ExaHyPE 2: An engine for hyperbolic PDEs solving codes

and how we build a numerical relativity code based on this

Han Zhang[1 2], Tobias Weinzierl[1], Baojiu Li[2], Sean Baccas[1], Mario Wille[3], Holger Schulz[1]…

[1] Department of Computational Science, Durham University
[2] Institute for Computational Cosmology, Durham University
[3] Computer Science, Technische Universität München
…

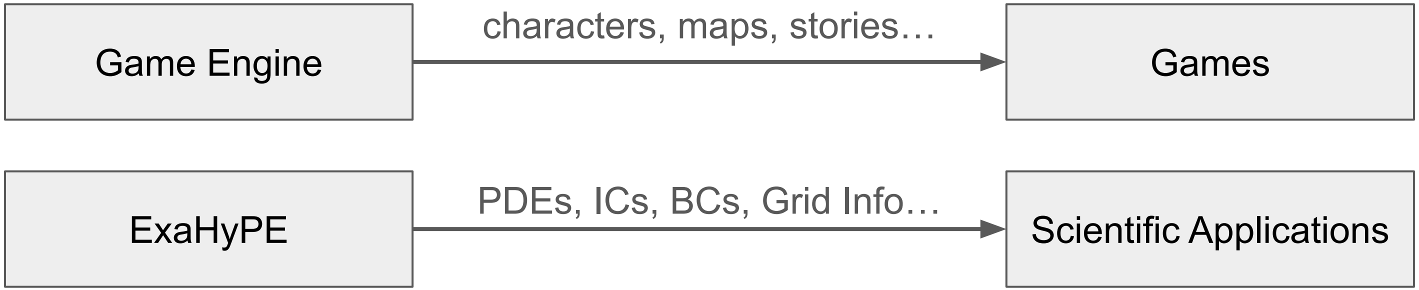
June 25, 2024, Cambridge

Outline

- ExaHyPE 2 Engine
  - Design and Architecture
  - Data Structure and Decomposition
  - Python API and Usage
  - User-defined Features
- ExaGRyPE: numerical relativity on ExaHyPE 2
  - Physics Remarks
  - Implementation
  - Preliminary Results

What is ExaHyPE 2?

An Exascale Hyperbolic PDE Engine[1], 2nd generation



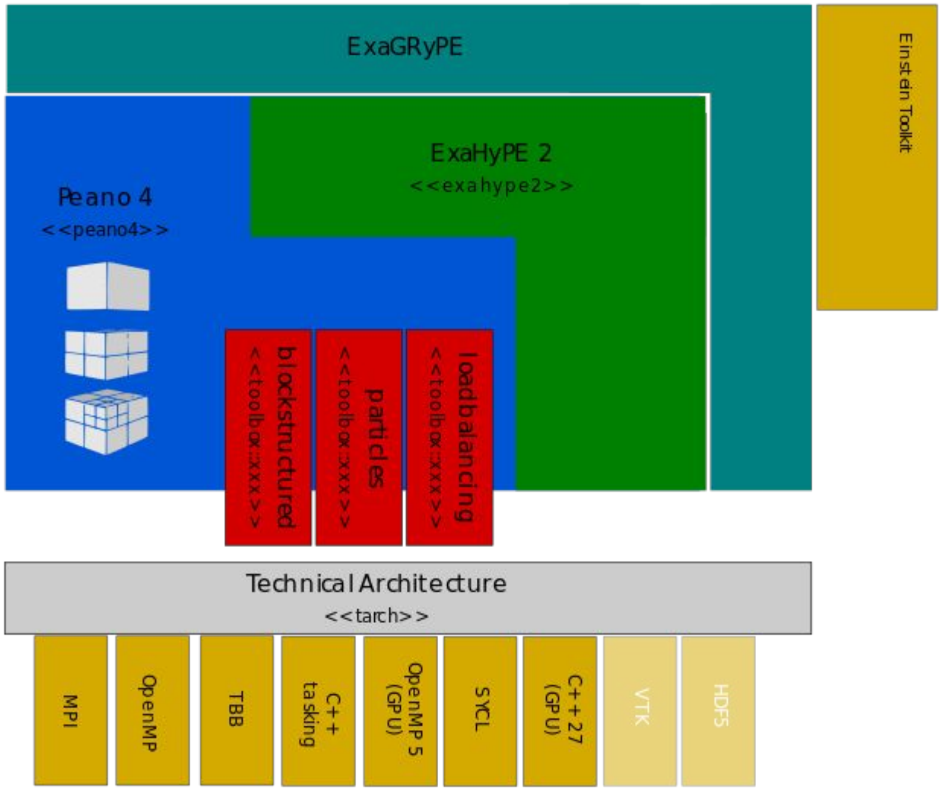
Users:

- Pick numerical solvers i.e., numerical scheme (FV, RKFD, RKDG…)
- Provide PDE terms, Initial conditions, Boundary conditions
- Decide the size, resolution and refine criteria for the grid

Engine:

- Generate and combine actual simulation code
- Handle data storage, parallelization, optimization automatically
- Determine *where*, *when*, *in which order* and *how* to call compute kernels

[1] Anne Reinarz, et al *(2020)*



ExaHyPE 2:

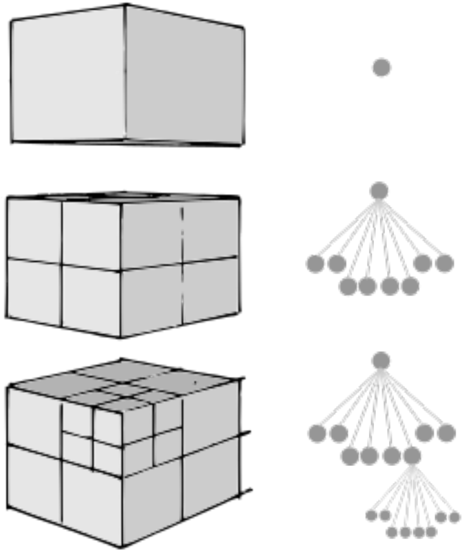
- Based on *Peano 4*
  a framework for solvers operating on dynamically adaptive Cartesian meshes
- Supported by *Technical Architecture*
  collections of libraries responsible for underlying technical realizations

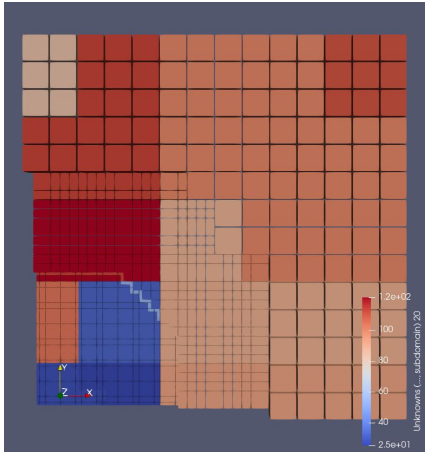
ExaGRyPE

- Application example
- build upon ExaHyPE 2
- also access to external library, e.g. `EinsteinToolkit`

ExaHyPE
An Exascale Hyperbolic PDE Engine

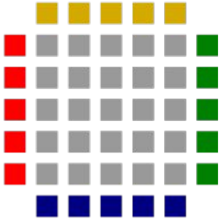
Durham University
Department of Computer Science

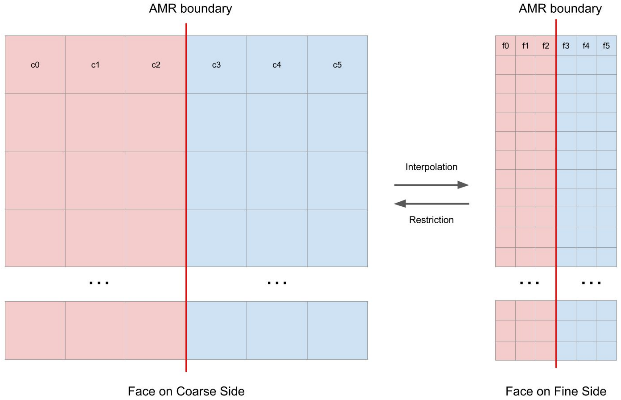




Data Structure

- Top-down generalized octree
- Tree carries actuall computate data (*Patches*)
- the code traverse the octree and apply kernel (basic idea)
- *Hollywood principle* in control logic





Decomposition

- Divided into subdomain for parallelization
- Following the space-filling curves
- Patches attached with halos, decoupled the communications and computations
- Resolution transitions happen in halos

Python API and Usage

Construct the application project

```
project = exahype2.Project(namespace = ["benchmarks", "exahype2", "ccz4"], name = "CCZ4",
executable="test ")
project.set_global_simulation_parameters(dimensions = 3, offset = [-0.5, -0.5, -0.5], domain_size =
[1, 1, 1], periodic_boundary_conditions = [True, True, True], end_time = 1.0,. . . )
project.set_Peano4_installation(directory=..., build_mode=peano4.output.CompileMode.Release)
…… #Actual solver construction
peano4_project = project.generate_Peano4_project()
peano4_project.generate()
```

Pick the solver(numerical schme)

```
my_solver = exagrype.api.CCZ4Solver_FD_GlobalAdaptiveTimeStep( name="CCZ4FD", patch_size=9,
min_meshcell_h=0.01, max_meshcell_h=0.1, rk_order=4, ...)
#if min and max h are not the same, the AMR is switched on automatically
project.add_solver(my_solver)
```

Python API and Usage

Specify the PDEs and other ingredients

```
my_solver.set_implementation(
    boundary_conditions=exahype2.solvers.PDETerms.User_Defined_Implementation,
    ncp=exahype2.solvers.PDETerms.User_Defined_Implementation,
    flux=exahype2.solvers.PDETerms.None_Implementation,
    source_term=exahype2.solvers.PDETerms.User_Defined_Implementation,
    refinement_criterion = exahype2.solvers.PDETerms.User_Defined_Implementation,
    eigenvalues = exahype2.solvers.PDETerms.User_Defined_Implementation)
```

$$\frac{\partial}{\partial t}Q + \boldsymbol{\nabla}\cdot\mathbf{F}(Q) + \sum_i \boldsymbol{\mathcal{B}}_i \frac{\partial Q}{\partial x_i} = \mathbf{S}(Q).$$

The first call of the python will then created a separated ***.cpp*** files which contains empty functions for corresponding terms specified above.

Python API and Usage

Fill functions accordingly (Head file showed)

```
class ...::exahype2::ccz4::CCZ4FD: public ...::exahype2::ccz4::AbstractCCZ4FD {

 RefinementCommand refinementCriterion ( double* Q, const Vector<Dimensions,double>& x, ...) override;

 void initialCondition (double* Q, const Vector<Dimensions,double>& x, ...) override;

 void nonconservativeProduct(
  double* Q,
  const Vector<Dimensions,double>& x,
  ...
  int normal,
  double* BgradQ // out
 ) override;
...
};
```

Now you are all set for you simulations. A second call of python would build the executable, which is compatible with from student laptops to exascale clusters.

User-defined Features

$$\frac{\partial}{\partial t}Q + \boldsymbol{\nabla}\cdot\mathbf{F}(Q) + \sum_i \boldsymbol{\mathcal{B}}_i \frac{\partial Q}{\partial x_i} = \mathbf{S}(Q).$$

- Adding numerical schemes
  - new numerical schemes can be incorporated in a module approach
  - First-order formulation or Second-order with auxiliary variables
- Control flow manipulation
  - e.g., add extra mesh traversal within timestep and call another solver
- Multiple Solver Coupling
  - different kernel updated simultaneously
  - restriction and interaction of solutions can be coded
    - e.g., use a FV solver to limit FD4 solver [1]

[1] Michael Dumbser, et al *(2018)*

User-defined Features

- Postprocessing within mesh traversals
  - add extra manipulation on compute data
  - e.g. impose algebraic constraints, calculate accuracy metric (H and M)
  - imposed with “one-line” via python API
- Particles
  - can be served as static data probes or moving field tracers
  - also in more “physics approach”: matter representer and interactions(WIP)
- Load-balancing and Performance
  - Users can specify the octree parameters or even write their own load-balancing strategies.

Outline

- ExaHyPE 2 Engine
  - Design and Architecture
  - Data Structure and Decomposition
  - Python API and Usage
  - User-defined Features
- ExaGRyPE: numerical relativity on ExaHyPE 2
  - Physics Remarks
  - Implementation
  - Preliminary Results

ExaHyPE An Exascale Hyperbolic PDE Engine
Durham University Department of Computer Science

ExaGRyPE: Physics Remarks

$$\frac{\partial}{\partial t}Q + \boldsymbol{\nabla}\cdot\mathbf{F}(Q) + \sum_i \boldsymbol{\mathcal{B}}_i \frac{\partial Q}{\partial x_i} = \mathbf{S}(Q).$$

- Current version on black hole spacetimes
- CCZ4[1] system under 3+1 foliations
  - Implemented both in First-order and Second-order formulations
  - Auxiliary variables are introduced to degenerate the system into FO

Second-order (24 primary variables)

$$\vec{Q}(t) = \left(\tilde{\gamma}_{ij}, \alpha, \beta^i, \phi, \tilde{A}_{ij}, K, \Theta, \hat{\Gamma}^i, b^i\right)$$

\+

Auxiliary variables (34 variables)

$$A_i := \partial_i\alpha,\ B_k^i := \partial_k\beta^i,\ D_{kij} := \frac{1}{2}\partial_k\tilde{\gamma}_{ij},\ P_i := \partial_i\phi$$

$$\vec{Q}(t) = \left(\tilde{\gamma}_{ij}, \alpha, \beta^i, \phi, \tilde{A}_{ij}, K, \Theta, \hat{\Gamma}^i, b^i, A_k, B_k^i, D_{kij}, P_k\right).$$

First-order (58 variables)

[1] Daniela Alic, Carles Bona-Casas, Carles Bona, Luciano Rezzolla, and Carlos Palenzuela *(2011)*

ExaGRyPE: Physics Remarks

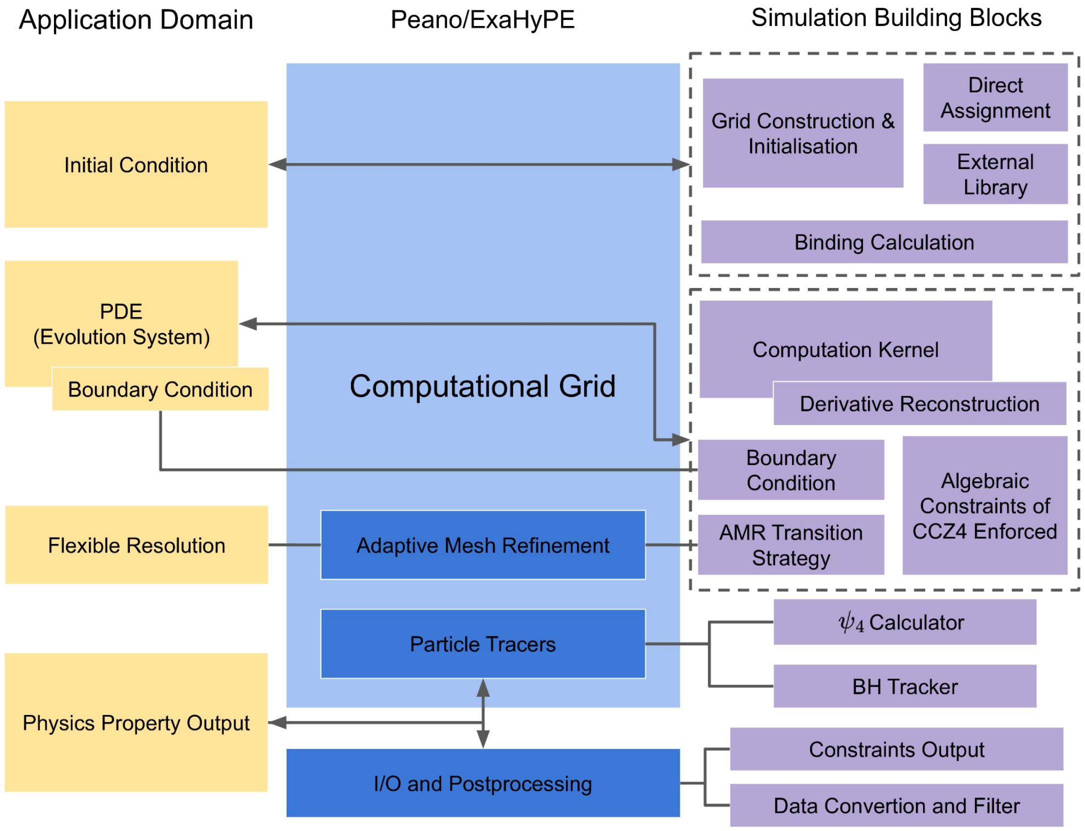
- Puncture Initial condition (from `EinsteinToolkit`[1])
- 1+log and Gamma-driven Gauges
- Boundary conditions
  - Periodic (e.g., Gauge waves)
  - Sommerfeld radiation condition[2]
- Newman-Penrose scalar[3] ψ_4 is calculated for Gravitational wave signal
- No matter is included (yet)

[1] *Frank Löffler, et al (2011)*

[2] *Miguel Alcubierre, Bernd Brugmann (2003)*

[3] *Ezra Newman, Roger Penrose (1962)*

Implementation



- Initial condition involves a call from external library and quantities conversion.
- fourth-order Finite difference (FD4) kernel as default.
- Derivative reconstruction called only in the Second-order formulation.
- Trilinear interpolation and averaging restriction
- Most of the feature utilizes the flexibility of ExaHyPE 2.

Exahype
An Exascale Hyperbolic PDE Engine

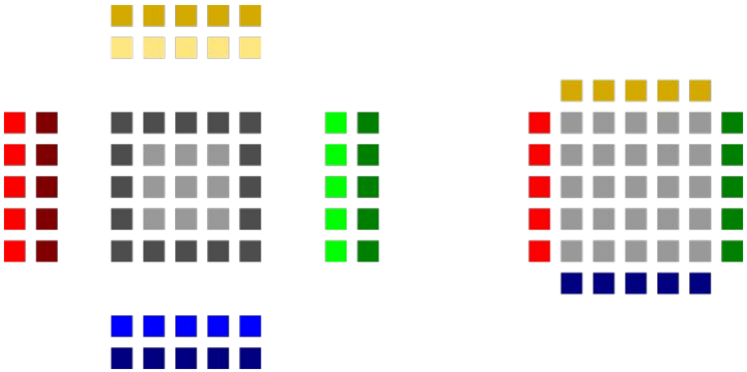
Durham University
Department of Computer Science

Implementation

FD4 kernel solver in second-order formulation

for each patch in Domain **do**
  for each volume in patch **do**
    compute Source $S(\vec{Q}_{pri})$
    $RHS_{\vec{Q}_{pri}} \leftarrow \Delta t \cdot S(\vec{Q}_{pri})$
    for $i = x, y, z$ **do**
      compute fourth-order FDs $(\Delta\vec{Q}_{pri})_i$
      compute NCP $B_i(\vec{Q}_{pri})(\Delta\vec{Q}_{pri})_i$
      compute KO term $(KO_{\vec{Q}_{pri}})_i$
    end for
    $RHS_{\vec{Q}_{pri}} \leftarrow RHS_{\vec{Q}_{pri}} - \sum_i \Delta t \cdot B_i(\vec{Q}_{pri})(\Delta\vec{Q}_{pri})_i$
    $RHS_{\vec{Q}_{pri}} \leftarrow RHS_{\vec{Q}_{pri}} + \sum_i \Delta t \cdot (KO_{\vec{Q}_{pri}})_i$
    $\vec{Q}_{pri} \leftarrow \vec{Q}_{pri} + RHS_{\vec{Q}_{pri}}$
    for $i = x, y, z$ **do**
      compute fourth-order FDs of the primary $(\Delta\vec{Q}_{pri})_i$
      Assign the auxiliary variables $\vec{Q}_{aux} \leftarrow (\Delta\vec{Q}_{pri})_i$
    end for
    $t \leftarrow t + \Delta t$
    compute λ_{max} to inform next time step
  end for
end for

$$\frac{\partial}{\partial t}Q + \nabla \cdot \mathbf{F}(Q) + \sum_i \boldsymbol{\mathcal{B}}_i \frac{\partial Q}{\partial x_i} = \mathbf{S}(Q).$$



The derivative in the patch halos need to be updated before(or after) the actual timestep, thus an extra mesh traversal is called responsible for derivative reconstruction.

Implementation

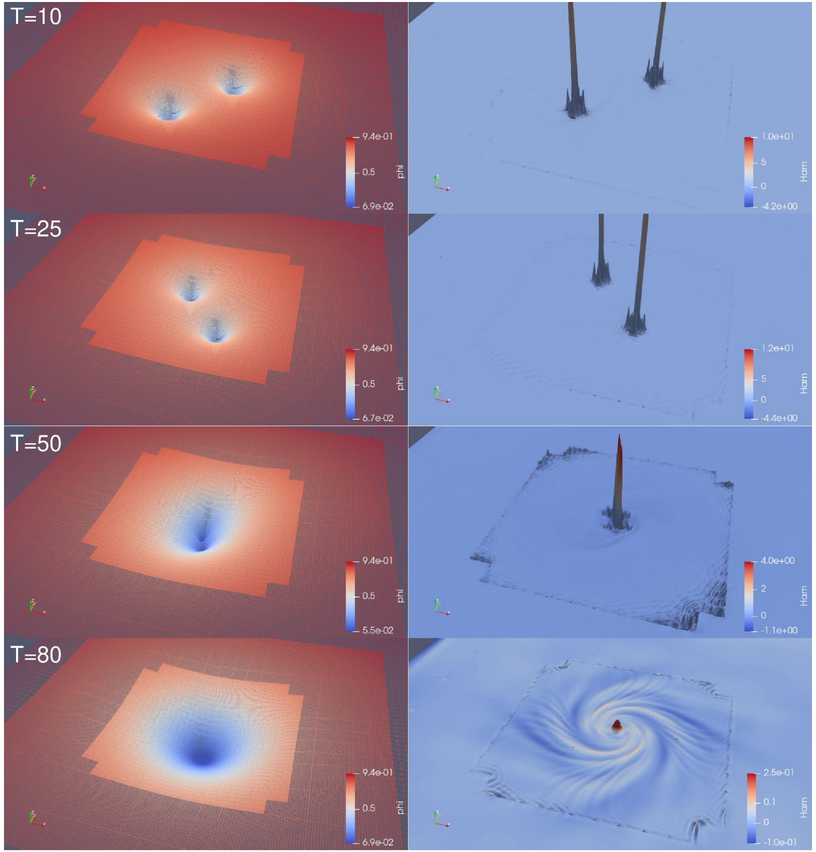
FD4 kernel solver in first-order formulation

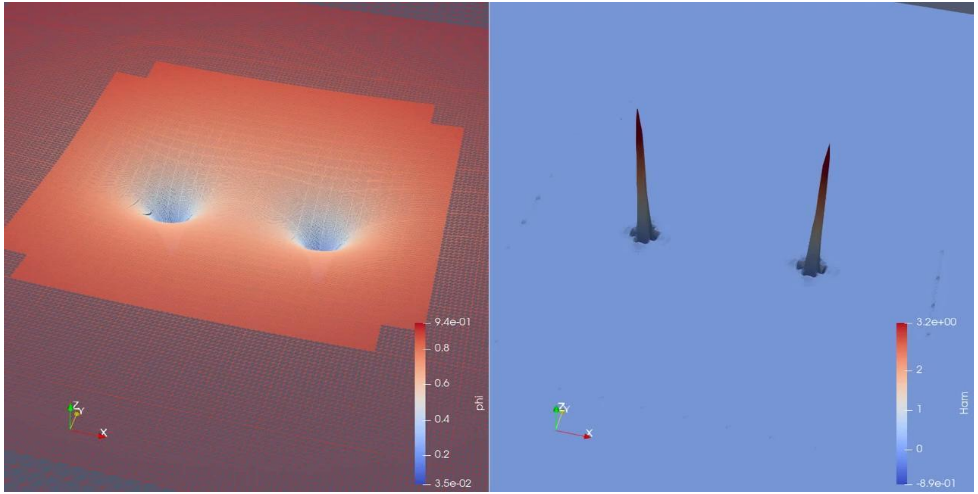
for each patch in Domain **do**
  for each volume in patch **do**
    compute source term $S(\vec{Q})$
    $RHS_{\vec{Q}} \leftarrow \Delta t \cdot S(\vec{Q})$
    for $i = x, y, z$ **do** ▷ Run over all elements in the patch
      compute fourth-order Finite Difference $(\Delta\vec{Q})_i$
      compute non-conservative product $B_i(\vec{Q})(\Delta\vec{Q})_i$
      compute KO term $(KO_{\vec{Q}})_i$
    end for
    $RHS_{\vec{Q}} \leftarrow RHS_{\vec{Q}} - \sum_i \Delta t \cdot B_i(\vec{Q})(\Delta\vec{Q})_i$
    $RHS_{\vec{Q}} \leftarrow RHS_{\vec{Q}} + \sum_i \Delta t \cdot (KO_{\vec{Q}})_i$
    $\vec{Q} \leftarrow \vec{Q} + RHS_{\vec{Q}}$
    $t \leftarrow t + \Delta t$
    compute λ_{max} to inform next time step
  end for
end for

The evolution equations for the auxiliary variables are from the Commutativity of Differentials

$$\partial_t \vec{Q}_{aux} = \partial_t(\partial_i \vec{Q}_{pri}) = \partial_i(\partial_t \vec{Q}_{pri})$$

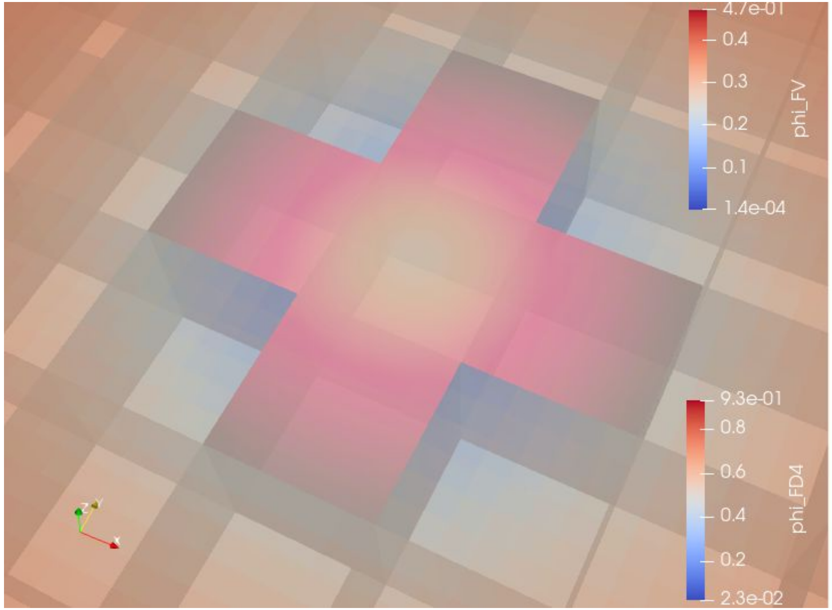
Preliminary Results

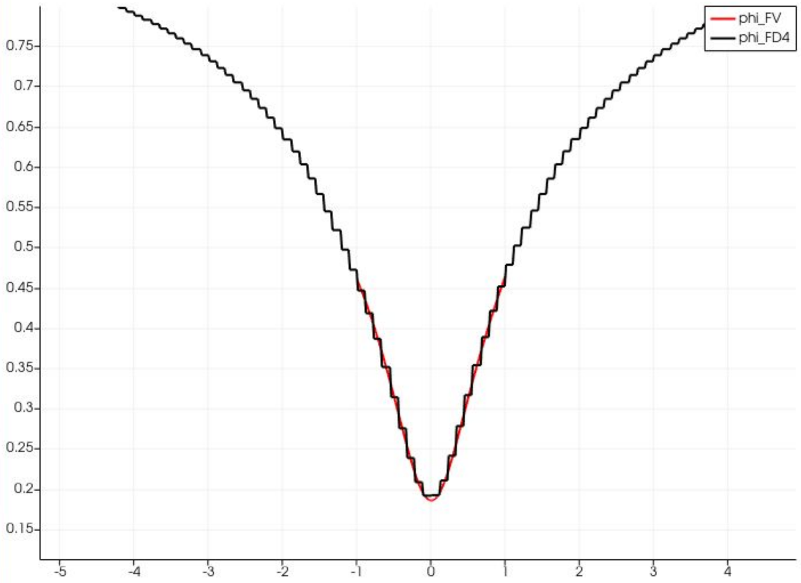




Preliminary Results

Solver Coupling





Summary

- ExaHyPE 2: a code Engine designed to implement hyperbolic PDEs system in a simple and efficient way
- Users only take care about sciences and engine handles the rest automatically
- Flexible module design allow advances users to tune the code quite freely
- completely open-source at https://gitlab.lrz.de/hpcsoftware/Peano

- ExaGRyPE: numerical relativity code on ExaHyPE 2
- work properly, but limited by current scalability bottleneck and AMR transition strategies (WIP, also GPU porting)
- collect various numerical schemes and ingredients allowing a thorough investigation on technical realization of numerical relativity
- Release paper under reviewing (CPC), and arxiv preprint: https://arxiv.org/abs/2406.11626