

Non-intrusive Multirate Time-Integration for High-Order accurate Compressible Fluid Dynamics with Trixi.jl

Daniel Doebring¹

in collaboration with

Michael Schlottke-Lakemper², Gregor Gassner³,
and Manuel Torrilhon¹

¹ACoM, RWTH Aachen University ²HPSC, University of Augsburg ³CDS, University of Cologne

2024/07/02



Applied and
Computational
Mathematics

RWTHAACHEN
UNIVERSITY

Outline

① About Trixi.jl

② Multirate Time-Integration with P-ERK

③ Applications

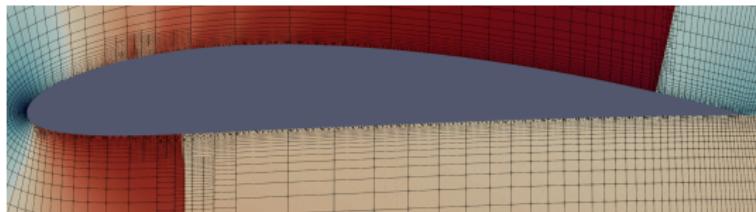
④ Conclusion & Outlook



About Trixi.jl

Trixi.jl → <https://github.com/trixi-framework/Trixi.jl>

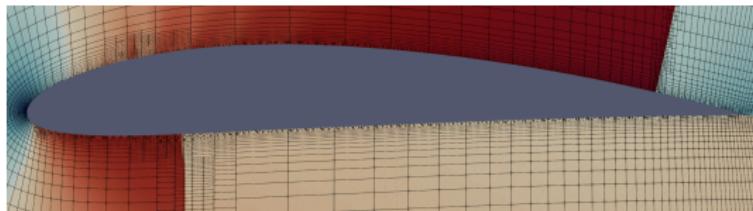
- is a **Julia** package for simulation of compressible flows
→ Euler, NSF, (VR)MHD, APE



About Trixi.jl

Trixi.jl → <https://github.com/trixi-framework/Trixi.jl>

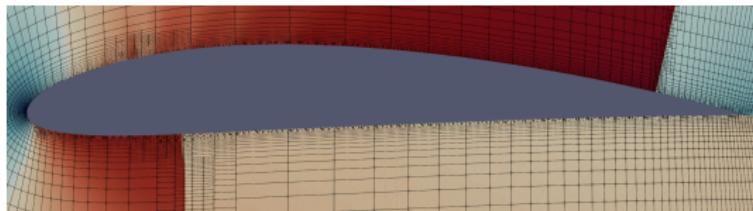
- is a **Julia** package for simulation of compressible flows
→ Euler, NSF, (VR)MHD, APE
- implements the DGSEM method



About Trixi.jl

Trixi.jl → <https://github.com/trixi-framework/Trixi.jl>

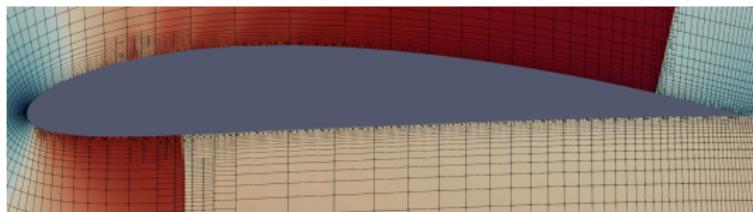
- is a **Julia** package for simulation of compressible flows
→ Euler, NSF, (VR)MHD, APE
- implements the DGSEM method
- is *h*-adaptive (AMR) through p4est and t8code



About Trixi.jl

Trixi.jl → <https://github.com/trixi-framework/Trixi.jl>

- is a **Julia** package for simulation of compressible flows
→ Euler, NSF, (VR)MHD, APE
- implements the DGSEM method
- is *h*-adaptive (AMR) through p4est and t8code
- discretizes PDEs via the **method-of-lines (MoL)** approach



About Trixi.jl

Method-of-Lines approach

- PDE

$$\partial_t \mathbf{u}(t, \mathbf{x}) + \partial_{x_i} \mathbf{f}_i(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}$$

```
using Trixi # Trixi.jl is reg. Julia package  
  
# Define PDE  
equations = LinearScalarAdvectionEquation1D(1.0)
```

Actual runnable Julia code snippet

About Trixi.jl

Method-of-Lines approach

- PDE → ODE

$$\partial_t \mathbf{u}(t, \mathbf{x}) + \partial_{x_i} \mathbf{f}_i(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}$$

$$\stackrel{\text{DG}}{\Rightarrow} \frac{d}{dt} \mathbf{U}(t) = \mathbf{F}(\mathbf{U}(t))$$

```
using Trixi # Trixi.jl is reg. Julia package

# Define PDE
equations = LinearScalarAdvectionEquation1D(1.0)

# Vanilla DGSEM with k = 3, Rusanov/LLF flux
solver = DGSEM(polydeg = 3,
                 surface_flux = flux_lax_friedrichs)
# 16-cell discretization of Ω = (-1,1)
mesh = StructuredMesh((16,), (-1,), (1,))
# Some initial condition function
initial_condition = initial_condition_const

# Perform MoL/Semidiscretization
semi = SemidiscretizationHyperbolic(mesh,
                                      equations,
                                      initial_condition, solver)

# Set up ODE problem
tspan = (0, 1)
ode = semidiscretize(semi, tspan)
```

Actual runnable Julia code snippet



About Trixi.jl

Method-of-Lines approach

- PDE → ODE

$$\partial_t \mathbf{u}(t, \mathbf{x}) + \partial_{x_i} \mathbf{f}_i(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}$$

$$\stackrel{\text{DG}}{\Rightarrow} \frac{d}{dt} \mathbf{U}(t) = \mathbf{F}(\mathbf{U}(t))$$

- a) Hand-off ODE problem to OrdinaryDiffEq.jl

```
using Trixi # Trixi.jl is reg. Julia package

# Define PDE
equations = LinearScalarAdvectionEquation1D(1.0)

# Vanilla DGSEM with k = 3, Rusanov/LLF flux
solver = DGSEM(polydeg = 3,
                 surface_flux = flux_lax_friedrichs)
# 16-cell discretization of Ω = (-1,1)
mesh = StructuredMesh((16,), (-1,), (1,))
# Some initial condition function
initial_condition = initial_condition_const

# Perform MoL/Semidiscretization
semi = SemidiscretizationHyperbolic(mesh,
                                      equations,
                                      initial_condition, solver)

# Set up ODE problem
tspan = (0, 1)
ode = semidiscretize(semi, tspan)
```

Actual runnable Julia code snippet



About Trixi.jl

Method-of-Lines approach

- PDE → ODE

$$\partial_t \mathbf{u}(t, \mathbf{x}) + \partial_{x_i} \mathbf{f}_i(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}$$

$$\stackrel{\text{DG}}{\Rightarrow} \frac{d}{dt} \mathbf{U}(t) = \mathbf{F}(\mathbf{U}(t))$$

- Hand-off ODE problem to OrdinaryDiffEq.jl
- Multirate Time-Integration via partitioned Runge-Kutta

```
using Trixi # Trixi.jl is reg. Julia package

# Define PDE
equations = LinearScalarAdvectionEquation1D(1.0)

# Vanilla DGSEM with k = 3, Rusanov/LLF flux
solver = DGSEM(polydeg = 3,
                 surface_flux = flux_lax_friedrichs)
# 16-cell discretization of Ω = (-1,1)
mesh = StructuredMesh((16,), (-1,), (1,))
# Some initial condition function
initial_condition = initial_condition_const

# Perform MoL/Semidiscretization
semi = SemidiscretizationHyperbolic(mesh,
                                      equations,
                                      initial_condition, solver)

# Set up ODE problem
tspan = (0, 1)
ode = semidiscretize(semi, tspan)
```

Actual runnable Julia code snippet



Contents

① About Trixi.jl

② Multirate Time-Integration with P-ERK

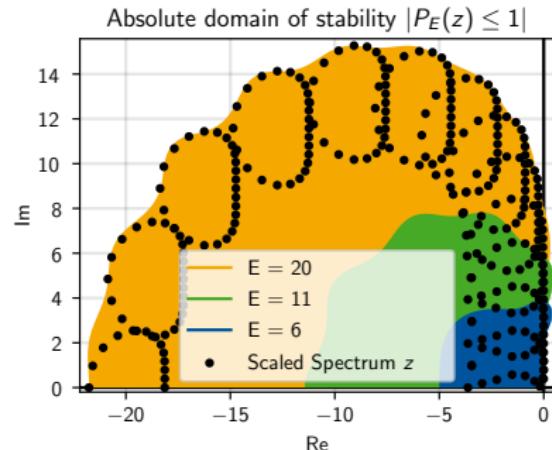
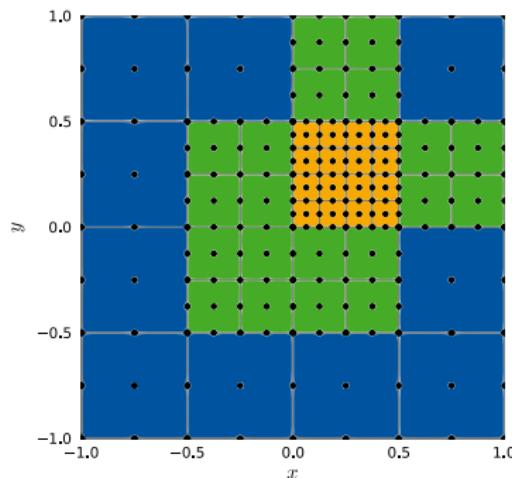
③ Applications

④ Conclusion & Outlook



Stabilized Multirate Time-Integration

Achieve multirate time-integration by using **stabilized/optimized** schemes in regions with higher characteristic speeds (stricter CFL)



For convection dominated problems:

$$\Delta t \stackrel{!}{\leq} C_t(E) \cdot C_x(k) \min_i \underbrace{\min_j \frac{\Delta x_i}{|\mu_j(\mathbf{u}_i)|}}_{=: a_i}$$



Stabilized Multirate Time-Integration

Partition ODE according to characteristic speeds a_i

$$\mathbf{U}'(t) = \begin{pmatrix} \mathbf{U}^{(1)}(t) \\ \vdots \\ \mathbf{U}^{(R)}(t) \end{pmatrix}' = \begin{pmatrix} \mathbf{F}^{(1)}(t, \mathbf{U}^{(1)}(t), \dots, \mathbf{U}^{(R)}(t)) \\ \vdots \\ \mathbf{F}^{(R)}(t, \mathbf{U}^{(1)}(t), \dots, \mathbf{U}^{(R)}(t)) \end{pmatrix} = \mathbf{F}(t, \mathbf{U}(t))$$



Stabilized Multirate Time-Integration

Partition ODE according to characteristic speeds a_i

$$\mathbf{U}'(t) = \begin{pmatrix} \mathbf{U}^{(1)}(t) \\ \vdots \\ \mathbf{U}^{(R)}(t) \end{pmatrix}' = \begin{pmatrix} \mathbf{F}^{(1)}(t, \mathbf{U}^{(1)}(t), \dots, \mathbf{U}^{(R)}(t)) \\ \vdots \\ \mathbf{F}^{(R)}(t, \mathbf{U}^{(1)}(t), \dots, \mathbf{U}^{(R)}(t)) \end{pmatrix} = \mathbf{F}(t, \mathbf{U}(t))$$

and employ different RKM

$$\mathbf{K}_i^{(r)} = \mathbf{F}^{(r)} \left(t_n + c_i^{(r)} \Delta t, \mathbf{U}_n + \Delta t \sum_{j=1}^S \sum_{k=1}^R a_{i,j}^{(k)} \mathbf{K}_j^{(k)} \right),$$

$$\mathbf{U}_{n+1} = \mathbf{U}_n + \Delta t \sum_{i=1}^S \sum_{k=1}^R b_i^{(r)} \mathbf{K}_i^{(r)}$$

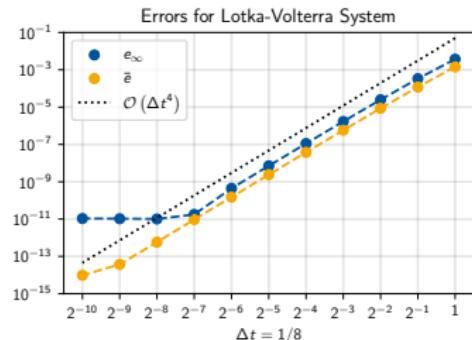
optimized for the spectrum $\sigma(J(\mathbf{U}_0)) = \sigma\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}|_{\mathbf{U}_0}\right)$.



Multirate Time-Integration with P-ERK

Paired-Explicit Runge-Kutta (P-ERK) methods¹ are a class of partitioned RKM^s that satisfy

- Order conditions:
 $p = 2^1$, $p = 3^2$ or $p = 4^3$



¹Vermeire. JCP. 2019

²Nasab, Vermeire. JCP. 2022

³Doehring et al. To Appear. 2024

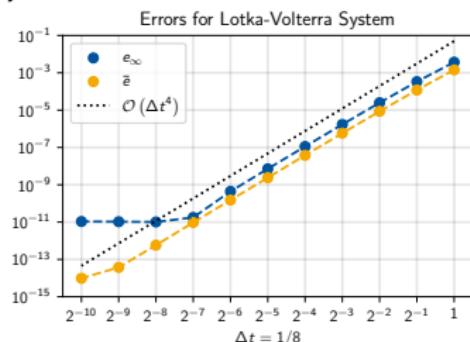
⁴Hundsdorfer, Ketcheson, Savostianov. J. Sci. Comput. 2015



Multirate Time-Integration with P-ERK

Paired-Explicit Runge-Kutta (P-ERK) methods¹ are a class of partitioned RKM^s that satisfy

- Order conditions:
 $p = 2^1$, $p = 3^2$ or $p = 4^3$
- Conservation⁴



	P-ERK _{4;{6,10,16}}	SSP _{4;10}
e_{ρ}^{Cons}	$3.78 \cdot 10^{-13}$	$2.11 \cdot 10^{-13}$
$e_{\rho v_x}^{\text{Cons}}$	$5.70 \cdot 10^{-14}$	$1.26 \cdot 10^{-13}$
$e_{\rho v_y}^{\text{Cons}}$	$6.53 \cdot 10^{-14}$	$1.29 \cdot 10^{-13}$
$e_{\rho e}^{\text{Cons}}$	$9.40 \cdot 10^{-13}$	$3.66 \cdot 10^{-12}$

¹Vermeire. JCP. 2019

²Nasab, Vermeire. JCP. 2022

³Doehring et al. To Appear. 2024

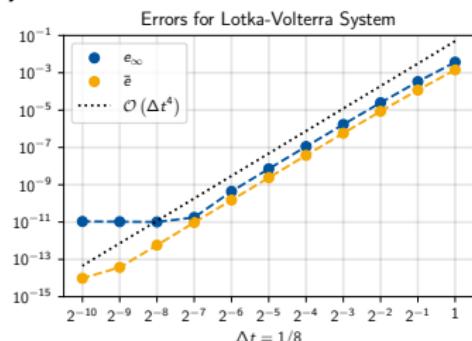
⁴Hundsdorfer, Ketcheson, Savostianov. J. Sci. Comput. 2015



Multirate Time-Integration with P-ERK

Paired-Explicit Runge-Kutta (P-ERK) methods¹ are a class of partitioned RKM's that satisfy

- Order conditions:
 $p = 2^1$, $p = 3^2$ or $p = 4^3$
- Conservation⁴
- Internal Consistency⁴



	P-ERK _{4;{6,10,16}}	SSP _{4;10}	Method	$\ e_\rho(x, y)\ _{L^\infty(\Omega)}$
e_ρ^{Cons}	$3.78 \cdot 10^{-13}$	$2.11 \cdot 10^{-13}$	P-ERK _{4;6}	$1.1497 \cdot 10^{-5}$
$e_{\rho v_x}^{\text{Cons}}$	$5.70 \cdot 10^{-14}$	$1.26 \cdot 10^{-13}$	P-ERK _{4;10}	$1.1497 \cdot 10^{-5}$
$e_{\rho v_y}^{\text{Cons}}$	$6.53 \cdot 10^{-14}$	$1.29 \cdot 10^{-13}$	P-ERK _{4;16}	$1.1497 \cdot 10^{-5}$
$e_{\rho e}^{\text{Cons}}$	$9.40 \cdot 10^{-13}$	$3.66 \cdot 10^{-12}$	P-ERK _{4;{6,10,16}}	$1.1499 \cdot 10^{-5}$

¹Vermeire. JCP. 2019

²Nasab, Vermeire. JCP. 2022

³Doehring et al. To Appear. 2024

⁴Hundsdorfer, Ketcheson, Savostianov. J. Sci. Comput. 2015



Multirate Time-Integration with P-ERK

Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- Elements



Multirate Time-Integration with P-ERK

Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- Elements
- Interfaces



Multirate Time-Integration with P-ERK

Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- Elements
- Interfaces
- Mortars



Multirate Time-Integration with P-ERK

Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- Elements
- Interfaces
- Mortars
- Boundaries



Multirate Time-Integration with P-ERK

Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- Elements
- Interfaces
- Mortars
- Boundaries

⇒ Assign these to partitions $r = 1, \dots, R$



Multirate Time-Integration with P-ERK

Non-intrusiveness

Trixi.jl operates on four (convenience) data structures

- Elements
- Interfaces
- Mortars
- Boundaries

⇒ Assign these to partitions $r = 1, \dots, R$

```
function calc_volume_integral!(du, u, mesh, nonconservative_terms, equations,
                               volume_integral::VolumeIntegralWeakForm,
                               dg::DGSEM, cache)
    # Loop over all elements in the cache
    @threaded for element in eachelement(dg, cache)
        weak_form_kernel!(du, u, element, mesh, nonconservative_terms,
                          equations, dg, cache)
    end
    return nothing
end
```



Multirate Time-Integration with P-ERK

Non-intrusiveness

Trixi.jl operates on four (convenience) data structures

- Elements
- Interfaces
- Mortars
- Boundaries

⇒ Assign these to partitions $r = 1, \dots, R$

```
function calc_volume_integral!(du, u, mesh, nonconservative_terms, equations,
                               volume_integral::VolumeIntegralWeakForm,
                               dg::DGSEM, cache, elements_r::Vector{Int64})
    # Loop over elements of the r'th level
    @threaded for element in elements_r
        weak_form_kernel!(du, u, element, mesh, nonconservative_terms,
                          equations, dg, cache)
    end
    return nothing
end
```



Contents

① About Trixi.jl

② Multirate Time-Integration with P-ERK

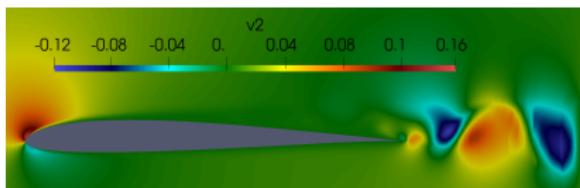
③ Applications

④ Conclusion & Outlook



Application: Laminar flow around 2D SD7003 airfoil

- $\text{Re} = 10^4, \text{Ma} = 0.2$
- $k = 3, p = 4, 7605 \text{ quads}$
- $\Omega = [-20, 40] \times [-20, 20]$



Source	\bar{C}_L	\bar{C}_D
P-ERK $p = 4$	0.3827	0.04995
P-ERK $p = 2^a$	0.3841	0.04990
P-ERK $p = 3^b$	0.3848	0.04910
Uranga et al. ^c	0.3755	0.04978
López-Morales et al. ^d	0.3719	0.04940

^aVermeire. JCP. 2019

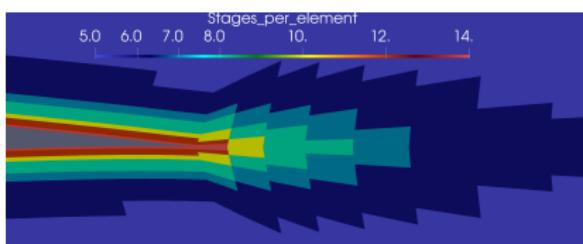
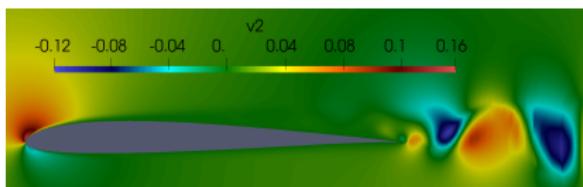
^bNasab, Vermeire. JCP. 2022

^cUranga et al. Int. J. Num Methods Eng. 2011

^dLópez-Morales et al. 32nd AIAA AAC. 2014

Application: Laminar flow around 2D SD7003 airfoil

- $\text{Re} = 10^4, \text{Ma} = 0.2$
- $k = 3, p = 4, 7605 \text{ quads}$
- $\Omega = [-20, 40] \times [-20, 20]$



Source	\bar{C}_L	\bar{C}_D
P-ERK $p = 4$	0.3827	0.04995
P-ERK $p = 2^a$	0.3841	0.04990
P-ERK $p = 3^b$	0.3848	0.04910
Uranga et al. ^c	0.3755	0.04978
López-Morales et al. ^d	0.3719	0.04940

Method	τ/τ^*	$N_{\text{RHS}}/N_{\text{RHS}}^*$
P-ERK _{4;{5,6,...14}}	1.0	1.0
P-ERK _{4;12}	1.37	2.05
RK _{4;4}	2.20	2.33

(Lower is better)

^aVermeire. JCP. 2019

^bNasab, Vermeire. JCP. 2022

^cUranga et al. Int. J. Num Methods Eng. 2011

^dLópez-Morales et al. 32nd AIAA AAC. 2014

Application: Euler-Acoustic Simulation

Co-rotating vortices

- Acoustic-Purturbation Equations (APE)¹

¹Ewert, Schröder. JCP. 2003

²Schlottke-Lakemper et al. Computers & Fluids. 2017



Application: Euler-Acoustic Simulation

Co-rotating vortices

- Acoustic-Purturbation Equations (APE)¹
- with Lamb-vector source term from CEE

¹Ewert, Schröder. JCP. 2003

²Schlottke-Lakemper et al. Computers & Fluids. 2017



Application: Euler-Acoustic Simulation

Co-rotating vortices

- Acoustic-Purturbation Equations (APE)¹
- with Lamb-vector source term from CEE
- Hybrid method²: CAA + CFD

¹Ewert, Schröder. JCP. 2003

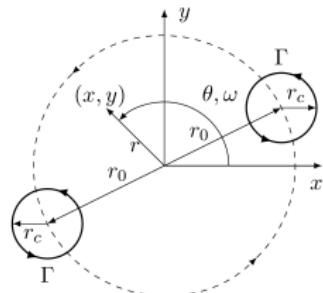
²Schlottke-Lakemper et al. Computers & Fluids. 2017



Application: Euler-Acoustic Simulation

Co-rotating vortices

- Acoustic-Perturbation Equations (APE)¹
- with Lamb-vector source term from CEE
- Hybrid method²: CAA + CFD



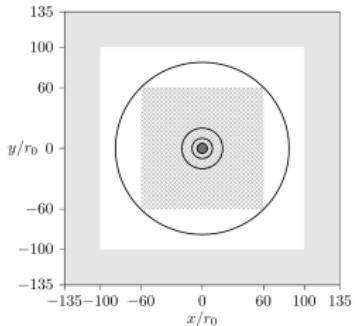
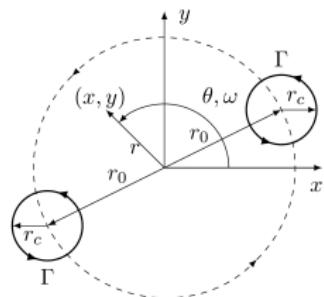
¹Ewert, Schröder. JCP. 2003

²Schlottke-Lakemper et al. Computers & Fluids. 2017

Application: Euler-Acoustic Simulation

Co-rotating vortices

- Acoustic-Perturbation Equations (APE)¹
- with Lamb-vector source term from CEE
- Hybrid method²: CAA + CFD



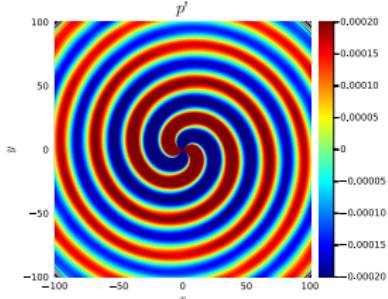
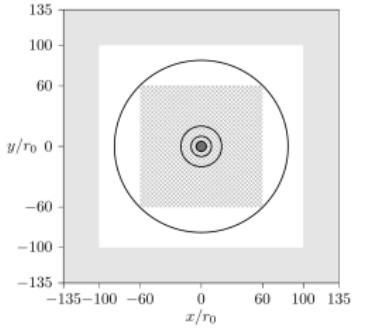
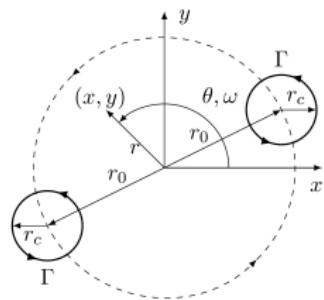
¹Ewert, Schröder. JCP. 2003

²Schlottke-Lakemper et al. Computers & Fluids. 2017

Application: Euler-Acoustic Simulation

Co-rotating vortices

- Acoustic-Perturbation Equations (APE)¹
- with Lamb-vector source term from CEE
- Hybrid method²: CAA + CFD



¹Ewert, Schröder. JCP. 2003

²Schlottke-Lakemper et al. Computers & Fluids. 2017

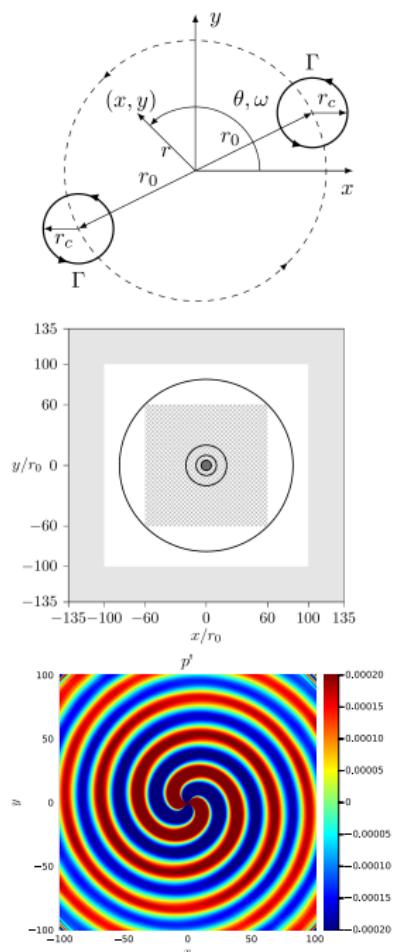
Application: Euler-Acoustic Simulation

Co-rotating vortices

- Acoustic-Perturbation Equations (APE)¹
- with Lamb-vector source term from CEE
- Hybrid method²: CAA + CFD

Method	τ / τ^*	$N_{\text{RHS}} / N_{\text{RHS}}^*$
P-ERK _{4;{5,6,8,13}} , P-ERK _{4;{5,6,9,14}}	1.0	1.0
P-ERK _{4;13} , P-ERK _{4;14}	1.60	1.87
NDB _{4;14}	1.80	2.20
TD _{4;8}	2.62	3.02
CFR _{4;6}	3.82	4.26
RK _{4;4}	4.72	4.52

(Lower is better)



¹Ewert, Schröder. JCP. 2003

²Schlottke-Lakemper et al. Computers & Fluids. 2017

Contents

① About Trixi.jl

② Multirate Time-Integration with P-ERK

③ Applications

④ Conclusion & Outlook



Conclusion & Outlook

Summary:

- Non-intrusive multirate time-integration

Up next:



Conclusion & Outlook

Summary:

- Non-intrusive multirate time-integration
→ MoL and stabilized schemes

Up next:



Conclusion & Outlook

Summary:

- Non-intrusive multirate time-integration
 - MoL and stabilized schemes
 - High-order in time & space

Up next:



Conclusion & Outlook

Summary:

- Non-intrusive multirate time-integration
 - MoL and stabilized schemes
 - High-order in time & space
 - Conservative & Consistent

Up next:



Conclusion & Outlook

Summary:

- Non-intrusive multirate time-integration
 - MoL and stabilized schemes
 - High-order in time & space
 - Conservative & Consistent

Up next:

- $p = 4$ P-ERK schemes with error-based timestep control



Conclusion & Outlook

Summary:

- Non-intrusive multirate time-integration
 - MoL and stabilized schemes
 - High-order in time & space
 - Conservative & Consistent

Up next:

- $p = 4$ P-ERK schemes with error-based timestep control
- Combination of local & multirate time-stepping



Thank you for your attention!

Questions?



Trixi.jl

GitHub Repository: <https://github.com/trixi-framework/Trixi.jl>

Documentation: <https://trixi-framework.github.io/Trixi.jl/stable/>

Slack: <https://join.slack.com/t/trixi-framework/>



RWTHAACHEN
UNIVERSITY