conservative part. Finally, $S(Q)$ is the source term and δ_i are the given n_{ps} point sources.

Hyperbolic systems in the form (4) can be used to model a wide range of applications that involve waves. In the following, we demonstrate the versatility of our formulation by introducing equations from three different application areas. Numerical experiments for these examples are provided in Section 6.

2.1. Waves in elastic media

Linear elastodynamics describes waves propagating through elastic heterogeneous media by relating displacements, velocities, stress and strain. The momentum equations of motion are derived from Hooke's law and the conservation of momentum. Following the form of Eq. (4) we write them as

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix}\sigma\\ \rho v\end{pmatrix}}_{=Q}+\underbrace{\begin{pmatrix}E(\lambda,\mu) & 0\\ 0 & 0\end{pmatrix}\cdot\nabla\begin{pmatrix}v\\ \sigma\end{pmatrix}}_{=B(Q)\cdot\nabla Q}+\nabla\cdot\underbrace{\begin{pmatrix}0\\ \sigma\end{pmatrix}}_{=F(Q)}=0,$$

where ρ denotes the mass density, v the velocity and the σ stress tensor, which can be written in terms of its six independent components as $\sigma = (\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{xz}, \sigma_{yz})$. In this paper we consider isotropic materials, i.e. the material matrix $E(\lambda, \mu)$ depends only on the two Lamé constants λ and μ of the material. However, the formulation holds also for more general anisotropic materials.

These equations can be used to simulate seismic waves, such as those radiated by earthquakes. In this context the restriction of ExaHyPE to Cartesian meshes seems to be restrictive. However, adaptive Cartesian meshes can be extended to allow the modelling of complex topography. Two methods have been implemented in ExaHyPE to represent complex topographies. The first approach treats ExaHyPE's adaptive Cartesian mesh as reference domain that is mapped to a complex topography via high-order curvilinear transformations [33,34]. The second approach, a diffuse interface method, represents the topography as a smooth field [2]. These approaches are described in more detail in Section 6.2.

2.2. Shallow water equations

In atmospheric and oceanic modelling of coastal areas, horizontal length scales are typically significantly greater than the vertical length scale. In this case, fluid flow can be modelled with the two-dimensional shallow water equations instead of the more complicated three-dimensional Navier–Stokes equations.

Following the form of Eq. (4) they can be written as

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix}h\\ hu\\ hv\\ b\end{pmatrix}}_{=Q}+\nabla\cdot\underbrace{\begin{pmatrix}hu & hv\\ hu^2 & huv\\ huv & hv^2\\ 0 & 0\end{pmatrix}}_{=F(Q)}+\underbrace{\begin{pmatrix}0\\ hg\,\partial_x(b+h)\\ hg\,\partial_y(b+h)\\ 0\end{pmatrix}}_{=B(Q)\cdot\nabla Q}=0, \tag{2}$$

where h denotes the height of the water column, (u, v) the horizontal flow velocity, g the gravity and b the bathymetry.

Hyperbolic systems of balance laws have non-trivial equilibrium solutions in which flux and source terms cancel. A well-balanced numerical scheme is capable of maintaining such an equilibrium state. In Section 3 we describe the ADER-DG scheme used in ExaHyPE and in Section 6 we will describe how to keep the scheme well balanced while allowing wetting and drying.

2.3. Compressible Navier–Stokes equations

ExaHyPE is extensible to non-hyperbolic equations. As an example of such an extension we show the compressible Navier–Stokes equations. They are used to model the dynamics of a viscous fluid, and are given by

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix}\rho\\ \rho v\\ \rho E\end{pmatrix}}_{=Q}+\nabla\cdot\underbrace{\begin{pmatrix}\rho v\\ v\otimes\rho v+Ip+\sigma(Q,\nabla Q)\\ v\cdot(I\rho E+Ip+\sigma(Q,\nabla Q))-\kappa\nabla(T)\end{pmatrix}}_{=F(Q,\nabla Q)}=\underbrace{\begin{pmatrix}0\\ -gk\rho\\ 0\end{pmatrix}}_{=S(Q)}, \tag{3}$$

where ρ denotes the density, ρv the momentum, ρE the energy density, T the temperature and p the pressure (this term includes gravitational effects). The temperature diffusion is given by $\kappa\nabla T$ with constant κ, these effects depend on the gradient of Q. In the source term, the vector k is the unit vector in z-direction and g is the gravitation of Earth. The viscous effects are modelled by the stress tensor $\sigma(Q, \nabla Q)$. Thus, the flux from (4) has been modified to allow ∇Q as input. More details on the implementation of these equations can be found in [35].

2.4. General relativistic magneto-hydrodynamics

The equations of classical magnetohydrodynamics (MHD) are used to model the dynamics of an electrically ideally conducting fluid with comparable hydrodynamic and electromagnetic forces. When modelling astrophysical objects with strong gravitational fields, e.g. neutron stars, it becomes necessary to model the background space–time as well. We use the standard $3 + 1$ split to decompose the four dimensional space–time manifold into 3D hyper-surfaces parameterised by a time coordinate t. The background space–time is introduced into the equations in the form of a non-conservative product.

Following the form of Eq. (4) they can be written as

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix}\sqrt{\gamma}D\\ \sqrt{\gamma}S_j\\ \sqrt{\gamma}\tau\\ \sqrt{\gamma}B^j\\ \phi\\ \alpha_j\\ \beta\\ \gamma_m\end{pmatrix}}_{Q}+\nabla\cdot\underbrace{\begin{pmatrix}\alpha v^i D-\beta^i D\\ \alpha T^i_j-\beta^i S_j\\ \alpha(S^i-v^i D)-\beta^i\tau\\ (\alpha v^i-\beta^i)B^j-(\alpha v^j-\beta^j)B^i\\ 0\\ 0\\ 0\\ 0\end{pmatrix}}_{=F(Q)}+\underbrace{\begin{pmatrix}0\\ \sqrt{\gamma}(\tau\,\partial_j\alpha-\frac{1}{2}T^{ik}\partial_j\gamma_{ik}-T^j_i\partial_j\beta^i)\\ \sqrt{\gamma}(S^j\partial_j\alpha-\frac{1}{2}T^{ik}\beta^j\partial_j\gamma_{ik}-T^j_i\partial_j\beta^j)\\ -\beta^j\partial_i(\sqrt{\gamma}B^i)+\alpha\sqrt{\gamma}\gamma^{ji}\partial_i\phi\\ \sqrt{\gamma}\alpha c_h^2\partial_j(\sqrt{\gamma}B^i)-\beta^j\partial^j\phi\\ 0\\ 0\\ 0\end{pmatrix}}_{=B(Q)\cdot\nabla Q}=0,$$

where $i, j = 1, 2, 3$ and $m = 1...6$.

The curved space–time is parameterised by several hypersurface variables: lapse α, spatial metric tensor γ shift vector β and extrinsic curvature K. The spatial metric tensor is given as a vector of its six independent components $\gamma = (\gamma_{11}, \gamma_{12}, \gamma_{13}, \gamma_{22}, \gamma_{23}, \gamma_{33})$ and has the determinant $\sqrt{\gamma} := \det\gamma$. Further,

Table 1
An overview of the main kernels implementing the algorithm, all are implemented for 2D and 3D and in linear and non-linear versions.

ADER-DG	
spaceTimePredictor	Linear: Cauchy–Kowalevsky, Nonlinear: weak element-local space–time DG
Integrals	Implements face, surface, and volume integration
FV	
Godunov	Implements the Godunov FV scheme
MUSCL-Hancock	Implements the MUSCL-Hancock FV scheme
slopeLimiter	Various slope limiters , default: minmod
Both (ADER-DG + FV)	
solutionUpdate	Updates the solution vector
riemannSolver	Riemann solver, default: Rusanov
stableTimeStepSize	Calculates next stable time step size
Limiter	
projectOnFVLimiterSpace	Projects DG solution on FV subcells
projectOnDGSpace	Projects FV solution back onto DG grid
discreteMaximumPrinciple	Limiter criterion, checks discrete maximum principle

$D = W\rho$ is the conserved density, which is related to the rest mass density ρ by the Lorentz factor W, v^i is the fluid velocity, T is the Maxwell 3-energy momentum tensor, S is the conserved momentum, B is the magnetic field and τ is the conserved energy density. Finally, ϕ is an artificial scalar introduced to ensure a divergence-free magnetic field, and c_h is the characteristic velocity of the divergence cleaning. For more details on this formulation see e.g. [36–38].

3. Solver components

In ExaHyPE's engine concept the numerical method is given and in general the user does not need to interact with any solver components. A layer of generated glue code separates the user from the kernel calls. The main kernels are given in Table 1, of these only the slope-limiter and Riemann solver can be modified by the user. However, all modifications to the kernels are optional since stable defaults are provided. This section briefly summarises the numerical algorithms used. For brevity, let us use a pared down form of (4), in which only the flux term is nontrivial:

$$\frac{\partial Q}{\partial t} + \nabla \cdot F(Q) = 0 \qquad \text{on } \Omega \subset \mathbb{R}^d,\ d = 2, 3. \tag{4}$$

Assume that (4) is subject to appropriate initial and boundary conditions:

$$Q(x, 0) = Q_0(x),\ \forall x \in \Omega,$$
$$Q(x, t) = Q_B(x, t),\ \forall x \in \partial\Omega,\ \forall t \in \mathbb{R}_0^+.$$

3.1. Discretisation

ExaHyPE allows for Cartesian grids in two and three dimensions. The computational domain is divided into a grid $\Omega = \bigcup_i T_i$ using a space-tree construction scheme [27,39]. Each cell T_i is recursively refined giving an adaptive Cartesian grid as shown in Fig. 1. Meshes can be defined with an arbitrary number of elements in each direction on the coarsest level. Refinement is based on tripartitioning.

The rationale behind a commitment to tripartitioning is twofold. On the one hand, we rely on the Peano AMR framework [39] which internally orders all cells along a Peano space-filling curve. This yields a high temporal and spatial locality of the cell and face accesses; a property the Peano curve shares with other space-filling curves. However, the Peano curve also allows us to realise all temporary data structures through cache-friendly stacks [27]. On the other hand, tripartitioning is an intrinsic match for patch-based Finite Volume methods which work with odd numbers of volumes per axis: A subdivision of a cell here makes a former volume centre coincide with a volume centre on the next finer level. This simplifies the structure of inter-resolution transfer operators and preserves them over many resolution levels.

3.2. Finite volumes

Classically systems of hyperbolic PDE have been solved using finite difference or finite volume schemes. In a finite volume method cell averages are calculated. Volume integrals in a PDE that contain a divergence term are converted to surface integrals using the divergence theorem. These terms are then evaluated as fluxes at the surfaces of each element. However, to achieve high order accuracy in a finite volume scheme, large stencils and expensive recovery or reconstruction procedures are needed. Examples include essentially non-oscillatory (ENO) or weighted ENO (WENO) schemes, see e.g. [40,41].

As shown in Table 1 ExaHyPE provides access to classical Godunov type schemes as well as MUSCL schemes [42,43]. Further, users can use parts of the generic compute routines while implementing other parts on their own. For instance, in both the ADER-DG and Finite Volume schemes, users can overwrite the Riemann Solver with their own implementation.

3.3. ADER-DG

Instead of representing the solution as cell averages, DG methods represent the solution within each cell by a (high-order) polynomial. The ADER-DG method is a one-step predictor–corrector scheme. The integration in time is initially performed only within the element, neglecting the element interfaces and then a single correction step is performed to take the element interfaces into account [44].

The method operates on a weak form of Eq. (4)

$$\int_{T_i}\int_{t^n}^{t^{n+1}} \theta_h \frac{\partial q_h}{\partial t}\,\mathrm{d}x\mathrm{d}t + \int_{T_i}\int_{t^n}^{t^{n+1}} \theta_h \nabla \cdot F(q_h)\,\mathrm{d}x\mathrm{d}t = 0. \tag{5}$$

This formulation has replaced the solution Q with a discrete function q_h, represented by Lagrange polynomials θ_h in time and space. This solution is referred to as the space–time predictor and intuitively corresponds to a Taylor-expansion in space and time. Hence, it is equal to the time-evolution of the discrete solution within the control element for a smooth solution. The space–time test function is in the space of piecewise polynomials, which is constructed as tensor products of Lagrange polynomials over Gauss–Legendre or Gauss–Lobatto points.

Integrating over each element T_i and over the current time interval $[t^n, t^{n+1}]$ then gives the element-local weak formulation. Note that the formulation does not take into account any information from neighbouring elements. The predictor is subsequently corrected using contributions from neighbouring cells using a Riemann-solver.

However, first we need to compute q_h. This can be evaluated directly by the Cauchy–Kovalevsky algorithm for linear problems. For non-linear models, we use the ADER-DG method proposed by Dumbser et al. [45]. It is a fixed-point iteration and can be seen as a discrete counterpoint to the well-known Picard iteration, see [26]. There are three phases per ADER-DG time step:

1. Per grid cell T_i and time interval $[t^n, t^{n+1}]$, we first implicitly solve (5), predicting the local evolution. The concurrent solves of (5) do not take into account any information from neighbouring elements and thus yield jumps along the cell faces in q_h and $F(q_h)$.

Please cite this article as: A. Reinarz, D.E. Charrier, M. Bader et al., ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems, Computer Physics Communications (2020) 107251, https://doi.org/10.1016/j.cpc.2020.107251.

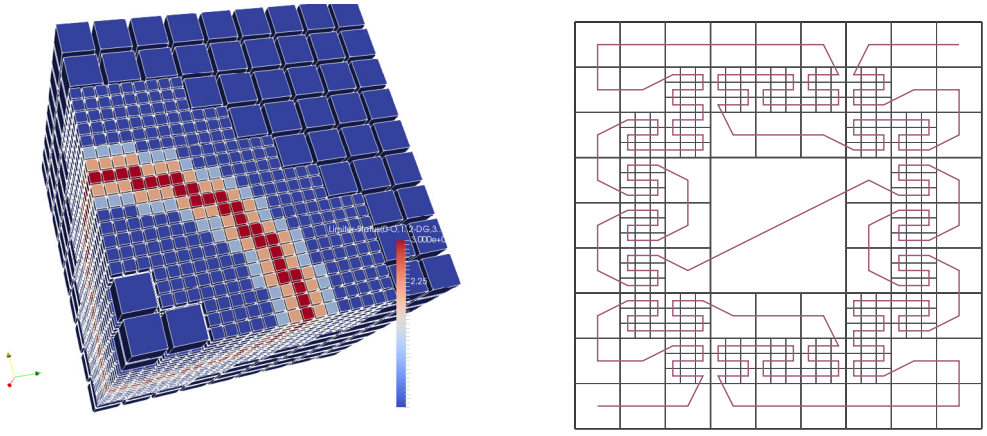

Fig. 1. Left: Adaptive Cartesian mesh in 3 dimensions. The FV limiter is active on the finest level. Right: Peano space-filling curve running through a 2D mesh that has a similar refinement pattern.

2. We traverse all faces of the grid and compute a numerical normal flux $G(q_h, F(q_h))$ from both adjacent cells. ExaHyPE uses a Rusanov flux by default; users can replace it with any other Riemann solver.
3. In the corrector step, we traverse the cells again and solve

$$\int_{T_i} \theta_h \, \Delta q_h \, \mathrm{d}x = -\int_{T_i} \int_{t^n}^{t^{n+1}} \nabla \theta_h \cdot F(q_h) \, \mathrm{d}x\mathrm{d}t + \int_{\partial T_i} \int_{t^n}^{t^{n+1}} \theta_h \, G(q_h, F(q_h)) \, \mathrm{d}s\mathrm{d}t \tag{6}$$

for $\Delta q_h = q_h(t^{n+1}) - q_h(t^n)$, see [16]. The one-step update (6) is derived by multiplying with a spatial testfunction and partially integrating (4). Eq. (6) can be easily inverted given that the ansatz and test space typically yield a diagonal mass matrix.

3.4. Time-step restrictions

Nonlinear effects and mesh adaptation require adjustments to the time step size during a simulation. This is expressed by the CFL condition, which gives an upper bound on the stable time step size for explicit DG schemes:

$$\Delta t \leq \frac{\mathrm{CFL}_N}{d\,(2N+1)} \frac{h}{|\lambda_{\max}|}, \tag{7}$$

where h and $|\lambda_{\max}|$ are the mesh size and the maximum signal velocity, respectively, and $\mathrm{CFL}_N < 1$ is a stability factor that depends on the polynomial order [16].

3.5. A-posteriori limiting

The unlimited ADER-DG algorithm will suffer from numerical oscillations (Gibbs phenomenon) in the presence of steep gradients or shock waves. Therefore a limiter must be applied. The approach followed in ExaHyPE is based on the a-posteriori MOOD method of Loubère et al. [25]. The solution is checked a-posteriori for certain admissibility or plausibility criteria and is recalculated with a robust FV scheme if it does not meet them. The FV patch size is chosen to have an order of $2N+1$, this is the smallest cell size that does not violate the CFL condition (7).

In contrast to the original approach, ExaHyPE's approach incorporates the observation that cells usually require a recalculation with FV multiple time steps in a row after the initial check failed. Therefore, ExaHyPE implements the a-posteriori limiting ADER-DG method as a hybrid ADER-DG-FV method (Fig. 2).

As a-posteriori detection criteria we use

1. *Physical Admissibility Criteria*: Depending on the system of PDEs being studied certain physical constraints can be placed on the solution. For the shallow water equations these are positivity of the water height. These criteria are supplied by the user along with the PDE terms.
2. *Numerical Admissibility Criteria*: To identify shocks we use a relaxed discrete maximum principle (DMP) in the sense of polynomials, for details see e.g. [25].

We call DG cells that do not satisfy the above criteria *troubled* cells. If a cell is flagged as troubled and has not been troubled in the previous time step, the scheme goes back to the old time step and recomputes the solution in all troubled cells (and their direct neighbours) with the FV scheme.

4. Engine architecture and programming workflow

This section briefly walks through the workflow of setting up an application in ExaHyPE using the shallow water equations given in (2) as an example. The architecture of ExaHyPE is illustrated in Fig. 3. ExaHyPE is a solver engine, domain-specific code has to be written by the user to obtain simulation code. In Fig. 3 a turquoise colour is used to highlight files written by the user. To write an ExaHyPE application users typically start from a specification file. The specification file is passed to the ExaHyPE toolkit, which creates glue code, empty application-specific classes and optionally application and architecture tailored core routines. The application specific classes are filled by the user with the PDE terms. This code can be written in C++ or Fortran. The generated glue code and the initially empty templates make up the ExaHyPE user solver.

The ExaHyPE core relies on the Peano framework (green) for its dynamically adaptive Cartesian meshes. In addition, it provides an efficient mesh traversal loop that ExaHyPE's algorithms plug into. Peano itself is a third-party component.

The number of dependencies in the ExaHyPE core is minimal. However, the architecture may not fulfil the requirements of all applications, so the user can extend and connect further software fragments to the ExaHyPE core. In red we show an optional dependency, libxsmm [46]. This package provides efficient kernels for small matrix multiplications, which are used by the optimised ADER-DG routines described in Section 5.2. Similarly, further software packages can be added by the user as needed.

In Table 2, a brief summary of the components of the user solver is given. In the following Section we highlight the parts

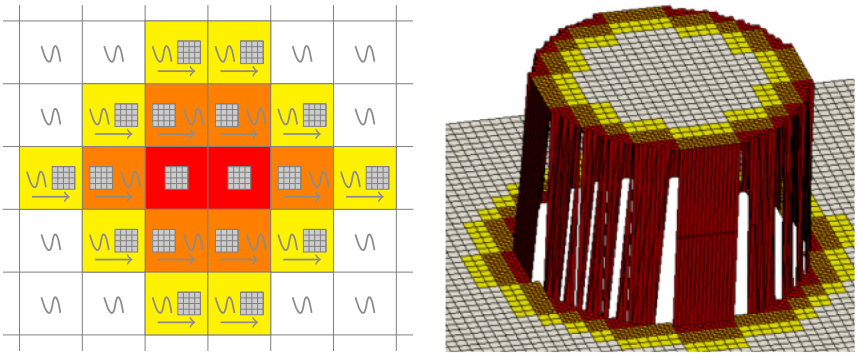

Fig. 2. Left: Limiter status stencil. Two FV cells (red) are surrounded by (ADER-)DG cells (white). Cells in the interface layers compute with FV and project to DG (orange) or with DG and project to FV (yellow). Right: The FV and DG subdomains are initialised along a discontinuity. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

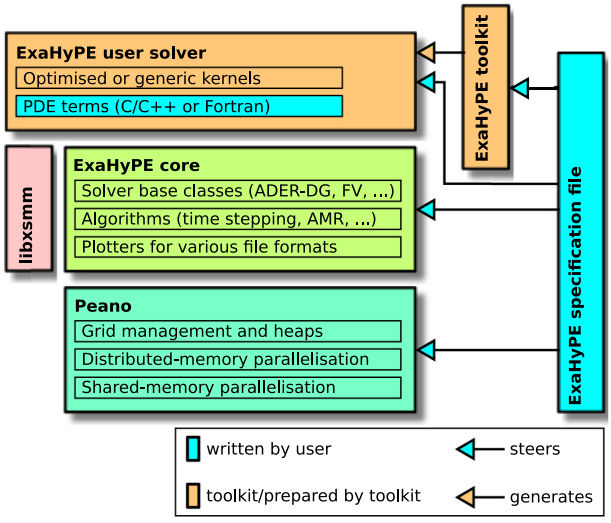

Fig. 3. Engine architecture.

Table 2
An overview over the optional and required user-implemented functions.

Required functions	
adjustPointSolution	Set values in the state vector. Required: initial values. Optional: constraints
eigenvalues	Eigenvalues to calculate time step restriction
boundaryValues	Define boundary conditions
Problem-dependent functions	
flux	Flux function
nonConservativeProduct	Non-conservative products
source, pointSource	Source terms
multiplyMaterialParameter-Matrix	Material parameters
viscousFlux	Flux terms with access to gradients
viscousEigenvalues	Eigevalues for time step restriction in viscous terms
Optional functions	
init	Initialisation of external libraries
refinementCriterion	Only for applications with AMR define additional areas to refine
isPhysicallyAdmissable	Only for applications with Limiter Gives physical admissibility criteria

of the specification file that need to be modified for each component and show the implementation of the user functions for the shallow water equations. The solver components can be set up flexibly by modifying the specification file. Users only need to write application-specific code that sets up their PDE system.

4.1. Code generation and compilation

After the specification file has been written it is handed over to the ExaHyPE toolkit. A minimal ExaHyPE specification file is shown below.

```
exahype-project SWE
 output-directory const = ./SWE

 computational-domain
   dimension const         = 2
   width                   = 1.0, 1.0
   offset                  = 0.0, 0.0
   end-time                = 1.0
 end computational-domain

 solver ADER-DG MySWESolver
   variables const    = h:1,hu:1,hv:1,b:1
   order const        = 3
   maximum-mesh-size  = 0.1
   time-stepping      = global
   type const         = non-linear
   terms const        = flux,ncp
   optimisation const = generic, usestack
 end solver
end exahype-project
```

To prepare this example for the simulation, run

```
> ./Toolkit/toolkit.sh Demonstrators/SWE.exahype
```

The toolkit generates a Makefile, glue code, as well as various helper files. Among them is one C++ class per solver that was specified in the specification file. Within each implementation file, the user can specify initial conditions, mesh refinement control, etc. For example, to set the eigenvalues the following function is generated in the file `MySWESolver.cpp`.

```
void SWE::MySWESolver::eigenvalues(const double* const Q,
                                   const int direction,
                                   double* const lambda) {
  // @todo Please implement/augment if required
  lambda[0] = 1.0;
  lambda[1] = 1.0;
  lambda[2] = 1.0;
  lambda[3] = 1.0;
}
```

This function can then be filled with the eigenvalues of the PDE system under consideration. Similarly, the other functions flux, initial conditions and boundary conditions can be defined in the file. For the shallow water equations this would be:

```
void SWE::MySWESolver::eigenvalues(const double* const Q,
                                   const int direction,
                                   double* const lambda) {
  ReadOnlyVariables vars(Q);
  Variables eigs(lambda);

  const double c = std::sqrt(gravity*vars.h());
  double u_n = Q[direction + 1] * 1.0/vars.h();

  eigs.h()  = u_n + c;
  eigs.hu() = u_n - c;
  eigs.hv() = u_n;
  eigs.b()  = 0.0;
}
```

In this example we have used named variables to enhance readability. However, ExaHyPE also allows the user to access the vectors directly as can be seen in the generated function above.

The whole build environment is generated. A simple make will create the ExaHyPE executable. ExaHyPE's specification files always act as both specification and configuration file, i.e. when running the code the specification file is passed in again.

```
> ./ExaHyPE-SWE ./SWE.exahype
```

A successful run yields a sequence of `vtk` files that you can open with Paraview or VisIt. In this example, we plot two quantities. Such an output is shown in Fig. 4.

Global metrics such as integrals can also be realised using a plotter with no output variables, i.e. `variable const = 0`.

5. Parallelisation and optimisation features

ExaHyPE relies on the Peano framework for mesh generation. Peano implements cache-efficient, tree-structured, adaptive mesh refinement. To traverse the cells or vertices the mesh traversal automaton of Peano runs along the Peano space-filling-curve (SFC), see Fig. 1. The action of a PDE operator is mapped to the SFC traversal by plugging into events triggered by the traversal automaton, these mappings can be generated by a toolkit.

Peano also takes care of the distributed-memory and shared-memory parallelisation. Domain decomposition for distributed-memory parallel simulations is realised by forking off or merging subtrees. The domain decomposition can be influenced via load balancing callbacks. The shared-memory parallelisation relies on identifying regular substructures in the tree and employing parallel-for loops in these areas [48]. Recently, we introduced a runtime tasking system to Peano and ExaHyPE that introduces additional multi-threading concurrency [49] in highly adaptive mesh regions and allows overlapping computations with MPI communication.

ExaHyPE builds on the Peano framework and it inherits a user model from Peano: Our engine removes the responsibility for algorithmic issues from the user. The time step, the mesh traversal and the parallelisation are implemented in a generic way. The user only controls the PDE system being solved, by specifying how many quantities are needed, which terms are required, and ultimately what all terms look like. Our goal is to allow users to focus on the physics only and to hide away as many implementation details as possible.

5.1. High-level optimisations

ExaHyPE contains several high level algorithmic optimisations that users can switch on and off at code startup through the specification file. To gain access to these optimisations, we add the following optional section to the minimal specification file shown in Section 4.

```
global-optimisation
    fuse-algorithmic-steps                = all
    spawn-predictor-as-background-thread = on
    spawn-amr-background-threads          = on
end global-optimisation
```

A discussion of each individual algorithmic tuning is beyond scope of this paper. Some techniques are:

- **Step fusion:** Each time step of an ExaHyPE solver consists of three phases: computation of local ADER-DG predictor (and projection of the prediction onto the cell faces), solve of the Riemann problems at the cell faces, and update of the predicted solution. We may speed up the code if we fuse these four steps into one grid traversal.
- **Spawn background jobs:** TBB has a thread pool in which idle threads wait for tasks, as soon as new tasks are spawned the threads in this pool start this task. We refer to these threads as background threads. Since spawning and scheduling these tasks has a overhead we wait until a certain number of tasks has accumulated and schedule these tasks together. Certain space–time-predictor computations can be spawned as a background job. Costly AMR operations such as the imposition of initial conditions and evaluation of refinement criteria can also be performed as a background jobs.
- **Modify storage precision:** ExaHyPE internally can store data in less-than-IEEE double precision. This can reduce the memory footprint of the code significantly.

Most users will not modify these options while they prototype. When they start production runs, they can tweak the engine instantiation through them. The rationale behind exposing these control values is simple: It is not clear at the moment which option combinations robustly improve performance. An auto-tuning/machine learning approach to find the optimal parameter combinations could be considered.

ExaHyPE allows running multiple simulations on the same computational mesh. This can be facilitated by adding multiple solver descriptions to the specification file. Each solver uses its own base grid and refinement criterion.

5.2. Optimised ADER-DG routines

One of ExaHyPE's key ideas is to use tailored, extremely optimised code routines whenever it comes to the evaluation of the ADER-DG scheme's steps and other computationally expensive routines. AMR projections and the space–time predictor (5) are typically the dominant steps. If the user decides to use these optimised variants in the specification file, the toolkit calls a specific code generator Python3 module and links to its output in the generated glue code so that the calls to the generic routines are replaced by calls to the generated optimised ones.

SIMD operations, notably introduced with AVX-512 on KNL and Skylake, become increasingly critical to fully exploit the potential of modern CPUs. Therefore, on Intel machines, the optimised routines' main goal is to either directly use SIMD or enable as much auto-vectorisation from the compiler as possible. To that end the optimised routines use Intel's libxsmm [46], which is

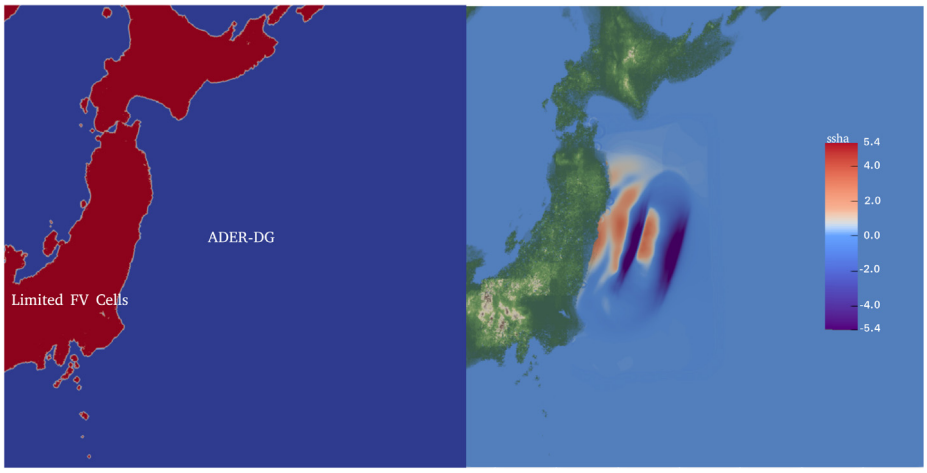

Fig. 4. Vtk output of a shallow water simulation of the Tohoku tsunami. Left: FV and DG domains. Right: the tsunami 5 min after the initial event. *Source:* These figures are taken from [47].

the second third-party building block in the basic ExaHyPE architecture. We map all tensor operations required by the ADER-DG algorithm to general matrix multiplications (gemms) and apply architecture-specific vectorised matrix operations. Furthermore, the code generation allows the introduction of data padding and alignment to get the most out of the compiler auto-vectorisation.

While improving SIMD vectorisation is our current prime use case for the optimised kernels, we provide alternative optimisation dimensions: Instead of a vectorisation of the native ADER-DG loops, the optimised routines can also be configured to work with vectorised PDE formulations. This implies that the PDE terms are computed through SIMD operations instead of scalar ones and requires some work from the user. This has been found to greatly improve performance [50].

Instead of the standard Picard iteration, we provide a continuous extension Runge–Kutta scheme (CERK) to yield an initial guess to the Picard iteration, leading to a much lower number of required iterations.

6. Numerical results

In this section, we show the ExaHyPE engine in action. The applications in this section are taken from our introductory discussion in Section 2. Only the Euler equations are added as an all-time classic starting point for solvers of hyperbolic systems. All specification and source files used to generate the results in this section are made publicly available, and all of the tests themselves can be run on a standard laptop. The scaling tests shown require the use of a larger cluster.

6.1. Euler equations

The non-linear compressible Euler equations model the flow of an inviscid fluid with constant density. Solutions of the Euler equations are sometimes used as approximations to real fluids problem, e.g. the lift of a thin airfoil. They are given by

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ j\\ E\end{pmatrix}+\nabla\cdot\begin{pmatrix}j\\ \frac{1}{\rho}j\otimes j+pI\\ \frac{1}{\rho}j(E+p)\end{pmatrix}=0, \tag{8}$$

where ρ denotes the mass density, $j\in\mathbb{R}^d$ denotes the momentum density, E denotes the energy density, p denotes the fluid pressure, to be given by an equation of state, and $\otimes$ denotes the outer product.

We extend the model by a colour function denoting the volume fraction of material in a cell starting from the Baer–Nunziato model [51]:

$$\frac{\partial}{\partial t}\begin{pmatrix}\alpha\rho\\ \alpha j\\ \alpha E\\ \alpha\end{pmatrix}+\nabla\cdot\begin{pmatrix}\alpha j\\ \frac{\alpha}{\rho}j\otimes j+\alpha pI\\ \frac{\alpha}{\rho}j(E+p)\\ 0\end{pmatrix}+\begin{pmatrix}0\\ -p\nabla\alpha\\ 0\\ 0\end{pmatrix}=0, \tag{9}$$

In the above PDE system α denotes the volume fraction of material present. The pressure is given by:

$$p=(\gamma-1)\left(E-\frac{1}{2\rho}\|j\|_2^2\right), \tag{10}$$

where $\gamma=1.4$ is the ratio of specific heats .

Note that the addition of α has introduced a non-conservative term into the equation. In the above we have given a *reduced* Baer–Nunziato model, similar to the approach presented in [2,30,31,52]. This model allows simulation of *fluid–structure-interaction (FSI)* problems on adaptive Cartesian grids, without requiring boundary-fitted grids.

In Fig. 5 we present computational results for flow over a thin airfoil. Initial conditions were:

$$j(x,0)=(1,0),\ \ \rho(x,0)=1.0,\ \ E(x,0)=2.5,$$

As boundary conditions we set an inflow boundary at the left, an outflow boundary at the right and $v\cdot\nabla n=0$ at the top and bottom boundaries. Results are shown for a NACA 4612 airfoil [53]. The results generate the expected bow shock.

To demonstrate the shared-memory scalability of the code we use a variant of the Sod shock tube problem, the circular, spherical explosion problem, also referred to as a "multi-dimensional Sod shock tube". This test case uses the following initial conditions

$$j(x,0)=0,\ \ \rho(x,0)=\begin{cases}1 & \text{if } \|x-x_0\|^2<r^2,\\ \frac{1}{8} & \text{else}\end{cases},$$

$$E(x,0)=\begin{cases}1 & \text{if } \|x-x_0\|^2<r^2,\\ \frac{1}{10} & \text{else.}\end{cases}$$

and wall boundary conditions.

In Fig. 6 we show the shared-memory scalability in two and three dimensions. This scaling test was run on the 14 cores of an Intel Xeon E5-v3 processor with 2.2 GHz core frequency (SuperMUC Phase 2). The tests were run on one processor with 14 cores using Intel's TBB [54] for parallelisation. We consider

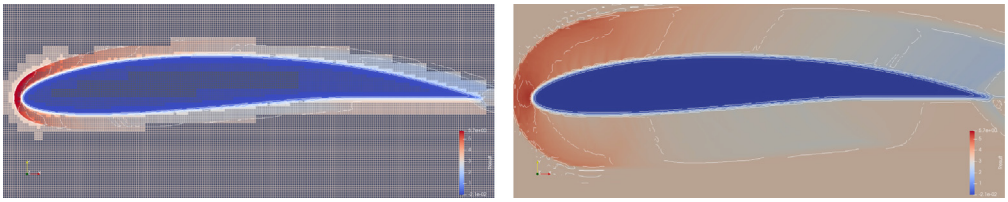

Fig. 5. An ADER-DG implementation of flow over an airfoil in 2D using a base grid of 79 × 25 cells. From left to right: The energy E at $t = 0.03$ and at $t = 0.1$.

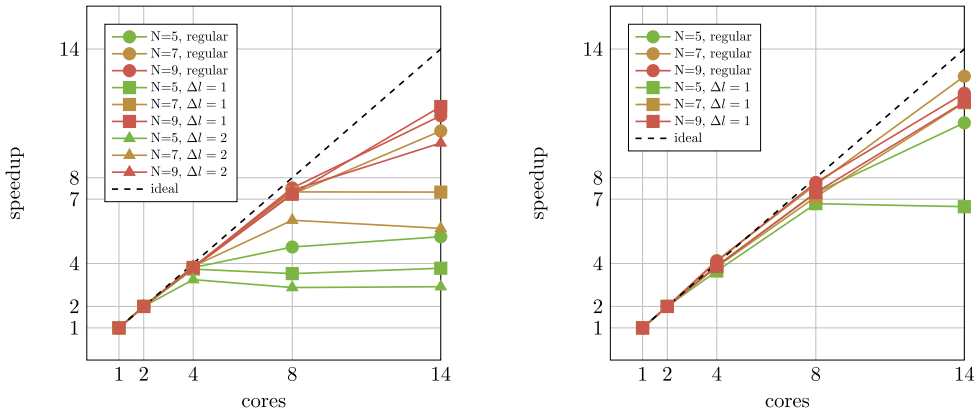

Fig. 6. Shared-memory scalability of the non-linear ADER-DG implementation for a circular, spherical explosion problem experiments using a base grid with up to two levels of adaptive refinement. Left: 2D test with a base grid of 81^2 cells. Right: 3D test with a base grid of 27^3 cells.

a regular base grid and allow a dynamic refinement criterion to add up to two additional levels $\Delta\ell \in \{1, 2\}$ of cells around the shock front. In these tests all kernel level optimisations that are available in ExaHyPE have been used. As a result, scalability is more challenging, however, we are interested mainly in reducing the overall run-time.

The scalability improves as the work per element increases, i.e. it improves with higher polynomial degree N or for increasing dimension. In general good scalability can be observed on up to 14 cores at higher orders. All ExaHyPE codes employ a hybrid parallelisation strategy with at least two MPI ranks per node [55]. Results for this hybrid parallelisation strategy are provided in Section 6.6.

6.2. Elastic wave equation

The restriction of ExaHyPE to Cartesian meshes seems restrictive in the context of most seismic applications. In this section, we introduce two methods for incorporating complex geometries into such a mesh *without modifying the underlying numerical method*. We would like to highlight that both of these methods omit manual mesh-generation, which is typically required as a pre-processing step in computational seismology applications and poses a major bottleneck. Hexahedral mesh generation can easily consume weeks to months and is limited for complex geometries of boundary and interface conditions, while form-fitted unstructured tetrahedral meshes allow for automatised meshing and complex geometries [56,57], however, pose numerical challenges, e.g. in form of misshaped sliver elements [58].

6.2.1. Diffuse interface approach

As in the Euler equations we can extend the elastic wave equation by a parameter α, which represents the volume fraction of the solid medium present in a control volume [2]. Diffuse interfaces completely avoid the problem of mesh generation, since all that is needed for the definition of the complex surface topography is to set a scalar colour function to unity inside the regions covered by the solid and to zero outside.

The diffuse interface model is given by:

$$\frac{\partial \boldsymbol{\sigma}}{\partial t} - E(\lambda,\mu)\cdot\frac{1}{\alpha}\nabla(\alpha\boldsymbol{v}) + \frac{1}{\alpha}E(\lambda,\mu)\cdot\boldsymbol{v}\otimes\nabla\alpha = 0,$$
$$\frac{\partial \alpha\boldsymbol{v}}{\partial t} - \frac{\alpha}{\rho}\nabla\cdot\boldsymbol{\sigma} - \frac{1}{\rho}\sigma\nabla\alpha = 0,$$

where $E(\lambda,\mu)$, λ, μ, ρ and the stress tensor $\boldsymbol{\sigma}$ are defined as in Section 2.1.

The material parameters are assumed to remain constant, i.e.

$$\frac{\partial\alpha}{\partial t} = 0, \qquad \frac{\partial\lambda}{\partial t} = 0, \qquad \frac{\partial\mu}{\partial t} = 0, \qquad \frac{\partial\rho}{\partial t} = 0.$$

Physically, α represents the volume fraction of the solid medium present in a control volume. The equations become non-linear in those regions in which $0 < \alpha < 1$.

This formulation can be extended to allowing moving materials. Instead of solving $\frac{d\alpha}{dt} = 0$, we can solve

$$\frac{d\alpha}{dt} + v\nabla\alpha = 0.$$

In this way the free surface boundary is allowed to move according the local velocity field [2].

To verify the accuracy of this method we solve the layer over homogeneous halfspace (LOH.1) benchmark problem described by Day et al. [59]. This problem is a well-known reference benchmark for seismic wave propagation in numerical codes. The LOH.1 benchmark considers way propagation in a hexahedral geometry filled with two materials that are stacked on top of each other. The first material is characterised by a lower density and smaller seismic wave speeds. The exact material parameters are defined in Table 3. The parameters λ and μ of the equation can be derived from these values.

A point source is placed 2 km below the surface at the centre of the domain, such that the resulting wave propagates through the change of material.

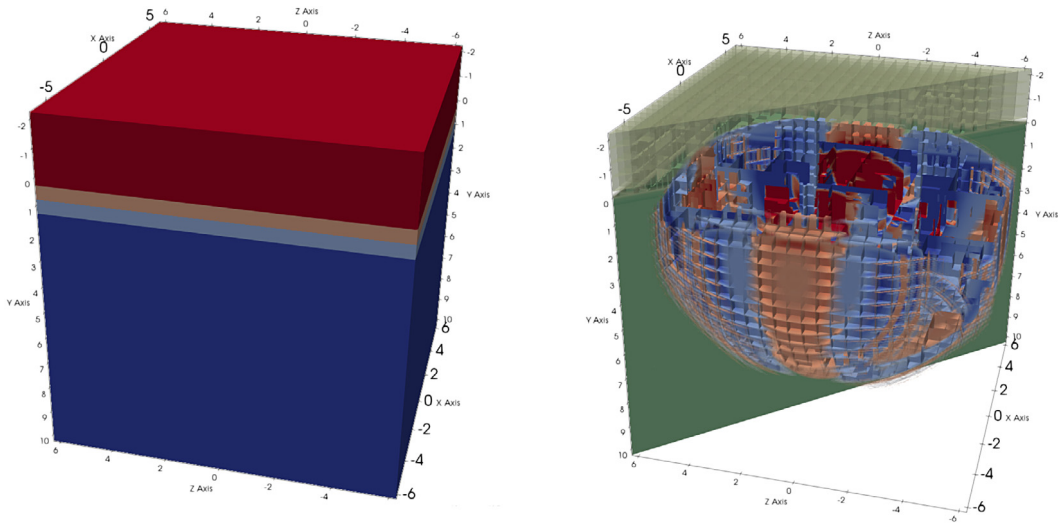

Fig. 7. From left to right: Plot of the limited area indicated by α; The velocity field in x with the simulated topography in the background at $t = 1.1$(transparent indicates the free surface). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 3
Material parameters of the LOH.1 benchmark.

x	c_p[m/s]	c_s[m/s]	ρ[kg/m^3]
< 1 km	4000	2000	2600
≥ 1 km	6000	3464	2700

In the diffuse interface method we limit the free surface to be able to resolve the local discontinuity of α. The left picture in Fig. 7 shows the resulting distribution of limited (red) and unlimited areas (blue) and indicates the transition of both (light blue and red). The right picture shows the velocity field in x direction at $t = 1.1$. Our implementation of the diffuse interface method is able to successfully resolve both the changing material parameters and the absorbing surface boundary conditions.

6.2.2. Curvilinear meshes

Our second approach models complex topographies by applying a curvilinear transformation to the elements of an adaptive Cartesian mesh. To do so we generate surface quadrature nodes depending on the topography, create a 2D curvilinear interpolation of those quadrature nodes on domain boundaries with topography curves and domain edges as constraints, and finally generate a Jacobian mapping J at each node. Taking this mapping into account, the linear elastic wave equations can be written as:

$$\frac{\partial \sigma}{\partial x} = \frac{1}{J}\left(\frac{\partial}{\partial q}(Jq_x\sigma) + \frac{\partial}{\partial r}(Jr_x\sigma) + \frac{\partial}{\partial s}(Js_x\sigma)\right)$$
$$\frac{\partial v}{\partial x} = q_x\frac{\partial v}{\partial q} + r_x\frac{\partial v}{\partial r} + s_x\frac{\partial v}{\partial s}$$

where J is the Jacobian matrix of the mapping from mesh to topography. This approach allows us to model meshes with complex topographies including faults and inner branches [33,34]. The ADER-DG algorithm as described previously, can then be applied directly to this version of the elastic wave equation.

Fig. 8 shows a snapshot of a numerical experiment that simulates propagation of seismic waves in the area around the mountain Zugspitze, in Germany. Both the curvilinear approach and the diffuse interface method are able to resolve the complicated topographies in this scenario which is motivated by the AlpArray experiment.

Fig. 9 shows the shared-memory scalability of the linear ADER-DG implementation on SuperMUC's Phase 2 when running the LOH.1 benchmark comparing the scalability of the two meshing approaches. We consider a regular base grid with 27^3 cells and allow a dynamic refinement criterion to add one additional level of cells to the layer over the halfspace. Memory constraints prevented further refinement. As in the non-linear test case the scalability improves as the work per element increases. However, due to the lower amount of total work in the linear setting a higher polynomial degree needs to be reached to attain scalability in this case. This means that the overall scalability of the diffuse interface approach is higher, however, this comes at the price of a higher overall computational cost due to the non-linearity of the formulation.

The question of which method should be used is highly problem dependent. The curvilinear method is purely linear and as such computationally cheap. The non-linear diffuse interface approach, on the other hand, requires additional limiting on the surface. However, the time step size for the DIM is independent of the simulated topography, while in the curvilinear method it is highly dependent on the distortion introduced by the topography. Simulations with a relatively smooth, flat surface are expected to be faster with the curvilinear method, while highly varying topographies are better discretised with the DIM. DIM has the added advantage of allowing moving free surface boundaries, a feature that is difficult to realise with curvilinear meshes.

6.3. Shallow water equations

We demonstrate the dynamic mesh refinement capabilities of ExaHyPE via the shallow water equations (2). Even more importantly, we present a non-toy problem which uses ExaHyPE's 2D facilities. To solve (2), we use the a-posteriori limiting ADER-DG method and equip its FV limiter with a recently developed HLLEM Riemann solver [60]. The solver allows for wetting and drying. This also demonstrates that users can inject their own Riemann solvers in ExaHyPE.

The same scenario is available in ExaHyPE using a Finite Volume scheme instead of an ADER-DG scheme with FV limiter [47,61]. In both implementations the same Riemann solver can be used. The Riemann solver considers a dry tolerance $\epsilon > 0$ below which cells are marked as *dry*. The constant ϵ is necessary to avoid negative water heights, which usually lead to unphysical non-linear effects. If one of the cells is flooded the jump in bathymetry is taken into account by the Riemann solver. When

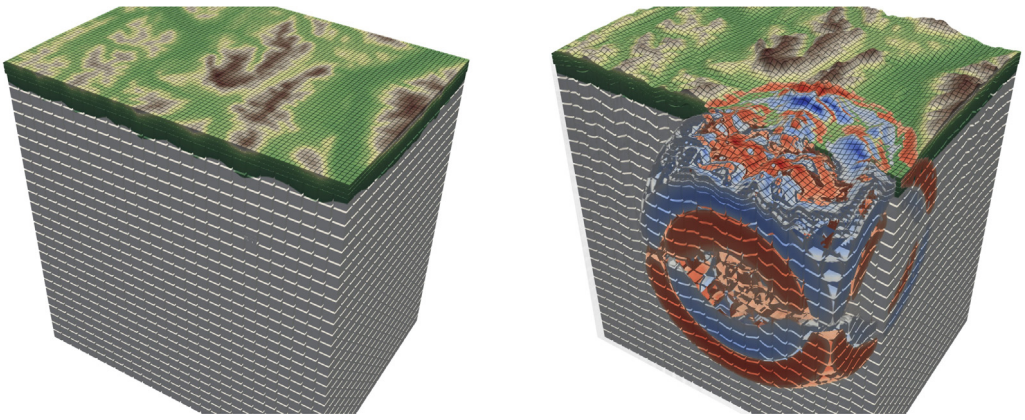

Fig. 8. Simulation of seismic waves around the Mount Zugspitze with curvilinear meshes. The internal Cartesian mesh (left) is used to simulate a complex topography (right).

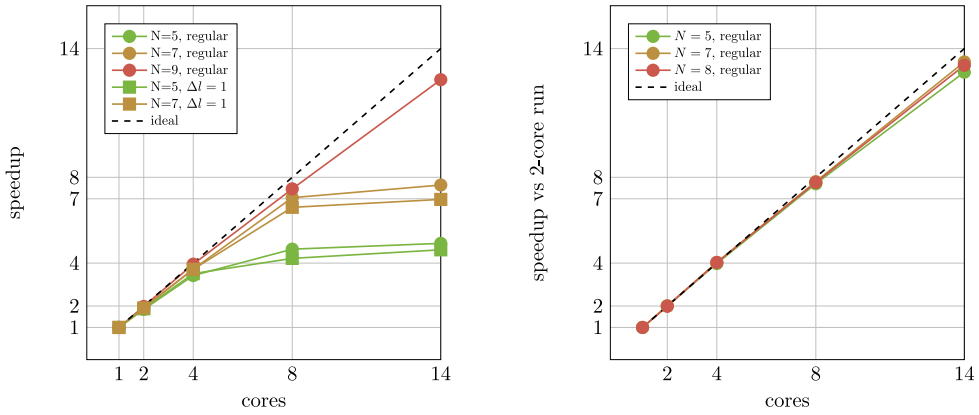

Fig. 9. Shared-memory scalability of the linear ADER-DG implementation on SuperMUC's Phase 2 when running the LOH.1 benchmark. Left: Curvilinear elements are used. Right: The diffuse interface method is used.

reconstructing the DG solution from the FV limiter the water level is reconstructed first and then the bathymetry is subtracted. This ensures that the non-linear reconstruction process does not produce artificial waves. In tsunami simulations, this scheme allows us to simulate areas close to the coast with the FV subcell limiter, while areas in the deep ocean are processed by the high order ADER-DG method.

To test the shallow water equations given in (2) we use the oscillating lake scenario. Here, a water droplet travels in circular motion over a dry basin. The topography of the basin is resolved using the final variable b, which remains constant throughout the simulation. The analytical solution, which is used as an initial condition is given by

$$Q = \begin{pmatrix} h \\ hu \\ hv \\ b \end{pmatrix} = \begin{pmatrix} \max\left\{0, \frac{1}{10}\left(x_0 \cos(\omega \cdot t) + x_1 \sin(\omega \cdot t) + \frac{3}{4}\right) - b\right\} \\ \frac{1}{2}\omega \sin(\omega \cdot t) h \\ \frac{1}{2}\omega \cos(\omega \cdot t) h \\ \frac{1}{10}(x_0^2 + x_1^2) \end{pmatrix},$$

where $\omega = \sqrt{0.2g}$ and $g = 9.81$ denotes the gravitational constant. The initial conditions are supplemented by wall boundary conditions. In this setup we use polynomial order $N = 3$ and a base grid of 7×7 elements. We adaptively refine mesh elements where the water height is small but not zero. We allow up to two levels of adaptive refinement.

This scenario provides a challenging test case for numerical codes due to the continuous wetting and drying of cells. There exists an analytical solution for this benchmark scenario allowing us to verify the well-balancedness of a scheme, as well as in the resolution of drying or inundated cells.

In the top row of Fig. 10, we plot the water height at $t = 0, 1, 2, 3$. Despite the frequent wetting and drying the scheme performs well. In those areas in which the DG solution is used, we note the accuracy of the higher order scheme on a coarse grid. In those areas in which a FV solution is necessary, the dynamic AMR is useful to retain accuracy and avoid diffusion. The bottom row of Fig. 10 shows the locations in which the FV limiter is active and adaptive mesh refinement is used. Furthermore, it shows the error in the water height. We observe that the FV limiter is used only in those cells in which it is required, the limiter locations move with the solution. The error remains below 10^{-3} in all cells.

In Fig. 11, we show the shared-memory scalability for the oscillating lake. This scaling test was run on one 2.2 GHz 14 core Intel Xeon E5-v3 processor of SuperMUC Phase 2. We consider a regular base grid and allow a dynamic refinement criterion to add up to two additional levels of cells around the water droplet. In this area the FV limiter is active. As in the previous test case the scalability improves for higher polynomial degree N.

6.4. General relativistic magneto-hydrodynamics

To uncover and validate ExaHyPE's high convergence order, we require a test case which does not contain discontinuities, as the limiter reduces the convergence order. Furthermore, the

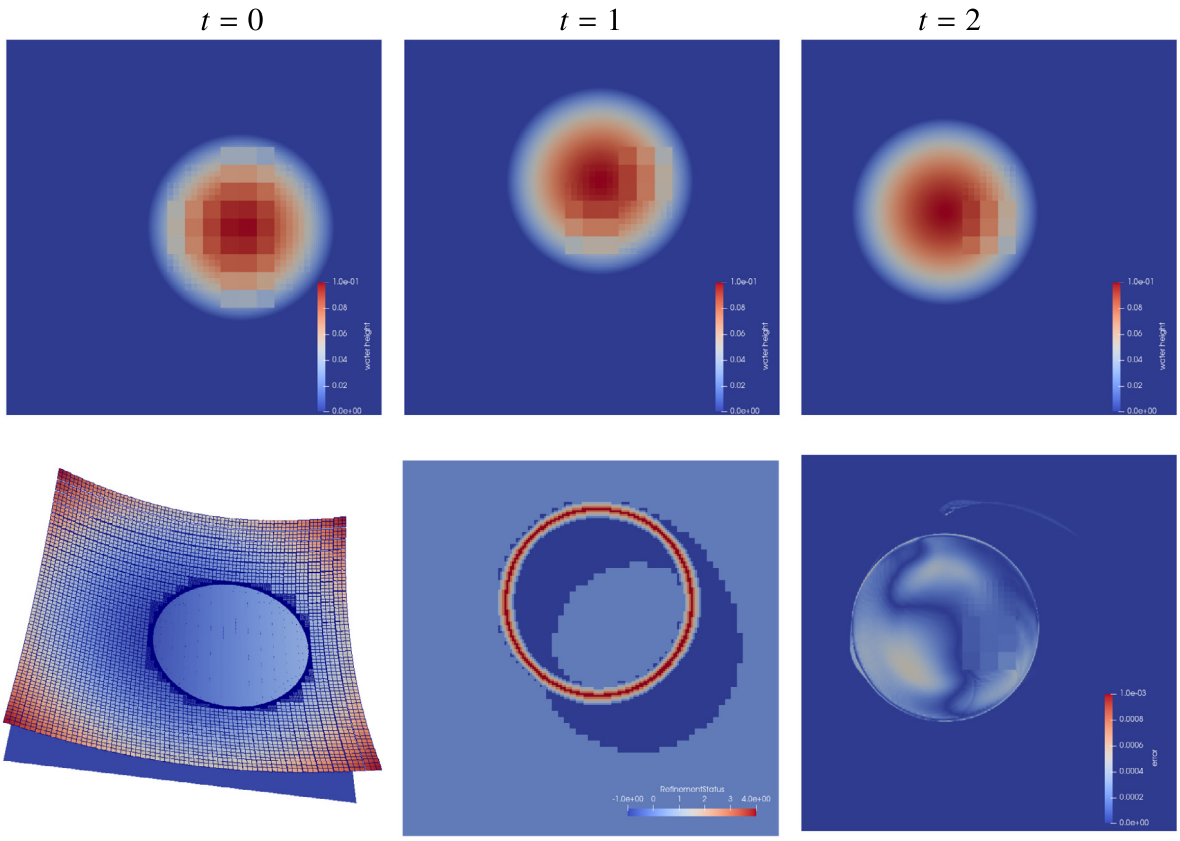

Fig. 10. ADER-DG with a-posteriori limiting for the oscillating lake scenario (shallow water equations). Top row: water height at time $t = 0, 1, 2, 3$. Bottom row: The locations (red) in which the FV limiter is active, the adaptively refined mesh and the error at times $t = 2$ and 3. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

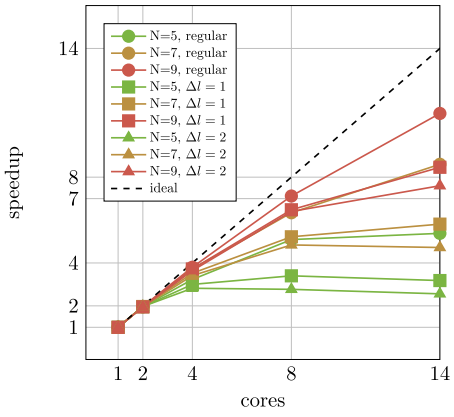

Fig. 11. Shared-memory scalability of the non-linear ADER-DG implementation for the oscillating lake experiments using a base grid of 243^2 cells with up to two levels of adaptive refinement.

analytical solution has to be known. We simulate the spherical accretion onto a stationary black hole. This well-known test case has a known analytical solution [62]. Details of the setup in the context of ADER-DG methods can also be found in [6]. This setup is challenging since GRMHD is a non-linear equation of 19 variables containing both fluxes and non-trivial non-conservative products. Further complexity arises from the closeness of the singularity at the critical radius to the computational domain.

We use the analytical solution as the external state vector for the Riemann solver at the boundary of the domain $\partial\Omega$. The test is performed in Kerr–Schild coordinates on the spatial domain $[2.2, 12.2]^3$, away from the critical radius.

In Fig. 12 we show the solution at time $t = 1.0$ and the convergence of the error in terms of L^1-norm for three different polynomial degrees $N = 2, 3$ and 6. Table 4 lists L^1-errors and convergence order for each polynomial degree. This shows that our scheme converges to the expected order of $N + 1$.

For the next test, we move to a simulation for which the analytical solution is not known. In this test, a stable non-rotating and non-magnetised neutron star is simulated in three space dimensions by solving the GRMHD equations in the Cowling approximation, i.e. assuming a static background spacetime. The initial state has been obtained as a solution to the Tolman–Oppenheimer–Volkoff (TOV) equations. The corresponding fluid and metric variables are compatible with the Einstein field equations. We set the magnetic field to zero for TOV stars.

The TOV equations constitute a non-linear ODE system in the radial space coordinate, that has been solved numerically by means of a fourth order Runge–Kutta scheme on a very fine grid with step size $dr = 0.001$. The parameters of the problem have been chosen to be: $\rho_c = 1.28 \cdot 10^{-3}$, adiabatic exponent $\Gamma = 2$ and a constant atmospheric pressure $p_{\text{atm}} = 10^{-12}$.

The star sits at the origin of the domain $[-15, 15]^3$. We apply reflection boundary conditions at the three simulation boundary surfaces $x = 0$, $y = 0$ and $z = 0$. At the surfaces $x = R$, $y = R$ and $z = R$, we apply exact boundary conditions, evaluating the initial data at the boundary.

In Fig. 13, we show the shared memory of scalability of the non-linear hybrid ADER-DG – FV implementation of the GRMHD equations for the TOV star run on SuperMUC phase 2. We consider a regular base grid with 27^3 cells and allow a dynamic

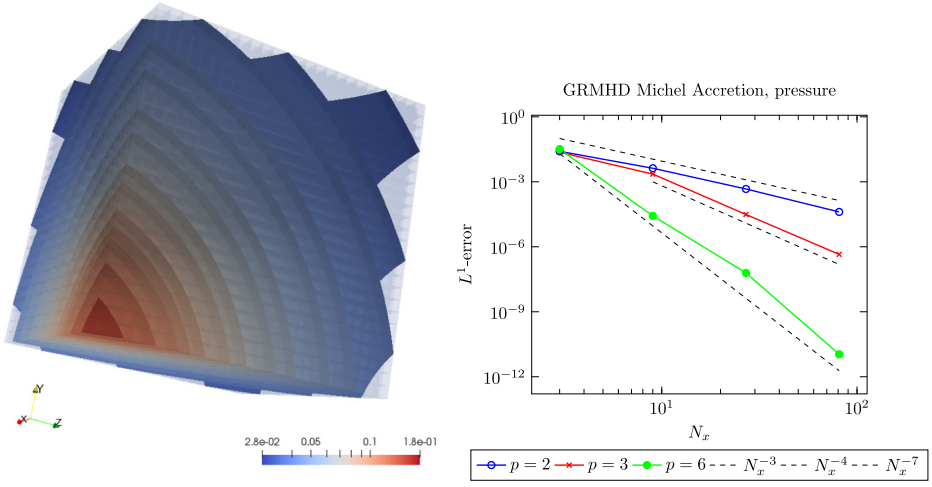

Fig. 12. (Left) The solution for pressure of the Michel accretion test at $t = 1.0$. (Right) Convergence of the ADER-DG scheme at $t = 1.0$. The dotted lines show the expected convergence rates.

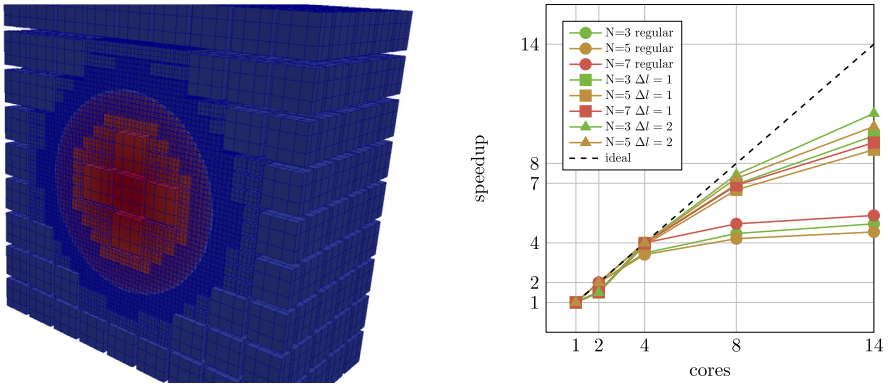

Fig. 13. Left: The TOV star setup showing two levels of adaptive mesh refinement along a sphere around the star. Right: Shared-memory scalability of the non-linear hybrid ADER-DG – FV implementation when running the TOV star example.

refinement criterion to add one additional level of cells along a sphere around the TOV star.

6.5. Compressible Navier–Stokes

The compressible Navier–Stokes equations (3) demonstrate that ExaHyPE is not only able to solve hyperbolic PDEs, but is also capable of handling viscous effects. We use the numerical flux of [63] to integrate this into our ADER-DG framework.

To test the equations, we utilise the Arnold–Beltrami–Childress (ABC) flow

$$\rho(x,t) = 1,$$

$$v(x,t) = \exp(-1\mu t)\begin{pmatrix}\sin(z)+\cos(y)\\ \sin(x)+\cos(z)\\ \sin(y)+\cos(x)\end{pmatrix},$$

$$p(x,t) = -\exp(-2\mu t)\,(\cos(x)\sin(y)+\sin(x)\cos(z) + \sin(z)\cos(y)) + C,$$

where the constant $C = 100/1.4$ governs the Mach number and $\mu = 0.01$ is the viscosity [64]. This equation has an analytical solution in the incompressible limit. We impose this solution at the boundary. In Fig. 14 we show a comparison between the analytical solution and a simulation with order $N = 2$ on a regular mesh of 27^3 cells. We see a good agreement with the analytical solution a time $t = 1.0$.

In addition, we evaluate our scenario for the colliding bubbles scenario of [65]. The initial conditions of this scenario are obtained by an atmosphere with a background state that is in hydrostatic balance, i.e. where

$$\frac{\partial}{\partial z}p(z) = -g\rho(z).$$

Initially, the domain has the same potential temperature Θ. This is then perturbed by

$$\Theta' = \begin{cases} A & r \le a, \\ A\exp\left(-\frac{(r-a)^2}{s^2}\right) & r > a, \end{cases}$$

where r is the Euclidean distance from the centre of the bubble (x_c, z_c) and the spatial position (x, z). We have two bubbles, with constants

warm:	$A = 0.5$, $x_c = 500\,\text{m}$,	$a = 150\,\text{m}$, $z_c = 300\,\text{m}$,	$s = 50\,\text{m}$,
cold:	$A = -0.15$, $x_c = 560\,\text{m}$,	$a = 0\,\text{m}$, $z_c = 640\,\text{m}$.	$s = 50\,\text{m}$,

Table 4
Convergence of the ADER-DG scheme for the Michel accretion test at $t = 1.0$.

#cells	$N = 2$		$N = 3$		$N = 6$	
	Error L^1	Order	Error L^1	Order	Error L^1	Order
3^3	2.56e−02		2.31e−02		3.23e−02	
9^3	4.22e−03	1.65	2.27e−03	2.11	2.28e−05	6.45
27^3	4.66e−04	2.01	3.09e−05	3.91	6.24e−08	5.52
81^3	4.06e−05	2.21	4.47e−07	3.86	1.09e−11	7.88

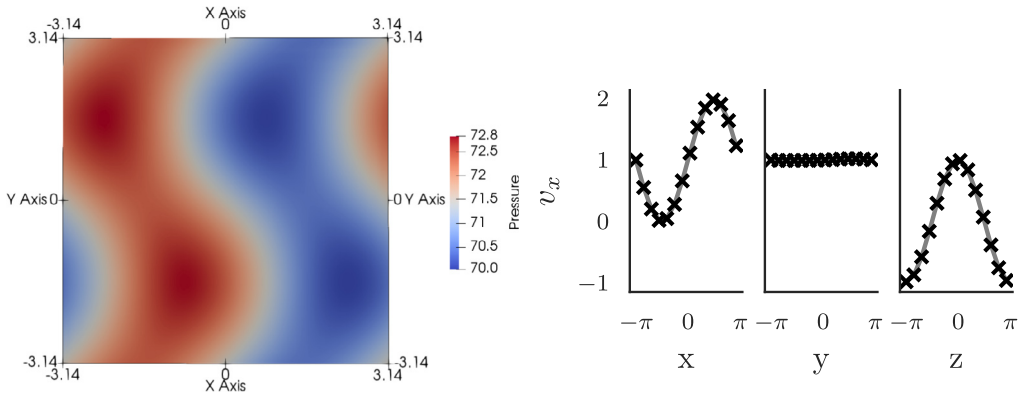

Fig. 14. Left: Pressure of the ABC-flow, *x*-*y* slice. Right: Velocity slices of the ABC-flow. Markers indicate samples of our solution, line indicates analytical solution. *Source:* Image reproduced from [35].

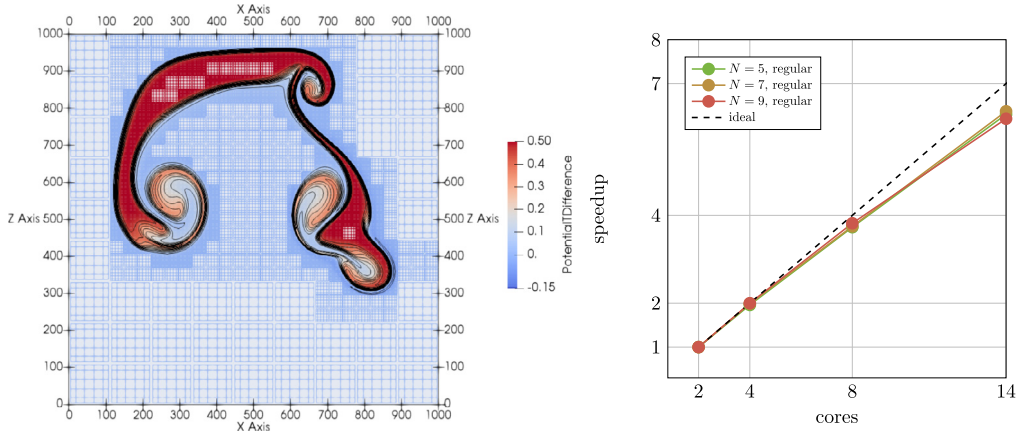

Fig. 15. Left: The colliding bubble scenario with two levels of dynamic adaptive mesh refinement. Image taken from [35] Right: Shared-memory scalability of the nonlinear ADER-DG implementation when running the ABC-flow example with viscous effects.

The simulation runs for 600 s and uses a viscosity of $\mu = 0.01$ for regularisation. We use order $N = 5$ and a mesh with up to two adaptive AMR levels. The results of this simulation are shown in Fig. 15 (left). There is an excellent agreement with the reference solution of [65]. For further details on the setup, we refer to [35]. In Fig. 15 (right), we show the shared-memory scalability using the ABC-flow on SuperMUC phase 2. In this test case the shared memory scalability is very good for all tested polynomial orders.

6.6. Hybrid parallelisation strategy

In the previous sections we showed the shared memory scalability of various test cases in two and three dimensions. In general we observed that the scalability improves as the work per element increases, i.e. it improves with higher polynomial degree N, for increasing dimension, or PDE complexity. In this section we demonstrate the effectiveness of a hybrid parallelisation strategy using the GRMHD TOV star setup described in Section 6.4. We consider a regular base grid of 79^3 cells and allow up to two levels of adaptive mesh refinement. As before all kernel level optimisations have been switched on.

This scaling test was run on up to 56 nodes of SuperMUC Phase 2 consisting of two 2.2 GHz 14 core Intel Xeon E5-v3 processors each. The tests were run using both Intel's TBB and MPI for parallelisation. As a reference value we use a test run on 56 cores and calculate the corresponding parallel efficiency compared to this baseline. Table 5 contains the results of this strong scaling test. Here, efficiency measures the speedup obtained divided by the expected optimal speedup. The efficiency is over 1.0 in some cases can be explained by the chosen baseline of 56 cores.

Table 5
Hybrid scaling of the GRMHD application. A $N = 6$ ADER-DG approximation plus Finite Volumes limiting is used. Up to two levels of adaptive mesh refinement are employed.

# cores (nodes)	Threads	Regular		$\Delta l = 1$		$\Delta l = 2$	
		Time	Efficiency	Time	Efficiency	Time	Efficiency
56 (2)	14	91.93	1.00	139.43	1.00	395.90	1.00
112 (4)	7	37.68	1.22	57.03	1.29	151.11	1.31
224 (8)	4	30.88	0.77	34.17	0.89	139.42	0.93
448 (16)	2	17.67	0.43	26.66	0.65	57.02	0.85
784 (28)	1	13.88	0.32	22.89	0.44	34.17	0.67

7. Conclusions and future work

This paper introduces a software engine that allows users to write higher-order ADER-DG codes for hyperbolic PDE systems with both conservative and non-conservative terms and a-posteriori limiters. The engine, ExaHyPE, has been designed to work on a wide range of computer systems from laptops to large high-performance compute clusters. Its core vision is to provide an engine rather than a framework: Technical details both on the computational science and numerical side are hidden from the user. A specification file plus very few routines realising the PDE terms are typically the only customisation points modified by user codes. Users can focus on the physics. To achieve this

high level of abstraction, writing code in ExaHyPE requires the use of a pre-specified set of numerical methods. The code thus clearly stands in the tradition of software packages such as Clawpack [66], as opposed to more generic, general-purpose software packages such as AMReX, deal.II or DUNE.

There are three general directions for future work. First, the sketched application areas have to make an impact in their respective domain. This comprises application-specific performance engineering and studies of real-world setups and data. It notably also comprises the coupling of engine applications with other code building blocks. Examples for first work into this direction are [2,33,34]. Second, we have to continue to investigate and to invest into the methods under the hood of ExaHyPE. Examples for such new ingredients on our agenda are accelerator support, local time stepping, and more dynamic load balancing and autotuning [67,68]. Finally, we plan extensions of the core paradigm of the engine. The development of ExaHyPE from scratch has been possible as we restricted ourself to ADER-DG only. In the future, we plan to elaborate to which degree we are able to add particle-based (Particle-in-Cell methods or simple tracers) algorithms or multigrid solvers (for elliptic subterms) on top of ExaHyPE. This will open new user communities to the engine. First feasibility studies for this [69–71] already exist.

Declaration of competing interest

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

Acknowledgements and funding

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671698, www.exahype.eu.

The ExaHyPE team acknowledges additional support by the European Union's Horizon 2020 research and innovation program (ChEESE, grant no. 823844).

The authors gratefully acknowledge the support by the Leibniz Supercomputing Centre, Germany (www.lrz.de), which also provided the computing resources on SuperMUC (Grant No. pr48ma and Grant No. pr63qo) .

We would especially like to thank the many people who have made contributions to ExaHyPE, in particular our previous team members Benjamin Hazelwood, Angelika Schwarz, Vasco Varduhn and Olindo Zanotti.

Appendix A. Dependencies and prerequisites

ExaHyPE requires the following prerequisites:

- For sequential simulations, only a C++ compiler is required. The code uses only few C++14 features, but for many older versions enabling those features through `--std=c++0x` is required.
- Python 3
- ExaHyPE's default build environment uses GNU Make.

Further, ExaHyPE has the following optional dependencies:

- ExaHyPE user code can also be written in Fortran. In this case, a Fortran compiler is needed.
- To exploit multi- or manycore computers, Intel's TBB 2017 is required. It is open source and works with GCC and Intel compilers.
- To run ExaHyPE on a distributed memory cluster, MPI is needed. ExaHyPE uses only very basic MPI routines (use e.g. MPI 1.3).
- To use ExaHyPE's optimised compute kernels, Intel's `libxsmm`[1] and Python's module `Jinja2`[2] are required. A local installation script is made available.

Appendix B. Obtaining ExaHyPE

ExaHyPE is free software is hosted at www.exahype.eu. There a two different options to obtain ExaHyPE, the first is to download a complete snapshot of ExaHyPE, in this case a snapshot of Peano is included. The second option is to clone the repository, in this case Peano has to be added manually. The repository is available at https://gitlab.lrz.de/exahype/ExaHyPE-Engine. The ExaHyPE guidebook contains documentation, detailed rationale and links to further resources and is available at http://www.peano-framework.org/exahype/guidebook.pdf.

References

[1] K. Duru, L. Rannabauer, A.-A. Gabriel, H. Igel, A new discontinuous Galerkin spectral element method for elastic waves with physically motivated numerical fluxes, 2017, arXiv:1802.06380.
[2] M. Tavelli, M. Dumbser, D.E. Charrier, L. Rannabauer, T. Weinzierl, M. Bader, J. Comput. Phys. 386 (2019) 158–189.
[3] N.T. Bishop, L. Rezzolla, Living Rev. Relativ. 19 (1) (2016) 2, http://dx.doi.org/10.1007/s41114-016-0001-9.
[4] Michael Dumbser, Federico Guercilena, Sven Köppel, Luciano Rezzolla, Olindo Zanotti, Phys. Rev. D 97 (8) (2018) 084053, http://dx.doi.org/10.1103/PhysRevD.97.084053, arXiv:1707.09910.
[5] Olindo Zanotti, Michael Dumbser, Computational Astrophysics and Cosmology 3 (2016).
[6] F. Fambri, M. Dumbser, O. Zanotti, Space–time adaptive ADER-DG schemes for dissipative flows: Compressible Navier–Stokes and resistive MHD equations, Comput. Phys. Commun. 219.
[7] W. Reed, T.R. Hill, Triangular mesh methods for the neutron transport equation.
[8] B. Cockburn, C.-W. Shu, ESAIM Math. Model. Numer. Anal. 25 (3) (1991) 337–361.
[9] B. Cockburn, C.-W. Shu, Math. Comp. 52 (186) (1989) 411–435.
[10] B. Cockburn, S.-Y. Lin, C.-W. Shu, J. Comput. Phys. 84 (1) (1989) 90–113.
[11] B. Cockburn, S. Hou, C.-W. Shu, Math. Comp. 54 (1990) 545–581.
[12] B. Cockburn, C.-W. Shu, J. Comput. Phys. 141 (2) (1998) 199–224.
[13] E.F. Toro, R.C. Millington, L.A.M. Nejad, Towards Very High Order Godunov Schemes, Springer US, Boston, MA, 2001, pp. 907–940, http://dx.doi.org/10.1007/978-1-4615-0663-8_87.
[14] V.A. Titarev, E.F. Toro, J. Sci. Comput. 17 (1) (2002) 609–618, http://dx.doi.org/10.1023/A:1015126814947.
[15] G. Gassner, M. Dumbser, F. Hindenlang, C.-D. Munz, J. Comput. Phys. 230 (11) (2011) 4232–4247, http://dx.doi.org/10.1016/j.jcp.2010.10.024.
[16] M. Dumbser, C. Enaux, E.F. Toro, J. Comput. Phys. 227 (8) (2008) 3971–4001.
[17] R. Hartmann, P. Houston, J. Comput. Phys. 183 (2) (2002) 508–532.
[18] P.-O. Persson, J. Peraire, Sub-cell shock capturing for discontinuous Galerkin methods, in: 44th AIAA Aerospace Sciences Meeting and Exhibit, p. 112.
[19] D. Radice, L. Rezzolla, Phys. Rev. D 84 (2011) 024010.
[20] J. Qiu, C. Shu, SIAM J. Sci. Comput. 26 (3) (2005) 907–929.
[21] J. Qiu, C.-W. Shu, Hermite WENO schemes and their application as limiters for runge-kutta discontinuous Galerkin method: one-dimensional case, 193 (2003) 115–135.
[22] R. Loubère, M. Dumbser, S. Diot, Commun. Comput. Phys. 16 (3) (2014) 718–763, http://dx.doi.org/10.4208/cicp.181113.140314a.
[23] S. Diot, R. Loubère, S. Clain, International Journal of Numerical Methods in Fluids 73 (2013) 362–392.
[24] Steven Diot, Stéphane Clain, Raphaël Loubère, Journal of Computational Physics 230 (10) (2011) 4028–4050.
[25] M. Dumbser, O. Zanotti, R. Loubère, S. Diot, J. Comput. Phys. 278 (C) (2013) 47–75.
[26] M. Dumbser, O. Zanotti, R. Loubère, S. Diot, J. Comput. Phys. 278 (2014) 47–75.
[27] T. Weinzierl, M. Mehl, SIAM J. Sci. Comput. 33 (5) (2011) 2732–2760, http://dx.doi.org/10.1137/100799071.
[28] T. Weinzierl, ACM Trans. Math. Software 45 (2) (2019) 14:1–14:41.

[1] Libxsmm is open source: https://github.com/hfp/libxsmm.
[2] Jinja2 is open source: http://jinja.pocoo.org/.

[29] O. Zanotti, F. Fambri, M. Dumbser, A. Hidalgo, Comput. & Fluids 118 (2015) 204–224.
[30] M. Dumbser, Comput. Methods Appl. Mech. Engrg. 200 (2011) 1204–1219.
[31] M. Dumbser, Comput. Methods Appl. Mech. Engrg. 257 (2013) 47–64.
[32] The jenkins project, https://jenkins.io/.
[33] K. Duru, A. Gabriel, G. Kreiss, Comput. Methods Appl. Mech. Engrg. 350 (2019) 898–937.
[34] K. Duru, L. Rannabauer, O.-K.A. Ling, A.-G. Gabriel, H. Igel, M. Bader, A stable discontinuous Galerkin method for linear elastodynamics in geometrically complex media using physics based numerical fluxes.
[35] Lukas Krenz, Leonhard Rannabauer, Michael Bader, Parallel Processing and Applied Mathematics, 13th International Conference PPAM 2019, in: Lecture Notes in Computer Science, vol. 12043, 2020, arXiv:1905.05524.
[36] Sven Köppel, Journal of Physics Conference Series 1031 (2018) 012017, http://dx.doi.org/10.1088/1742-6596/1031/1/012017, arXiv:1711.08221.
[37] F. Fambri, M. Dumbser, S. Köppel, L. Rezzolla, O. Zanotti, Mon. Not. R. Astron. Soc. 477 (4) (2018) 4543–4564, http://dx.doi.org/10.1093/mnras/sty734, arXiv:/oup/backfile/content_public/journal/mnras/477/4/10.1093_mnras_sty734/1/sty734.pdf.
[38] L. Rezzolla, O. Zanotti, Relativistic Hydrodynamics, Oxford University Press, Oxford UK, 2013.
[39] T. Weinzierl, ACM Trans. Math. Software (2020) (2nd version submitted for publication).
[40] R. Abgrall, J. Comput. Phys. 114 (1) (1994) 45–58.
[41] M. Dumbser, A. Hidalgo, O. Zanotti, Comput. Methods Appl. Mech. Engrg. 268 (2014) 359–387.
[42] V.P. Kolgan, J. Comput. Phys. 230 (2011) 2384–2390.
[43] B. van Leer, J. Comput. Phys. 135 (2) (1997) 229–248, http://dx.doi.org/10.1006/jcph.1997.5704.
[44] V. Titarev, E. Toro, J. Sci. Comput. 17 (2002) 609–618.
[45] O. Zanotti, F. Fambri, M. Dumbser, A. Hidalgo, Comput. & Fluids 118 (2015) 204–224, http://dx.doi.org/10.1016/j.compfluid.2015.06.020.
[46] A. Heinecke, G. Henry, M. Hutchinson, H. Pabst, SC16: International Conference for High Performance Computing, Networking, Storage and Analysis(SC), Vol. 00, 2016, pp. 981–991, http://dx.doi.org/10.1109/SC.2016.83.
[47] L. Rannabauer, S. Haas, D.E. Charrier, T. Weinzierl, M. Bader, Environmental Informatics: Techniques and Trends. Adjunct Proceedings of the 32nd edition of the EnviroInfo, 2018.
[48] W. Eckhardt, T. Weinzierl, Parallel Processing and Applied Mathematics, 8th International Conference, PPAM 2009, Wroclaw, Poland, September 13–16, 2009. Revised Selected Papers, Part I, 2009, pp. 567–575, http://dx.doi.org/10.1007/978-3-642-14390-8_59.
[49] D. Charrier, B. Hazelwood, T. Weinzierl, SIAM Journal on Scientific Computing (2020) (in press; arXiv:1806.07984).
[50] M. Dumbser, F. Fambri, M. Tavelli, M. Bader, T. Weinzierl, Efficient implementation of ADER Discontinuous Galerkin schemes for a scalable hyperbolic PDE engine, Axioms 7.
[51] M.R. Baer, J.W. Nunziato, J. Multiphase Flow 12 (1986) 861–889.
[52] E. Gaburro, M. Castro, M. Dumbser, Comput. & Fluids 175 (2018) 180–198.
[53] J. Moran, An Introduction to Theoretical and Computational Aerodynamics, in: Dover Books on Aeronautical Engineering, Dover Publications, 2003, URL https://books.google.de/books?id=4eVP3yWZ1LgC.
[54] J. Reinders, Intel Threading Building Blocks, first ed., O'Reilly & Associates, Inc, Sebastopol, CA, USA, 2007.
[55] Dominic E. Charrier, Benjamin Hazelwood, Ekaterina Tutlyaeva, Michael Bader, Michael Dumbser, Andrey Kudryavtsev, Alexander Moskovsky, Tobias Weinzierl, The International Journal of High Performance Computing Applications 33 (5) (2019) 973–986, http://dx.doi.org/10.1177/1094342019842645.
[56] S. Wollherr, A.-A. Gabriel, P.M. Mai, Landers 1992 reloaded: Integrative dynamic earthquake rupture modeling, J. Geophys. Res.: Solid Earth 124 (7) 6666–6702 http://dx.doi.org/10.1029/2018JB016355, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018JB016355.
[57] T. Ulrich, A.-A. Gabriel, J.-P. Ampuero, W. Xu, Dynamic viability of the 2016 mw 7.8 kaikÅura earthquake cascade on weak crustal faults, Nature Communications 10 (1213). http://dx.doi.org/10.1038/s41467-019-09125-w.
[58] C. Uphoff, S. Rettenberger, M. Bader, E.H. Madden, T. Ulrich, S. Wollherr, A.-A. Gabriel, SC '17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2017, URL https://dl.acm.org/citation.cfm?id=3126948.
[59] S.M. Day, C.R. Bradley, Memory-efficient simulation of anelastic wave propagation, Bull. Seismol. Soc. Amer. 3, 520–531. https://dx.doi.org/10.1785/0120000103.
[60] M. Dumbser, D.S. Balsara, J. Comput. Phys. 304 (C) (2016) 275–319, http://dx.doi.org/10.1016/j.jcp.2015.10.014.
[61] L. Rannabauer, M. Dumbser, M. Bader, ADER-DG with a-posteriori finite-volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework, Comput. Fluids https://dx.doi.org/10.1016/j.compfluid.2018.01.031.
[62] F.C. Michel, Accretion of Matter by Condensed Objects 15 (1972) 153–160 http://dx.doi.org/10.1007/BF00649949.
[63] G. Gassner, F. Lörcher, C.-D. Munz, J. Sci. Comput. 34 (3) (2008) 260–286, http://dx.doi.org/10.1007/s10915-007-9169-1.
[64] M. Tavelli, M. Dumbser, J. Comput. Phys. 319 (2016) 294–323, http://dx.doi.org/10.1016/j.jcp.2016.05.009.
[65] A. Müller, J. Behrens, F.X. Giraldo, V. Wirth, Fifth European Conference on Computational Fluid Dynamics, ECCOMAS CFD, 2010.
[66] Clawpack Development Team, Clawpack software, version 5.4.0, 2017, http://dx.doi.org/10.5281/zenodo.262111, URL http://www.clawpack.org.
[67] M. Schreiber, T. Weinzierl, HiPEAC 2018—3rd COSH Workshop on Co-Scheduling of HPC Applications, 2018.
[68] D.E. Charrier, T. Weinzierl, Parallel Processing and Applied Mathematics - 12th International Conference, PPAM 2017, Lublin, Poland, September 10–13, 2017, 2017, pp. 3–13, http://dx.doi.org/10.1007/978-3-319-78054-2_1.
[69] B. Reps, T. Weinzierl, ACM Trans. Math. Software 44 (1) (2017) 2:1–2:36.
[70] T. Weinzierl, B. Verleye, P. Henri, D. Roose, Parallel Comput. 52 (2016) 42–64.
[71] M. Weinzierl, T. Weinzierl, ACM Trans. Math. Software 44 (3) (2018) 32:1–32:44.

Please cite this article as: A. Reinarz, D.E. Charrier, M. Bader et al., ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems, Computer Physics Communications (2020) 107251, https://doi.org/10.1016/j.cpc.2020.107251.