Towards an Exascale EDE Engine

ExaHyPE [1] is designed to enable medium-sized interdisciplinary research teams to quickly realize extreme-scale simulations of grand challenges. The ExaHyPE Engine solves systems of first-order hyperbolic PDEs of the form:

\[ \frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F} = 0 \]

ExaHyPE employs higher-order AED-DG on tree-structured adaptive Cartesian grids using a-priori/based Finite-Volume limiting [6].

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

Using the ExaHyPE Toolkit:

- Create a specification file that defines the domain, PDE system, required architecture, parallelisation, etc.
- ExaHyPE toolkit creates code, application-specific template classes and core routines (tailored application and architecture)
- Implement the application classes with PDE- and scenario-specific methods

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, etc.; for example, creates the glue code for either a finite volume solver or an AED-DG solver
- View: Jinjia2 template engine is invoked to render templates that are tailored to Model-provided contexts.

Creating an ExaHyPE Application: View for the Application Expert

Specification file:

```python
def Jinjia2_template_contexts(tesla_mem),order=2):
    return [0,1,2,3,4,5,6,7,8]

jinja2_contexts = Jinjia2_template_contexts(tesla_mem)
```

Implementation of flux function:

```flux```

```code```
```
```

Architecture-Oblivious Templates and Architecture-Aware Optimisation Macros

Using Jinjia2’s macros and variables, we can design architecture-oblivious algorithmic templates that are rendered by Jinjia2 with custom made architecture-aware optimisation macros. These include algorithm selection, selection of numerical schemes and/or level-architecture-aware optimisation separated and the role of algorithm and optimisation expert independent from one another. Example: tensor contraction to compute the x-component of the gradient of scalar tensor \( \frac{\partial S}{\partial x} \):

```java
def flux_tensor_contraction(S):
    M = 10
    for i in range(0, M):
        S[i][j][k] = S[i][j][k] + S[i][j][k]
```

Depending on the context — number of degrees of freedom (c dof), used architecture, etc. — the Code Generator resolves the template variables, using hardware/autotuning in the fix for the tensor size and/or mesh size, non conservative products, etc. (e.g., tensor contractions, eigenvalues for (Femion solvers) traverse/Filter (L\__filter)).

References


Role-Oriented Code Generation

We have observed the following rules 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.
- Desisdes straightforward API that hides complexity of solver and optimisation.
- Jinjia2 template(s) implements efficient numerical schemes that shall design architecture-oblivious algorithms via custom macros that tailor low-level optimisation.
- Jinjia2 template(s) implements thorough, automatic optimisation of performance-critical components of the solver — grids or abstractions by algorithmic templates.

Role Expert(s) may be adopted by multiple users. Any user may adopt more 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 Jinjia2 template engine.

Multi-Physics Dynamic Rupture

We simulate multi-physics spontaneous dynamic rupture, across complex fault geometries. The automated mesh generator allows to model fault structures, including, including, that is, a k-d tree. The rupture is incorporated as boundary condition, which we solve with a new developed physics based rupture solver. Our code is verified against community benchmarks (Paper: SSEC PVP28).

Acknowledgements

ExaHyPE is a joint development of:


Download the ExaHyPE engine from: https://www.ExaHyPE.org

ExaHyPE was developed as a joint project of:

- CHEESE PI Demonstration: Towards UG for Seismic Hazard Analysis
- We link ExaSeis—the collection of seismic wave simulation models, including our own and the community’s—via the ExaHyPE Toolkit to the uncertainty quantification toolbox for uncertainty quantification (muq.mit.edu) and do deep experimentation with novel UQ approaches to seismic hazard analysis.
- Towards UG for Seismic Hazard Analysis

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 628477 (ExaHyPE, agreement No 671168 (ExaSeis), and No 671168 (project ExaSeis, www.exaseis.eu).