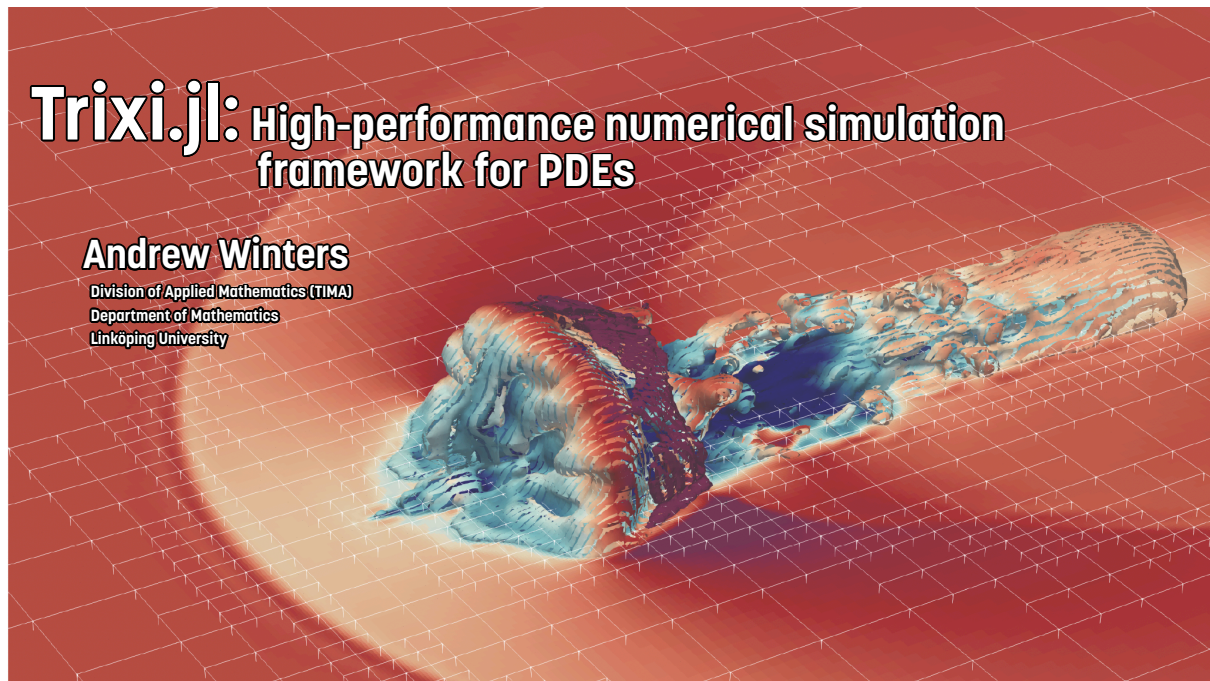
Trixi.jl: High-performance numerical simulation framework for PDEs

Andrew Winters

Division of Applied Mathematics (TIMA)
Department of Mathematics
Linköping University



Acknowledgements

Funded by

Daimler und Benz Stiftung

NUMFOCUS

OPEN CODE = BETTER SCIENCE

Who are we?

LINKÖPING UNIVERSITY

Meet the team

- Contributors from 5 countries and 10+ locations

Why did we choose Julia?

Past experiences

- Came from Fortran development: FLUXO
- Very fast 3D DGSEM HPC code
- Steep learning curve for student involvement

Moving mesh | Wave scattering | GPU + shallow water

- Codes with singular use from student projects
- Often unusable for anything else
- Abandoned after use

A Julian way forward

- Easy to get up and running; fast prototyping
- Student interest and easier to incorporate work into the main codebase
- Allows a composable and modular structure for flexibility
- Code can be simple (MATLAB-like)
  - But beware of type instabilities and (evil!) allocations
- Performant code and good HPC scaling is competitive with Fortran
- Testing framework for CI and reproducibility very strong

What is Trixi.jl?

What can it do?

At a glance

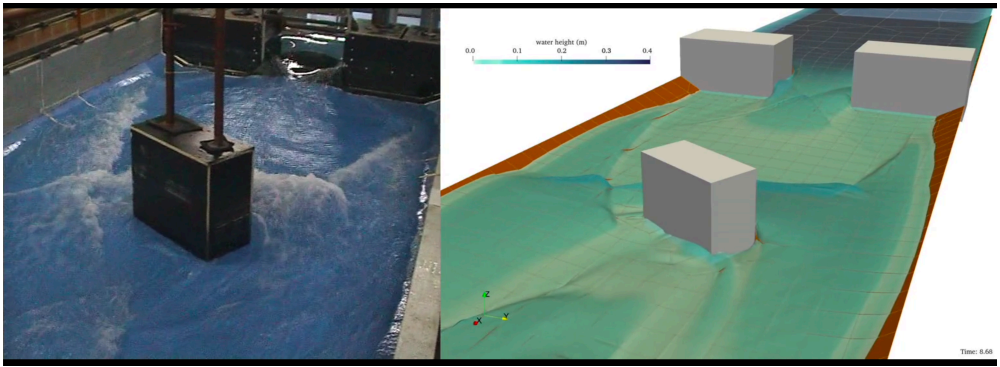
Trixi.jl

A high-performance numerical simulation framework for hyperbolic or mixed hyperbolic-parabolic PDEs

- Flexible and extensible solver in 1D, 2D or 3D
- High-order accuracy with discontinuous Galerkin spectral element methods (DGSEM)
- Structured and unstructured quadrilateral / hexahedral meshes
- Entropy stable formulations via flux differencing
- Positivity preservation for flows near vacuum (or wet/dry transitions)
- Element-wise or node-wise IDP limiting for shock capturing
- Compatible with the SciML ecosystem for ODEs
- Many equations available like compressible Euler or Navier-Stokes equations

Some lighthouse examples

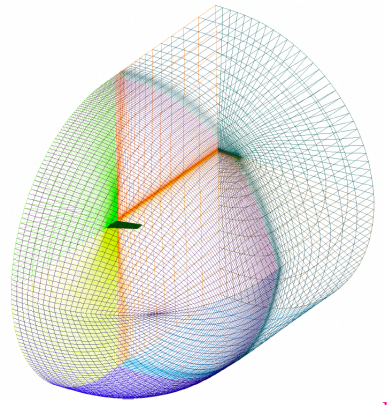
- Shallow water equations in a channel

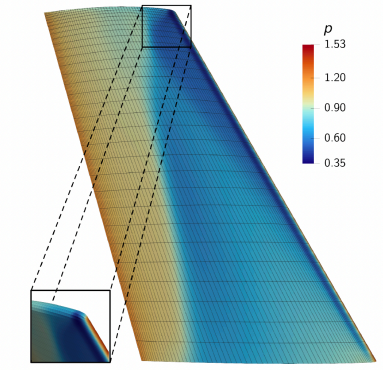


credit: Andrew Winters and Patrick Ersing

Some lighthouse examples

- Compressible flow past an ONERA M6 Wing geometry

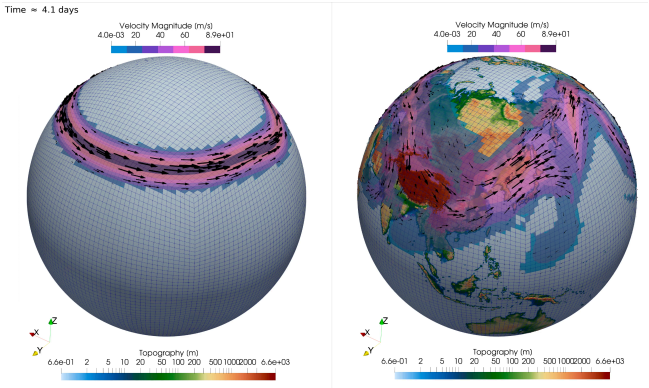




credit: Daniel Döhring

Some lighthouse examples

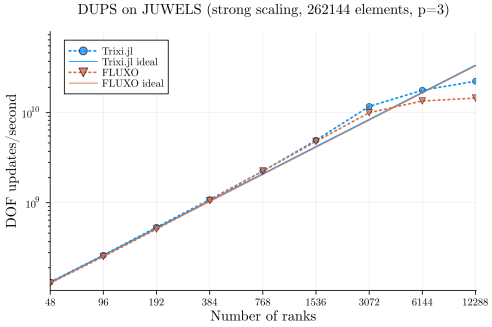
- Atmospheric flows on cubed sphere meshes



credit: Andrés Rueda-Ramírez

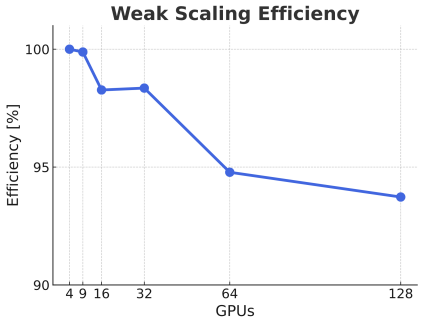
Some lighthouse examples

- Parallel scaling of compressible Euler for 3D Taylor-Green vortex turbulent box



CPUs with MPI.jl

credit: Michael Schlottke-Lakemper and Lars Christmann

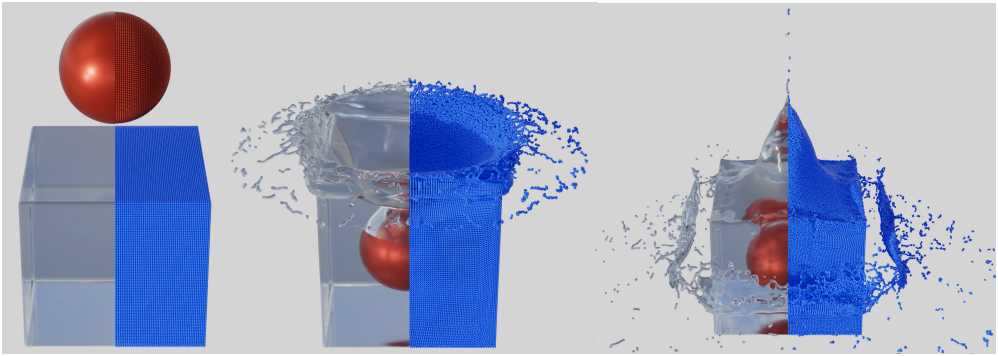


multi-GPU with KernelAbstracts.jl

credit: Benedict Geihe

Some lighthouse examples

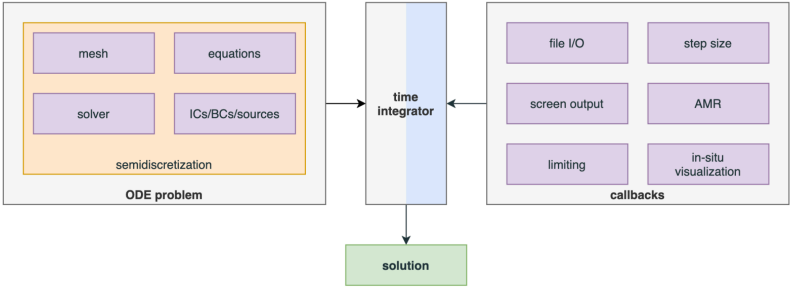
- Sibling package TrixiParticles.jl implements Smoothed Particle Hydrodynamics (SPH)



credit: Erik Faulhaber, Sven Berger, and Niklas Neher

How is Trixi.jl built?

Guiding principles for the framework



- Modular architecture (“Trixi as a library”)
- Exploit multiple dispatch to reuse kernels and prototype
- Make it work; make it fast; make it nice

- Discuss aspects of `FluxDifferencing` and `ShockCapturing` kernels

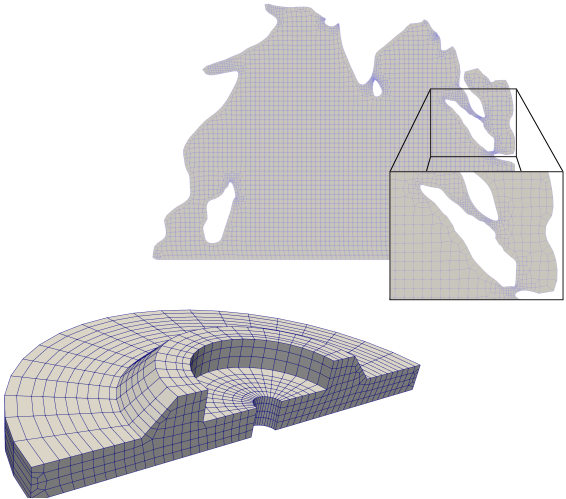
Problem of interest for Trixi.jl

- Numerical approximation of a system of conservation laws

$$\frac{\partial \mathbf{u}}{\partial t} + \vec{\nabla} \cdot \vec{\mathbf{f}}\left(\mathbf{u}, \vec{\nabla}\mathbf{u}\right) = 0, \quad \vec{x} \in \Omega$$

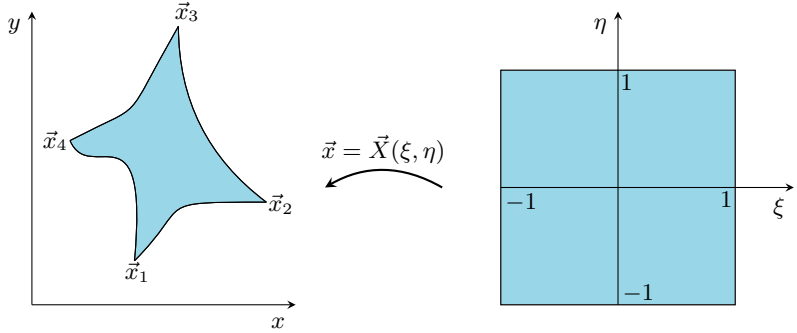
- Domain Ω is a portion of $\mathbb{R}^d$ with $d = \{1, 2, 3\}$
- Initial conditions prescribe the starting configuration of the flow
- Boundary conditions dictate how the flow behaves on the bounded domain
  - Inflow boundaries: information enters the domain
  - Outflow boundaries: information leaves the domain
  - Wall boundaries: information interacts with solid obstacles
- Source terms can add external forces like gravity or Coriolis

Domain $\Omega \rightarrow$ sub-domains



- Use specific mesh generation software HOHQMesh or gmsh
- Divide a complex domain into smaller easy to work with pieces
- Quadrilateral (2D) or hexahedral (3D) meshes for computational efficiency
- High-order methods need high-order boundary information

Conservation law in either place



- Equations remain a conservation law on reference element

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{1}{\mathcal{J}} \vec{\nabla}_\xi \cdot \vec{\tilde{\mathbf{f}}}(\mathbf{u}, \vec{\nabla}_\xi \mathbf{u}) = 0$$

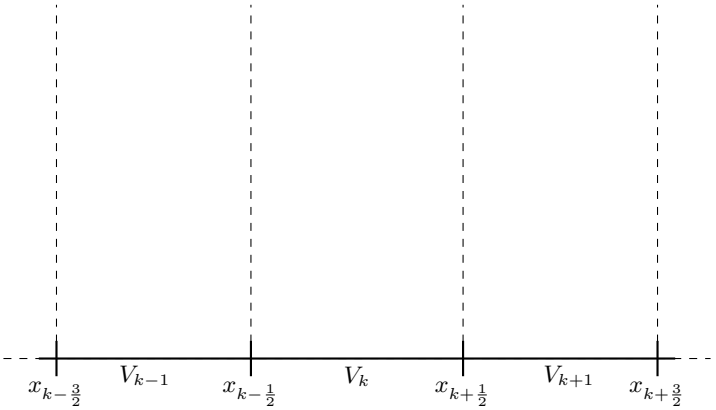
Let's build some numerical schemes!

1. Finite volume (FV): low-order approximation
2. Nodal discontinuous Galerkin (DG): high-order approximation

- Arise naturally from the integral or weak form of the governing equations
- Capable, flexible solvers for systems of conservation laws
- Keep the general motivation techniques to one spatial dimension

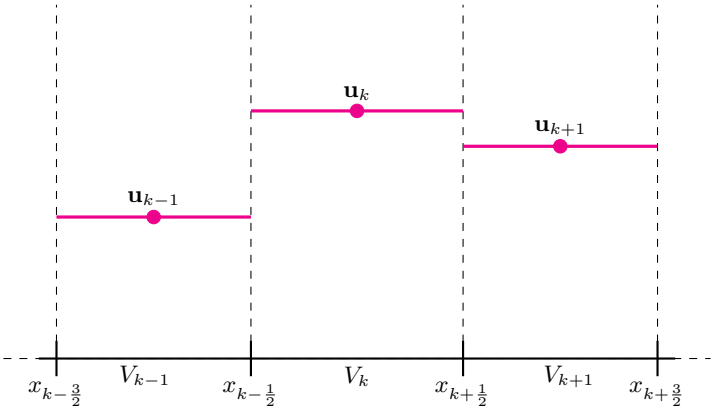
Numerical method: Finite volume

- Divide into non-overlapping cells V_k



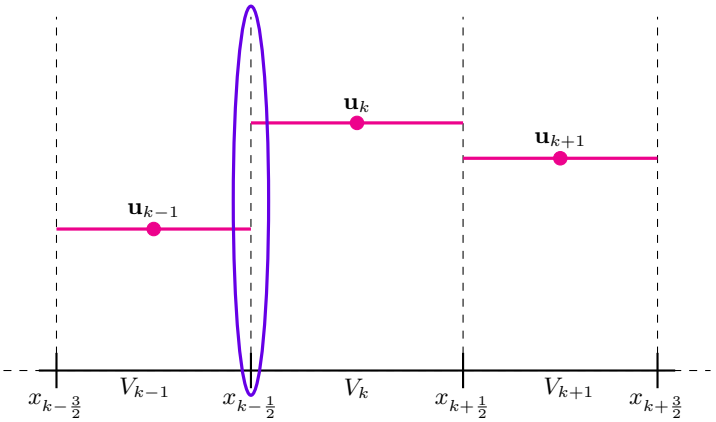
Numerical method: Finite volume

- Approximate solution vector $\mathbf{u}$ with cell averages



Numerical method: Finite volume

- Numerical flux $\mathbf{f}$ at cell boundaries resolves multiple values



Numerical method: Finite volume

- So from integral form of the PDE in cell V_k

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{V_k}\mathbf{u}\,\mathrm{d}x = -\left[\mathbf{f}(\mathbf{u})\Big|^{x_{k+\frac{1}{2}}} - \mathbf{f}(\mathbf{u})\Big|_{x_{k-\frac{1}{2}}}\right]$$

- Apply finite volume spatial discretization and resolve discontinuity with a numerical flux

$$\mathbf{f}^*(\mathbf{u}_L, \mathbf{u}_R)$$

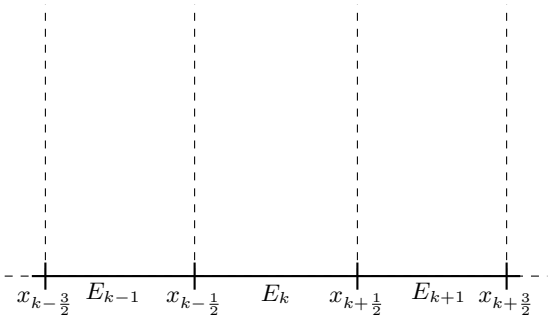
- Yields an ordinary differential equation (ODE)

$$\dot{\mathbf{u}}_k^{\mathrm{FV}} = -\frac{1}{\Delta x_k}\left[\mathbf{f}^*(\mathbf{u}_k, \mathbf{u}_{k+1}) - \mathbf{f}^*(\mathbf{u}_{k-1}, \mathbf{u}_k)\right]$$

- Now able to approximate and update solution in each cell

Numerical method: Nodal discontinuous Galerkin

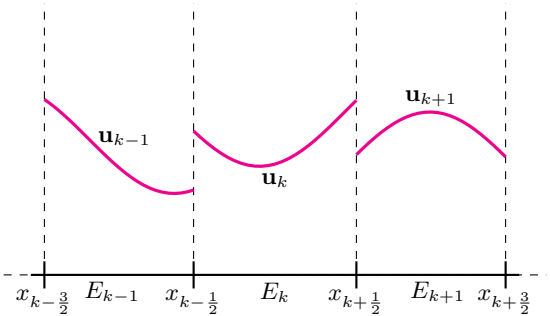
- Divide the spatial domain into non-overlapping elements



Numerical method: Nodal discontinuous Galerkin

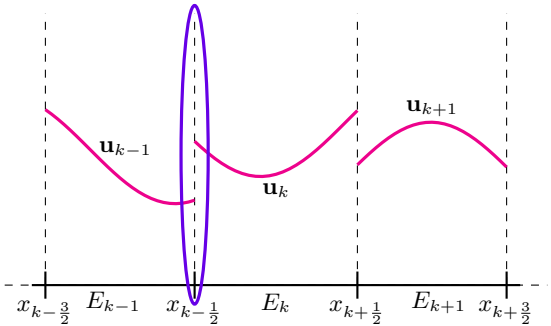
- Map each element V_k to $E = [-1, 1]$ with coordinate ξ
- On each element approximate $\mathbf{u}$ as nodal polynomial of degree N

$$\mathbf{u}(x(\xi), t) \approx \sum_{j=0}^{N} \mathbf{u}_j(t)\ell_j(\xi)$$



Numerical method: Nodal discontinuous Galerkin

- Solution ansatz still requires numerical flux at boundaries



Numerical method: Nodal discontinuous Galerkin

- From the PDE on each element E_k

$$\mathbf{u}_t + \mathbf{f}_x = 0 \qquad x \in \left[x_{k-\frac{1}{2}}, x_{k+\frac{1}{2}}\right]$$

- Map onto the reference element $E = [-1, 1]$

$$\frac{\Delta x_k}{2}\mathbf{u}_t + \mathbf{f}_\xi = 0 \qquad \xi \in [-1, 1]$$

- Multiply by Lagrange polynomials $\ell_j(\xi)$ and integrate over E

$$\frac{\Delta x_k}{2}\int_{-1}^{1} \ell_j(\xi)\mathbf{u}_t \,\mathrm{d}\xi + \int_{-1}^{1} \ell_j(\xi)\mathbf{f}_\xi \,\mathrm{d}\xi = 0$$

Numerical method: Nodal discontinuous Galerkin

- Use integration-by-parts to generate boundary terms
- Numerical flux function $\mathbf{f}^*$ couples the elements
- Approximate integrals with Legendre-Gauss-Lobatto quadrature

$$\frac{\Delta x_k}{2}\frac{\partial}{\partial t}\int\limits_{E,N} \ell_j(\xi)\,\mathbf{u}\,\mathrm{d}\xi + [\mathbf{f}^* - \mathbf{f}]\,\big|_{-1}^{1} + \int\limits_{E,N} \ell_j(\xi)\,\mathbf{f}_\xi\,\mathrm{d}\xi = 0$$

- Legendre-Gauss-Lobatto quadrature exact for polynomials up to degree $2N-1$
- Collocate interpolation and quadrature nodes to reduce cost
- Exploits the Kronecker-delta property of the nodal Lagrange basis functions

Numerical method: Nodal discontinuous Galerkin

- Again have ordinary differential equations (ODEs) on each element k

$$\dot{\mathbf{u}}_j^{\mathrm{DG}} = -\frac{2}{\Delta x_k}\left\{\frac{\delta_{jN}}{\mathcal{M}_{jj}}\left[\mathbf{f}^* - \mathbf{f}_N\right] - \frac{\delta_{j0}}{\mathcal{M}_{jj}}\left[\mathbf{f}^* - \mathbf{f}_0\right] + \sum_{m=0}^{N}\mathcal{D}_{jm}\mathbf{f}_m\right\}$$

  where $j = 0, \ldots, N$ and

  - $\mathcal{M} = \mathrm{diag}(\omega_0, \ldots, \omega_N)$ is the integration matrix (or norm matrix)
  - $\mathcal{D}_{jm} = \ell_m'(\xi_j)$ is the polynomial derivative matrix

- High-order method has surface and volume contributions

Comparison of the two methodologies

- Low-order FV method:

$$\dot{\mathbf{u}}_k^{\mathrm{FV}} = -\frac{1}{\Delta x_k}\left[\mathbf{f}^*(\mathbf{u}_k, \mathbf{u}_{k+1}) - \mathbf{f}^*(\mathbf{u}_{k-1}, \mathbf{u}_k)\right]$$

- High-order nodal DG method:

$$\dot{\mathbf{u}}_j^{\mathrm{DG}} = -\frac{2}{\Delta x_k}\left\{\frac{\delta_{jN}}{\mathcal{M}_{jj}}\left[\mathbf{f}^* - \mathbf{f}_N\right] - \frac{\delta_{j0}}{\mathcal{M}_{jj}}\left[\mathbf{f}^* - \mathbf{f}_0\right] + \sum_{m=0}^{N} \mathcal{D}_{jm}\mathbf{f}_m\right\}$$

  where $j = 0, \ldots, N$

- DG similar to FV but with extra solution "information" from interior

- For multiple dimensions use the tensor product ansatz
- Fluxes in the normal direction require the metric terms

Nodal DG commits variational crimes

- Collocated DGSEM contains aliasing errors that can drive instabilities
- Borrow two ideas from the finite difference community
  1. Split formulations act as a built-in dealiasing mechanism
  2. Summation-by-parts (SBP) to create discrete stability statements

- Example split form: average conservative and non-conservative terms

$$\frac{1}{2}(ab)_x + \frac{1}{2}\left(ba_x + ab_x \right)$$

- SBP property mimics integration-by-parts

$$(\mathcal{M}\mathcal{D}) + (\mathcal{M}\mathcal{D})^T = \mathcal{B}$$

where $\mathcal{B} = \mathrm{diag}(-1, 0, \ldots, 0, 1)$

Flux differencing DG framework

- Recast volume terms to an equivalent form

$$\sum_{m=0}^{N} \mathcal{D}_{jm} \mathbf{f}_m \quad \Rightarrow \quad 2 \sum_{m=0}^{N} \mathcal{D}_{jm} \mathbf{f}^{\#}(\mathbf{u}_j, \mathbf{u}_m)$$

- Introduce a two point volume flux that is consistent and symmetric

$$\mathbf{f}^{\#}(\mathbf{u}, \mathbf{u}) = \mathbf{f}(\mathbf{u}) \qquad \mathbf{f}^{\#}(\mathbf{u}_j, \mathbf{u}_m) = \mathbf{f}^{\#}(\mathbf{u}_m, \mathbf{u}_j)$$

- Construct different volume fluxes from mean solution states
- Surface terms untouched; still use a numerical flux function $\mathbf{f}^{*}(\mathbf{u}_L, \mathbf{u}_R; \vec{n})$

Action of the volume flux term

- Notation for the arithmetic mean

$$\{\{a\}\} = \frac{1}{2}(a_j + a_m)$$

- Example: Product of averages $\mathbf{f}^{\#}(\mathbf{u}_j, \mathbf{u}_m) = \{\{a\}\}\{\{b\}\}$ yields

$$\begin{aligned} 2\sum_{m=0}^{N} \mathcal{D}_{jm}\mathbf{f}^{\#}(\mathbf{u}_j, \mathbf{u}_m) \\ &= \frac{1}{2}\sum_{m=0}^{N} \mathcal{D}_{jm} a_m b_m + \frac{b_j}{2}\sum_{m=0}^{N} \mathcal{D}_{jm} a_m + \frac{a_j}{2}\sum_{m=0}^{N} \mathcal{D}_{jm} b_m \\ &\approx \frac{1}{2}(ab)_x + \frac{1}{2}\left(b a_x + a b_x \right) \end{aligned}$$

- Powerful framework for split form translation

Surprising result about nodal DG methods

Nodal DG and SBP property; Gassner (2013)

On collocated Legendre-Gauss-Lobatto nodes the integration and derivative matrices, $\mathcal{M}$ and $\mathcal{D}$, are SBP operators for all polynomial degrees N.

- Precise interplay between discrete integration and derivative
- Flux differencing numerical scheme remains conservative
- Specific choice of $\mathbf{f}^{\#}$ conserves mathematical energy / entropy
- Framework recovers action of the skew-symmetric form!

Flux differencing nodal DG method

$$\dot{\mathbf{u}}_j^{\mathrm{DG}} = -\frac{2}{\Delta x_k}\left\{\frac{\delta_{jN}}{\mathcal{M}_{jj}}\left[\mathbf{f}^* - \mathbf{f}_N\right] - \frac{\delta_{j0}}{\mathcal{M}_{jj}}\left[\mathbf{f}^* - \mathbf{f}_0\right] + 2\sum_{m=0}^{N}\mathcal{D}_{jm}\mathbf{f}^{\#}(\mathbf{u}_j, \mathbf{u}_m)\right\}$$

- Can link the choice of volume and surface flux for Lax-Friedrichs-type dissipation

$$\mathbf{f}^*(\mathbf{u}_L, \mathbf{u}_R) = \mathbf{f}^{\#}(\mathbf{u}_L, \mathbf{u}_R) - \frac{\lambda_{\max}}{2}\left[\mathbf{u}_R - \mathbf{u}_L\right]$$

- Drastically improves robustness of the discretisation
- Design of volume flux can discretely capture extra physics
- Extension to 2D / 3D splits apart metric terms and flux terms

Issues near shocks

- Well-known that solution of nonlinear hyperbolic systems can develop discontinuities
- High-order DG method generates spurious oscillations near discontinuities
- Unphysical overshoots (Gibbs phenomena) might lead to negative density / pressure
- Unfortunately, split form / entropy stable method does not guarantee a monotone scheme
- Low-order FV methods avoid such issues

Split form DG and its hidden FV

Flux differencing DG and subcell FV; Fisher and Carpenter (2013)

Every consistent and symmetric $\mathbf{f}^{\#} = \mathbf{f}^{\#}(\mathbf{u}_L, \mathbf{u}_R) = \mathbf{f}^{\#}(\mathbf{u}_R, \mathbf{u}_L)$ is related to a unique, consistent subcell FV method

$$2 \sum_{m=0}^{N} \mathcal{D}_{jm} \mathbf{f}^{\#}(\mathbf{u}_j, \mathbf{u}_m) = \frac{\widehat{\mathbf{f}}_{(j,j+1)} - \widehat{\mathbf{f}}_{(j-1,j)}}{\mathcal{M}_{jj}} = \frac{\widehat{\mathbf{f}}_{(j,j+1)} - \widehat{\mathbf{f}}_{(j-1,j)}}{\omega_j}$$

- Uniquely interpret diagonal norm SBP operator as subcell FV
- Flux differencing form implies local subelement-wise conservation
- Retain computational efficiency as methods re-use the same computational nodes

Illustration of DG as subcell FV

- Solution jump at element interfaces exaggerated for clarity

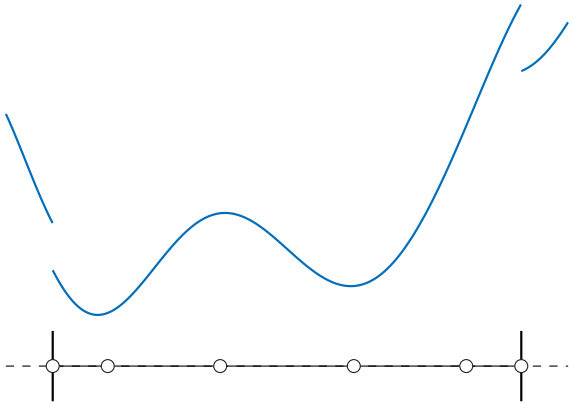


Illustration of DG as subcell FV

- Solution jump at element interfaces exaggerated for clarity

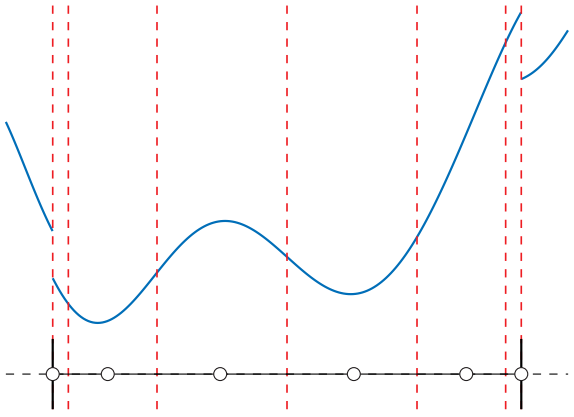


Illustration of DG as subcell FV

- Solution jump at element interfaces exaggerated for clarity

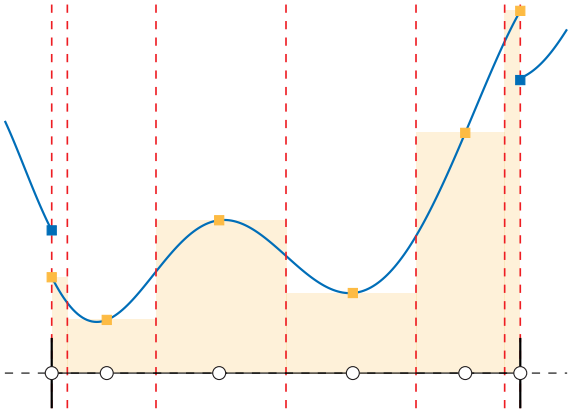
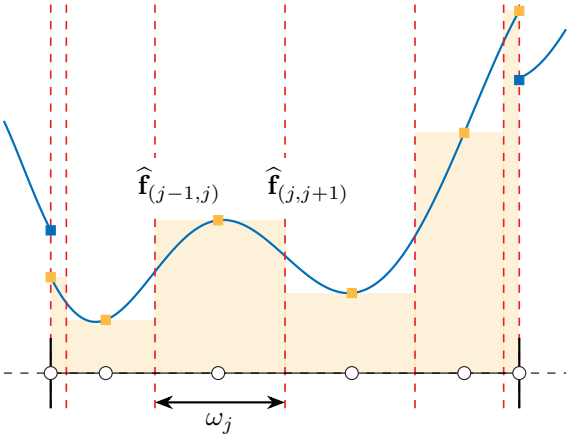


Illustration of DG as subcell FV

- Solution jump at element interfaces exaggerated for clarity



Mysterious auxiliary fluxes

- Subcell numerical fluxes (Hennemann et al. 2021)

$$\widehat{\mathbf{f}}_{(-1,0)} = \mathbf{f}^*(\mathbf{u}_L, \mathbf{u}_0)$$

$$\widehat{\mathbf{f}}_{(j,j+1)} = 2\sum_{k=0}^{j}\sum_{m=0}^{N} \mathcal{D}_{km}\mathbf{f}^{\#}(\mathbf{u}_k, \mathbf{u}_m) \qquad j = 0, \ldots, N-1$$

$$\widehat{\mathbf{f}}_{(N,N+1)} = \mathbf{f}^*(\mathbf{u}_N, \mathbf{u}_R)$$

- For consistency define fluxes at each boundary to be the surface numerical fluxes that use neighbor data
- Possible to exploit symmetry for interior fluxes and compute them recursively

A curious but effective blend

- Compatible subcell FV scheme inside each DG element
- Create a mixture of high-order DG and low-order FV

$$\dot{\mathbf{u}} = (1-\alpha)\dot{\mathbf{u}}^{\mathrm{DG}} + \alpha\dot{\mathbf{u}}^{\mathrm{FV}}$$

- Overall approximation remains provably (nonlinearly) stable

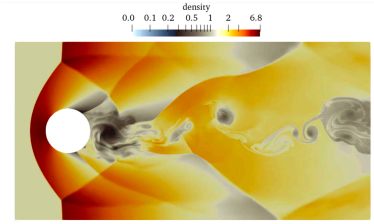
Factor we want

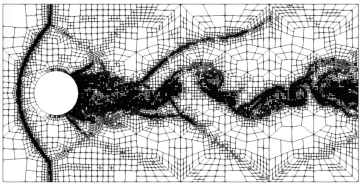
Value of α: Near 0 in smooth regions and near 1 in shocked regions.

- Exploit that we can convert DG solution to modal basis
- Compute solution "energy" in modal space according to user chosen indicator function

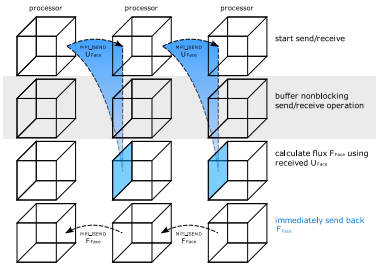
Complicated solutions? Back to the mesh!

- Solution is dynamic and features like shocks move
- Further exploit geometric flexibility of DG methods
- Mesh-based methods can have adaptive mesh refinement (AMR)
- Adaptive mesh managed by `p4est` via P4est.jl
- Concentrates degrees of freedom where they are most needed
- Mortar method to guarantee conservation on adaptive meshes





Big problem? To the supercomputer!



- DGSEM well-known for its locality and weak coupling via element interfaces
- Embarrassingly parallel on distributed memory CPU architectures
- Asynchronous communication yields very good scaling and efficiency (even with 1 element per rank!)
- Flux differencing kernel perfect candidate for implementation on GPUs
- More computational overhead without the need for more data storage and transfer

Where is Trixi.jl going?

Active areas and future directions

- Entropy stability **not enough** to guarantee nonlinear stability!?
- Time integration move away from CFL based explicit methods
- Further improvements in node-wise IDP limiting
- Better support for other node types and simplex elements
- Hone and benchmark the GPU and multi-GPU compute kernels
- Expand to other physical models, like dispersive waves

⋮

Current Trixi ecosystem

Trixi.jl

TrixiShallowWater.jl

TrixiAtmo.jl

TrixiP

TrixiParticles.jl

Upstream dependencies and partner packages

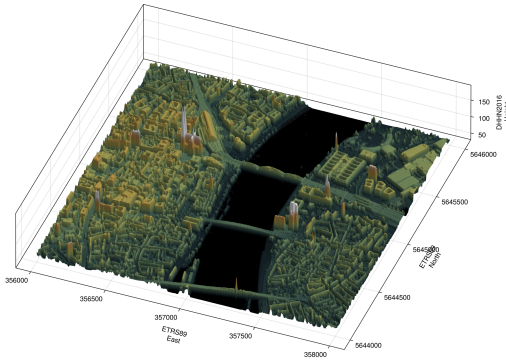
- HOHQMesh(.jl)
- PointNeighbors.jl
- libtrixi
- P4est.jl
- ParaViewCatalyst.jl
- TrixiBottomTopography.jl
- SummationByPartsOperaors.jl
- Trixi2Vtk.jl
- StartUpDG.jl
- T8code(.jl)
- TrixiLW.jl
- more...

How to get involved?

Don’t be shy! Drop us a line

- Join us in the trixi-framework Slack channel
- Stop by the docs, read some issues, open a PR
- Try out one the `examples/` (there are ≈ 500)
- Active development in many components:
  - Solver framework in Trixi.jl
  - Shallow water extensions in TrixiShallowWater.jl
  - Atmospheric flows in TrixiAtmo.jl
  - Bathymetry approximation via TrixiBottomTopography.jl
  - Mesh generation that provides high-order boundary information with HOHQMesh

  ⋮



credit: Andrew Winters

References

Fisher, T. C. and Carpenter, M. H. (2013).
High-order entropy stable finite difference schemes for nonlinear conservation laws: Finite domains.
J. Comput. Phys., 252:518–557.

Gassner, G. J. (2013).
A skew-symmetric discontinuous Galerkin spectral element discretization and its relation to SBP-SAT finite difference methods.
SIAM J. Sci. Comput., 35(3):A1233–A1253.

Hennemann, S., Rueda-Ramírez, A. M., Hindenlang, F. J., and Gassner, G. J. (2021).
A provably entropy stable subcell shock capturing approach for high order split form DG for the compressible Euler equations.
J. Comput. Phys., 426:109935.

Rueda-Ramírez, A. M., Pazner, W., and Gassner, G. J. (2022).
Subcell limiting strategies for discontinuous Galerkin spectral element methods.
Comput. Fluids, 247:105627.

Schlottke-Lakemper, M., Winters, A. R., Ranocha, H., and Gassner, G. J. (2021).
A purely hyperbolic discontinuous Galerkin approach for self-gravitating gas dynamics.
J. Comput. Phys., 442:110467.

Winters, A. R., Kopriva, D. A., Gassner, G. J., and Hindenlang, F. (2021).
Construction of modern robust nodal discontinuous Galerkin spectral element methods for the compressible Navier–Stokes equations.
In Kronbichler, M. and Persson, P.-O., editors, *Efficient High-Order Discretizations for Computational Fluid Dynamics*, pages 117–196. Springer International Publishing, Cham.