Controlling Parallel CFD Simulations in Julia from C/Fortran with `libtrixi`

Gregor Gassner, Benedict Geihe (University of Cologne, Division of Mathematics)
Michael Schlottke-Lakemper (High-Performance Computing Center Stuttgart, University of Stuttgart)

What is it all about?

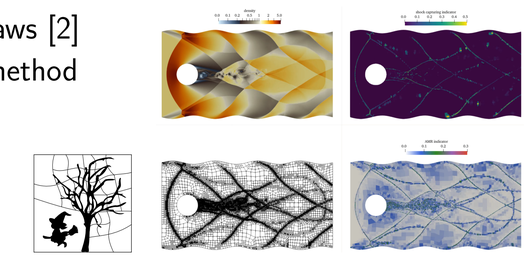
- Enhance legacy applications with modern programming concepts
- Keep original C/C++/Fortran code
- Add new features written in a high-level, user-friendly language
- Do not sacrifice performance
- Support heterogeneous HPC environments

Why Julia?

- Modern high-level programming language
- Lots of ready-to-use packages
- Dynamic typing, rapid prototyping capability
- Just-in-time compilation
- Solves two-language problem: easy to use and performant, HPC-ready

Who is that `Trixi.jl`?

- Numerical simulation framework for conservation laws [2]
- Adaptive high-order discontinuous Galerkin (DG) method
- Entropy stable and entropy conserving fluxes
- Focus: extensibility, usability, performance
- ∼20 main developers, ∼75k lines of code
- Performance on par with classical codes [3]



Who would possibly be using this?

- `MESSy`, the *Modular Earth Submodel System*, written in Fortran
- Project *ADAPTEX* (BMBF call *SCALEXA*):
  Adaptive Earth system modelling with strongly reduced computation time for exascale-supercomputers
- `Trixi.jl` as an alternative dynamical core, i. e. computational fluid dynamics (CFD) solver
- Makes `MESSy` exascale-ready, allows `MESSy` to leverage Julia's vendor-agnostic GPU support

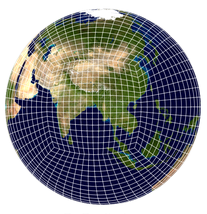
Can Fortran speak Julia?

Introduce `libtrixi`:

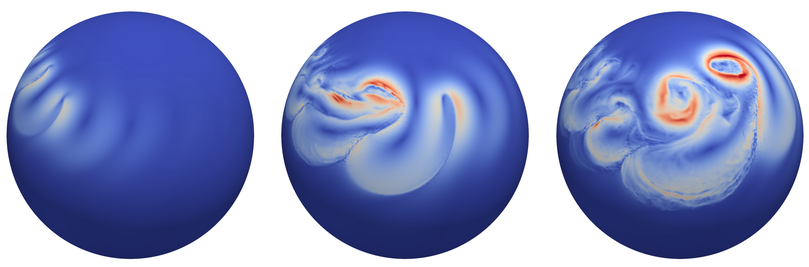
- `LibTrixi.jl`: Julia interface to `Trixi.jl`
- C API to `LibTrixi.jl`
- Fortran wrapper for C API

Does it work?

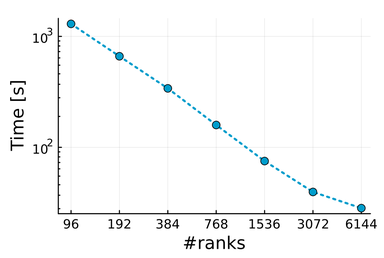
- Simulation of a baroclinic instability [1]
- Cubed sphere discretization of earth's atmosphere
- 3D compressible Euler equations
- Controlled from Fortran main program
- 98 304 mesh cells, 6^3 degrees of freedom each
- Computed on 6144 MPI ranks



E. Faulhaber



Velocity magnitude near surface after about 7.5, 10 and 12 days



Strong scaling (1h simulated time)

How fast is it?

- Benchmarks: 2D Sedov blast wave, 3D baroclinic instability (compressible Euler equations)
- Minimum runtime in seconds from 5 runs
- Negligible overhead for practical applications!

	`Trixi.jl`	`LibTrixi.jl`	`libtrixi`	
			classic	`PackageCompiler.jl`
2D	7.9	11.3	11.2	11.1
3D	327.0	332.0	329.0	334.0

How to use it?

- Build and install `libtrixi` using `cmake`
- Set up Julia runtime environment (`libtrixi` comes with a script!)
- Alternative: `PackageCompiler.jl`, skips C API, no runtime required
- Configure simulation through code (*elixir*)

```julia
using Trixi, LibTrixi
function init_simstate()
  equations = CompressibleEulerEquations2D(gamma = 1.4)
  solver = DGSEM(polydeg = 3, surface_flux = flux_lax_friedrichs)
  mesh = P4estMesh((8, 8), (-1.0, -1.0), (1.0, 1.0), polydeg = 3)
  my_init(x, t, equations) = ...
  semi = SemidiscretizationHyperbolic(mesh, equations, my_init, solver)
  ...
  # OrdinaryDiffEq
  integrator = init(ode, ...)
  return SimulationState(semi, integrator)
end
```

- Include header/module, link `libtrixi.so`, run

```
program trixi_controller_simple_f
  use LibTrixi
  ...
  ! Initialize Trixi
  call trixi_initialize(julia_runtime_dir)
  ! Set up the Trixi simulation
  handle = trixi_initialize_simulation(elixir)
  do
    if ( trixi_is_finished(handle) ) exit
    call trixi_step(handle)
  end do
  ! Finalize Trixi simulation
  call trixi_finalize_simulation(handle)
  ! Finalize Trixi
  call trixi_finalize()
end program
```

- Handle (integer) to address multiple simulations

How is it done?

- Fortran API: thin wrapper around C code

```
interface
  integer(c_int) function trixi_nelements(handle) bind(c)
    integer(c_int), value, intent(in) :: handle
  end function
end interface
```

- C API: Obtain, store, and call Julia function pointers
- Uses Julia C API

```c
int trixi_nelements(int handle) {
  const char* command = "@cfunction(trixi_nelements, Cint, (Cint,))";
  jl_value_t* res = jl_eval_string(command);
  int (*nelements)(int) = (void *) jl_unbox_voidpointer(res);
  return nelements(handle);
}
```

- Julia API: Get simulation state, call `Trixi.jl` function
- Retrieves non-integral Julia data

```julia
Base.@ccallable function trixi_nelements(simstate_handle::Cint)::Cint
    simstate = load_simstate(simstate_handle)
    _, _, solver, cache = mesh_equations_solver_cache(simstate.semi)
    return nelements(solver, cache)
end
```

What else?

Reproducibility

repository `libtrixi` `Trixi.jl` *ADAPTEX* `MESSy`

[1] C. Jablonowski and D. Williamson. "A baroclinic instability test case for atmospheric model dynamical cores". *Q. J. R. Meteorol. Soc.* (2006). DOI: https://doi.org/10.1256/qj.06.12.

[2] M. Schlottke-Lakemper et al. *Trixi.jl: Adaptive high-order numerical simulations of hyperbolic PDEs in Julia*. https://github.com/trixi-framework/Trixi.jl. 2020.

[3] H. Ranocha et al. "Efficient Implementation of Modern Entropy Stable and Kinetic Energy Preserving Discontinuous Galerkin Methods for Conservation Laws". *ACM Trans. Math. Softw.* (2023). DOI: 10.1145/3625559.

We gratefully acknowledge the resources granted through the ESM partition on the supercomputer JUWELS at the Jülich Supercomputing Centre, Germany!

SPONSORED BY THE
Federal Ministry of Education and Research

Funded by the European Union
NextGenerationEU

UNIVERSITY OF COLOGNE

H L R S