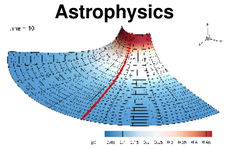
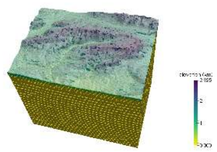
Engine Design

Towards an Exascale *PDE Engine*

Astrophysics



Seismology



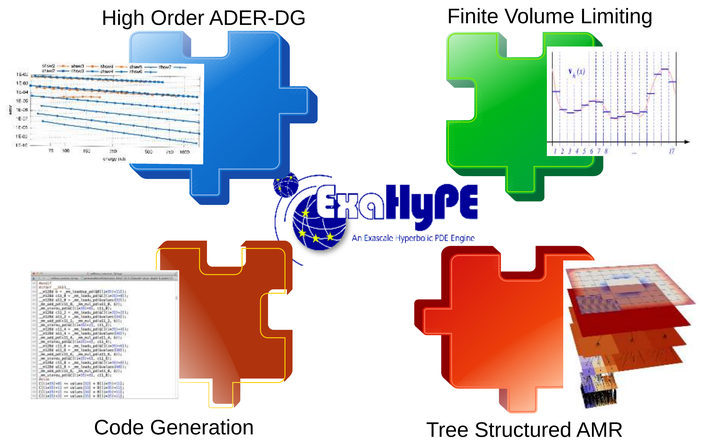
ExaHyPE Goal:

- 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

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

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

with
- material matrix $\mathbf{P}$
- conserved flux vector $\mathbf{F}$
- state vector $\mathbf{Q}$
- non-conservative fluxes
- point sources $\sum \delta$
- algebraic source terms $\mathbf{S}$



ADER DG

The computational domain $\Omega \subset \mathbb{R}^d$ with $d = 2, 3$ is discretised with a cartesian grid.

1. Insert the DG ansatz function $u_h(x, t^n) = \sum_\ell \phi_\ell(x)\hat{u}_\ell^n$ into equation (1) and multiply with a test function ϕ_k from the space of piecewise polynomias of degree N
2. integrate over a space-time control volume $T_i \times [t^n, t^{n+1}]$
   - for surface integrals introduce a classical Riemann solver as its used in Godunov-type FV schemes
   - The non-conservative product on the element boundaries is discretized via a path-conservative jump term

Finite Volume Limiting

The new approach followed in ExaHyPE extends the successful a posteriori MOOD method of Loubère et al. also to the DG-FEM framework.
As very simple a posteriori detection criteria we use

- A relaxed discrete maximum principle (DMP) in the sense of polynomials, absence of floating point errors (NaN)
- Positivity of the solution, i.e. physical constraints on the solution

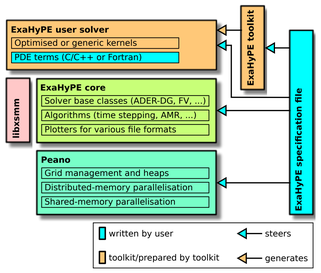
If one of these criteria is violated after a time step, the scheme goes back to the old time step and recomputes the solution in the troubled cells, using a more robust high resolution shock capturing FV scheme subcells

- Limiter can be interpreted as element-local checkpointing and restarting of the solver with a more robust scheme on a fine subgrid.
- Method is by construction positivity preserving, if and only if the FV scheme is positive

API

How do you create code that is easy to use and extend without losing flexibility and efficiency?

Using code generation to generate efficient programs tailored to the hardware and application.



To write an ExaHyPE code

1. Start from a specification file defining the domain, PDE system and required architecture
2. From this the ExaHyPE toolkit creates glue code, empty application-specific classes and optionally application and architecture tailored core routines
3. Fill the empty application classes with domain specific code that sets up the PDE system being solved

The ExaHyPE Engine comprises the following key features

- 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 optimised ADER-DG routines
- Distributed memory parallelisation with MPI
- Shared memory parallelisation through Intel's Threading Building Blocks (TBB)

Creating an Application

```
exahype-project EulerFV
 peano-kernel-path const = ./Peano
 exahype-path const      = ./ExaHyPE
 output-directory const  = ./Euler

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

 solver Finite-Volumes MyEulerSolver
   variables const    = rho:1,j:3,E:1
   patch-size const   = 10
   maximum-mesh-size  = 2e-2
   time-stepping      = global
   type const         = godunov
   terms const        = flux
   optimisation const = generic
   language const     = C
 end solver
end exahype-project
```

specification file

→

```
void EulerFV::MyEulerSolver
              ::adjustPointSolution(
     const double* const x,
     const double t,const double dt,
     double* Q) {
 // @todo Please implement
}
```

empty application class

↓

```
void EulerFV::MyEulerSolver
              ::adjustPointSolution(
     const double* const x,
     const double t,const double dt,
     double* Q) {
 Variables vars(Q);
 double energy = vars.E();
 getInitialProfile(x,energy,t,dt);
 vars.E() = energy;

 double density = vars.rho();
 getInitialProfile(x,density,t,dt);
 vars.rho() = density;
}
```

completed initial conditions

↓

Code Optimisation and Generation

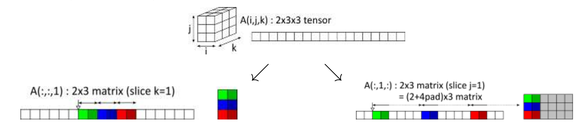
How to make a user friendly code fast?

In ExaHyPE's API the code generator lies between the toolkit and the kernels

- Automatically called by the toolkit *if* required by the specification file without changes required by the user
- Generates kernels tailored to the hardware and application

The ExaHyPE code generator uses:

- Jinja2 a python template engine
- and LIBXSMM a library for small matrix-matrix products, see A. Heinecke, G. Henry, M. Hutchinson, H. Pabst, 2016



Extracting matrix slices consisting of 1 space dimension and the dimension of quantities from a (2+1)-dimensional tensor.

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

References:

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

[2] M. Dumbser, F. Guercilena, S. Köppel, L. Rezzolla, O. Zanotti: A strongly hyper- bolic first-order CCZ4 formulation of the Einstein equations and its solution with discontinuous Galerkin schemes (2017) Physical Review D

[3] D.E. Charrier and T. Weinzierl, Stop talking to me – a communication avoiding ADER-DG realisation.