ChEESE

www.cheese-coe-eu

Center of Excellence for Exascale in Solid Earth

# Seismic Simulations using the ExaHyPE Engine

Leonhard Rannabauer
Anne Reinarz
**Technical University of Munich**
**Durham University**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823844

# The ExaHyPE Project

- EU Horizon 2020 project in the FETHPC call
  *"Towards Exascale High Performance Computing"*
  (New mathematical and algorithmic approaches)
- ExaHyPE has received followup funding through *ChEESE*
  The main objective of ChEESE is to establish a new Center of Excellence (CoE) in the domain of Solid Earth (SE) targeting the preparation of 10 Community flagship European codes for the upcoming pre-Exascale (2020) and Exascale (2022) supercomputers.

# People

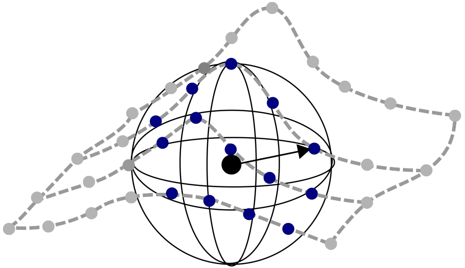
- **PI:** Michael Bader
- **Instructors:** Anne Reinarz, Jean-Matthieu Gallard, Leonhard Rannabauer, Philipp Samfass, Lukas Krenz






# Overview

1. The ExaHyPE Engine
2. The elastic wave equation
   - Curvilinear Meshes
   - Diffuse Interface Method
3. Perfectly Matched Layers
4. The GPR Model



# Towards an Exascale Hyperbolic PDE Engine

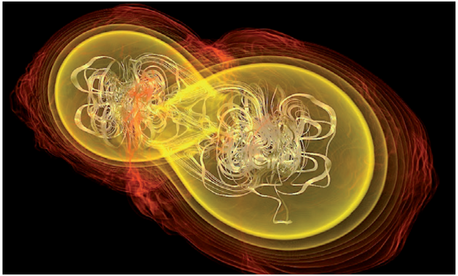
*ExaHyPE Goal:* a PDE "engine" (as in "game engine")

- enable medium-sized interdisciplinary research teams to realise extreme-scale simulations of grand challenges quickly
- efficiently solve hyperbolic PDE systems on Cartesian grids using higher-order ADER DG schemes with subcell limiting

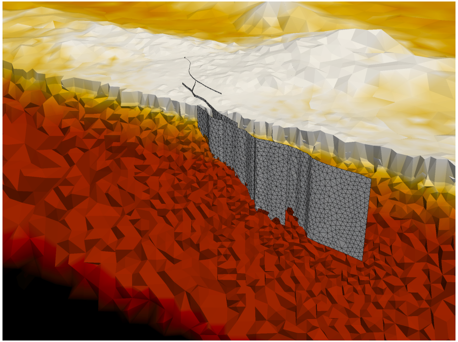
# Towards an Exascale Hyperbolic PDE Engine

- primary focus on two application areas:

**Astrophysics**



**Seismology**



# Hyperbolic PDE systems

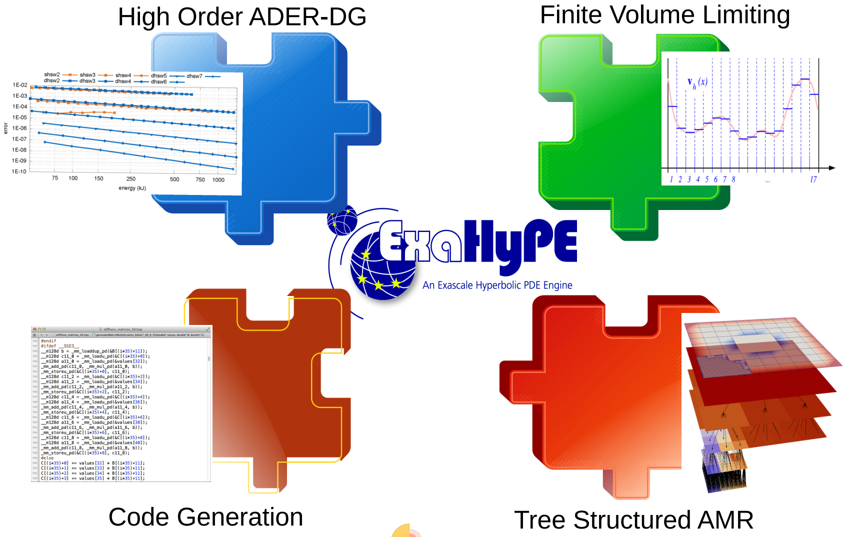
The ExaHyPE Engine solves systems of first-order hyperbolic PDEs in the following form:

$$\mathrm{P}\frac{\partial \mathrm{Q}}{\partial t} + \nabla \cdot \mathrm{F}(\mathrm{Q}) + \sum_{i=1}^{d} \mathrm{B}_i(\mathrm{Q})\frac{\partial \mathrm{Q}}{\partial x_i} = \mathrm{S}(\mathrm{Q}) + \sum \delta,$$

with

- material matrix P
- state vector Q
- conserved flux vector F
- non-conservative fluxes $\sum \mathrm{B}_i(\mathrm{Q})\frac{\partial \mathrm{Q}}{\partial x_i}$
- algebraic source terms S
- point sources $\sum \delta$

# The ExaHyPE Engine



# Engine Architecture and Application Interface

**Application Layer** – user provides:

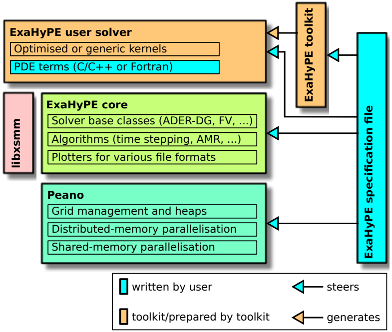
- C/Fortran code for fluxes: $F(Q)$, $G(Q)$, etc.
- C/Fortran code for eigenvalues: $\lambda_1 = u + \sqrt{gh}$, etc.

**ExaHyPE toolkit** generates:

- core routines, templates for application-specific functions
- kernels tailored to discretisation order, number of quantities, etc.

**Peano framework**:

- hybrid MPI+Intel TBB parallelism
- data structures for parallel AMR

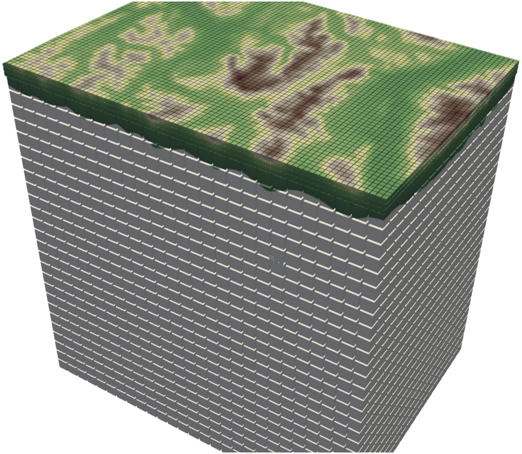


# Available Equations

*The Flexibility* of the Engine allows the implementation of highly different PDE systems:

- Euler Equations
- Tsunamis with the Shallow Water Equations
- **Curvilinear Meshes for the Elastic Wave Equation**
- **Diffuse Interface Approach**
- **Perfectly Matching Layers for the Elastic Wave Equation (PML)**
- Clouds with the Compressible Navier-Stokes Equations
- General Relativistic Magneto-Hydrodynamics
- **Godunov-Peshkov-Romenski (GPR) Model**
- Gravitational waves with the Einstein's Equations in Vacuum

# Elastic Wave Equation

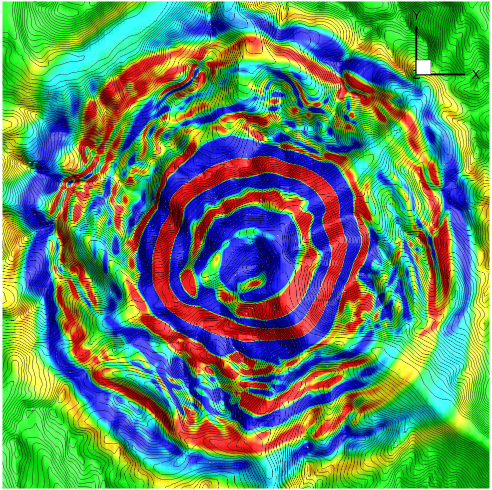


*Note:* This is a linear equation

$$\frac{\partial \boldsymbol{\sigma}}{\partial t} - \mathrm{E}(\lambda, \mu) \cdot \nabla \vec{\boldsymbol{v}} = 0,$$
$$\frac{\partial \rho \boldsymbol{v}}{\partial t} - \nabla \cdot \boldsymbol{\sigma} = 0.$$

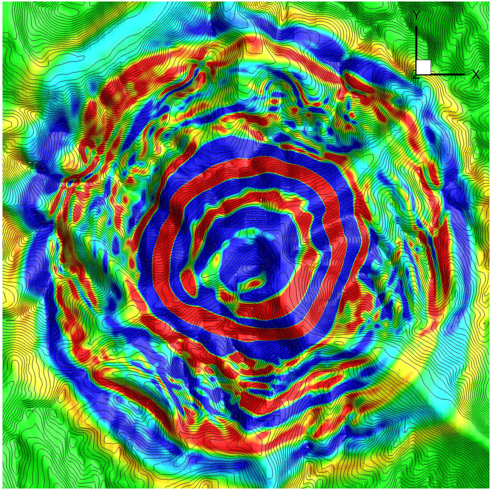
- $\mathrm{E}(\lambda, \mu)$ is a matrix depending on the two Lamé constants $\lambda$ and $\mu$
- $\rho$ is the mass density
- $\boldsymbol{\sigma} \in \mathbb{R}^d \times \mathbb{R}^d$ the stress tensor
- v is the velocity field

# Motivation




- Mesh generation traditionally requires a large fraction of the time in simulations, both in terms of run time and set up time
- Meshing often requires commercial software
- *Example:* topography and fault profile → CAD model → mesh generator
- *Goal:* Require only topography and fault profile to initialize the simulation

# Motivation

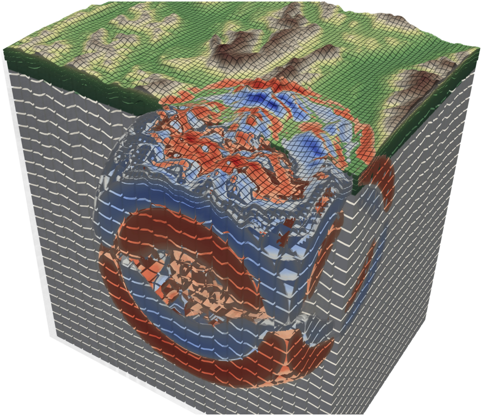



- Mesh generation traditionally requires a large fraction of the time in simulations, both in terms of run time and set up time
- Meshing often requires commercial software
- *Example:* topography and fault profile → CAD model → mesh generator
- *Goal:* Require only topography and fault profile to initialize the simulation

**Two approaches**:
*Curvilinear Meshes* with automated mesh generation.
*Diffuse Interface Approach* avoiding mesh generation.

# Curvilinear Meshes

K. Duru and L. Rannabauer



- Maps each element from Cartesian mesh onto a boundary fitting curvilinear mesh.
- Requires initial (automated) Mesh generation.
- Flux and source terms are transformed with the Jacobian.
- Eigenvalues and time-step size highly depend on the norm of the transformation.

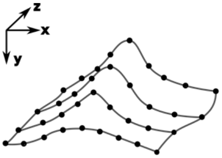
# Curvilinear Meshes

Modification to the flux

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

$$\frac{\partial v}{\partial x} = q_x\frac{\partial v}{\partial q} + r_x\frac{\partial v}{\partial r} + s_x\frac{\partial v}{\partial s}$$

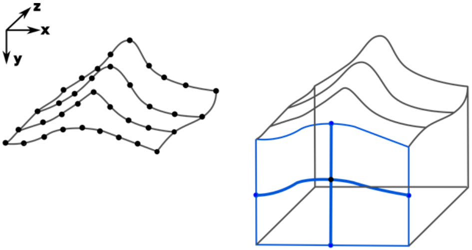
# Curvilinear Meshes




1. Generate surface quadrature nodes depending on topography (Interpolation with the easi[1] library).

[1] github.com/SeisSol/easi

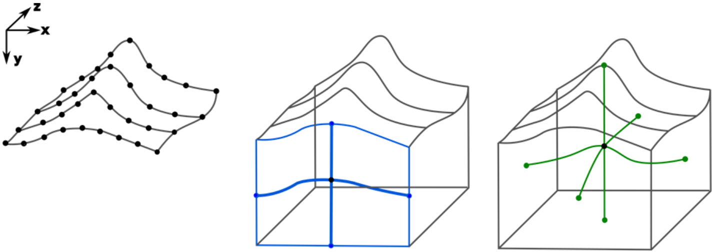
# Curvilinear Meshes




1 Generate surface quadrature nodes depending on topography (Interpolation with the easi[1] library).
2 2D curvilinear interpolation of quadrature nodes on domain boundaries with topography curves and domain edges as constraints.

[1] github.com/SeisSol/easi

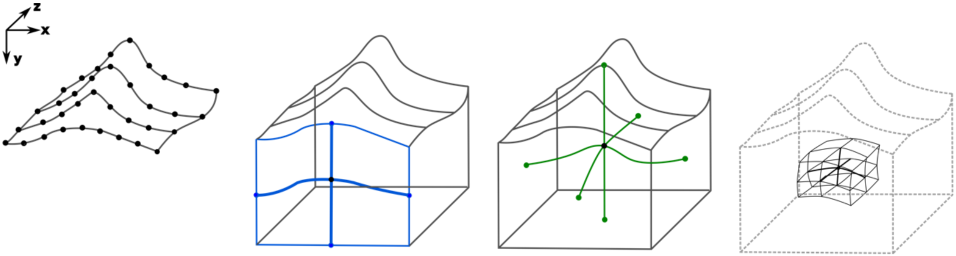
# Curvilinear Meshes




1. Generate surface quadrature nodes depending on topography (Interpolation with the easi[1] library).
2. 2D curvilinear interpolation of quadrature nodes on domain boundaries with topography curves and domain edges as constraints.
3. 3D curvilinear interpolation of all volume quadrature nodes with boundary faces and topography surface as constraints.

[1] github.com/SeisSol/easi

# Curvilinear Meshes

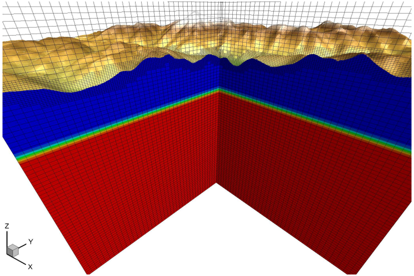



1. Generate surface quadrature nodes depending on topography (Interpolation with the easi[1] library).
2. 2D curvilinear interpolation of quadrature nodes on domain boundaries with topography curves and domain edges as constraints.
3. 3D curvilinear interpolation of all volume quadrature nodes with boundary faces and topography surface as constraints.
4. From the whole transformation of an element we can generate the Jacobian in each node.

[1] github.com/SeisSol/easi

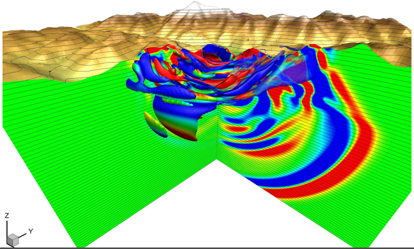
# Diffuse Interface Approach

M. Tavelli and M. Dumbser



*Idea:* Introduce a parameter $\alpha$, which identifies the location of solid medium

$$Q = \begin{pmatrix} \sigma & \alpha \mathbf{v} & \lambda & \mu & \rho & \alpha \end{pmatrix}^T,$$
$$\partial_t \alpha = \partial_t \lambda = \partial_t \rho = \partial_t \mu = 0$$



- At boundaries fluxes are *no longer linear*.
- This new approach *completely avoids* the problem of mesh generation
- The eigenvalues and time-step size are *independent* from the topography.
- Allows moving meshes

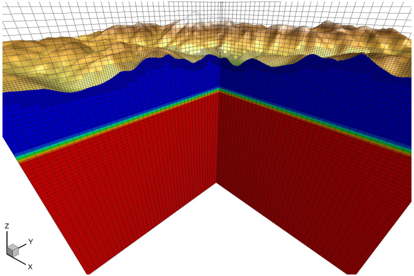
# Diffuse Interface Approach

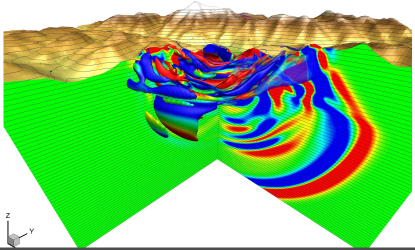
M. Tavelli and M. Dumbser

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

$$\frac{\partial \alpha \boldsymbol{v}}{\partial t} - \frac{\alpha}{\rho}\nabla\cdot\boldsymbol{\sigma} - \frac{1}{\rho}\sigma\nabla\alpha = 0,$$

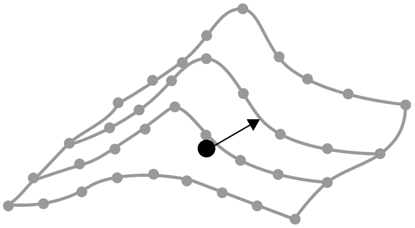
- At boundaries fluxes are *no longer linear*.
- $\alpha$ introduces a *discontinuity* at the topography which needs to be limited.
- This new approach *completely avoids* the problem of mesh generation
- The eigenvalues and time-step size are *independent* from the topography.





# Diffuse Interface Approach

Mesh initialization reduces to finding $\alpha$.
*But how do we find* $\alpha$*?*



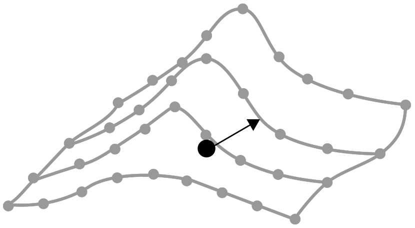
# Diffuse Interface Approach

Mesh initialization reduces to finding $\alpha$.
*But how do we find $\alpha$?*
For the free-surface boundary condition we require:

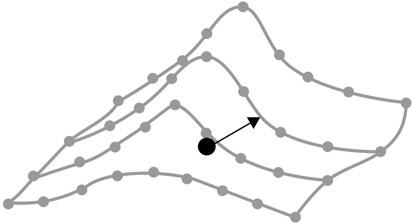
$$\nabla \alpha(t) \stackrel{!}{=} \vec{n_t},$$

where $t$ is an arbitrary point on the topography and $\vec{n_t}$ is normal.
Interpolation of the surface ends up with a non-linear optimisation problem:

$$\alpha(\vec{x}) = f(d(\vec{x}))$$



# Diffuse Interface Approach



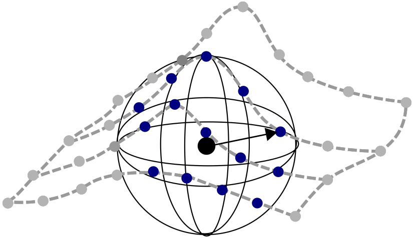
Mesh initialization reduces to finding $\alpha$.
*But how do we find $\alpha$?*
For the free-surface boundary condition we require:

$$\nabla\alpha(t) \stackrel{!}{=} \vec{n}_t,$$

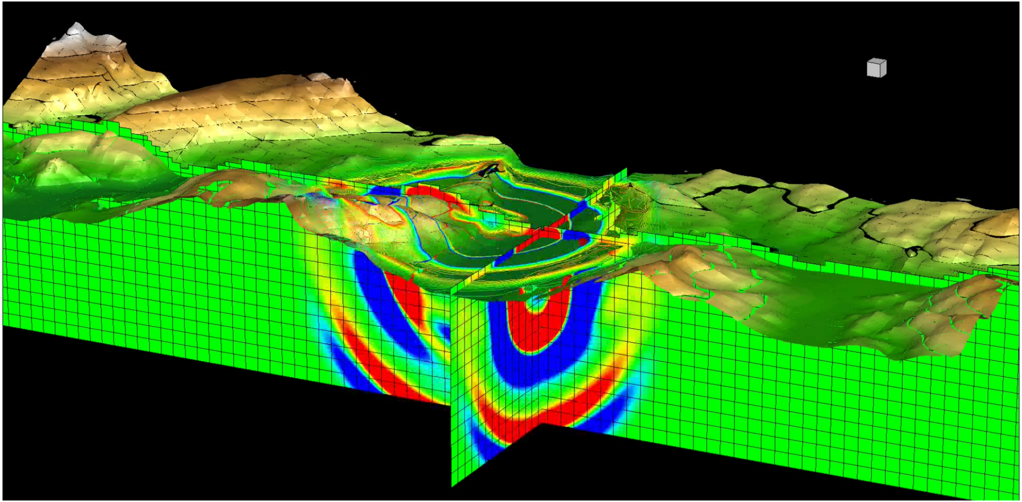
where $t$ is an arbitrary point on the topography and $\vec{n}_t$ is normal.
Interpolation of the surface ends up with a non-linear optimisation problem:

$$\alpha(\vec{x}) = f(d(\vec{x}))$$

$\rightarrow$ Approximation by only considering samples of the topography.



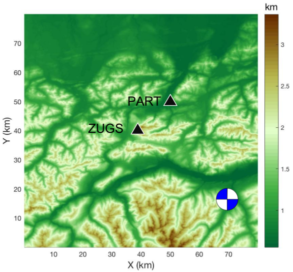
# Diffuse Interface Approach

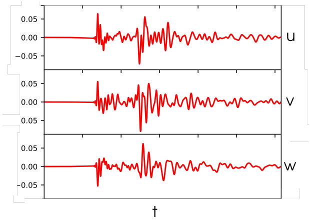


Uses open access topography data by Earth Observation Center (EOC), project Copernicus

# Studies of the Alpine area near Zugspitze

K. Duru and L. Rannabauer



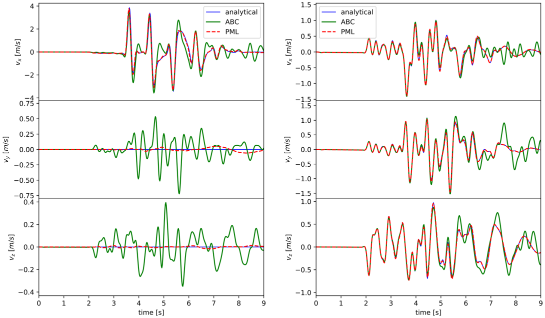




*Goal:* Track topographic effects on wave scattering.

- Time-steps of the DIM are larger by a factor of $\approx 16$ to 64
- Implies a point at which the additional cost for the DIM is evened out by requiring less time-steps.
- *Question:* What accuracy do we get for each method?

# Perfectly Matched Layers

K. Duru and L. Rannabauer

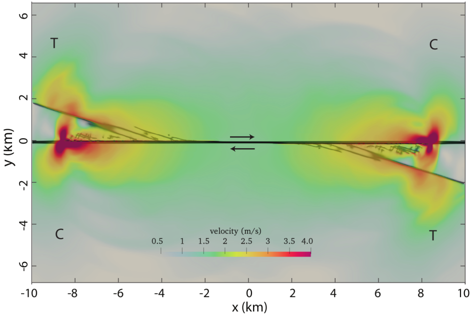



*Goal:* Remove reflections emanating from the not perfectly absorbing boundary of the computational domain.

- Based on complex coordinate stretching.
- Requires extension of the numerical DG fluxes, inter-element and boundary procedures.
- Allow us reduce the the computational domain and only simulate the area of interest.

# GPR: Godunov, Peshkov and Romenski model

AA. Gabriel, D. Li




*Goal: Numerical modeling of continuous damage and freely evolving dynamic rupture.*

- Based on the Godunov Peshkov Romenski, a unified framework for arbitrary rheological responses of material.
- Used for nonlinear elasto-plasticity, material damage and of viscous Newtonian flows with phase transition between solid and liquid phases.
- Fault geometry and secondary cracks are part of the PDE.
- A scalar function $\xi \in [0, 1]$ indicates the local level of material damage.

# Challenges for Engine Development:

- lots of functionality to be tested, high effort for software integration.
- "multiple targets" for parallelisation and optimisation.
- equal number of cells does not lead to equal execution time.

Thus,

- in ExaHyPE we use a *task-based paradigm* for unpredictable work loads.
- tasks processing is build on a *produce-consumer pattern*. We assume volume operations are significantly more expensive than boundary operations (Prediction vs Riemann-solver).
- strategy for AMR: different granularity of AMR required by applications
- *communication-avoiding traversal scheme* that minimizes data transfer.
- *code generation* tailored to required PDE kernels.

# Access to the Engine:

- snapshots of the engine, documentation, etc
  `www.exahype.org`
- webpage that comprises statistics, galleries, publication lists, etc.
  `exahype.eu`

# References

[1] The ExaHyPE consortium. The ExaHyPE Guidebook. www.exahype.eu

[2] Reinarz et al. ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems. Computer Physics Communications. 2020.

[3] Tavelli et al. A simple diffuse interface approach on adaptive Cartesian grids for the linear elastic wave equations with complex topography. Journal of Computational Physics 386.

[4] Duru et al. A stable discontinuous Galerkin method for the perfectly matched layer for elastodynamics in first order form. Submitted 2019.