Development of a Nodal DG Solver within the SU2 Framework

Edwin van der Weide
Department of Mechanical Engineering
University of Twente

UNIVERSITY OF TWENTE.

Jae hwan Choi, Paul Urbanczyk
Department of Aeronautics and Astronautics
Stanford University

aerospacedesignlab

Dheevatsa Mudigere
Intel Corporation

intel

Juan J. Alonso
Department of Aeronautics and Astronautics
Stanford University

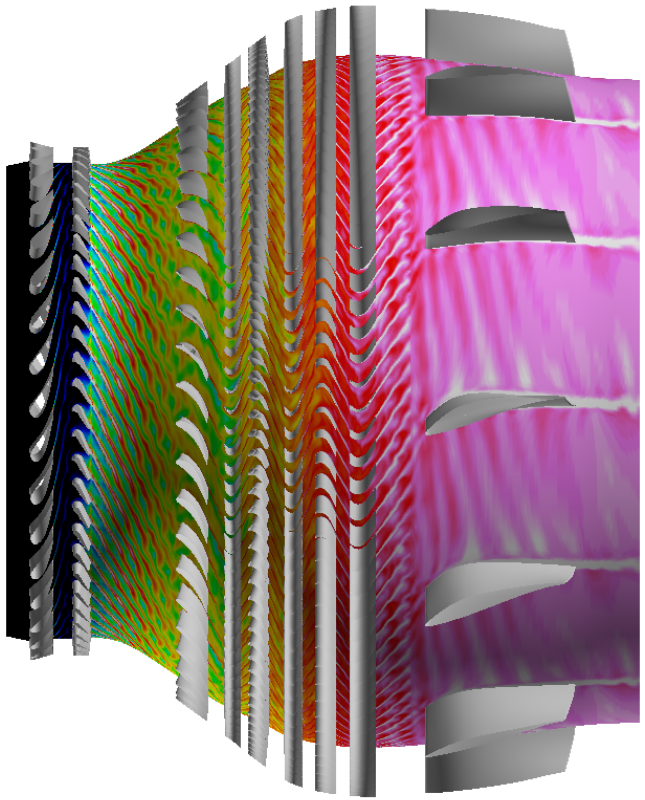
aerospacedesignlab

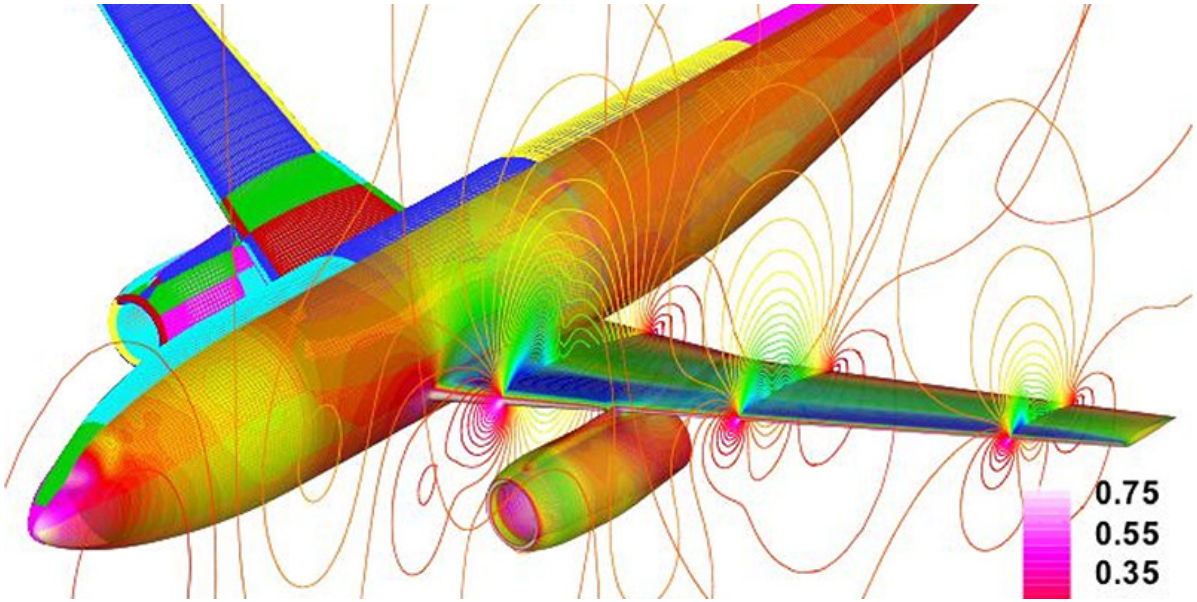
Outline

- Motivation for high-order schemes
- DG-FEM, the basic principles
- Current solver capabilities
- Access, compile and run the code
- Main DG-solver routines
- Time-accurate local time stepping
- Conservative vs. entropy variables
- Shock capturing
- LES models
- Performance optimization

Why do we need high-order schemes?

2nd order schemes have been extremely successful, certainly for RANS. SU2 FV is used for many applications…



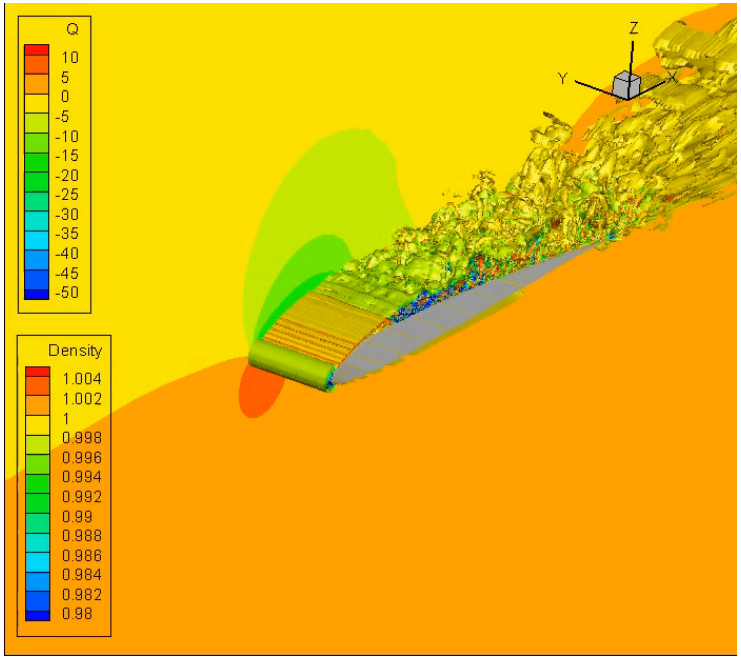


But…

For some applications 2nd order accuracy may not be sufficient

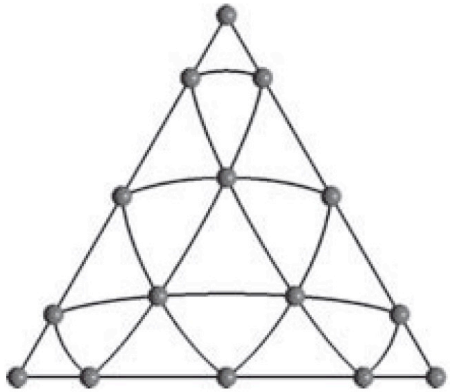
Examples

- Wake and vortex flows
- Noise prediction
- DES/LES/DNS

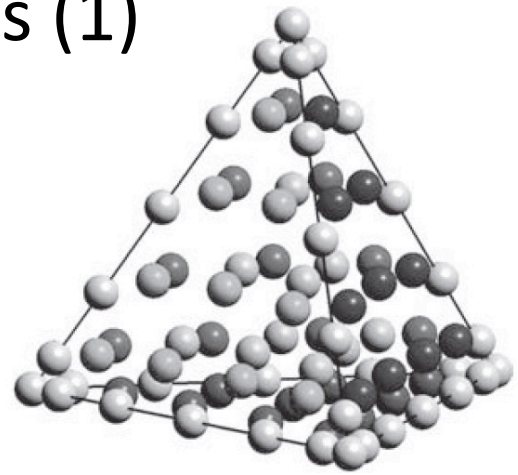


DG solver is intended for high-fidelity modeling of the turbulence, i.e. LES (wall resolved and wall modeled) and DNS

Nodal DG-FEM: the basic principles (1)



$$\text{Element } k: \; U(x_i) = \sum_{j=1}^{N_p} U_j^k \varphi_j^k(x_i)$$



Hyperbolic system of PDE's

Weak formulation

$$\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i} = 0 \quad \Rightarrow$$

$$\iint_{V_k} \frac{\partial U}{\partial t} \varphi_m^k \, dV - \iint_{V_k} F_i \frac{\partial \varphi_m^k}{\partial x_i} \, dV + \oint_{\partial V_k} F_i n_i \varphi_m^k \, d\Omega_k = 0, \quad m = 1, \ldots, N_p$$

Integrals are computed with high enough accuracy to prevent instabilities due to aliasing errors

Nodal DG-FEM: the basic principles (2)

Contribution from the contour integral

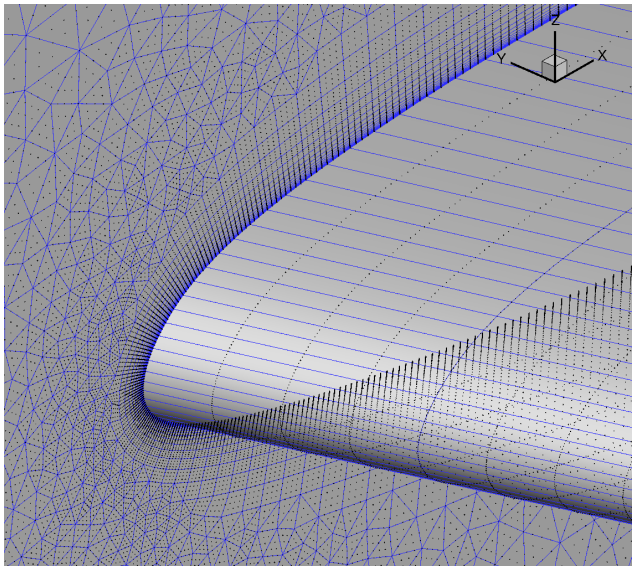
Solution at the interfaces is multiply defined and discontinuous

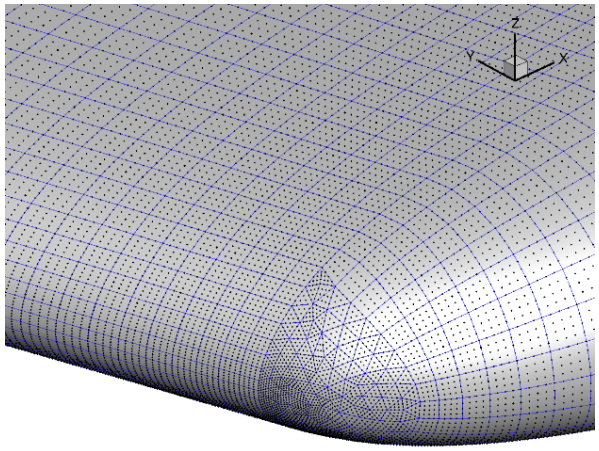
$$\oint_{\partial V_k} F_i n_i \varphi_m^k \, d\Omega_k \Rightarrow \oint_{\partial V_k} F_i\left(U_L, U_R\right) n_i \varphi_m^k \, d\Omega_k$$

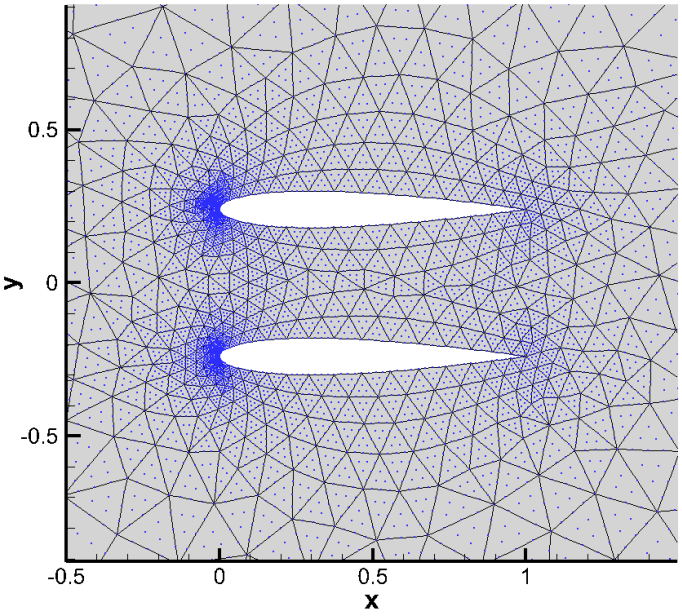
Riemann problem: Any approximate Riemann solver can be used => stabilizes the discretization

1st order DG-FEM equals 1st order FVM!!!

Higher-order Grids with Curved Elements







In-house capability to generate higher-order grids for simple cases

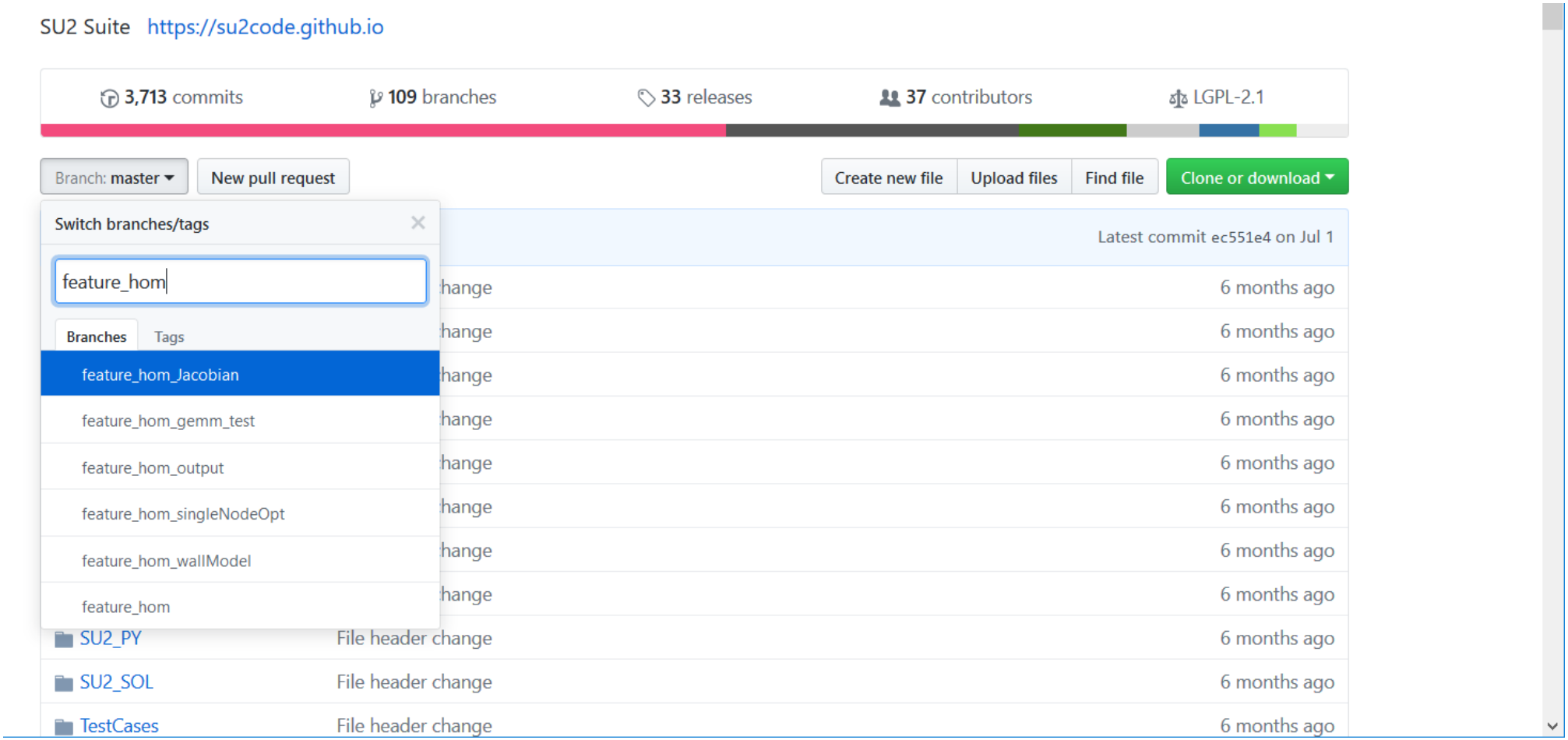
A lot more effort should be put into high-order grid generation.

Current Capabilities

- Both 2D and 3D, just like SU2-FV
- All standard elements (tri, quad, tet, pyra, prism, hex)
- Curved elements of arbitrary order
- Arbitrary polynomial order solution elements
- Polynomial order can differ in individual elements
- Symmetric Interior Penalty method for viscous fluxes
- Explicit time integration schemes (Runge-Kutta type)
- Time-accurate local time stepping via ADER-DG
- Task scheduling approach for efficient parallelization
- Preliminary implementation of LES models and shock capturing

Access the code

- *feature_hom* branch on GitHub is the main branch for the DG solver
- Several other development branches exist



Compile the code

- configure script, like the rest of SU2
- Some additional flags are required, for handling of matrix multiplications
  - Native implementation (use for debugging only)
  - System BLAS/LAPACK routines
  - Intel MKL
  - LIBXSMM (available on GitHub)

```
FLAGS="-fpic -O2 -funroll-loops -ftree-vectorize -ffast-math"
FLAGS="$FLAGS -DHAVE_LIBXSMM"
INCFLAGS="-I$LIBXSMM_include_dir"
LDFLAGS=
LIBFLAGS="$LIBFLAGS $LIBXSMM_lib_dir/libxsmm.lib -lopenblas -lmsmpi -static -limagehlp"

CFLAGS="$FLAGS"
CXXFLAGS="$FLAGS -std=c++11"

cd $SU2_dir

build_cmd="./configure --enable-mpi --enable-tecio --with-cc=gcc --with-cxx=g++ CXXFLAGS=\"$CXXFLAGS $
INCFLAGS\" CFLAGS=\"$CFLAGS $INCFLAGS\" LDFLAGS=\"$LDFLAGS\" LIBS=\"$LIBFLAGS\" --with-metis-cppflags=
\"-O3\" --with-parmetis-cppflags=\"-O3\" --prefix=$INSTALL_DIR --exec-prefix=$INSTALL_DIR"
```

Run the code

- Either sequential or parallel via mpirun (mpiexec)
- Additional parameters must be specified in the .cfg file
- Running via python should be possible, but currently no benefit (no multi-disciplinary applications yet)

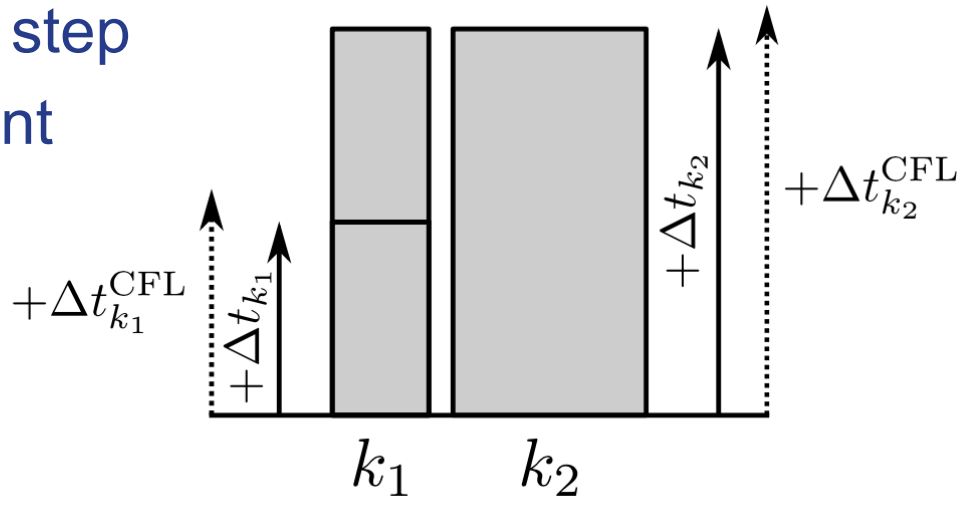
```
WeideETA@UT145406 MINGW64 ~/SU2_HOM/TestCases/hom_navierstokes/FlatPlate/nPoly4
$ mpiexec -n 8 ~/SU2_HOM/ExecParallel_LIBXSMM/bin/SU2_CFD.exe lam_flatplate_reg.cfg

-------------------------------------------------------------------------
|    ___ _    _ ___                                                     |
|   / __| |  | |_  )   Release 5.0.0  "Raven"                           |
|   \__ \ |_| |/ /                                                      |
|   |___/\___//___|   Suite (Computational Fluid Dynamics Code)         |
|                                                                       |
-------------------------------------------------------------------------
| SU2 Original Developers: Dr. Francisco D. Palacios.                   |
|                          Dr. Thomas D. Economon.                      |
-------------------------------------------------------------------------
| SU2 Developers:                                                       |
| - Prof. Juan J. Alonso's group at Stanford University.                |
| - Prof. Piero Colonna's group at Delft University of Technology.      |
| - Prof. Nicolas R. Gauger's group at Kaiserslautern U. of Technology. |
| - Prof. Alberto Guardone's group at Polytechnic University of Milan.  |
| - Prof. Rafael Palacios' group at Imperial College London.            |
| - Prof. Edwin van der Weide's group at the University of Twente.      |
| - Prof. Vincent Terrapon's group at the University of Liege.          |
-------------------------------------------------------------------------
| Copyright (C) 2012-2017 SU2, the open-source CFD code.                |
|                                                                       |
```

Main DG-solver routines

- Partitioning
  - Common/src/geometry_structure_fem_part.cpp
  - Common/src/fem_work_estimate_metis.cpp
- Preprocessing
  - Common/include/fem_geometry_structure.hpp
  - Common/include/fem_standard_element.hpp
  - Common/src/fem_geometry_structure.cpp
  - Common/src/fem_integration_rules.cpp
  - Common/src/fem_standard_element.cpp
- Solver
  - SU2_CFD/include/solver_structure.hpp
  - SU2_CFD/src/integration_time.cpp
  - SU2_CFD/src/solver_direct_mean_fem.cpp
  - Common/src/dense_matrix_product.cpp

Time-accurate local time stepping

- Explicit schemes: Global time step determined by smallest element
- Inefficient when element sizes differ significantly
- Solution: time-accurate local time stepping



- Difficult (impossible?) for Runge-Kutta schemes
- Possible with space-time formulations, e.g. ADER-DG
- Practical restrictions
  - Only finite number of time steps allowed, which differ by a factor 2, i.e. Δt, 2Δt, 4Δt, etc.
  - Neighboring elements: max. one time level difference
- Challenge for the load balancing => task scheduler

Conservative vs. entropy variables

- Numerical stability can only be proven for symmetric systems
- Not the case for Navier-Stokes equations
- However, NS can be *symmetrized* using entropy variables

$$B\frac{\partial v}{\partial t} + A_i\frac{\partial v}{\partial x_i} - \frac{\partial}{\partial x_i}\left(K_{ij}\frac{\partial v_j}{\partial x_j}\right) = 0$$

SPD Sym SPSD

$$v = \begin{bmatrix} \frac{\gamma - s}{\gamma - 1} - \frac{1}{2}\frac{\rho}{p}u_i u_i \\ \frac{\rho u_i}{p} \\ -\frac{\rho}{p} \end{bmatrix}$$

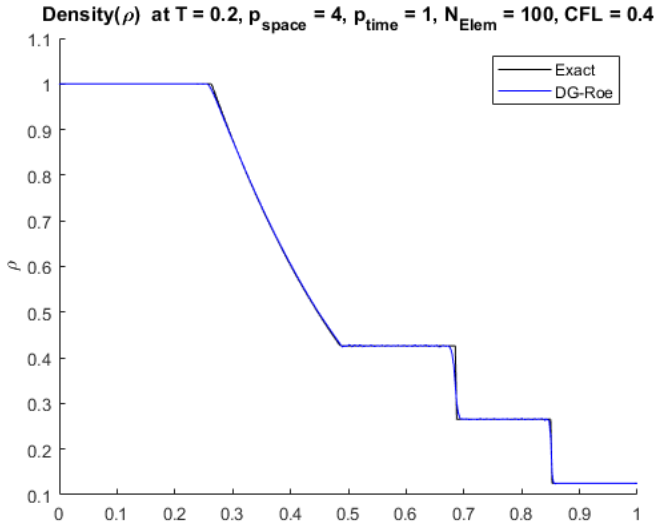
$$\mathrm{A_i} = \frac{\partial F_i}{\partial v},\ B = \frac{\partial Q}{\partial v}, Q : \mathrm{Cons.\ variables}$$

- Leads to an implicit formulation, even for explicit schemes

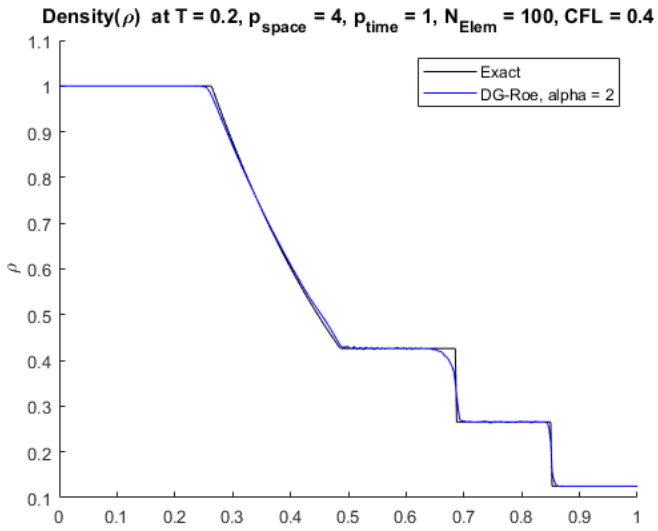
Shock capturing

- Shock capturing consists of two components:
  - Detecting the discontinuity
  - Resolving the discontinuity
- Detecting the discontinuity:
  - Persson and Peraire : Modal decay
  - Clain, Diot and Loubere : MOOD
- Resolving the discontinuity
  - Limiter
  - Artificial viscosity / filtering
  - Sub-cell limiting

<Sod's shock tube problem>



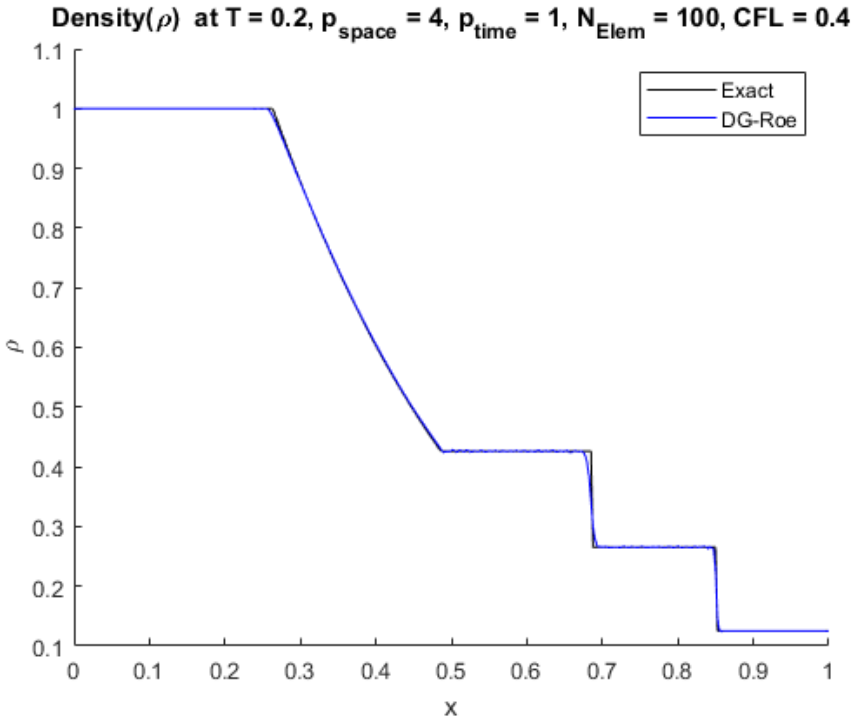
MOOD sensor / Subcell limiting

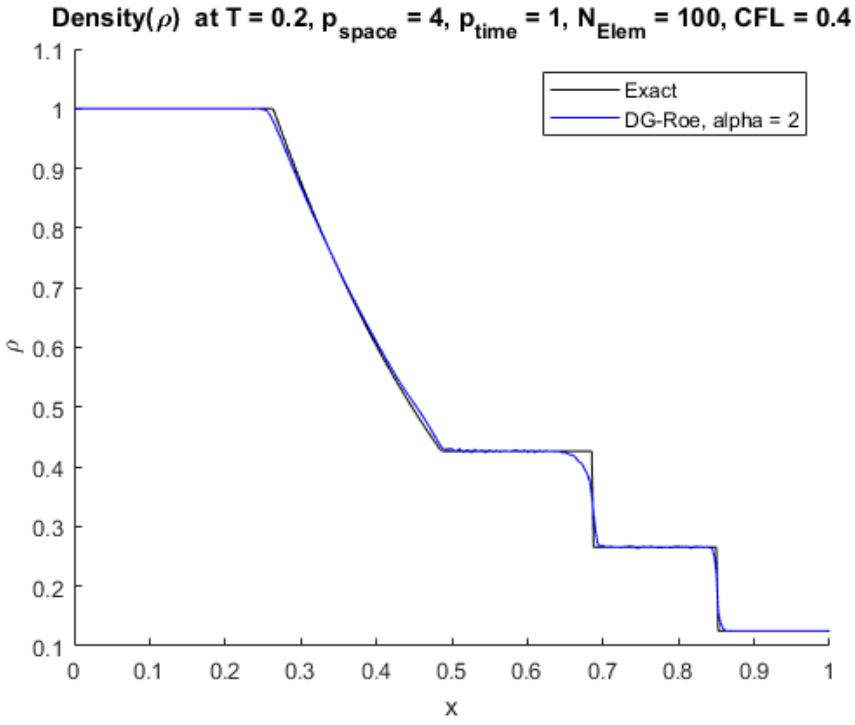


MOOD sensor / Filtering

Shock capturing

- Sod's shock tube problem
  - MOOD sensor
  - Sub-cell limiting
  - MOOD sensor
  - Filtering





LES Models

- SGS Models
  - Constant Smagorinsky

    $\nu_{sgs} = C_s^2 \Delta^2 |\tilde{S}|$
  - Wall-Adapting Local Eddy Viscosity (WALE)

$$\nu_{sgs} = (C_w \Delta)^2 \frac{\left(S_{ij}^d S_{ij}^d\right)^{(3/2)}}{\left(\widetilde{S_{ij}}\widetilde{S_{ij}}\right)^{(5/2)} + \left(S_{ij}^d S_{ij}^d\right)^{(5/4)}}$$

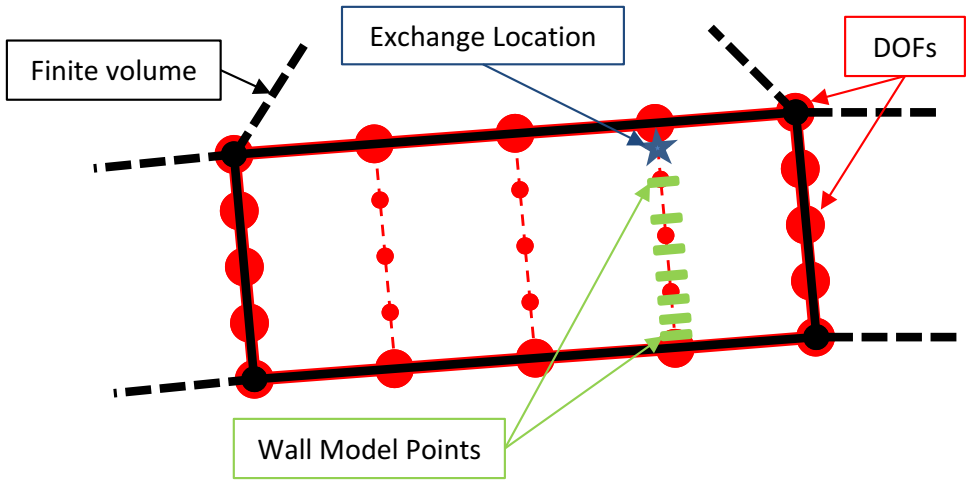
  - More sophisticated models to be implemented in future
    - Dynamic Smagorinsky, etc.

- Wall Models
  - One-dimensional Equilibrium BL Equations

$$\frac{d}{dy}\left[(\mu - \mu_t)\frac{d\tilde{u}_{\|}}{dy}\right] = 0$$

$$\frac{d}{dy}(\bar{p}) = 0$$

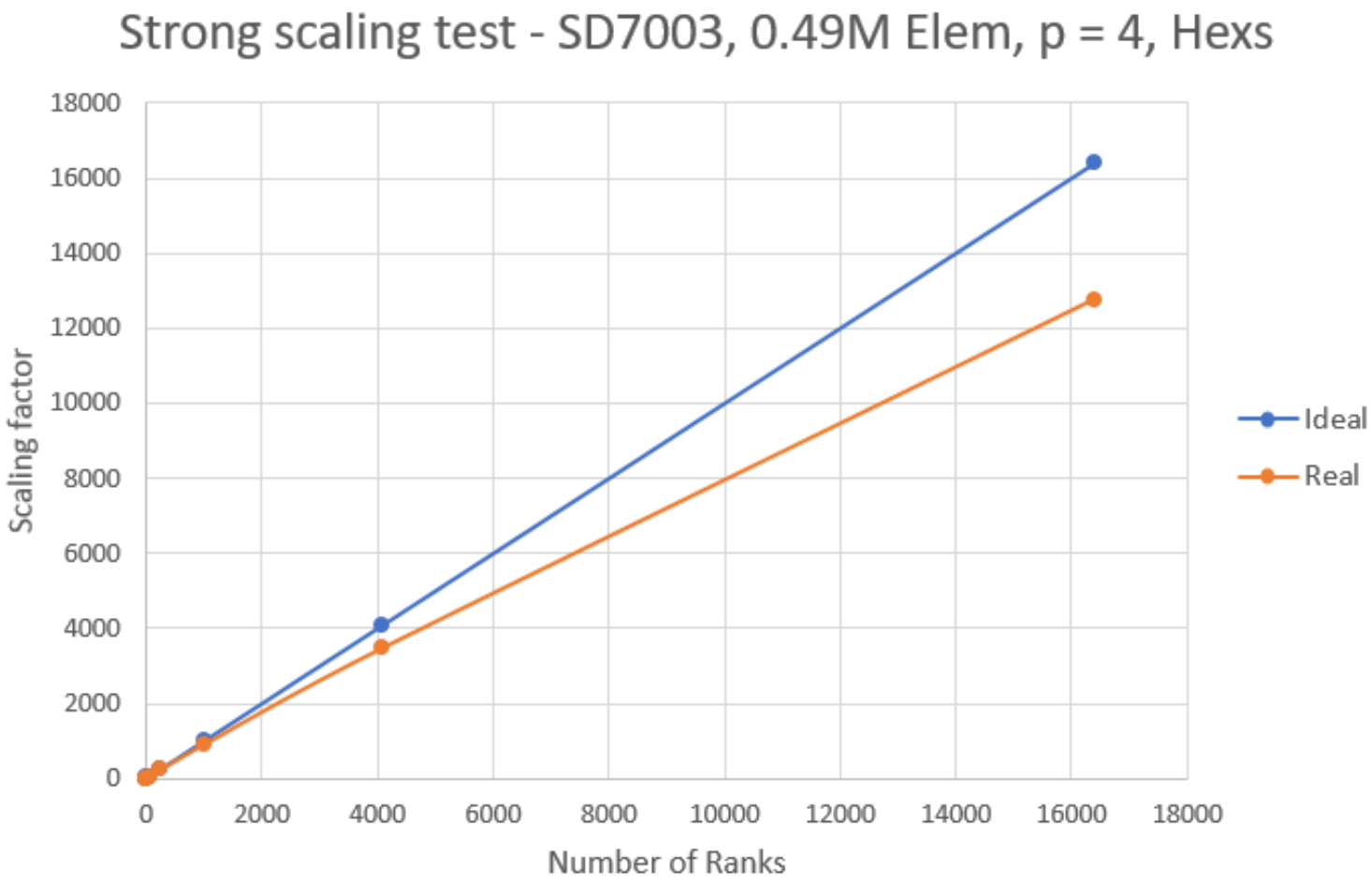
$$\frac{d}{dy}\left[\tilde{u}_{\|}(\mu + \mu_t)\frac{d\tilde{u}_{\|}}{dy} + \left(\frac{\mu c_p}{Pr} + \frac{\mu_t c_p}{Pr_t}\right)\frac{d\tilde{T}}{dy}\right] = 0$$



Performance optimization

- First implementation was very inefficient (< 5% peak on Xeon)
- Collaboration with Intel to improve efficiency
  - Specialized matrix multiplication software (BLAS, LIBXSMM)
  - Explicit unrolling of small loops (specialized 2D, 3D code)
  - Vectorization direction matrix multiplication: 128 byte aligned
  - Element-wise operation fusion for vectorization
- Current performance: ≈ 40% peak on Xeon
- Thoughts about hybrid MPI-OpenMP to increase flexibility
- Potential for advanced (Regent/Legion) methods to help scalability and portability

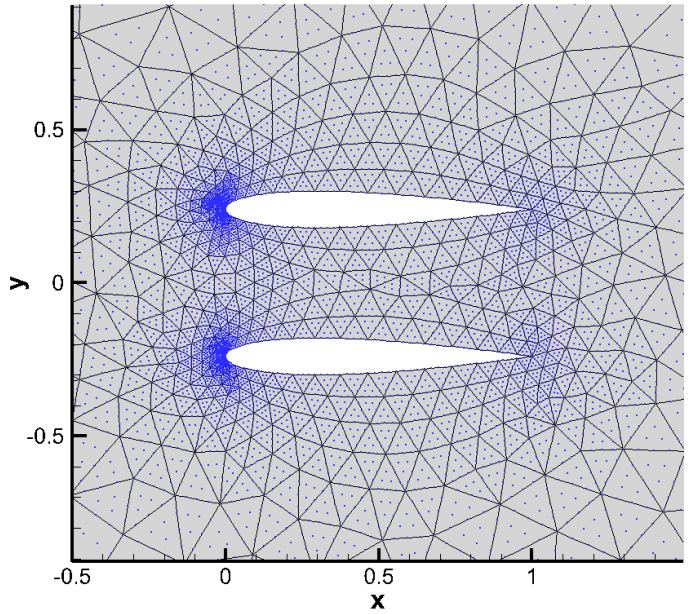
Strong scaling test on Theta (ANL)

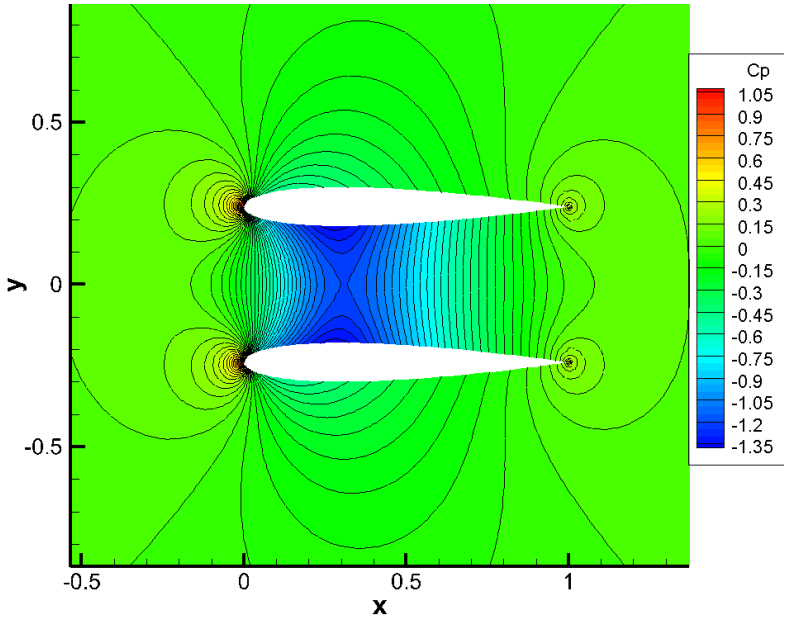


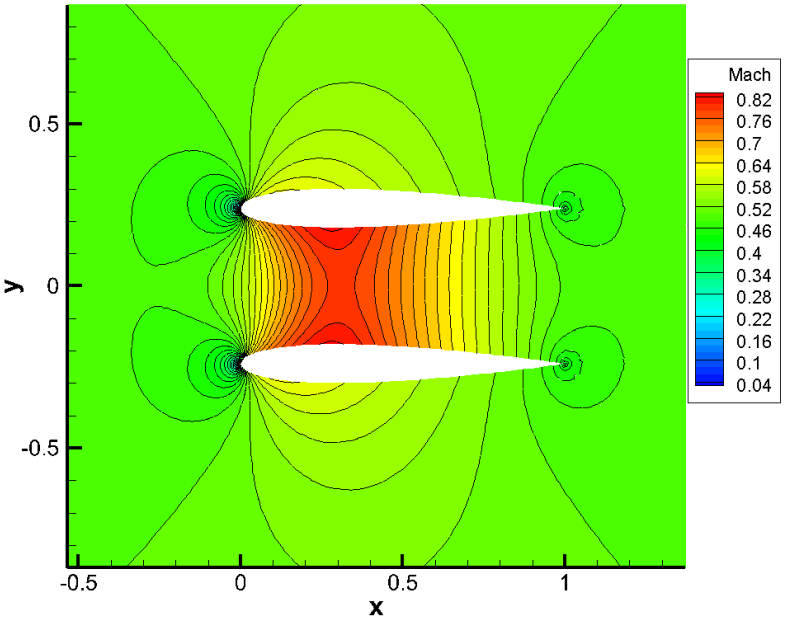
Elem/Rank	0.25M	0.12M	30.7K	7.68K	1.92K	480	120	30
DOF/Rank	30.7M	15.4M	3.84M	0.96M	0.24M	60K	15K	3.75K
Efficiency(%)	N/A	99.98	97.15	90.89	93.39	90.49	85.07	77.86

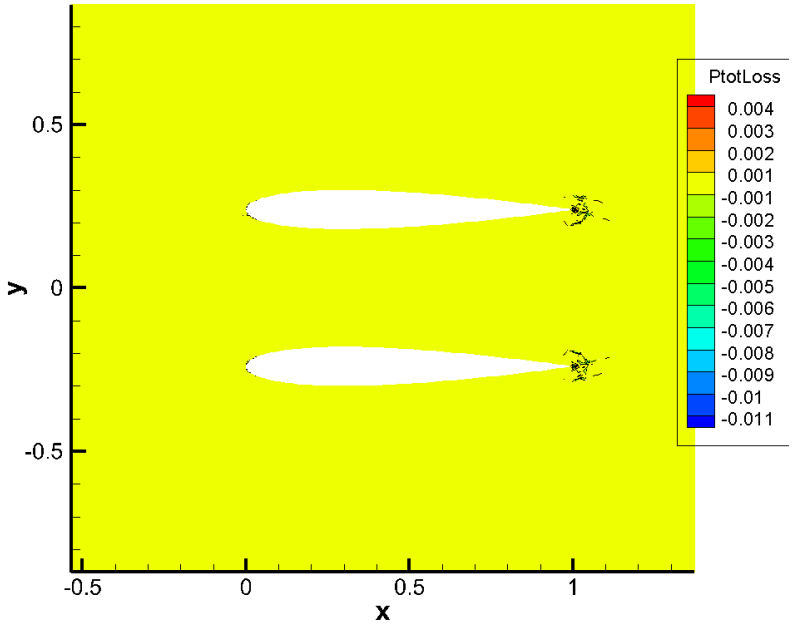
Results (p = 4, triangles, inviscid)

Visualization via linear sub-elements



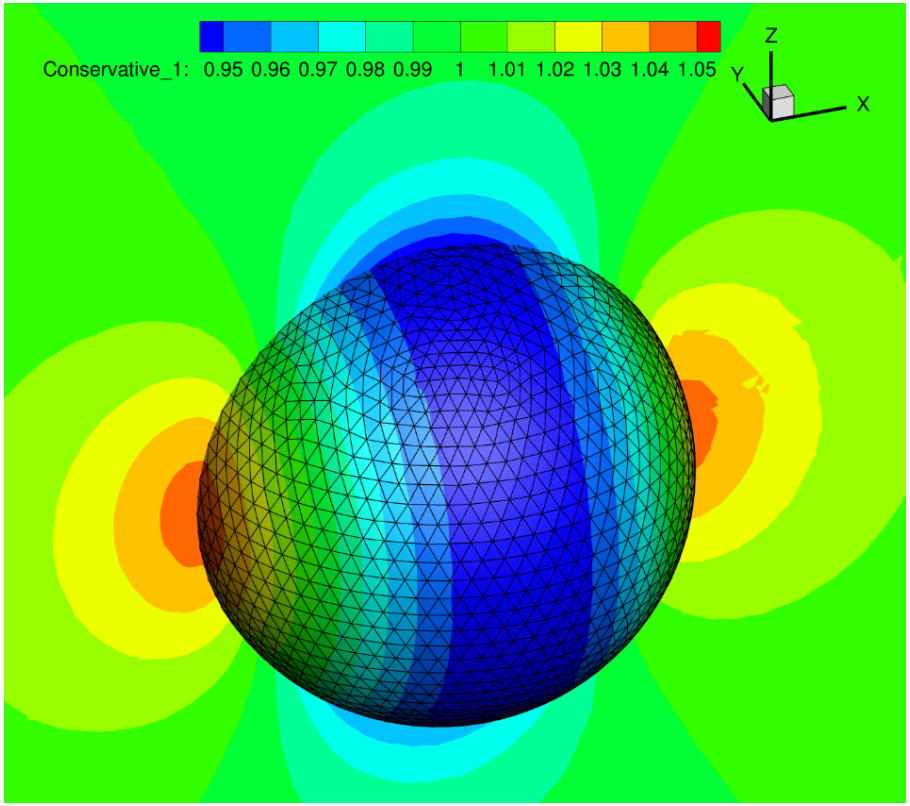




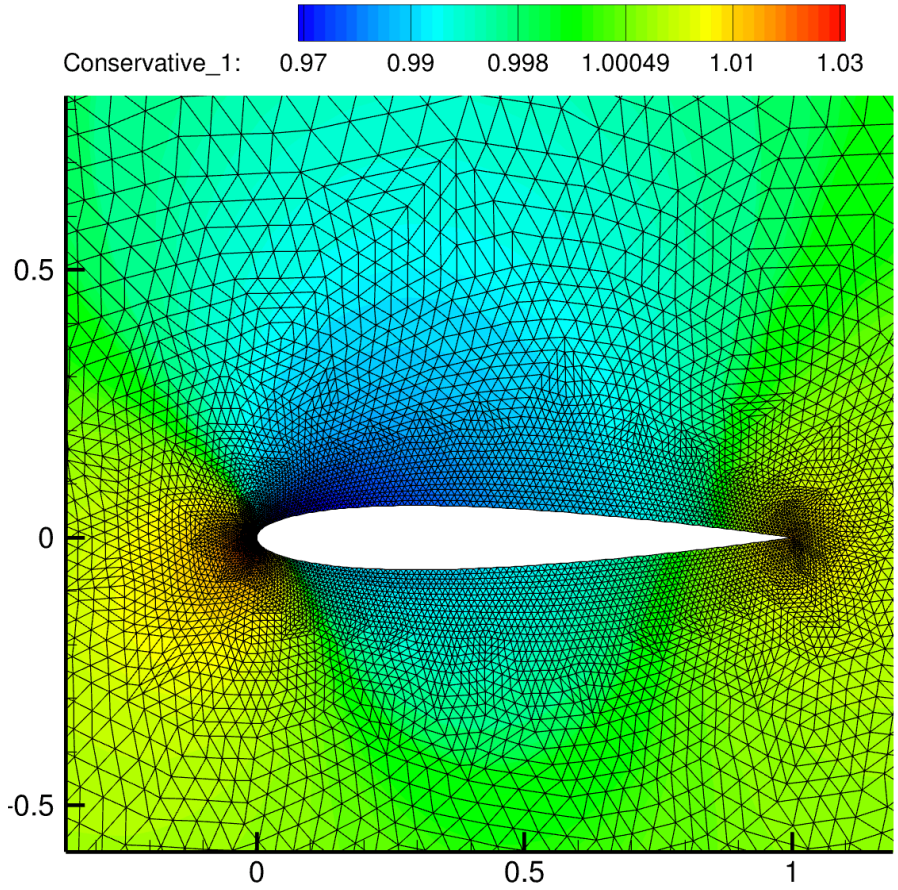


Results, inviscid

Visualization via linear sub-elements



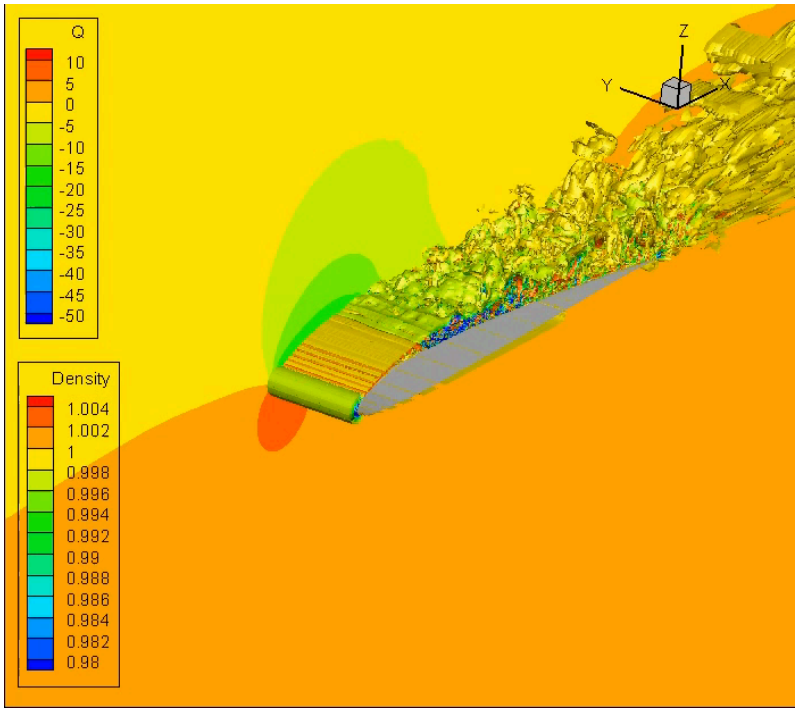
p = 3, tets

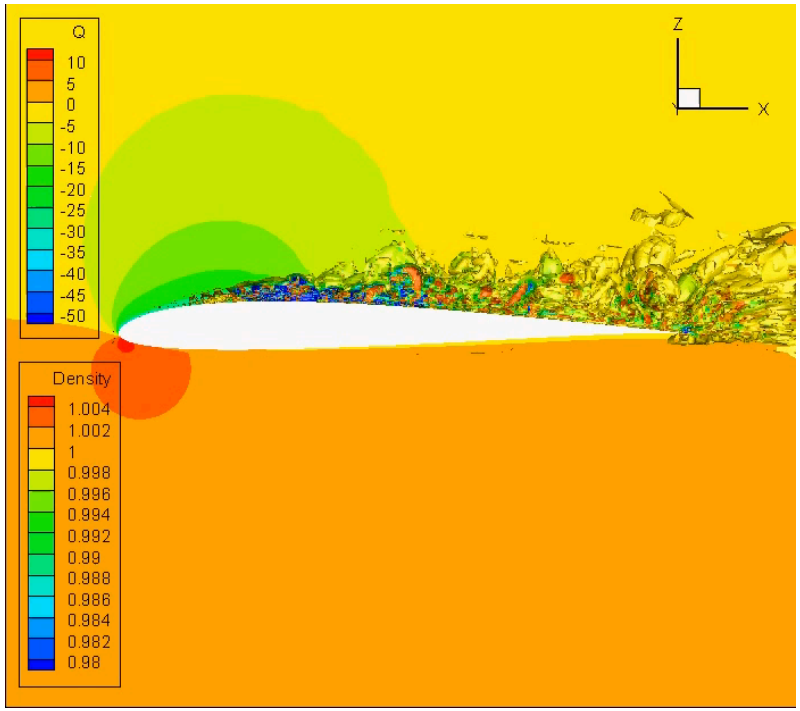


p = 4, triangles

Results, viscous

Implicit LES, SD 7003 (Reynolds = 60,000)





p = 4, hexahedra

Work to be done

- Thorough V&V of the implementation
- Finish shock capturing
- Finish LES wall models
- LES statistics (common to DDES and URANS)
- Improve boundary conditions (non-reflective)
- Performance optimization, including OpenMP parallelization
- Grid motion, sliding mesh interfaces
- Grid sequencing
- High-order grid generation