Role-Oriented Code Generation in ExaHyPE

M. Bader, J.-M. Gallard, L. Rannabauer (Technical University of Munich)
A. Reinarz, T. Weinzierl (Durham University)

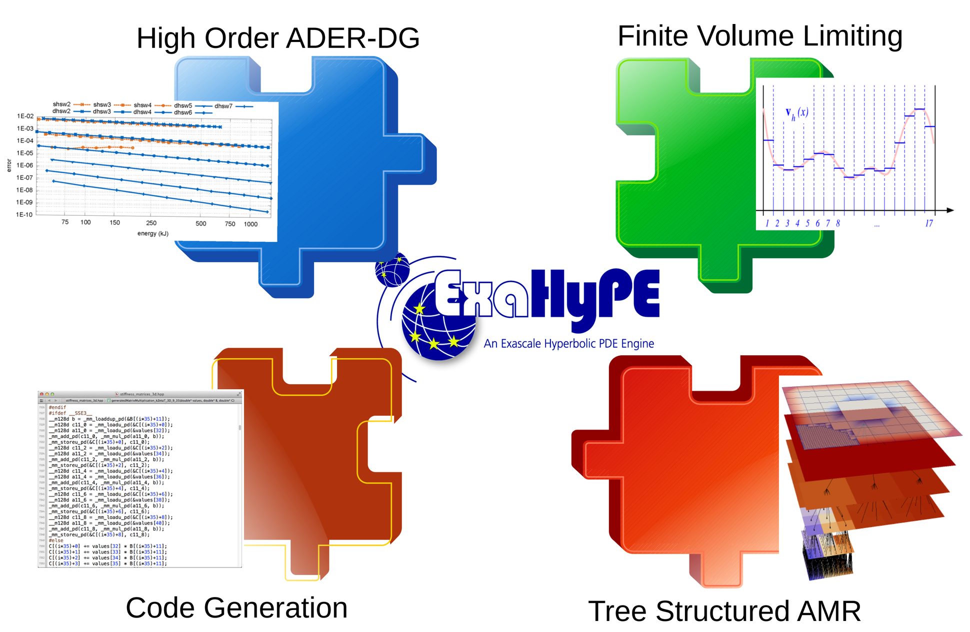
Towards an Exascale *PDE Engine*

ExaHyPE [1] is designed to enable medium-sized interdisciplinary research teams to quickly realise extreme-scale simulations of grand challenges.

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

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

ExaHyPE employs higher-order ADER-DG on tree-structured adaptive Cartesian grids using a-posteriori subcell Finite-Volume limiting [4]:



"What's an Engine?"

Similar to a "game engine", we aim for efficient core functionality but also application flexibility:

- **fixed parallel AMR framework**: Peano [3] (tree-structured adaptive Cartesian grids; MPI+Tasking parallelism, load balancing) → www.peano-framework.org
- **fixed numerics**: high-order discontinuous Galerkin with ADER time-stepping (ADER-DG) with a-posteriori Finite-Volume subcell limiting
- **flexible w.r.t. applications**: hyperbolic PDEs stemming from conservation laws

Code generation is our means to manage software complexity.

Role-Oriented Code Generation:

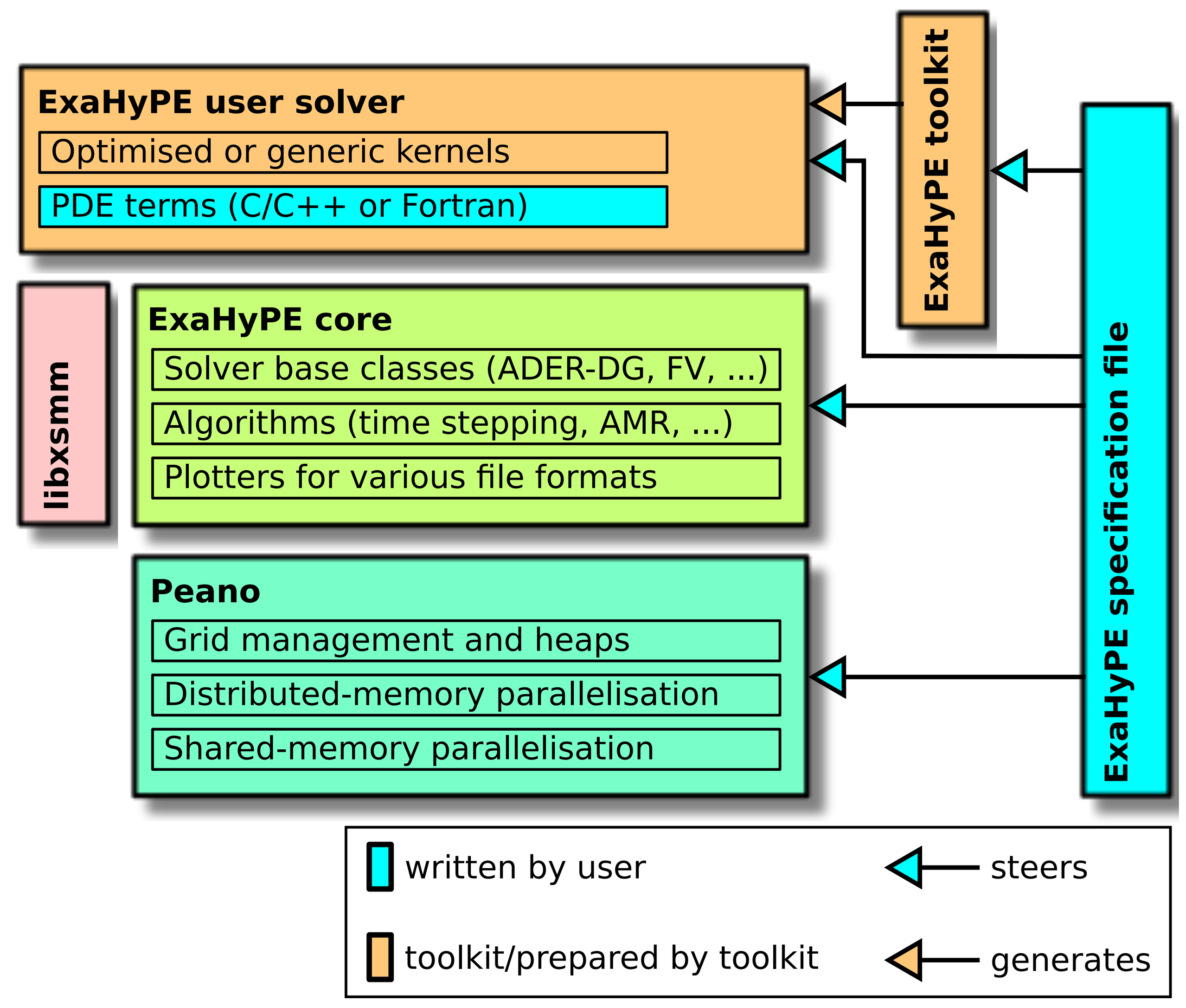
We have observed the following roles for software development on the engine and on its applications:

- **application expert(s)**: implements the PDE system, problem-specific initial/boundary conditions, etc., for a given application; desires straightforward user API that hides complexity of solver and optimisation
- **algorithms expert(s)**: implements efficient numerical schemes; shall design architecture-oblivious algorithms via custom macros that isolate low-level optimisation
- **optimisation expert(s)**: performs hardware-aware optimisation on performance-critical components of the solver – relies on abstractions by algorithmic templates.

Any role might be adopted by multiple users.
Any user may adopt multiple roles.

ExaHyPE's *Toolkit* and *Code Generator* [2] thus provide separate views for each role.
Toolkit and Code Generator are stand-alone applications based on the Jinja2 templating engine.

How to Create Code that is Easy to Use & Extend, Flexible, Efficient, . . . ?



Using the ExaHyPE Toolkit:

1. create a specification file that defines the domain, PDE system, required architecture, parallelisation, etc.
2. ExaHyPE toolkit creates glue code, application-specific template classes and core routines (tailored to application and architecture)
3. implement the application classes with PDE- and scenario-specific methods:
   - `flux(...)`, `ncp(...)`, ... for PDE terms (conservative fluxes, non-conservative products, etc.)
   - `eigenvalues(...)` to compute eigenvalues (for Riemann solvers)
   - `boundaryValues(...)`, etc.

Jinjia2 Templates and Model-View-Controller Design

ExaHyPE Toolkit and Code Generator follow a Model-View-Controller Design – e.g., for the Toolkit:

- **Controller**: builds multiple contexts from the specification file, such as type of PDE, choice of numerical solver, architecture, etc.
- **Model**: responsible for generating a specific View – e.g., generate the glue code for either a finite volume solver or an ADER-DG solver
- **View**: Jinja2 template engine is invoked to render templates that are tailored to Model-provided contexts.

Jinjia2 templates allow "logic" in the code representation, while keeping it close to the generated code and easily readable and expandable. For example

```
{% if initA %}
{{allocateArray('A', nDof)}}
for(int i=0; i<{{nDof}}; ++i) {
  A[i] = B[i+{{nDof*nVar}}] * {{C}}[i];
}
{% endif %}
```

may generate the following code:

```
double A[5] __attribute__((aligned(32)));
for(int i=0; i<5; ++i) {
  A[i] = B[i+20] * foo[i]
}
```

Creating an ExaHyPE Application: View for the Application Expert

Specification file:

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

  computational-domain
    dimension const     = 3
    width               = 1.0, 1.0, 1.0
    offset              = 0.0, 0.0, 0.0
    end-time            = 1.0
  end computational-domain

  solver ADER-DG ElasticWaveSolver
    variables const   = v:3,sigma:6
    parameters const  = rho:1,cp:1,cs:1
    order const       = 7
    maximum-mesh-size  = 2e-2
    maximum-mesh-depth = 2
    time-stepping      = global
    terms const = flux,ncp,
     material_parameters,point_sources
    optimisation const = optimised
    language const     = C
    basis              = Lobatto
  end solver
end exahype-project
```

Implementation of flux function:

```
void Elastic::ElasticWaveSolver
  ::flux(const double* const Q,
         double** const F) {
  VariableShortcuts s;
  double sigma_xx=Q[s.sigma + 0];
  double sigma_yy=Q[s.sigma + 1];
  double sigma_zz=Q[s.sigma + 2];
  double sigma_xy=Q[s.sigma + 3];
  double sigma_xz=Q[s.sigma + 4];
  double sigma_yz=Q[s.sigma + 5];
  F[0][ s.v + 0] = -sigma_xx;
  F[0][ s.v + 1] = -sigma_xy;
  F[0][ s.v + 2] = -sigma_xz;
  F[1][ s.v + 0] = -sigma_xy;
  F[1][ s.v + 1] = -sigma_yy;
  F[1][ s.v + 2] = -sigma_yz;
  F[2][ s.v + 0] = -sigma_xz;
  F[2][ s.v + 1] = -sigma_yz;
  F[2][ s.v + 2] = -sigma_zz;
}
```

References

[1] A. Reinarz et al.: *ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems.* Comp. Phys. Comm. 254, 2020. http://dx.doi.org/10.1016/j.cpc.2020.107251

[2] J.-M. Gallard et al.: *Role-oriented code generation in an engine for solving hyperbolic PDE systems.* 2019 Int. Workshop on Softw. Eng. for HPC-Enabled Research (SE-HER), SC19.

[3] T. Weinzierl: *The Peano software—parallel, automaton-based, dynamically adaptive grid traversals.* ACM Trans. Math. Softw. 45(2): 14, 2019.

[4] O. Zanotti, F. Fambri, M. Dumbser, A. Hidalgo: *Space-time adaptive ADER discontinuous Galerkin finite element schemes with a posteriori sub-cell finite volume limiting.* Computers & Fluids 118, 2015, p. 204–224.

Download the ExaHyPE engine from: www.ExaHyPE.org

ExaHyPE was developed as a joint project of:

in particular by:

Dominic Charrier, Benjamin Hazelwood, Tobias Weinzierl (University of Durham), Michael Dumbser, Francesco Fambri, Maurizio Tavelli, Olindo Zannotti (University of Trento), Alice Gabriel, Kenneth Duru (Ludwig-Maximilians-University Munich), Luke Bovard, Sven Köppel, Luciano Rezzolla (Frankfurt Institute for Advanced Studies), Jean-Mathieu Gallard, Leonhard Rannabauer, Anne Reinarz, Philipp Samfaß, Angelika Schwarz and Vasco Varduhn (Technical University of Munich).
We thank the Leibniz Supercomputing Centre and the Russian Academy of Sciences for their support.