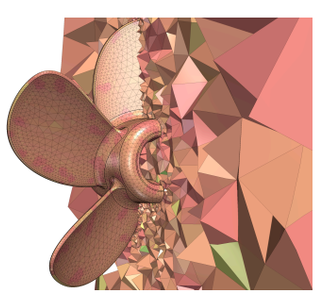
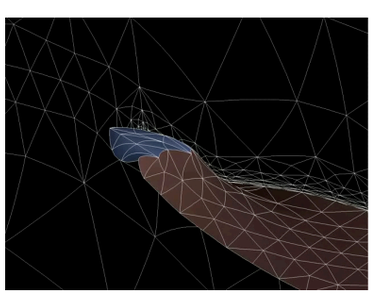
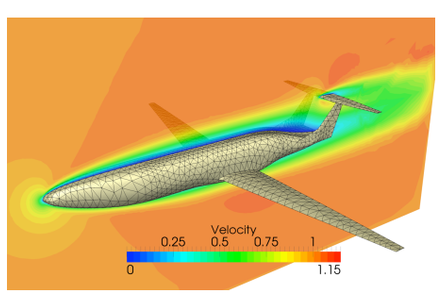
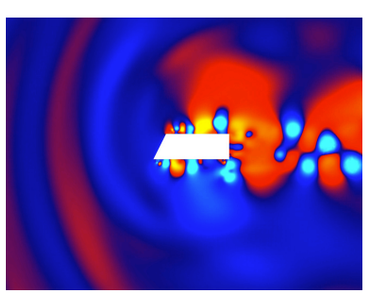
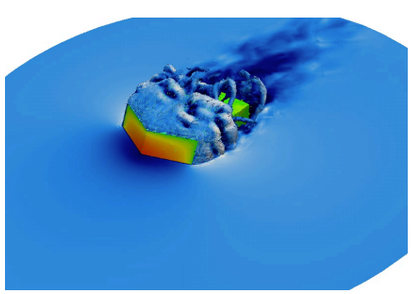
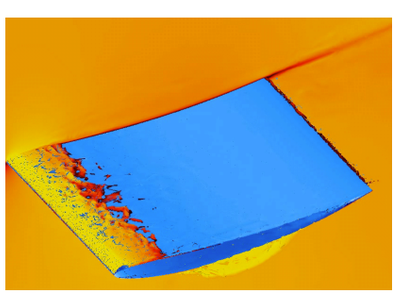
Paralellization of the hybridizable discontinuous Galerkin method

Xevi Roca
xevi.roca@bsc.es
http://web.mit.edu/xeviroca/www/index.html

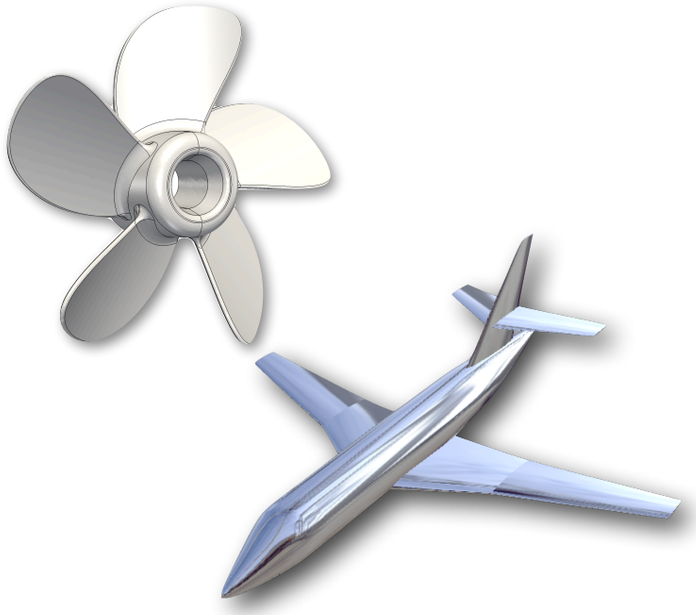
Geometry and Meshing for Simulation Team
Computer Applications in Science & Engineering (CASE)
Barcelona Supercomputing Center
Barcelona, Spain

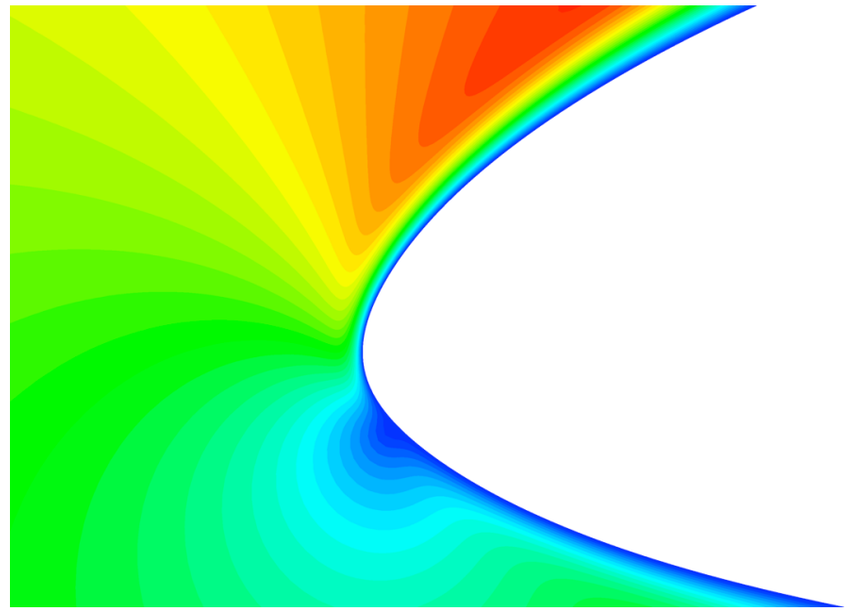




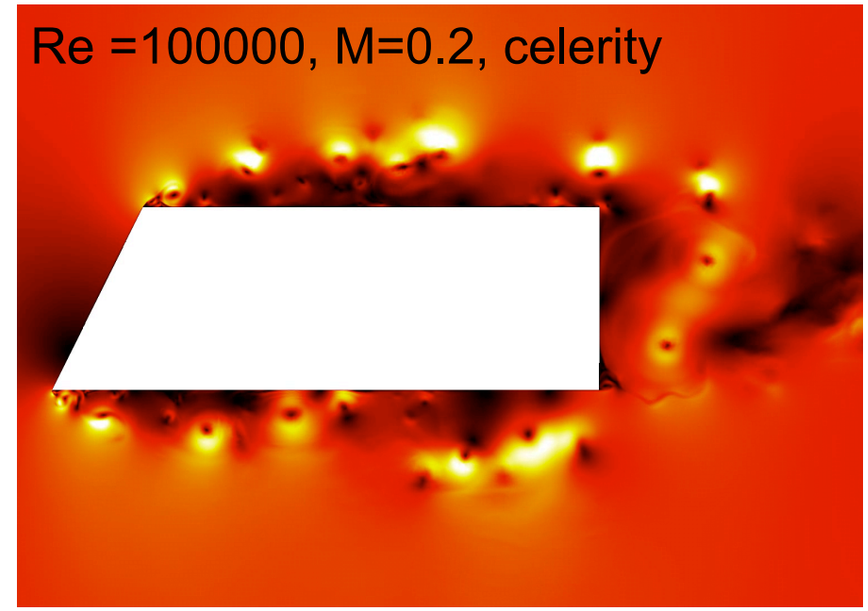




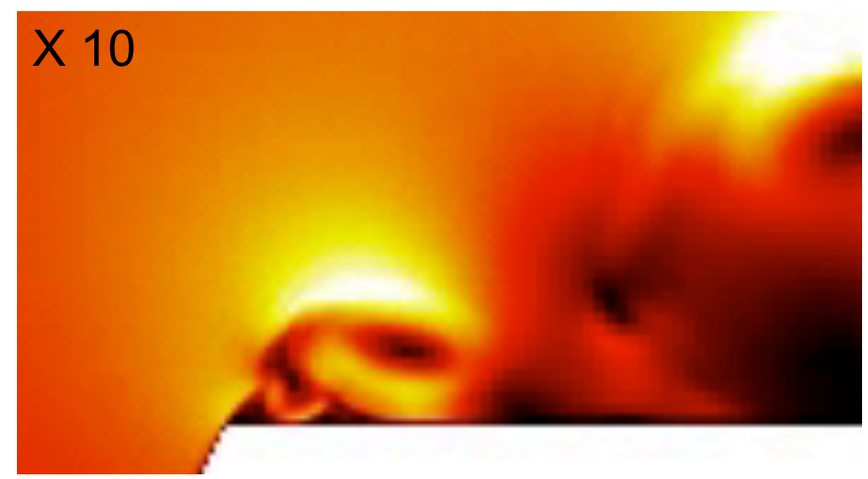
Motivation

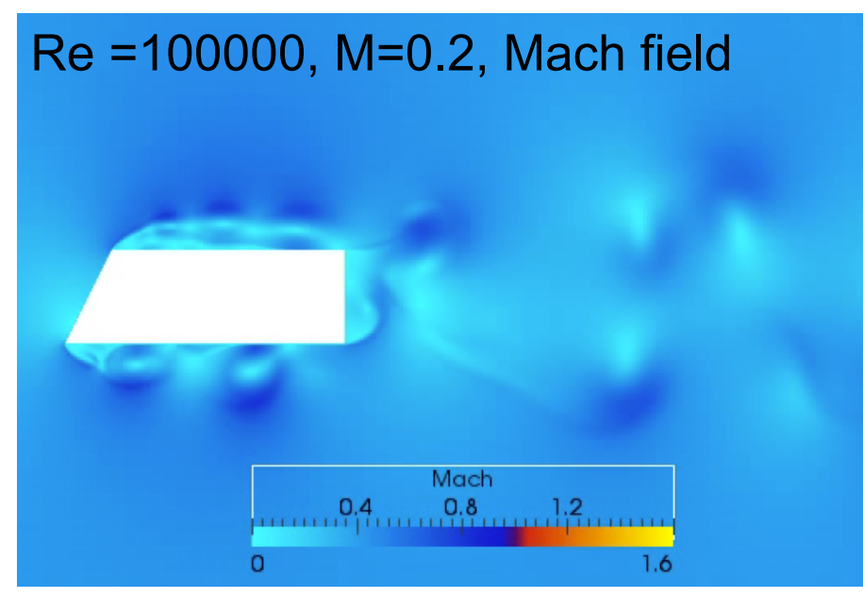
Challenges: high-fidelity simulation









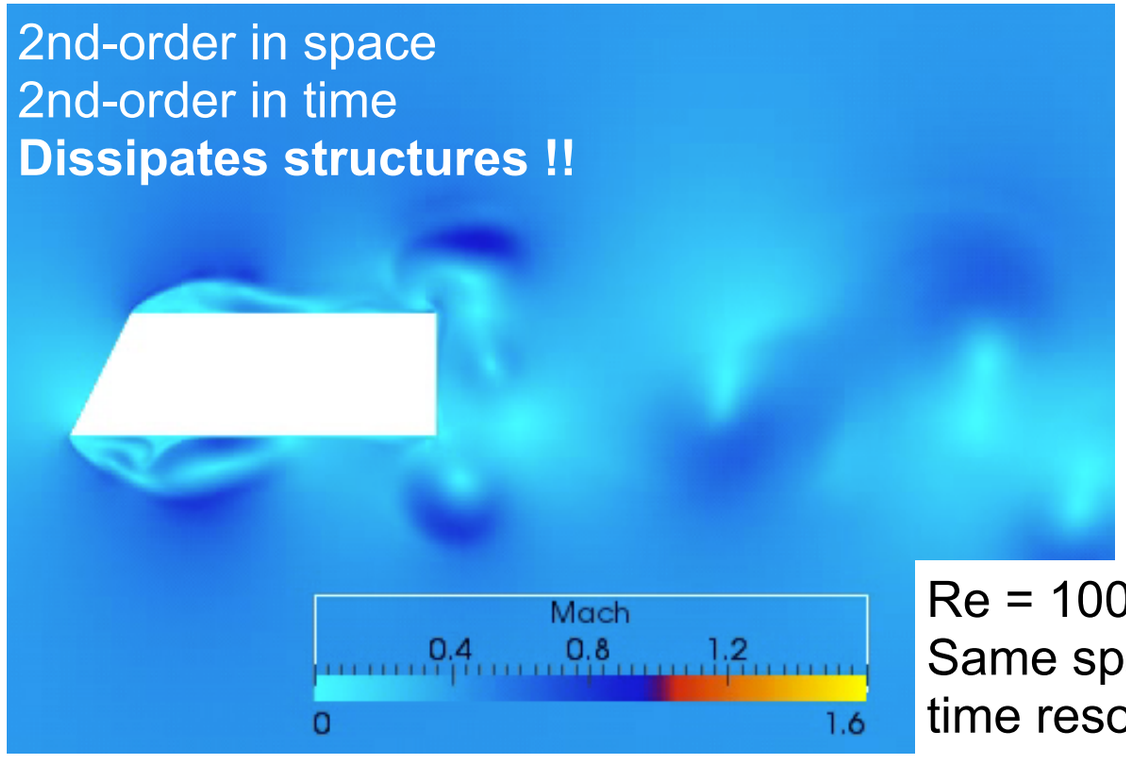


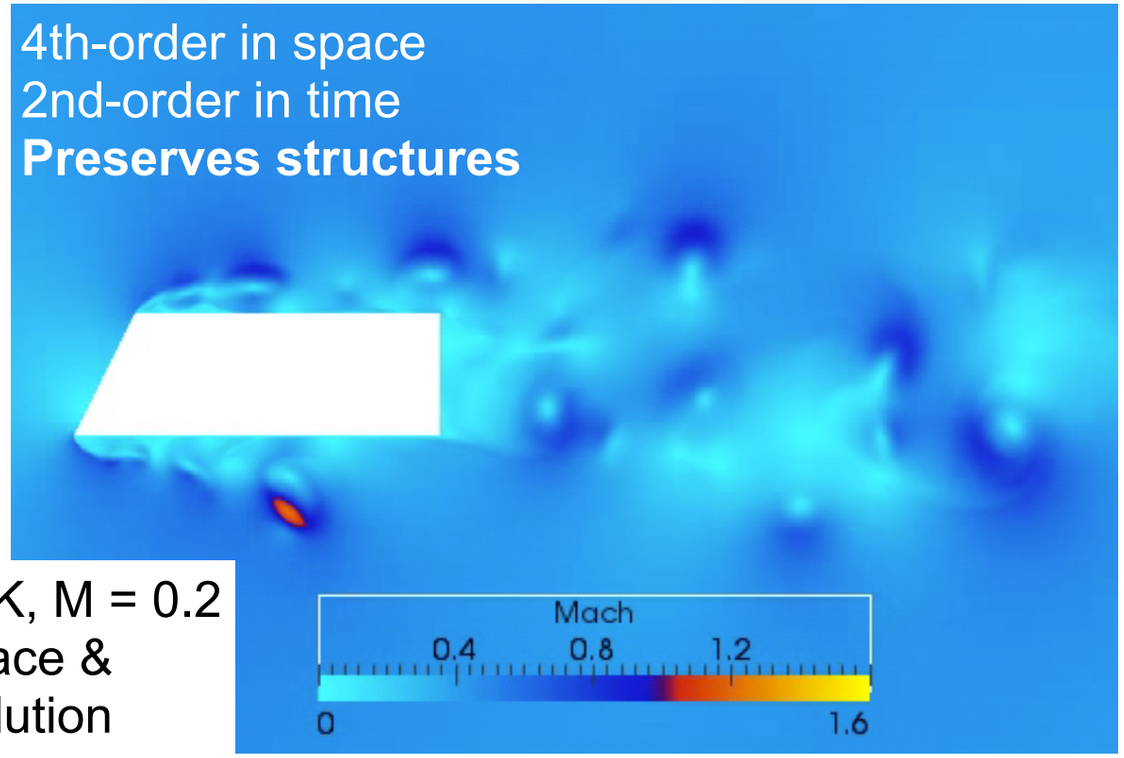
- **Geometrical**: accuracy, complexity, curved boundaries, sharp features …
- **Physical**: accurate model, boundary layers, separation, turbulence, …
- **Numerical**: accuracy, dissipation, dispersion, condition numbers, convergence, …
- **Computational**: rate of convergence, FLOPS, communication, parallelization, memory footprint, …

High-order: low dissipation & dispersion

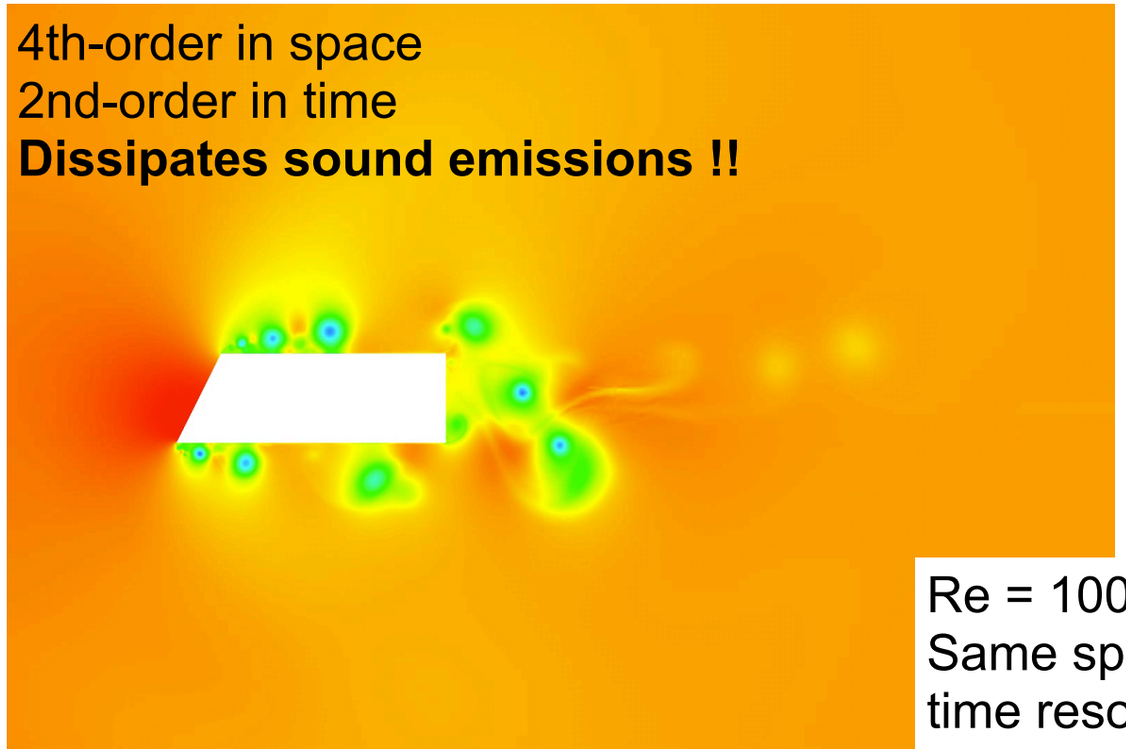
with N.C. Nguyen & J. Peraire

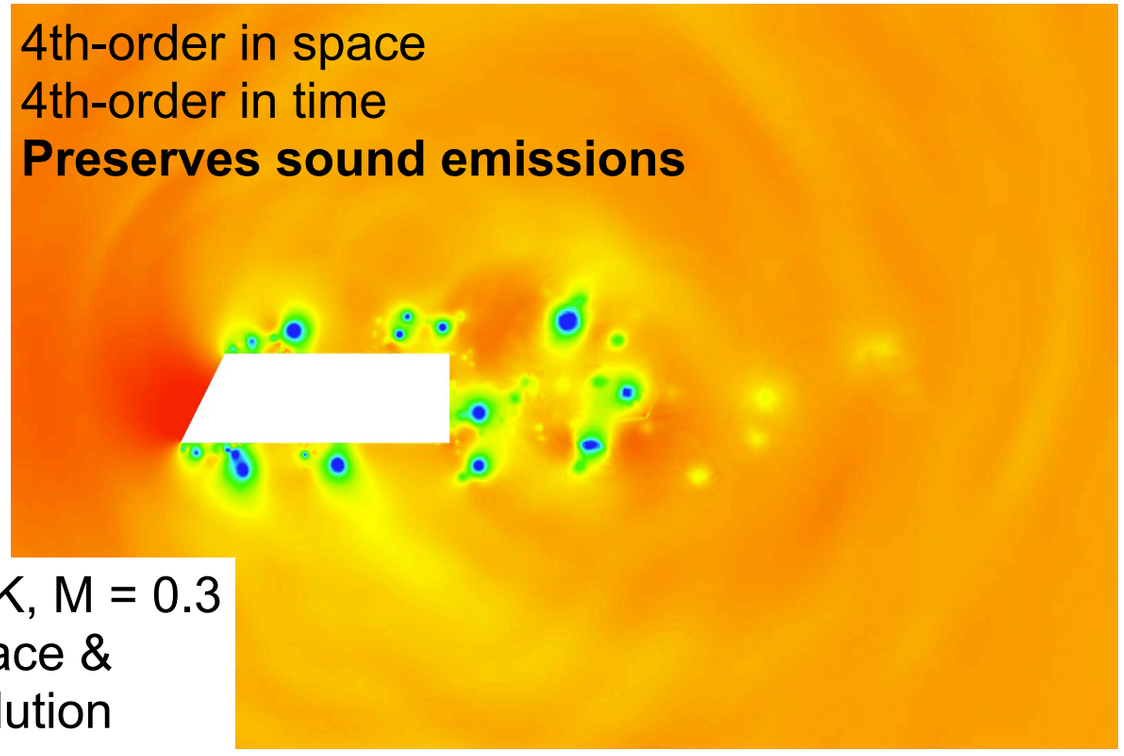
- Example: Compressible NS & Implicit Large Eddy Sim. (ILES) & high-order & HDG





Re = 100K, M = 0.2
Same space &
time resolution



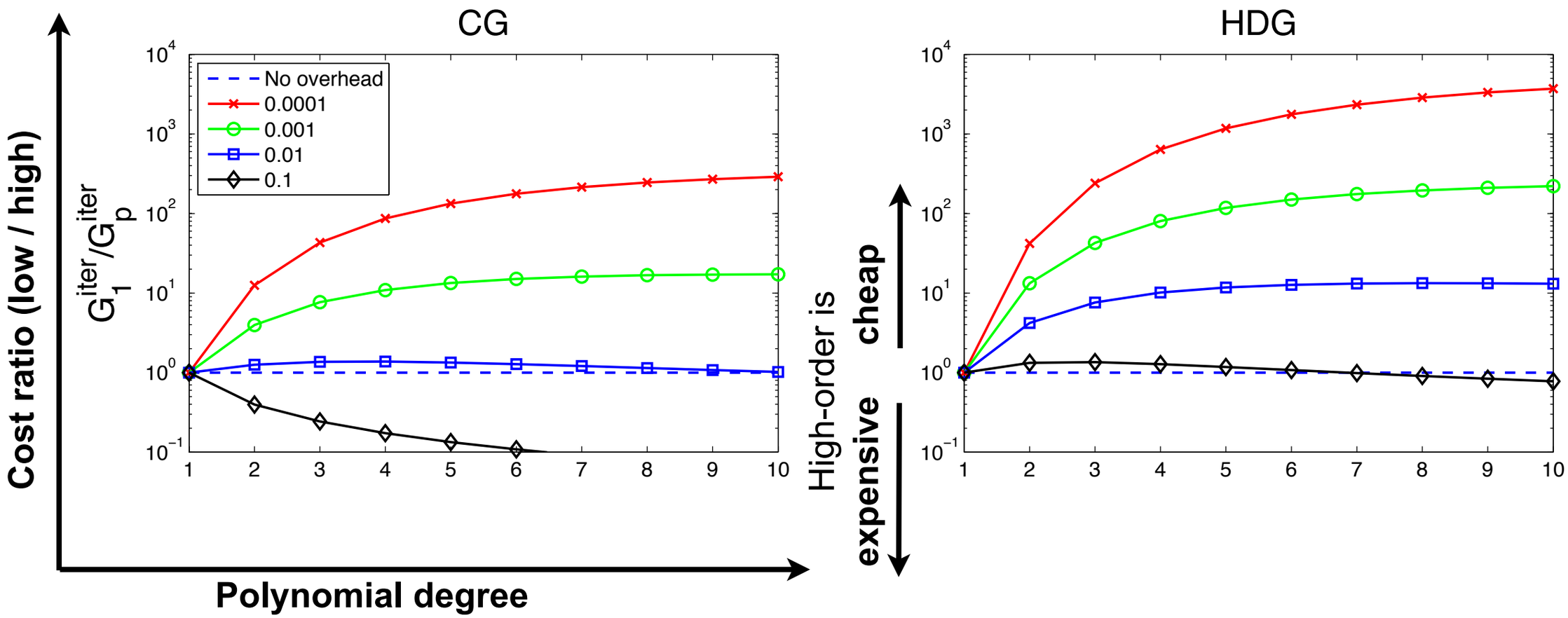


Re = 100K, M = 0.3
Same space &
time resolution

High vs. low-order: higher accuracy for a lower cost

- **Conditions**: Smooth solution, Galerkin & implicit time stepping
- **Computational cost**: number of floating point operations
- **Result**: high-order is cheaper for higher accuracies

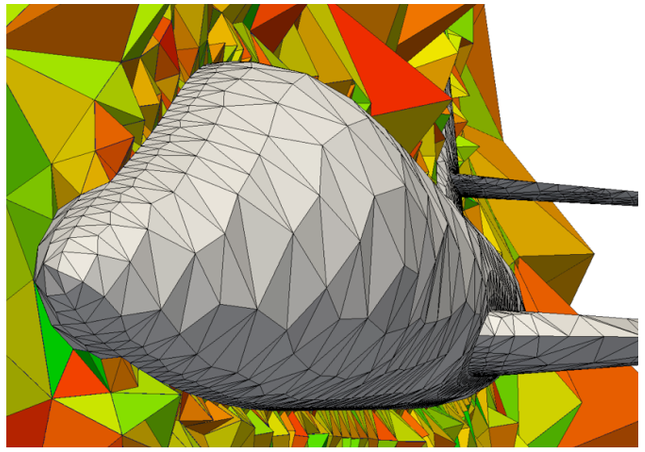
[Huerta, Angeloski, Roca, Peraire, IJNME, 2013]



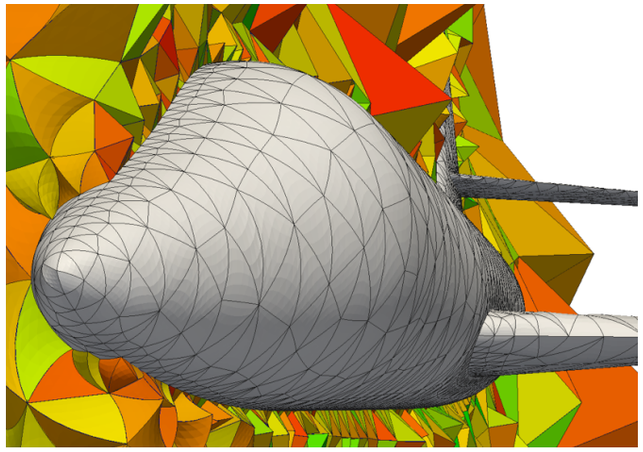
Example (Linear system). Cost ratio of a GMRES iteration pre-conditioned with ILU(0) for different degrees and accuracies

Curved boundaries & mesh quality are critical

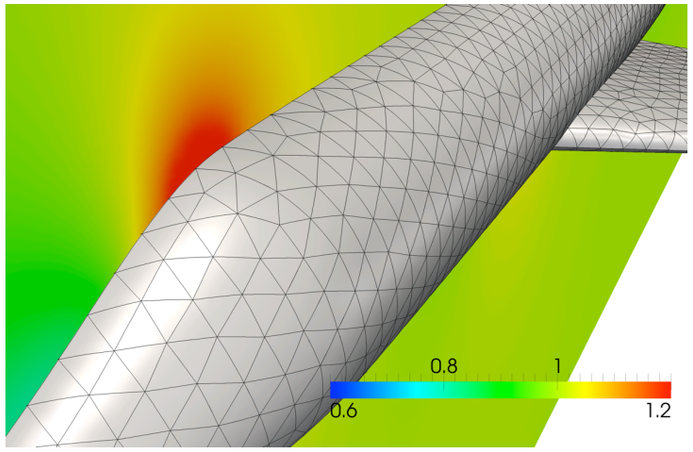
- 5th order approximation for inviscid flow: $\alpha = 0, M_\infty = 0.6, p = 4$
  - Straight-sided impedes convergence: artificial separation & entropy
    (as elucidated first for 2D cases by Bassi & Rebay'97)



Straight-sided: no convergence

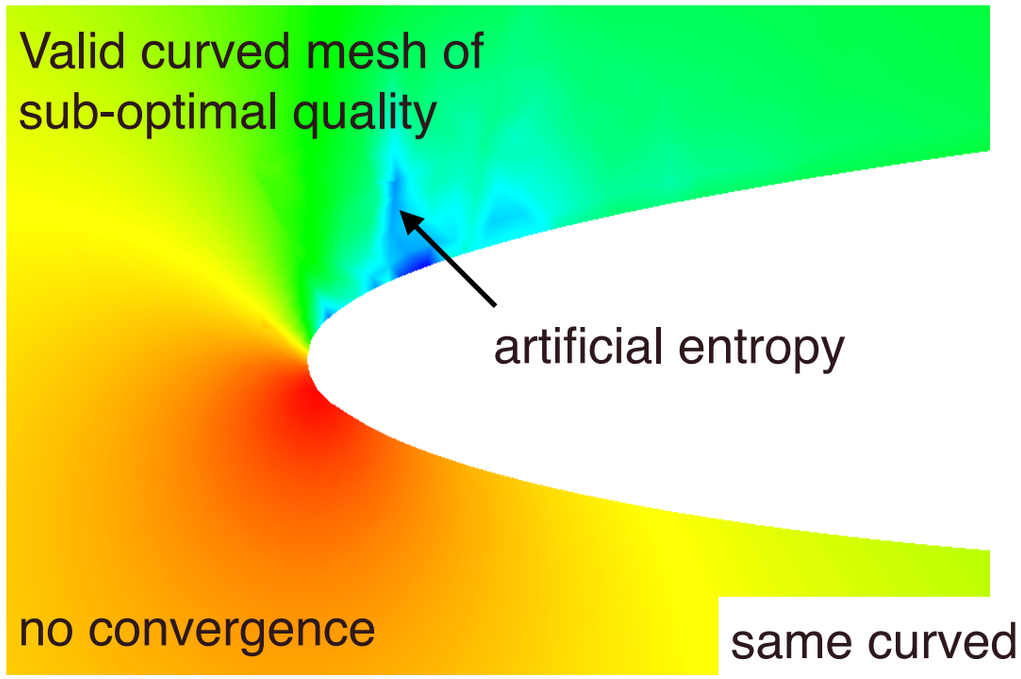


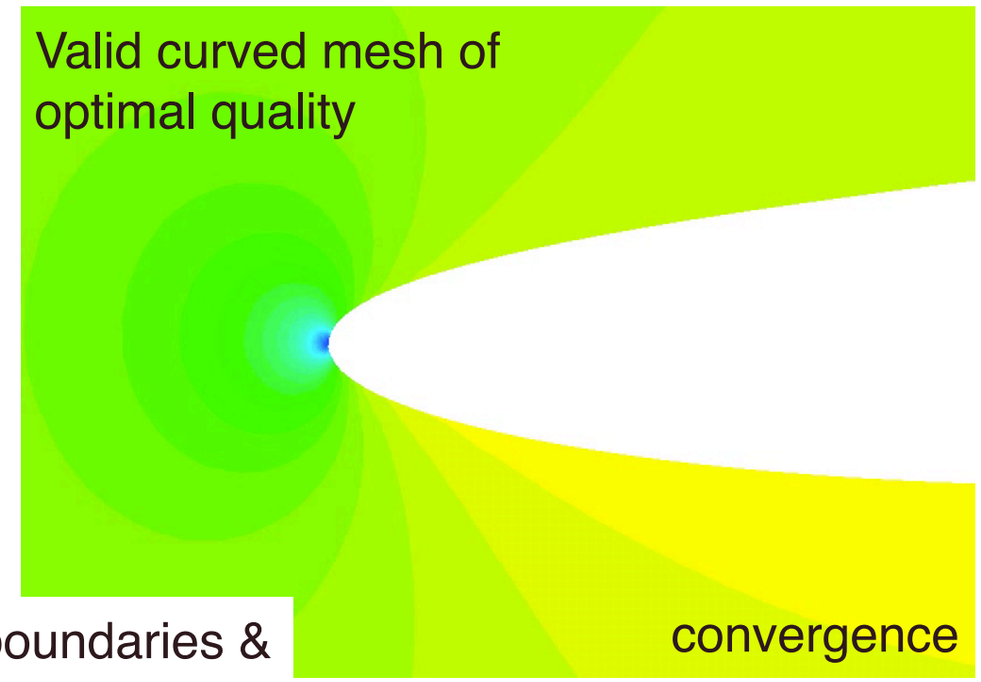
Curved: convergence



Curved: velocity magnitude

  - Low-quality can impede convergence: shape, smoothness, …





same curved boundaries &
mesh topology

Outline

- Motivation: high-fidelity simulation
- The HDG method: suitability to parallelization
- Parallel computing overview
- HDG linear systems
- Parallel HDG solver
- Numerical examples

The HDG method:

suitability to parallelization

The HDG method

Derivation of the HDG method for second-order partial differential equations in conservative form:

- First-order problem: introduce the gradient of the conserved quantities
- Finite element spaces: discontinuous on the elements and faces
- Inner product: on the element and face spaces
- Weak form: Galerkin projection and integration by parts

We will highlight the relation between the derivation and the *parallelization* of the method.

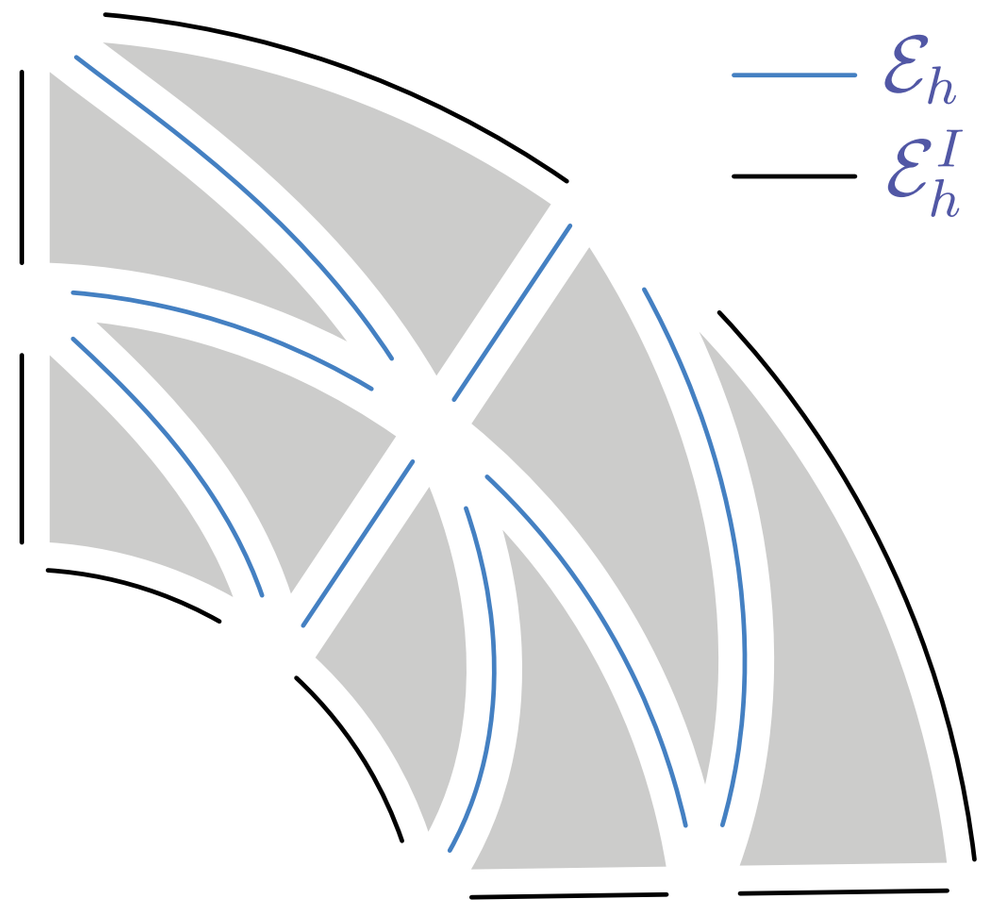
Mesh

For a domain Ω, the mesh is composed by discontinuous and curved:

- $\mathcal{T}_h$: elements,
- $\mathcal{E}_h$: faces

Let us denote:

- $\mathcal{E}_h^I$: interior faces,
- $\mathcal{E}_h^B$: boundary faces.



Parallel loop. The mesh elements correspond to element-wise loops.

Approximation spaces: illustration

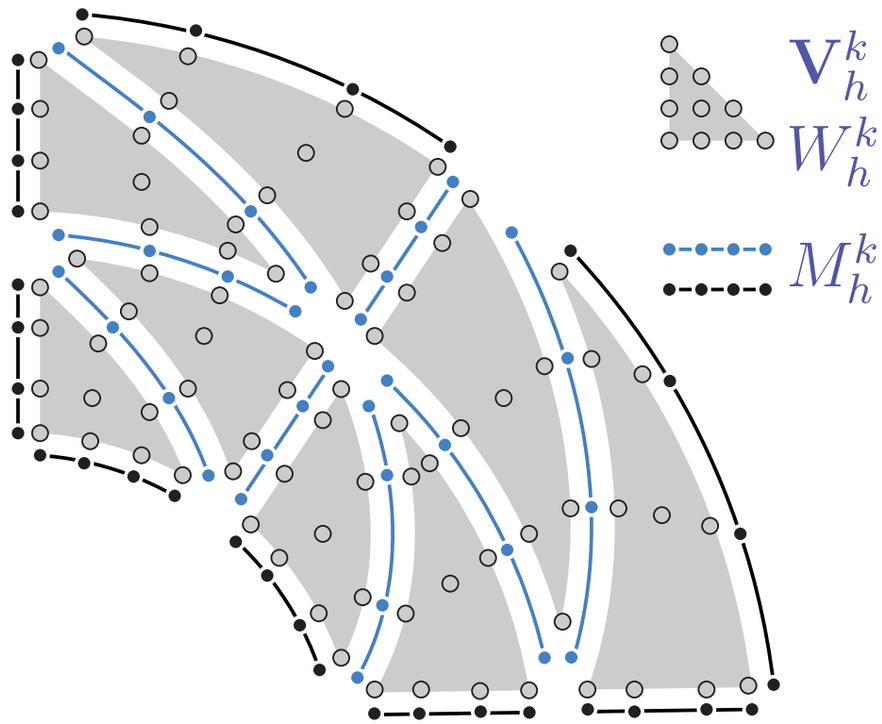


Figure : Element and face nodes for a curved mesh with $k = 3$.

Functions in $\mathcal{M}_h^k$ and $\boldsymbol{\mathcal{M}}_h^k$ are continuous inside the faces $F \in \mathcal{E}_h$ and discontinuous at their borders.

Parallel loop. The local problems are independent since nodal basis functions on an element are connected only to the basis functions of the surrounding faces.

Inner products: element spaces

We introduce the inner products for functions in the element spaces:

$$\left(w^1, w^2\right)_{\mathcal{T}_h} := \sum_{K \in \mathcal{T}_h} \left(w^1, w^2\right)_K, \text{ for } w^1, w^2 \in \mathcal{W}_h^k,$$

$$\left(\boldsymbol{w}^1, \boldsymbol{w}^2\right)_{\mathcal{T}_h} := \sum_{i=1}^{n_c} \left(\boldsymbol{w}_i^1, \boldsymbol{w}_i^2\right)_{\mathcal{T}_h}, \text{ for } \boldsymbol{w}^1, \boldsymbol{w}^2 \in \boldsymbol{\mathcal{W}}_h^k,$$

$$\left(\boldsymbol{v}^1, \boldsymbol{v}^2\right)_{\mathcal{T}_h} := \sum_{i=1}^{n_c} \sum_{j=1}^{d} \left(\boldsymbol{v}_{ij}^1, \boldsymbol{v}_{ij}^2\right)_{\mathcal{T}_h}, \text{ for } \boldsymbol{v}^1, \boldsymbol{v}^2 \in \boldsymbol{\mathcal{V}}_h^k,$$

where

$$\left(w^1, w^2\right)_K := \int_K w^1 w^2.$$

Parallel loop. By definition, the element inner products and their derivatives are computed element-wise.

Inner products: face spaces

We introduce the inner products for functions in the face spaces:

$$\left\langle \mu^1, \mu^2 \right\rangle_{\partial \mathcal{T}_h} := \sum_{K \in \mathcal{T}_h} \left\langle \mu^1, \mu^2 \right\rangle_{\partial K}, \text{ for } \mu^1, \mu^2 \in \mathcal{M}_h^k,$$

$$\left\langle \boldsymbol{\mu}^1, \boldsymbol{\mu}^2 \right\rangle_{\partial \mathcal{T}_h} := \sum_{i=1}^{n_c} \left\langle \boldsymbol{\mu}^1, \boldsymbol{\mu}^2 \right\rangle_{\partial \mathcal{T}_h}, \text{ for } \boldsymbol{\mu}^1, \boldsymbol{\mu}^2 \in \boldsymbol{\mathcal{M}}_h^k,$$

where

$$\left\langle \mu^1, \mu^2 \right\rangle_{\partial K} := \int_{\partial K} \mu^1 \mu^2.$$

Parallel loop. By definition, the face inner products and their derivatives are computed element-wise.

HDG method

Seeks a solution $(\boldsymbol{q}_h, \boldsymbol{u}_h, \hat{\boldsymbol{u}}_h) \in \boldsymbol{\mathcal{V}}_h^k \times \boldsymbol{\mathcal{W}}_h^k \times \boldsymbol{\mathcal{M}}_h^k$ such that $\forall \boldsymbol{v} \in \boldsymbol{\mathcal{V}}_h^k$, $\forall \boldsymbol{w} \in \boldsymbol{\mathcal{W}}_h^k$, and $\forall \boldsymbol{\mu} \in \boldsymbol{\mathcal{M}}_h^k$:

$$\mathbf{r_q} := (\boldsymbol{q}_h, \boldsymbol{v})_{\mathcal{T}_h} + (\boldsymbol{u}_h, \nabla \cdot \boldsymbol{v})_{\mathcal{T}_h} - \langle \hat{\boldsymbol{u}}_h, \boldsymbol{v} \cdot \boldsymbol{n} \rangle_{\partial \mathcal{T}_h} = 0,$$

$$\mathbf{r_u} := \alpha\, (\boldsymbol{u}_h, \boldsymbol{w})_{\mathcal{T}_h} - (\boldsymbol{F}, \nabla \boldsymbol{w})_{\mathcal{T}_h} + \left\langle \hat{\boldsymbol{F}} \cdot \boldsymbol{n}, \boldsymbol{w} \right\rangle_{\partial \mathcal{T}_h} - (s, \boldsymbol{w})_{\mathcal{T}_h} = 0,$$

$$\mathbf{r_{\hat{u}}} := \left\langle \hat{\boldsymbol{F}} \cdot \boldsymbol{n}, \boldsymbol{\mu} \right\rangle_{\partial \mathcal{T}_h \setminus \partial \Omega} + \left\langle \hat{\boldsymbol{F}}^b \cdot \boldsymbol{n}, \boldsymbol{\mu} \right\rangle_{\partial \Omega} - \langle g, \boldsymbol{\mu} \rangle_{\partial \Omega} = 0,$$

where $\hat{\boldsymbol{F}} = \hat{\boldsymbol{F}}(\boldsymbol{q}_h, \boldsymbol{u}_h, \hat{\boldsymbol{u}}_h)$ (numerical flux), and $\hat{\boldsymbol{F}}^b = \hat{\boldsymbol{F}}^b(\boldsymbol{q}_h, \boldsymbol{u}_h, \hat{\boldsymbol{u}}_h)$ (boundary conditions on Γ_D and Γ_N).

The HDG method:

linearization

Residual system

The weak form corresponds to the non-linear problem (eventually linear):

$$\mathbf{r}_{\mathbf{q}}(\boldsymbol{q}_h, \boldsymbol{u}_h, \hat{\boldsymbol{u}}_h) = \mathbf{0}, \tag{3}$$

$$\mathbf{r}_{\mathbf{u}}(\boldsymbol{q}_h, \boldsymbol{u}_h, \hat{\boldsymbol{u}}_h) = \mathbf{0}, \tag{4}$$

$$\mathbf{r}_{\hat{\mathbf{u}}}(\boldsymbol{q}_h, \boldsymbol{u}_h, \hat{\boldsymbol{u}}_h) = \mathbf{0}, \tag{5}$$

- Equations (3) and (4): parameterize $(\boldsymbol{q}_h, \boldsymbol{u}_h)$ in terms of $\hat{\boldsymbol{u}}_h$ element-by-element (locally)
- Equation (5): continuity of $\hat{\boldsymbol{F}}$ (globally) and boundary conditions ($\hat{\boldsymbol{F}}^b$)

Parallel loop. Grouping by elements, Equations (3) and (4) correspond to element-wise independent problems.

Linearization: Newton's method

To solve the residual system we use Newton's method. At iteration $k+1$, we have to solve the linear system:

$$\left[\begin{array}{c|c|c} \mathbf{J_{qq}} & \mathbf{J_{qu}} & \mathbf{J_{q\hat{u}}} \\ \hline \mathbf{J_{uq}} & \mathbf{J_{uu}} & \mathbf{J_{u\hat{u}}} \\ \hline \mathbf{J_{\hat{u}q}} & \mathbf{J_{\hat{u}u}} & \mathbf{J_{\hat{u}\hat{u}}} \end{array}\right]_k \left[\begin{array}{c} \delta\mathbf{q} \\ \hline \delta\mathbf{u} \\ \hline \delta\hat{\mathbf{u}} \end{array}\right]_{k+1} = \left[\begin{array}{c} -\mathbf{r_q} \\ \hline -\mathbf{r_u} \\ \hline -\mathbf{r_{\hat{u}}} \end{array}\right]_k , \tag{6}$$

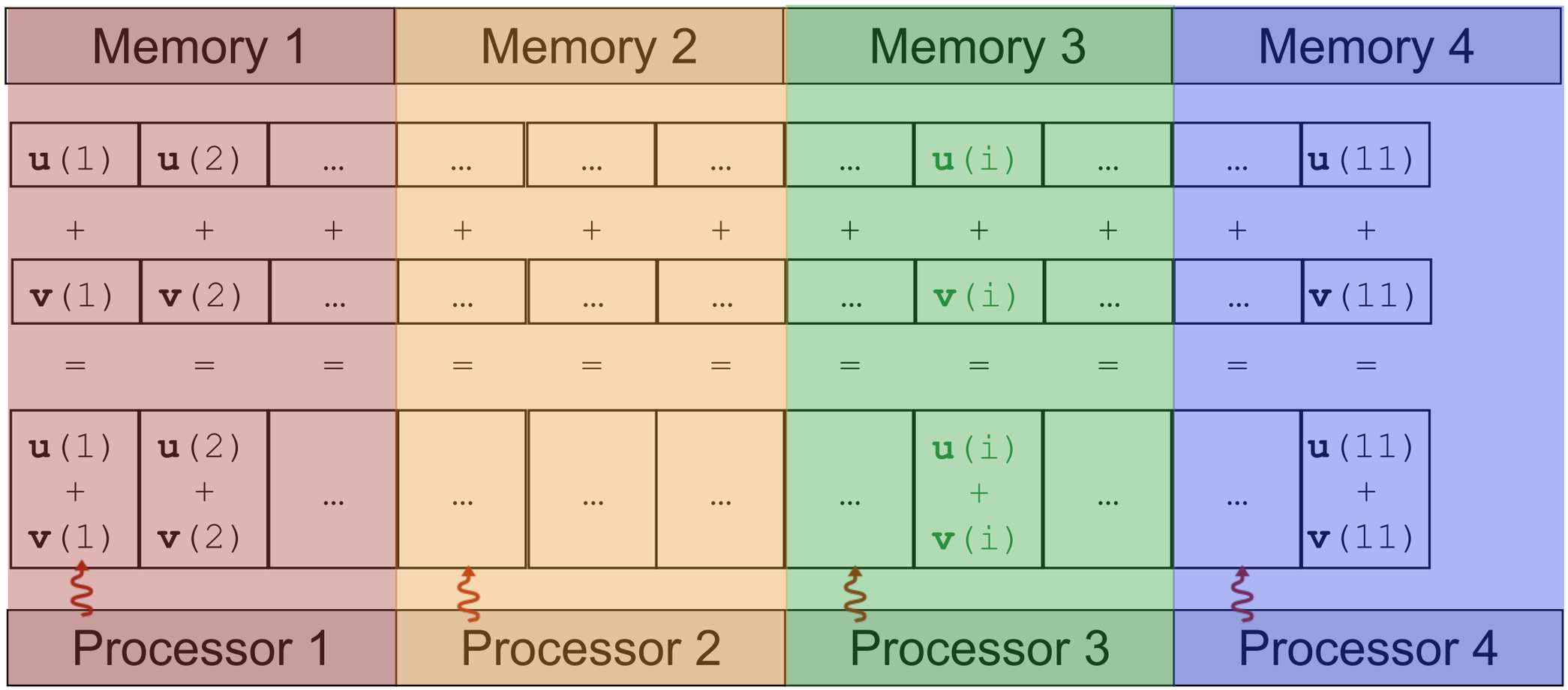
where $\delta\mathbf{q}$, $\delta\mathbf{u}$, and $\delta\hat{\mathbf{u}}$ are the degrees of freedom for $\boldsymbol{q}_h$, $\boldsymbol{u}_h$, and $\hat{\boldsymbol{u}}_h$. To solve the system, we express $\delta\mathbf{q}$ and $\delta\mathbf{u}$ in terms of $\delta\hat{\mathbf{u}}$:

$$\left[\begin{array}{c} \delta\mathbf{q} \\ \hline \delta\mathbf{u} \end{array}\right] = \left[\begin{array}{c} \delta\mathbf{q}(\delta\hat{\mathbf{u}}) \\ \hline \delta\mathbf{u}(\delta\hat{\mathbf{u}}) \end{array}\right] = \left[\begin{array}{c|c} \mathbf{J_{qq}} & \mathbf{J_{qu}} \\ \hline \mathbf{J_{uq}} & \mathbf{J_{uu}} \end{array}\right]^{-1} \left(\left[\begin{array}{c} -\mathbf{r_q} \\ \hline -\mathbf{r_u} \end{array}\right] - \left[\begin{array}{c} \mathbf{J_{q\hat{u}}} \\ \hline \mathbf{J_{u\hat{u}}} \end{array}\right] \delta\hat{\mathbf{u}} \right).$$

Parallel computing overview

Parallelization with a cluster: distributed memory

- Several processors computing in parallel and sharing data through messages
- **Basic idea**: point-to-point send and receive
  - Scatter the vector in blocks of components in different computers
  - Each processor computes a block



Explicit and implicit parallelization

- **Explicit parallelization**: write the code to perform the tasks in parallel (OpenMP / CUDA / MPI)
  - **Example**: vector addition code for shared memory, vector processor and distributed memory
- **Implicit parallelization**: cast a piece of code to a primitive that runs in parallel
  - **Examples**:
    - Vector-vector, matrix-vector, and matrix-matrix operations (BLAS)
    - Solvers for sparse linear systems: direct and iterative

Implicit parallelization: available tools

- Multi-threaded basic linear algebra systems (BLAS): ATLAS, MKL, gotoBLAS, ...
  `http://math-atlas.sourceforge.net/`
  `http://software.intel.com/en-us/intel-mkl`

- Multi-threaded direct solvers: UMFPACK, Pardiso, ...
  `http://www.cise.ufl.edu/research/sparse/umfpack/`
  `http://www.pardiso-project.org/`

- Distributed iterative solvers: PETSc, Trilinos, ... (**requires MPI coding**)
  `http://www.mcs.anl.gov/petsc/`
  `http://trilinos.sandia.gov/`

- MATLAB is multi-threaded:
  `maxNumCompThreads(numThreads)`
  - shared memory matrix-matrix product: `C = A*B`
  - shared memory sparse direct solver: `x = A\b`

HDG linear systems:

local and global problems

Linear system in terms of $\delta\hat{\mathbf{u}}$ (global problem)

Eliminating $\delta\mathbf{q}$ and $\delta\mathbf{u}$ from (6), we obtain the linear system:

$$\mathbf{H}\delta\hat{\mathbf{u}} = \mathbf{r}, \tag{7}$$

where

$$\mathbf{H} = \left[\ \mathbf{J}_{\hat{\mathbf{u}}\hat{\mathbf{u}}}\ \right] - \left[\ \mathbf{J}_{\hat{\mathbf{u}}\mathbf{q}} \,\middle|\, \mathbf{J}_{\hat{\mathbf{u}}\mathbf{u}}\ \right] \left[\begin{array}{c|c} \mathbf{J}_{\mathbf{qq}} & \mathbf{J}_{\mathbf{qu}} \\ \hline \mathbf{J}_{\mathbf{uq}} & \mathbf{J}_{\mathbf{uu}} \end{array}\right]^{-1} \left[\begin{array}{c} \mathbf{J}_{\mathbf{q}\hat{\mathbf{u}}} \\ \hline \mathbf{J}_{\mathbf{u}\hat{\mathbf{u}}} \end{array}\right], \tag{8}$$

$$\mathbf{r} = \left[-\mathbf{r}_{\hat{\mathbf{u}}}\right] - \left[\ \mathbf{J}_{\hat{\mathbf{u}}\mathbf{q}} \,\middle|\, \mathbf{J}_{\hat{\mathbf{u}}\mathbf{u}}\ \right] \left[\begin{array}{c|c} \mathbf{J}_{\mathbf{qq}} & \mathbf{J}_{\mathbf{qu}} \\ \hline \mathbf{J}_{\mathbf{uq}} & \mathbf{J}_{\mathbf{uu}} \end{array}\right]^{-1} \left[\begin{array}{c} -\mathbf{r}_{\mathbf{q}} \\ \hline -\mathbf{r}_{\mathbf{u}} \end{array}\right]. \tag{9}$$

The system (7) allows to obtain $\delta\hat{\mathbf{u}}$.

Parallel linear solver. Solve the sparse linear system (7) with a solver that is: direct and multi-threaded, or iterative and distributed.

Element-wise matrix inversion (local problem)

To create $\mathbf{H}$ and $\mathbf{r}$ we have to compute the inverse of

$$\left[\begin{array}{c|c} \mathbf{J}_{\mathbf{qq}} & \mathbf{J}_{\mathbf{qu}} \\ \hline \mathbf{J}_{\mathbf{uq}} & \mathbf{J}_{\mathbf{uu}} \end{array}\right].$$

Grouping the degrees of freedom $\delta\mathbf{q}$ and $\delta\mathbf{u}$ by elements, the matrix becomes block diagonal.

Parallel for. The matrix can be inverted independently for each element K in $\mathcal{T}_h$:

$$\left[\begin{array}{c|c} \mathbf{J}^K_{\mathbf{qq}} & \mathbf{J}^K_{\mathbf{qu}} \\ \hline \mathbf{J}^K_{\mathbf{uq}} & \mathbf{J}^K_{\mathbf{uu}} \end{array}\right]^{-1}. \tag{10}$$

Creation of $\mathbf{H}$ and $\mathbf{r}$: element contributions

Parallel for. For each element K in $\mathcal{T}_h$:

- Compute the elemental: matrices $\mathbf{J}^K_{\mathbf{qq}}$, $\mathbf{J}^K_{\mathbf{qu}}$, $\mathbf{J}^K_{\mathbf{q\hat{u}}}$, $\mathbf{J}^K_{\mathbf{uq}}$, $\mathbf{J}^K_{\mathbf{uu}}$, $\mathbf{J}^K_{\mathbf{u\hat{u}}}$, $\mathbf{J}^K_{\mathbf{\hat{u}q}}$, $\mathbf{J}^K_{\mathbf{\hat{u}u}}$, and $\mathbf{J}^K_{\mathbf{\hat{u}\hat{u}}}$; and the vectors $\mathbf{r}^K_{\mathbf{q}}$, $\mathbf{r}^K_{\mathbf{u}}$, and $\mathbf{r}^K_{\mathbf{\hat{u}}}$.
- Compute $\left[\begin{array}{c|c} \mathbf{J}^K_{\mathbf{qq}} & \mathbf{J}^K_{\mathbf{qu}} \\ \hline \mathbf{J}^K_{\mathbf{uq}} & \mathbf{J}^K_{\mathbf{uu}} \end{array}\right]^{-1}$
- Compute the element contributions to $\mathbf{H}$ and $\mathbf{r}$:

$$\mathbf{H}^K = \left[\ \mathbf{J}^K_{\mathbf{\hat{u}\hat{u}}}\ \right] - \left[\ \mathbf{J}^K_{\mathbf{\hat{u}q}} \,\middle|\, \mathbf{J}^K_{\mathbf{\hat{u}u}}\ \right] \left[\begin{array}{c|c} \mathbf{J}^K_{\mathbf{qq}} & \mathbf{J}^K_{\mathbf{qu}} \\ \hline \mathbf{J}^K_{\mathbf{uq}} & \mathbf{J}^K_{\mathbf{uu}} \end{array}\right]^{-1} \left[\begin{array}{c} \mathbf{J}^K_{\mathbf{q\hat{u}}} \\ \hline \mathbf{J}^K_{\mathbf{u\hat{u}}} \end{array}\right],$$

$$\mathbf{r}^K = \left[-\mathbf{r}^K_{\mathbf{\hat{u}}}\right] - \left[\ \mathbf{J}^K_{\mathbf{\hat{u}q}} \,\middle|\, \mathbf{J}^K_{\mathbf{\hat{u}u}}\ \right] \left[\begin{array}{c|c} \mathbf{J}^K_{\mathbf{qq}} & \mathbf{J}^K_{\mathbf{qu}} \\ \hline \mathbf{J}^K_{\mathbf{uq}} & \mathbf{J}^K_{\mathbf{uu}} \end{array}\right]^{-1} \left[\begin{array}{c} -\mathbf{r}^K_{\mathbf{q}} \\ \hline -\mathbf{r}^K_{\mathbf{u}} \end{array}\right].$$

Parallel (partially) matrix assembler. Multi-threaded or distributed.

Recover $\delta\mathbf{q}$ and $\delta\mathbf{u}$ from $\delta\hat{\mathbf{u}}$

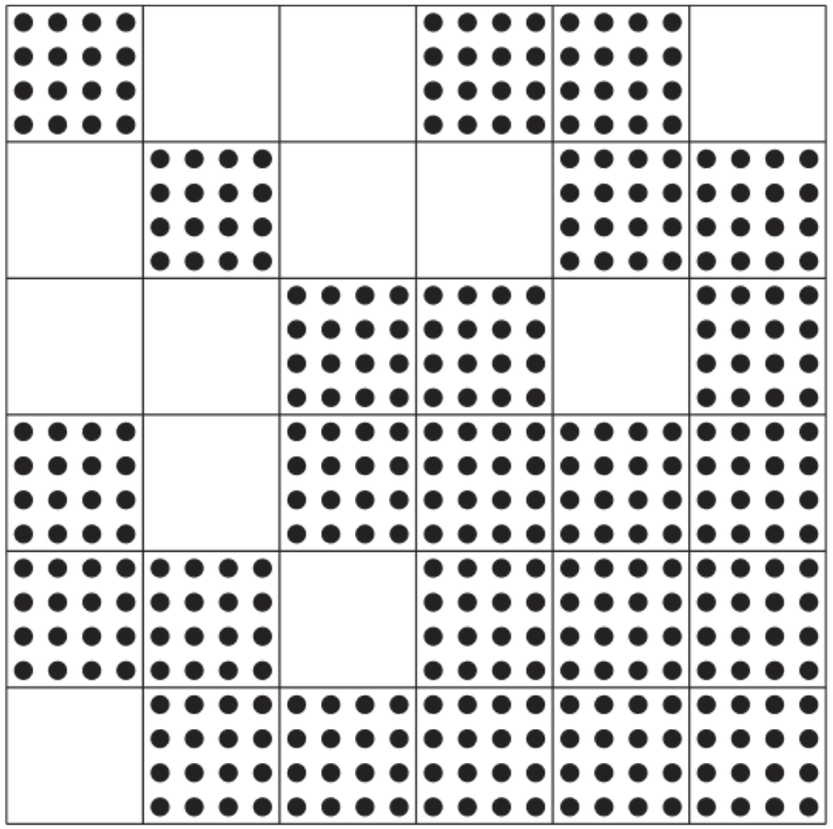
Once we have obtained $\delta\hat{\mathbf{u}}$ from the global linear system, we can compute $\delta\mathbf{q}$ and $\delta\mathbf{u}$.

Parallel for. For each element K in $\mathcal{T}_h$ we compute:

$$\left[\begin{array}{c}\delta\mathbf{q}^K\\ \hline \delta\mathbf{u}^K\end{array}\right] = \left[\begin{array}{c|c}\mathbf{J}^K_{\mathbf{qq}} & \mathbf{J}^K_{\mathbf{qu}}\\ \hline \mathbf{J}^K_{\mathbf{uq}} & \mathbf{J}^K_{\mathbf{uu}}\end{array}\right]^{-1} \left(\left[\begin{array}{c}-\mathbf{r}^K_{\mathbf{q}}\\ \hline -\mathbf{r}^K_{\mathbf{u}}\end{array}\right] - \left[\begin{array}{c}\mathbf{J}^K_{\mathbf{q}\hat{\mathbf{u}}}\\ \hline \mathbf{J}^K_{\mathbf{u}\hat{\mathbf{u}}}\end{array}\right] \delta\hat{\mathbf{u}}^{\partial K}\right). \quad (11)$$

HDG linear systems

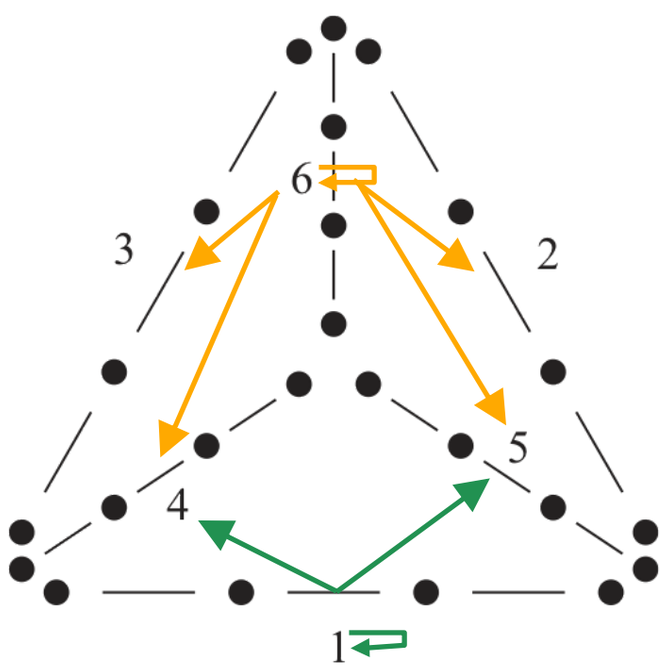
- HDG discretizations lead to linear systems where matrices are structured in sparse blocks:
  - Non-zero (dense)
  - Equal-sized
  - Non-overlapped
  - Constant number of blocks per row



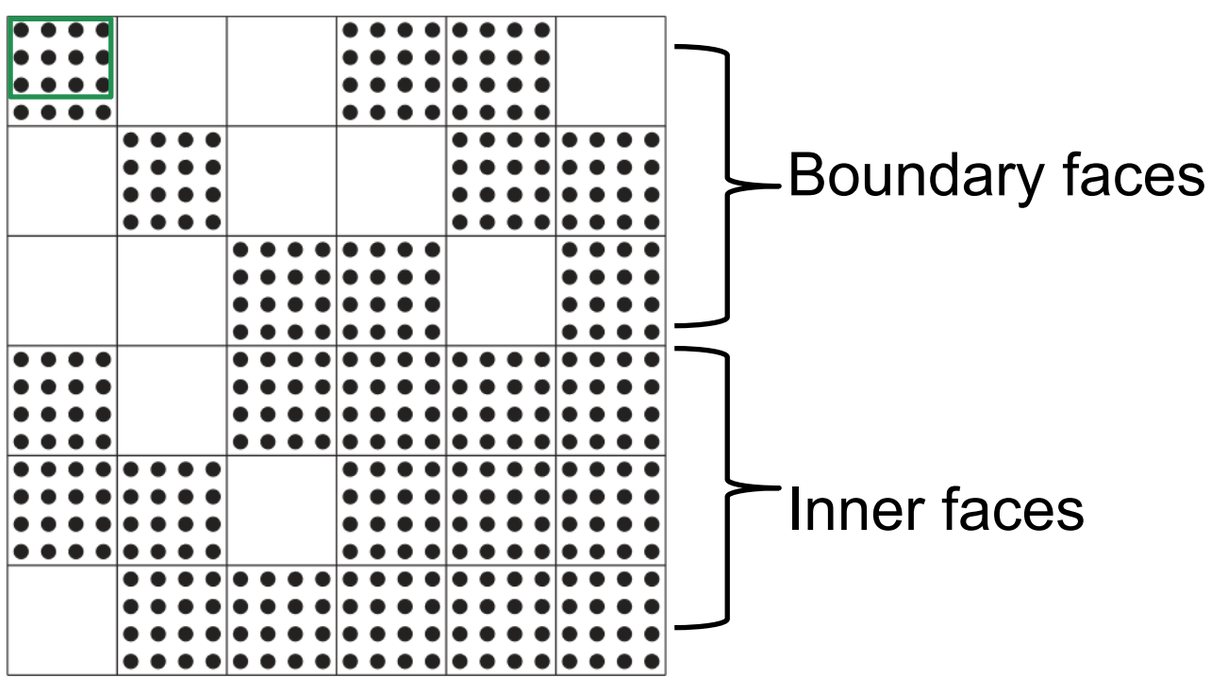
- Large linear systems due to: fine meshes, high-order, solution components, spatial dimensions …
- Iterative methods require fast sparse matrix-vector products (for DG and HDG matrices)

Structure of HDG matrices

- Blocks (#face unknowns) x (#face unknowns)
- Face unknowns are coupled if share an adjacent element
- HDG matrices for triangular (tetrahedral) meshes
  - 5 (7) blocks per row (inner faces)
  - 3 (4) blocks per row (boundary faces)



Mesh faces and points



Matrix structure

Parallel HDG solver (1/2):

parallel pre-conditioner, iterative solver and distribution

Solving HDG linear system: sequential

Algorithm 1: Sequential solve of the HDG linear system

Input: $\mathbf{q}$, $\mathbf{u}$, $\hat{\mathbf{u}}$

Output: $\delta\mathbf{q}$, $\delta\mathbf{u}$, $\delta\hat{\mathbf{u}}$

```
begin Compute matrix H and vector r from q, u, û
    for K in T_h do
        H^K, r^K ← Elemental contributions from q^K, u^K, û^K
    H, r ← Assemble H^K and r^K for all K in T_h
δû ← Solve Hδû = r
begin Obtain δq and δu from δû
    for K in T_h do
        δq^K, δu^K ← Recover element solution from δû^{∂K}
    δq, δu ← Assemble δq^K and δu^K for all K in T_h
```

Solving HDG linear system: distributed memory

Algorithm 3: Distributed solve of the HDG linear system

Input: $\underline{\mathbf{q}}$, $\underline{\mathbf{u}}$, $\underline{\hat{\mathbf{u}}}$

Output: $\underline{\delta\mathbf{q}}$, $\underline{\delta\mathbf{u}}$, $\underline{\delta\hat{\mathbf{u}}}$

$p \leftarrow$ Current processor

begin Compute distributed matrix $\underline{\mathbf{H}}$ and vector $\underline{\mathbf{r}}$ from $\underline{\mathbf{q}}$, $\underline{\mathbf{u}}$, $\underline{\hat{\mathbf{u}}}$

  Gather $\mathbf{q}^K$, $\mathbf{u}^K$, $\hat{\mathbf{u}}^{\partial K}$ from $\underline{\mathbf{q}}$,$\underline{\mathbf{u}}$,$\underline{\hat{\mathbf{u}}}$ for all K in ${\mathcal{T}_h}^p$

  for K *in* ${\mathcal{T}_h}^p$ **do**

    $\mathbf{H}^K$, $\mathbf{r}^K \leftarrow$ Elemental contributions from $\mathbf{q}^K$, $\mathbf{u}^K$, $\hat{\mathbf{u}}^{\partial K}$

  $\underline{\mathbf{H}}$, $\underline{\mathbf{r}} \leftarrow$ Assemble $\mathbf{H}^K$ and $\mathbf{r}^K$ for all K in ${\mathcal{T}_h}^p$

$\underline{\delta\hat{\mathbf{u}}} \leftarrow$ Distributed solve $\underline{\mathbf{H}}\underline{\delta\hat{\mathbf{u}}} = \underline{\mathbf{r}}$

begin Obtain $\underline{\delta\mathbf{q}}$ and $\underline{\delta\mathbf{u}}$ from $\underline{\delta\hat{\mathbf{u}}}$

  Gather $\delta\hat{\mathbf{u}}^{\partial K}$ from $\underline{\delta\hat{\mathbf{u}}}$ for all for all K in ${\mathcal{T}_h}^p$

  for K *in* ${\mathcal{T}_h}^p$ **do**

    $\delta\mathbf{q}^K$, $\delta\mathbf{u}^K \leftarrow$ Recover element solution from $\delta\hat{\mathbf{u}}^{\partial K}$

  $\underline{\delta\mathbf{q}}$, $\underline{\delta\mathbf{u}} \leftarrow$ Distribute $\delta\mathbf{q}^K$ and $\delta\mathbf{u}^K$ for all K in ${\mathcal{T}_h}^p$

Required parallel computations

Newton for HDG requires vector updates and, a linear HDG solver:

- Element-wise loops: elemental quantities, element inverses (local problem), element contributions, recovery of $\delta\mathbf{q}$ and $\delta\mathbf{u}$ from $\delta\hat{\mathbf{u}}$.
- Linear solver
  - Nested dissection or similar (UMFPACK and Pardiso)
  - Pre-conditioned GMRES (PETSc and Trilinos)
- Gather and distribute vector components (distributed memory)

Required parallel computations

Pre-conditioned GMRES:

- GMRES: dot products and sparse matrix-vector products
- Pre-conditioner
  - ILU has a sequential nature (low parallelization)
  - Additive Schwarz domain-decomposition.

Linear solver: Krylov methods

Krylov methods are projection (Galerkin) methods for solving

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

They are based on the generation of the Krylov subspace

$$\mathcal{K}_j := \mathrm{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{j-1}\mathbf{r}_0\},$$

where $\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}$.

- Large linear systems: iterative versions (low memory footprint) with a pre-conditioner.
- They require computing matrix-vector products (parallelizable).
- Our choice: Generalized Minimal RESidual (GMRES) method with restart (reduce memory footprint).

Distributed pre-conditioner

Additive Schwarz domain decomposition. Divide-and-conquer approach to pre-condition a linear system:

- *Mesh partition*. Determine n_p sub-domains from the graph of DOFs connections.
- the pre-conditioner is:

$$\mathbf{M}^{-1} = \sum_{p=1}^{n_p} \mathbf{R}_p^T \tilde{\mathbf{A}}_p^{-1} \mathbf{R}_p,$$

where
n_p: is the number of sub-domains (processors)
$\mathbf{R}_p$: restricts a vector to the p-th sub-domain (Boolean matrix)
$\tilde{\mathbf{A}}_p$: approximates $\mathbf{R}_p \mathbf{A} \mathbf{R}_p^T$ (*e.g.* its ILU factorization)

Gather and distribute vectors: element vectors

Element vectors (distributed in element sub-domains):

- Gather $\mathbf{q}^K$, $\mathbf{u}^K$ from $\underline{\mathbf{q}}$,$\underline{\mathbf{u}}$ for all K in ${\mathcal{T}_h}^p$
- $\delta\mathbf{q}$, $\underline{\delta\mathbf{u}} \leftarrow$ Distribute $\delta\mathbf{q}^K$ and $\delta\mathbf{u}^K$ for all K in ${\mathcal{T}_h}^p$
- Each processor p stores the components associated with the elements in ${\mathcal{T}_h}^p$ (element sub-domains)

Gather and distribute vectors: face vectors

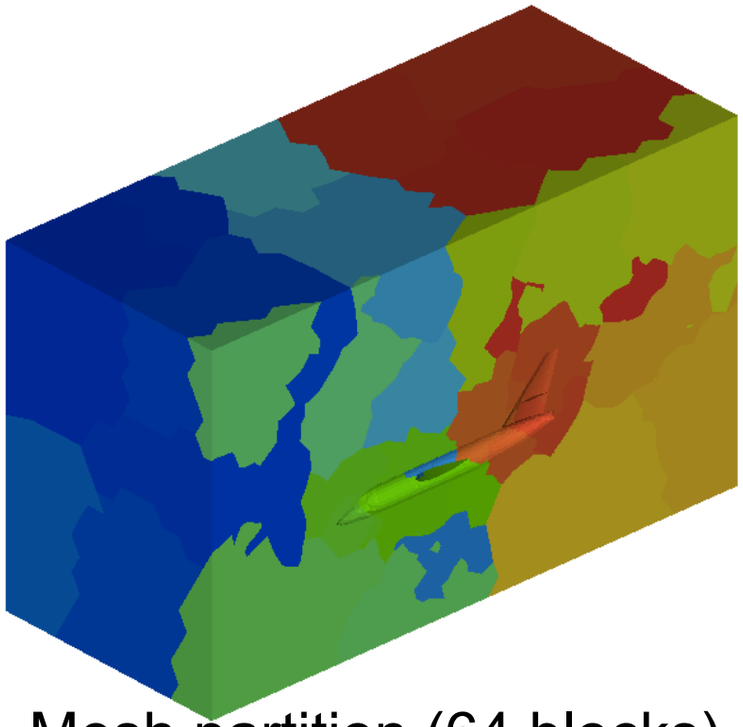
Face vectors (distributed in face sub-domains):

- $\hat{\mathbf{u}}^{\partial K}$ from $\underline{\hat{\mathbf{u}}}$ for all K in ${\mathcal{T}_h}^p$
- Gather $\delta\hat{\mathbf{u}}^{\partial K}$ from $\underline{\delta\hat{\mathbf{u}}}$ for all K in ${\mathcal{T}_h}^p$
- Each processor p stores the components associated with the faces in ${\mathcal{E}_h}^p$ (face sub-domains)
- A processor a has to receive from a processor b the components associated with the faces in ${\mathcal{E}_h}^b$ that are also in $\partial{\mathcal{T}_h}^a$

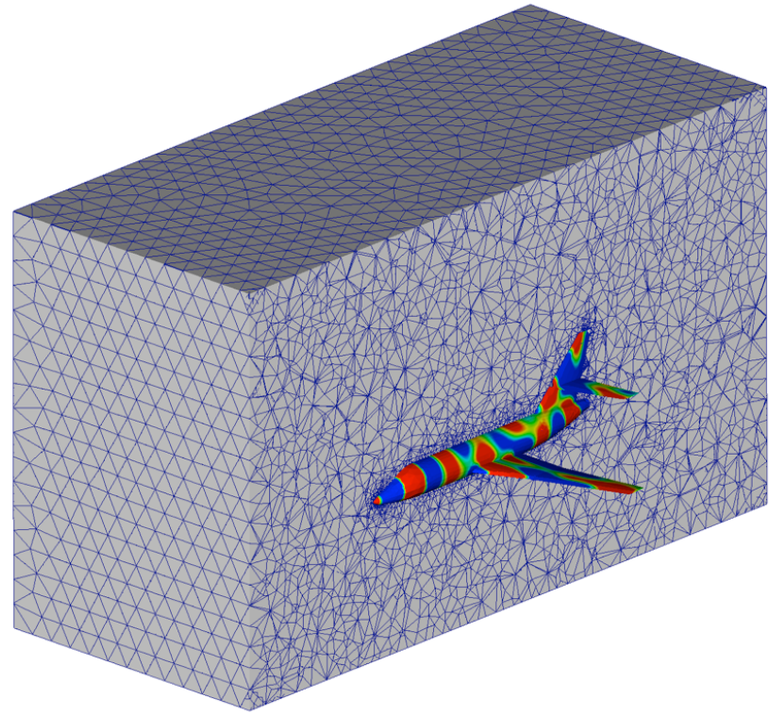
Parallel HDG solver (2/2):

partition, overlap and profiling

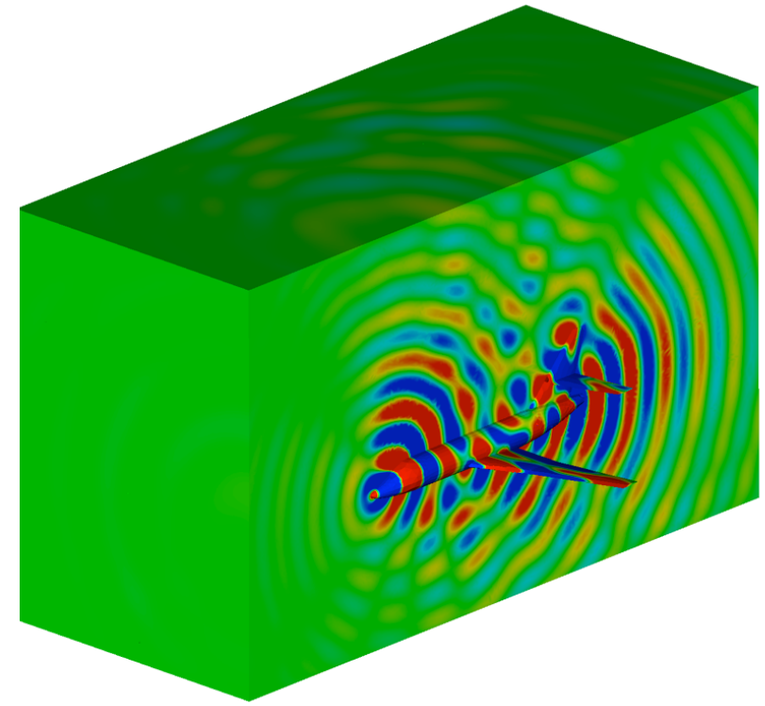
Parallel and distributed solver: wave scattering



Mesh partition (64 blocks)
METIS



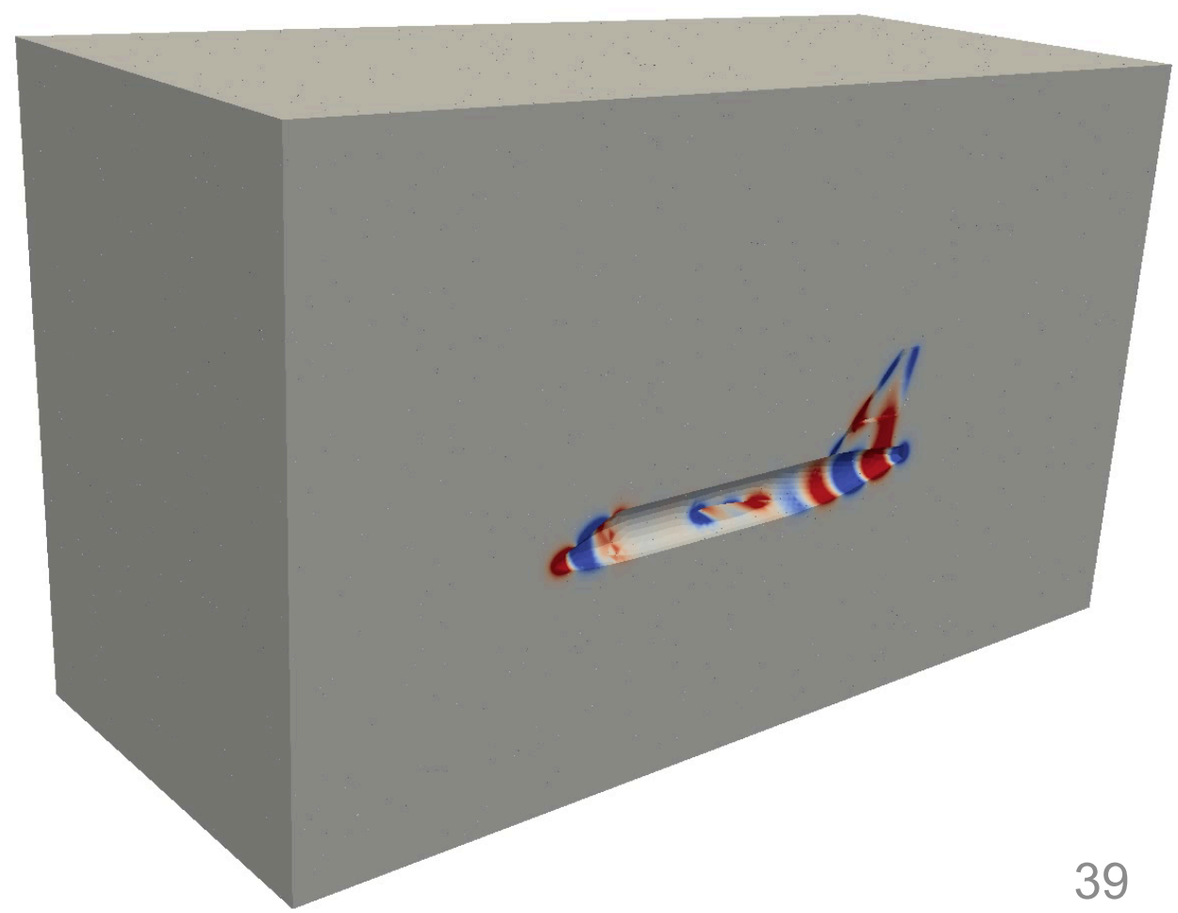
Mesh



Amplitude (clamped at [-0.1,0.1])

64 processors

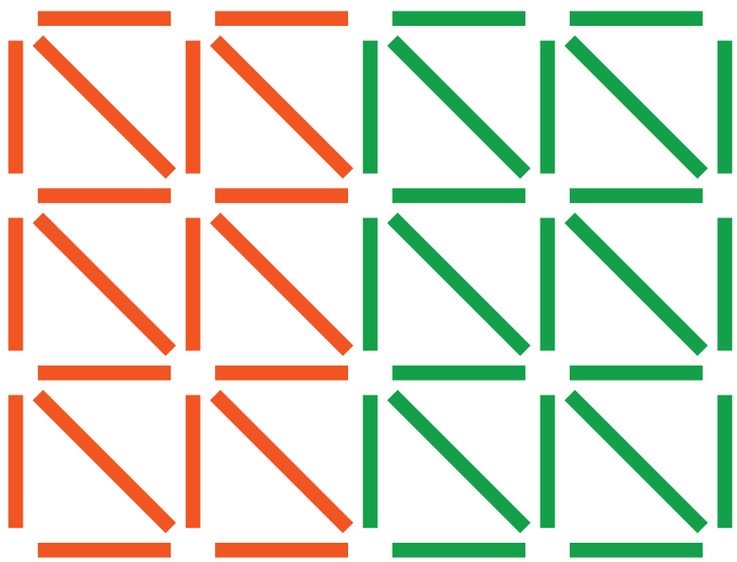
25.3M DOFS
u: 5.2M
q: 15.6M
û: 4.5M



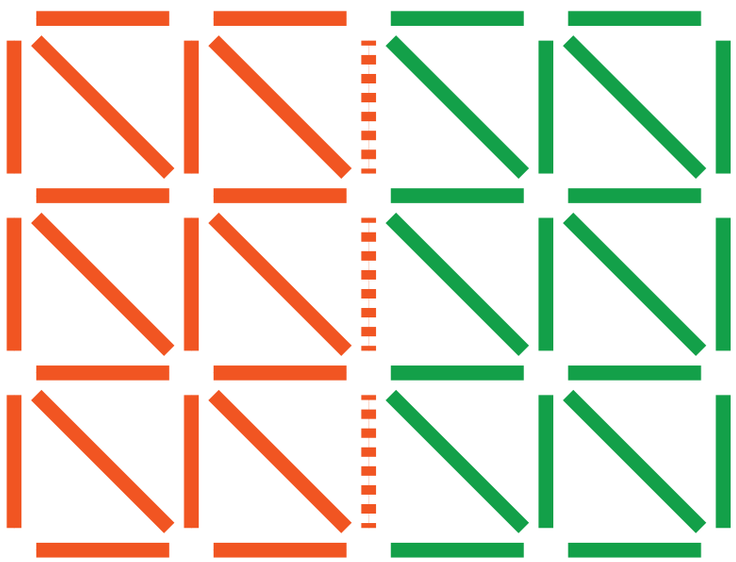
ASDD(*l*): *l*-levels of overlap for HDG

Overlap for mesh faces

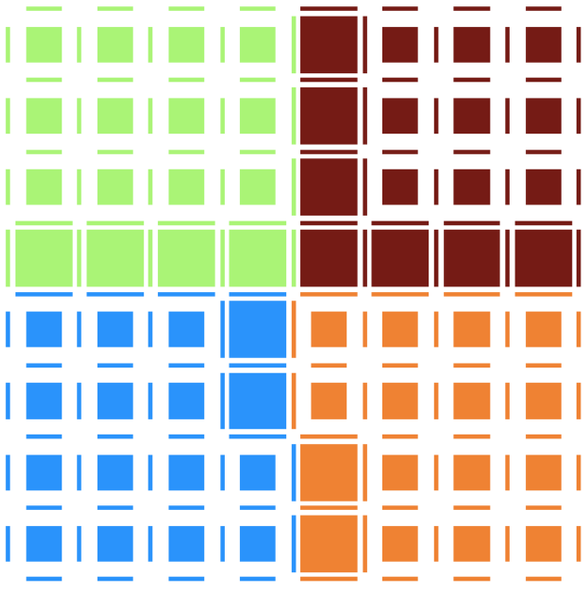
Face connections



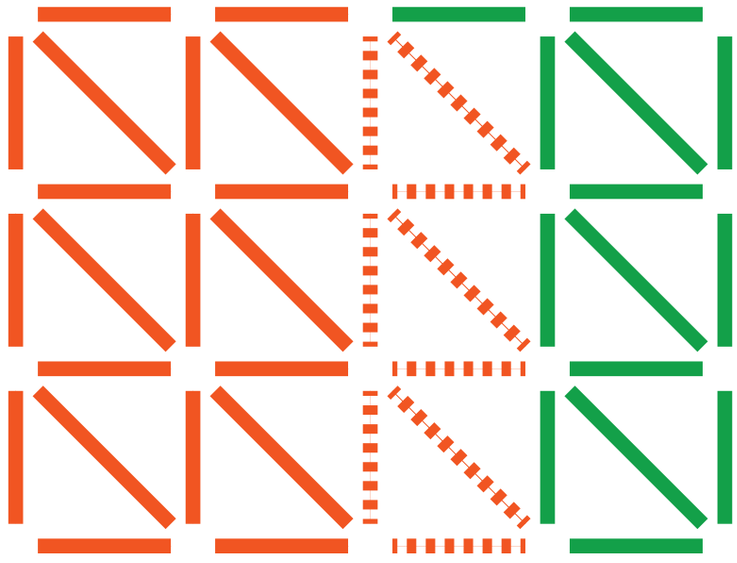
0-levels of overlap



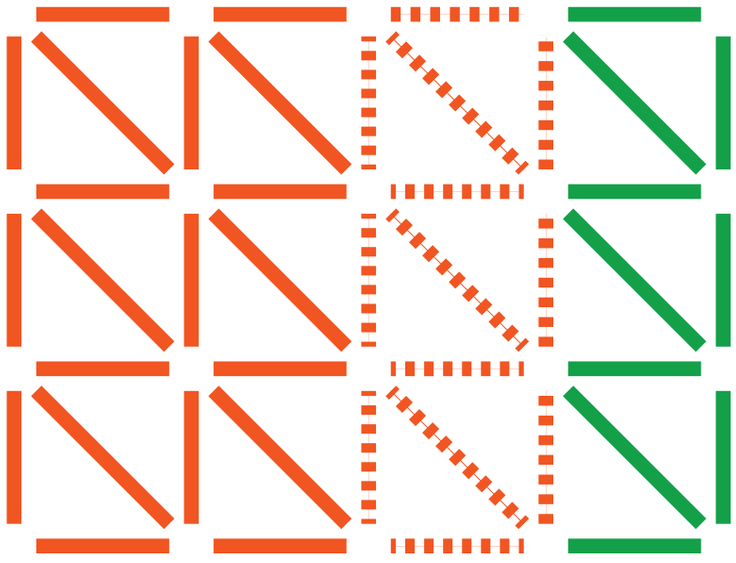
1-level of overlap



Element / faces partition (METIS)



2-levels of overlap

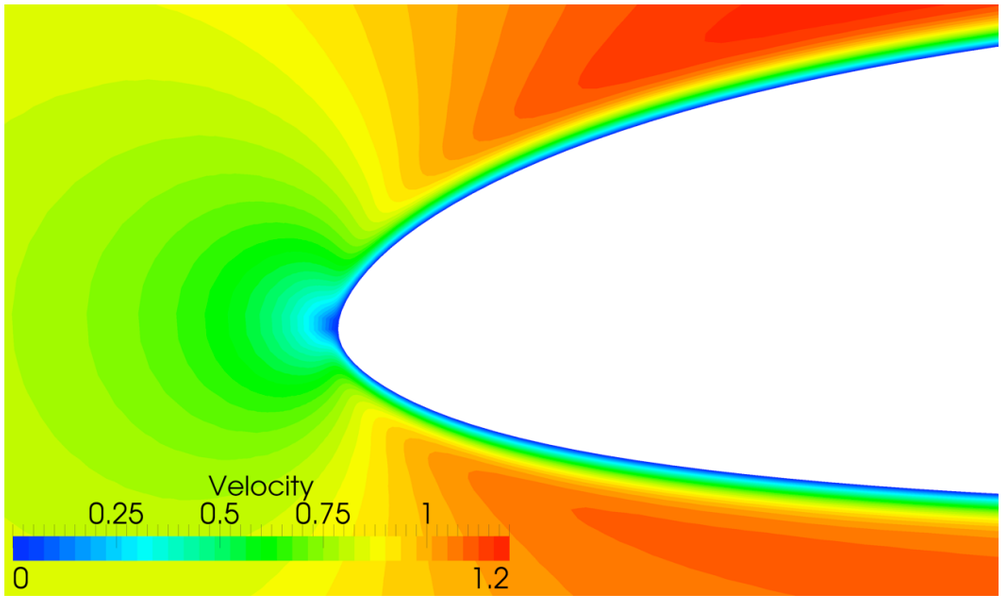


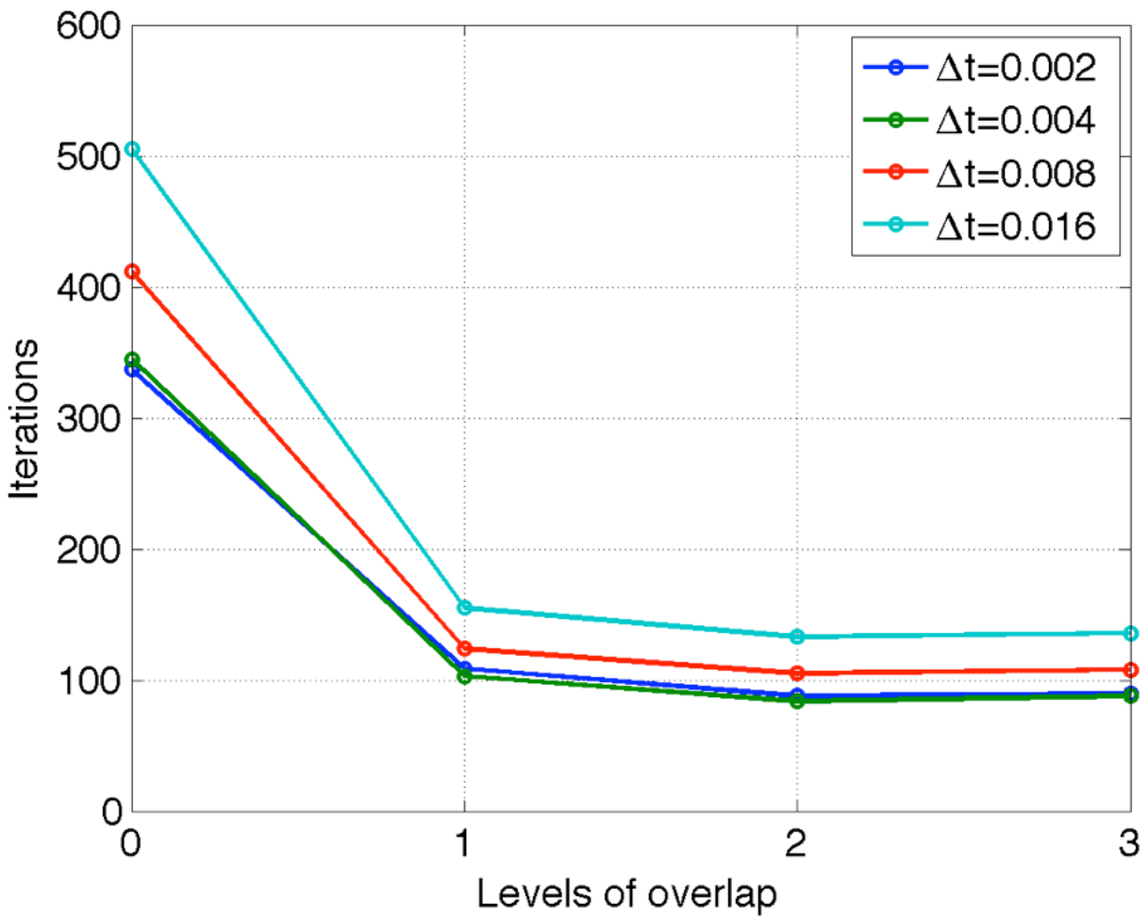
3-levels of overlap

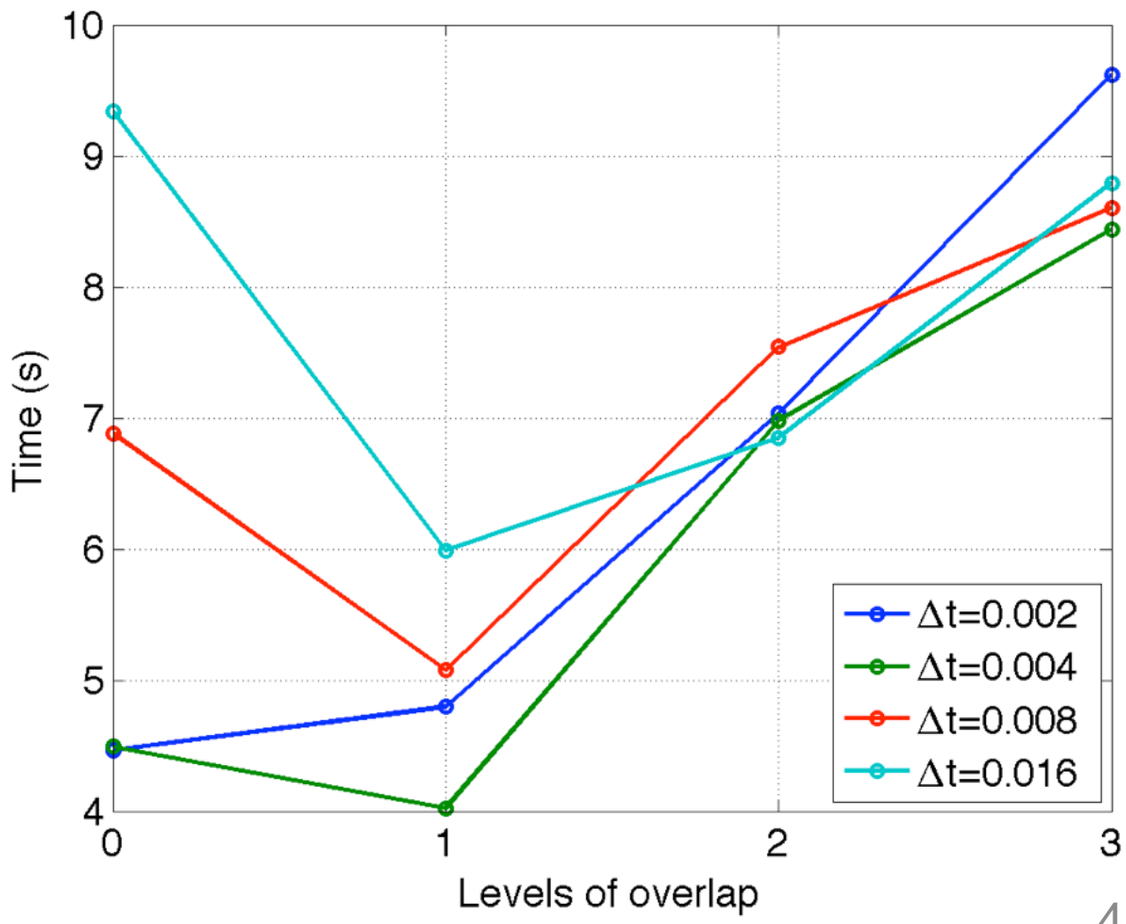
ASDD(1) / ILU(0)

Results: how many levels of overlap (ASDD) ?

- Unsteady, Re = 5000, M = 0.1, angle = 3, p = 5, 32 processors
  - **reduce** the number of iterations
  - **but**, slower iterations (memory):
    x1, x1.1, x1.4, x1.7
  - From 0-levels to 1-level:
    ASDD(1) / ILU(0)

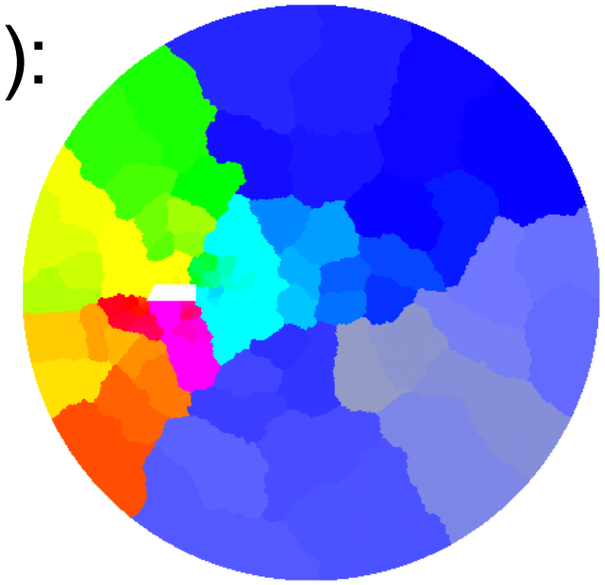


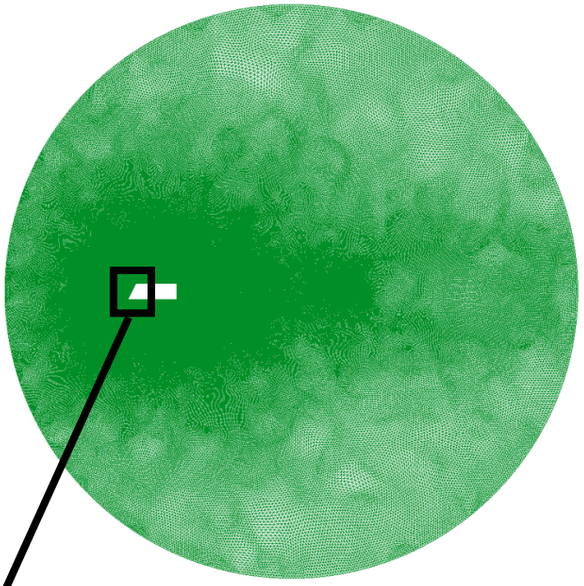




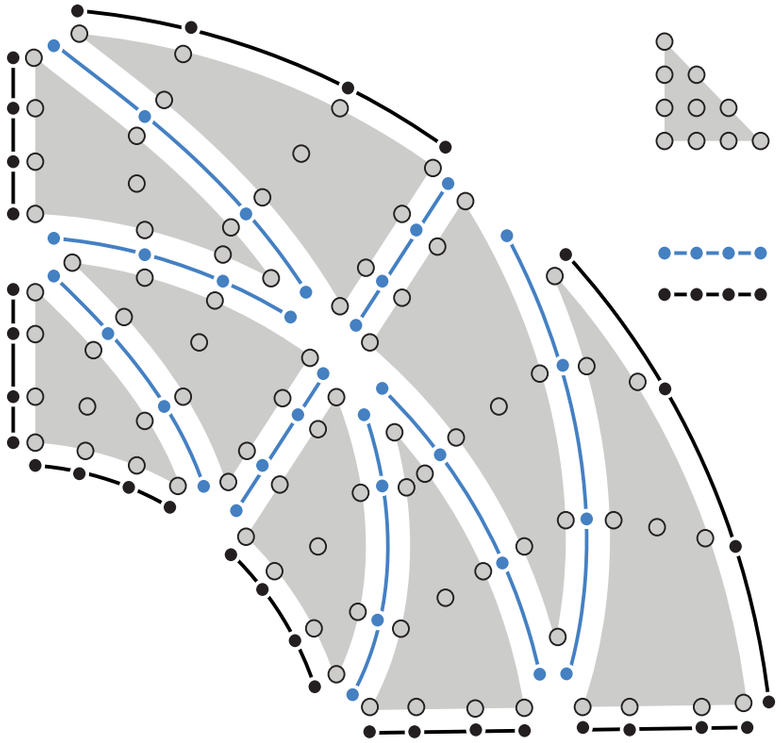
Profiling HDG linear solve: Re = 20000, M = 0.4, 20 solves

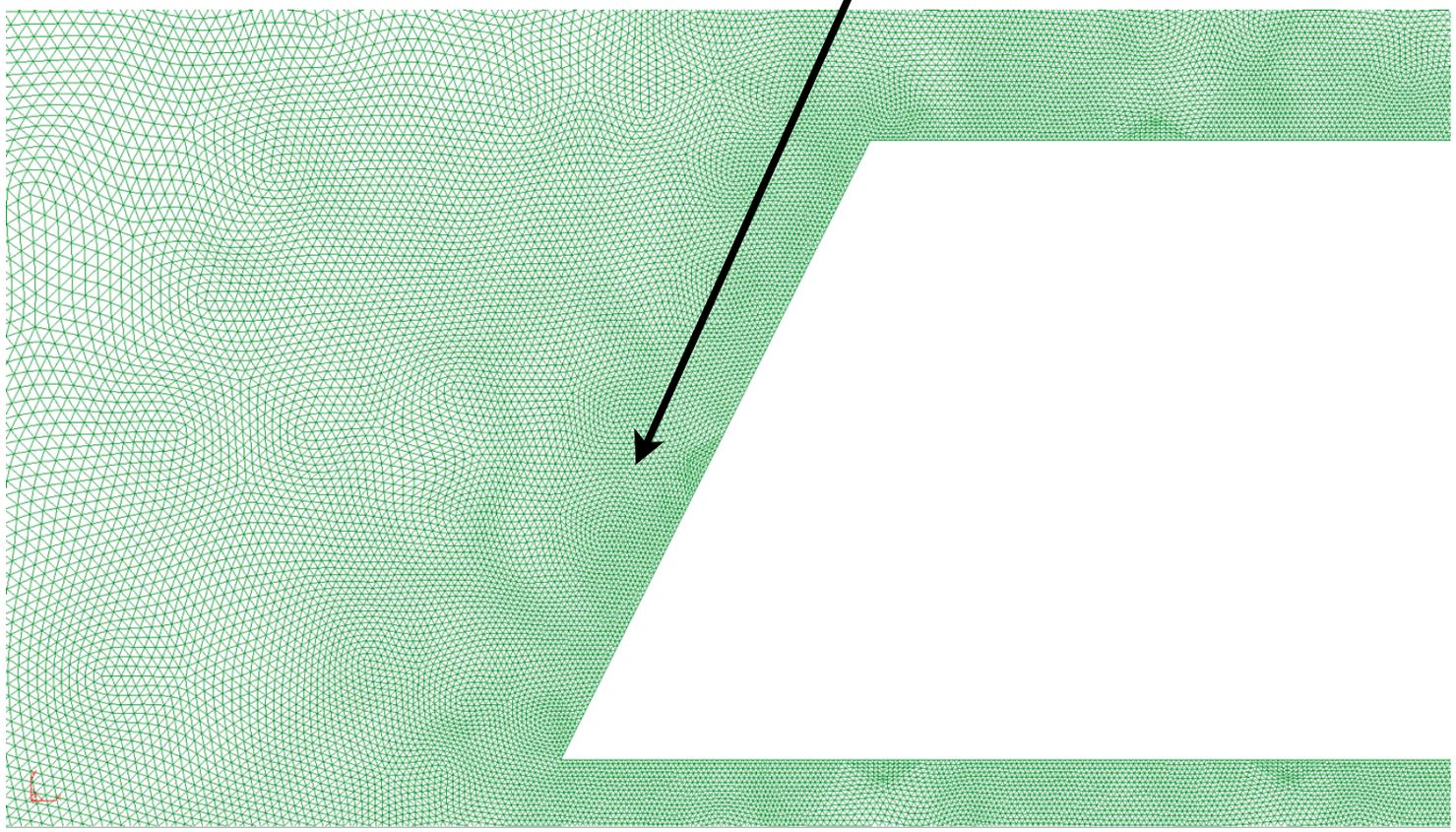
- Degrees of freedom (DOF):
  - **local** (q, u) = 86.4 M
  - **global** (û) = 19.2 M
  - **total** (q, u, û) = 105.6 M





Partition: 128 sub-domains **Mesh:** 480000 elements





Mesh detail: 480000 elements, p =4, dt = 0.0256

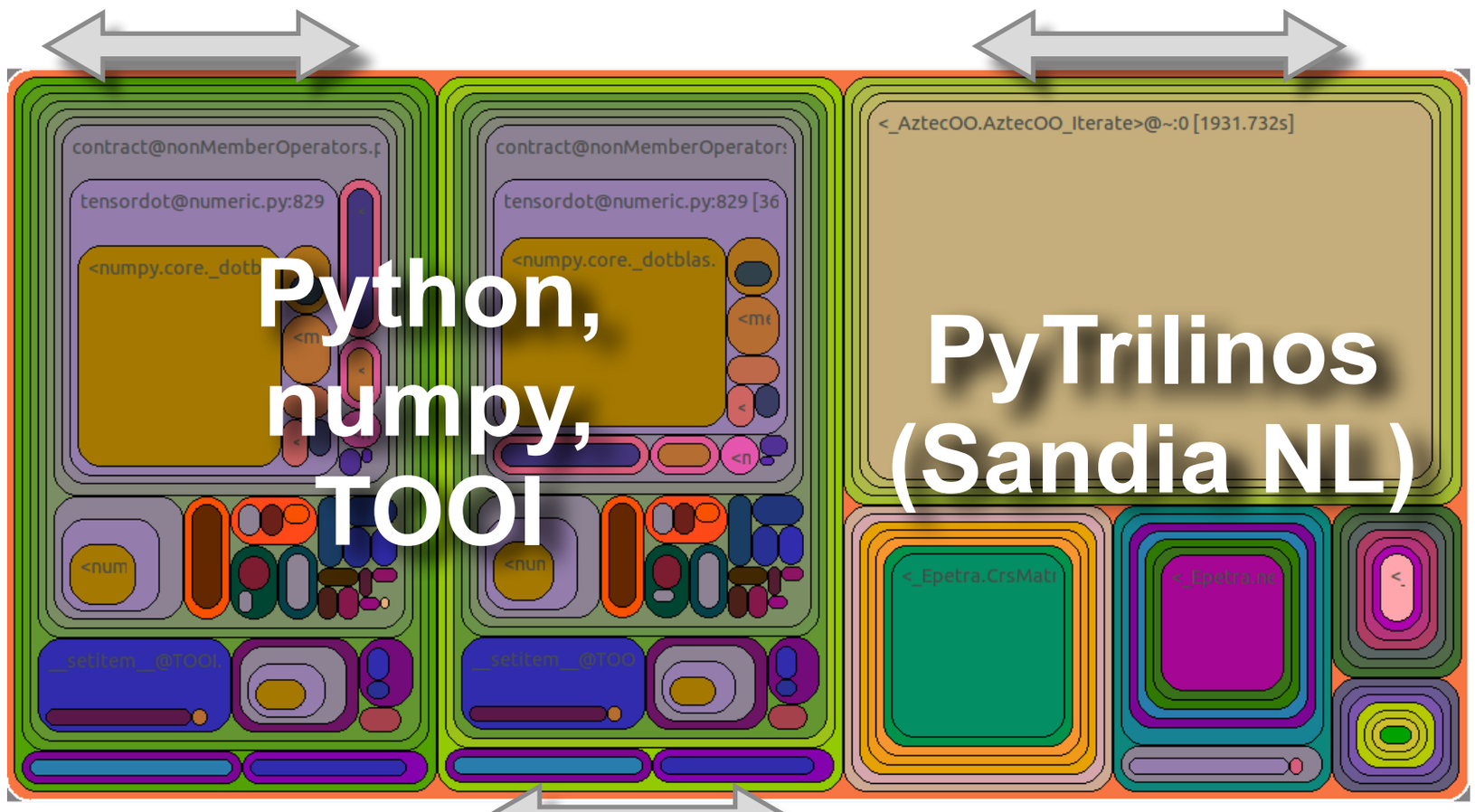
Profiling HDG solve: Re = 20000, M = 0.4, 20 solves, 128 cores

$$\left[\begin{array}{c|c} \mathbf{J_{qq}} & \mathbf{J_{qu}} \\ \hline \mathbf{J_{uq}} & \mathbf{J_{uu}} \end{array}\right]^{-1}$$

Solve local problems (element-wise):
30% (112 sec / solve)
86.4 M DOFs
15.5 billions of nnzs

$$\mathbf{H}\delta\hat{\mathbf{u}} = \mathbf{r}$$

Solve global problem (distributed):
42% (162 sec / solve)
216 iterations GMRES / ASDD(1) / ILU (0)
19.2 M DOFs
1.9 billions of nnzs



Solve HDG problem:
100% (380 sec / solve)
105.6 M

$$\left[\begin{array}{c|c|c} \mathbf{J_{qq}} & \mathbf{J_{qu}} & \mathbf{J_{q\hat{u}}} \\ \hline \mathbf{J_{uq}} & \mathbf{J_{uu}} & \mathbf{J_{u\hat{u}}} \\ \hline \mathbf{J_{\hat{u}q}} & \mathbf{J_{\hat{u}u}} & \mathbf{J_{\hat{u}\hat{u}}} \end{array}\right]_k \left[\begin{array}{c} \delta\mathbf{q} \\ \hline \delta\mathbf{u} \\ \hline \delta\hat{\mathbf{u}} \end{array}\right]_{k+1} = \left[\begin{array}{c} -\mathbf{r_q} \\ \hline -\mathbf{r_u} \\ \hline -\mathbf{r_{\hat{u}}} \end{array}\right]_k$$

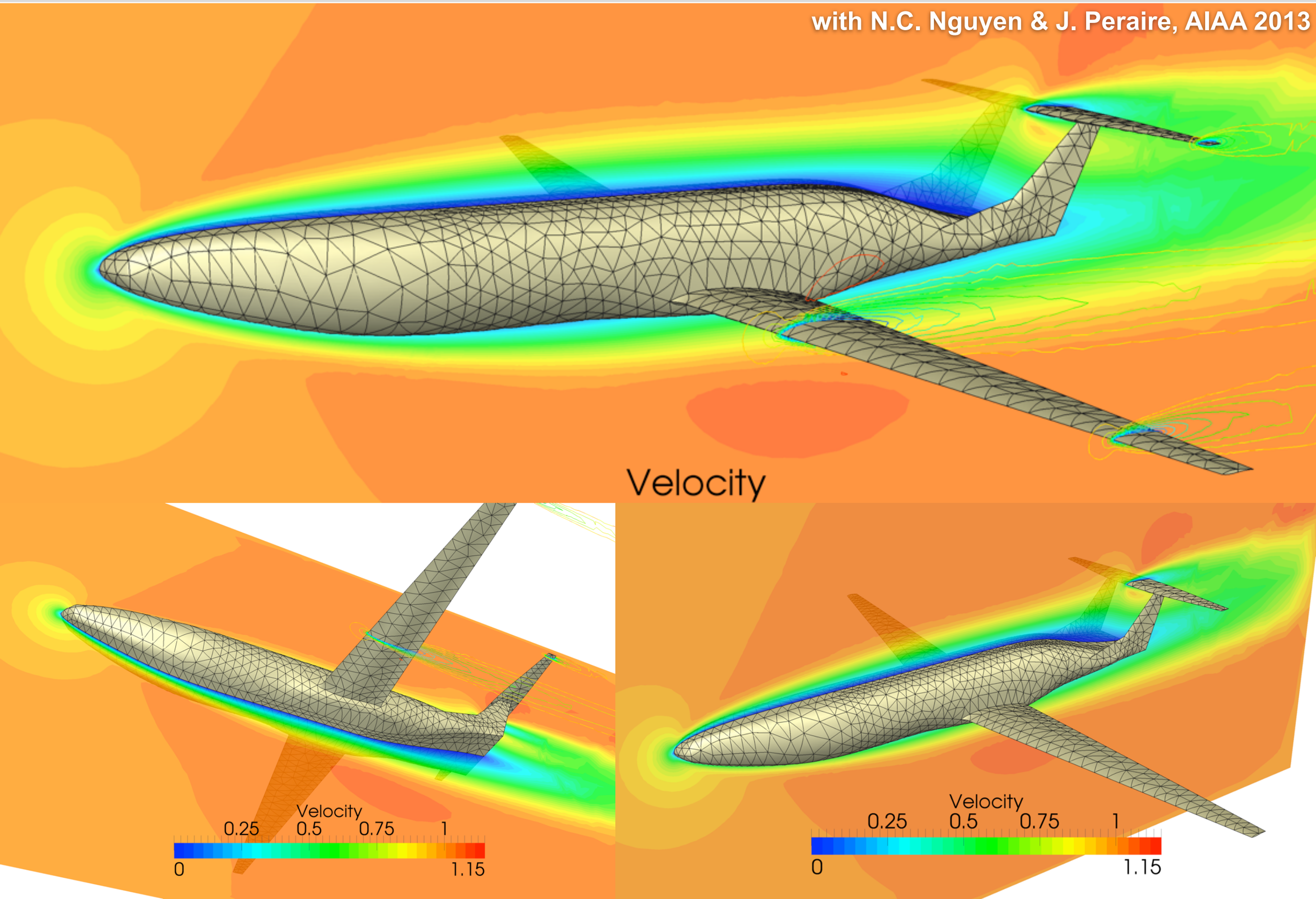
Recover local variables (element-wise):
28% (106 sec / solve)
86.4 M DOFs

$$\left[\begin{array}{c|c} \mathbf{J_{qq}} & \mathbf{J_{qu}} \\ \hline \mathbf{J_{uq}} & \mathbf{J_{uu}} \end{array}\right]^{-1}$$

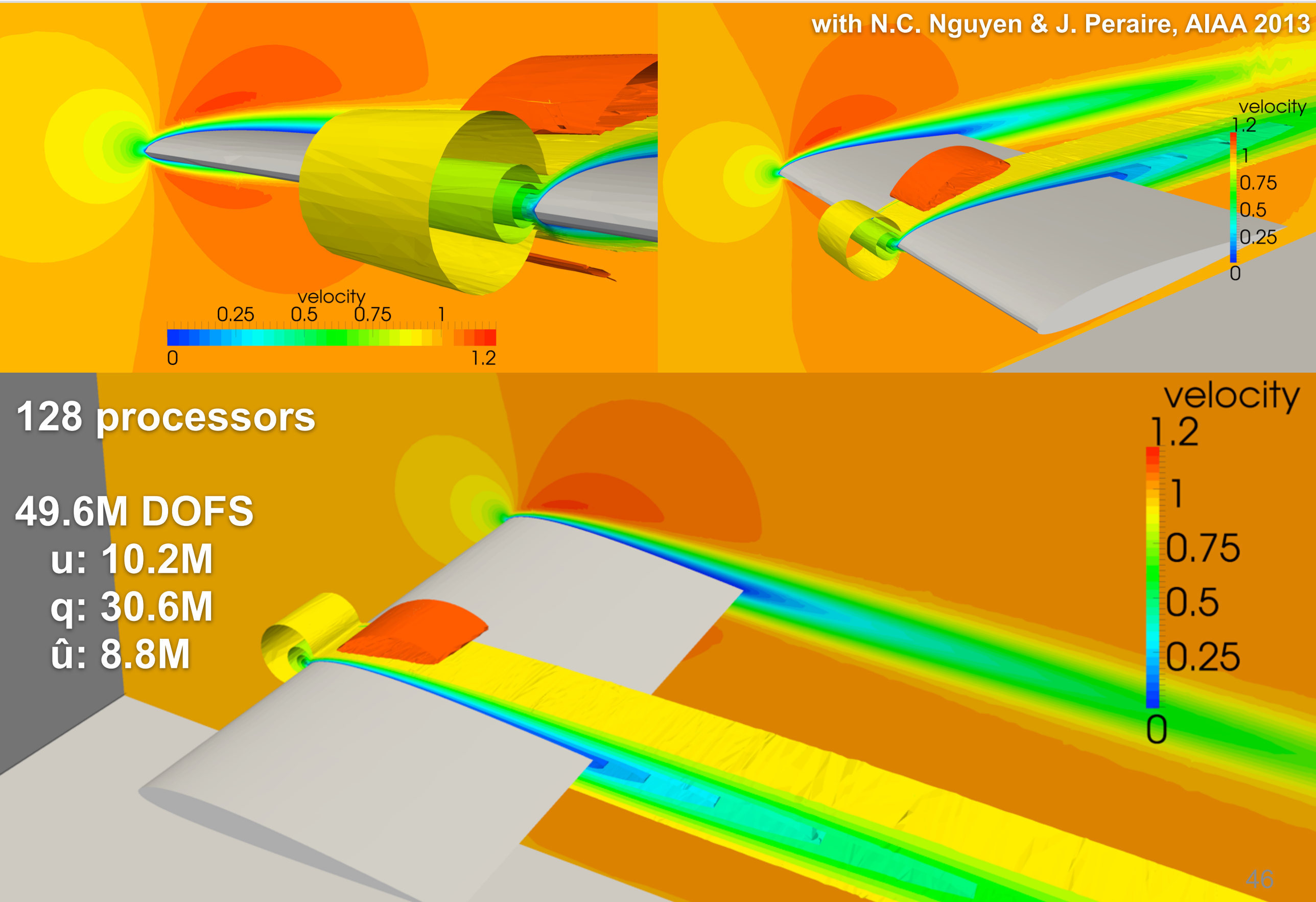
Numerical examples

3D steady flow, Re = 1000, p = 4, 70K elements

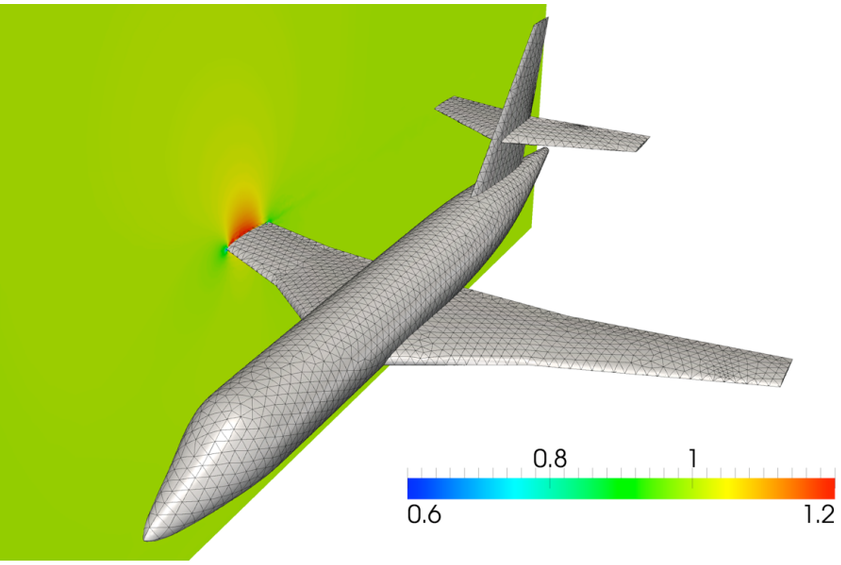
with N.C. Nguyen & J. Peraire, AIAA 2013

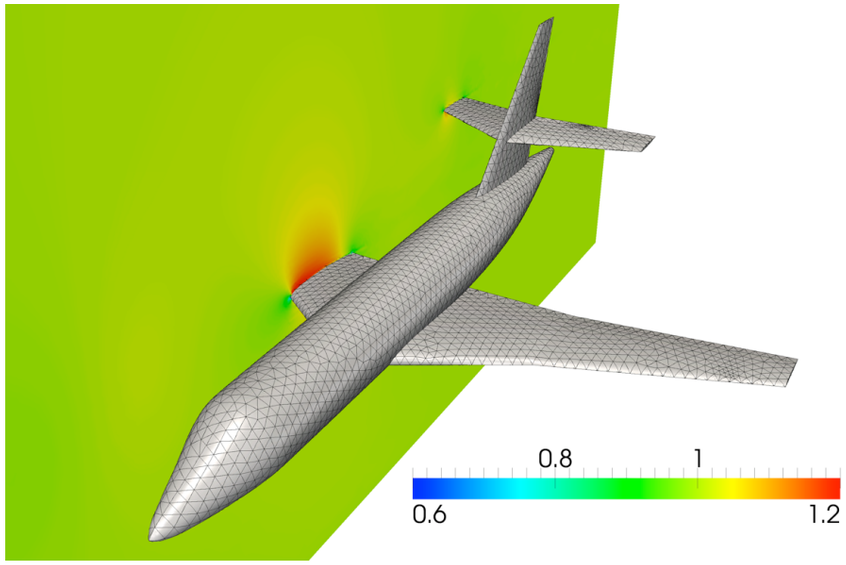


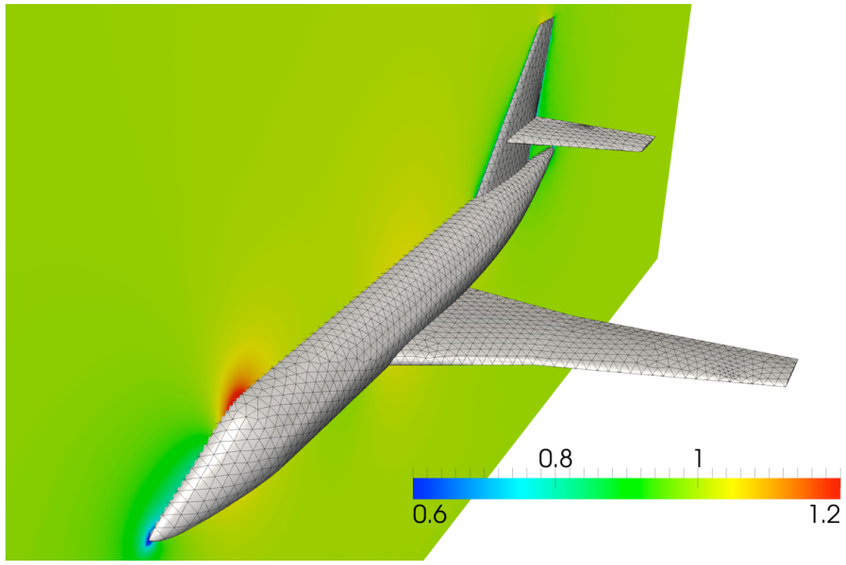
3D steady compressible flow, Re = 1000, p = 4, 60K



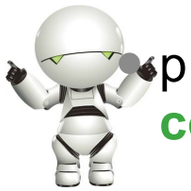
Inviscid compressible flow (Euler): M = 0.6, α = 0, p = 4







- Steady state **requires a curved mesh**: [Bassi & Rebay'97]



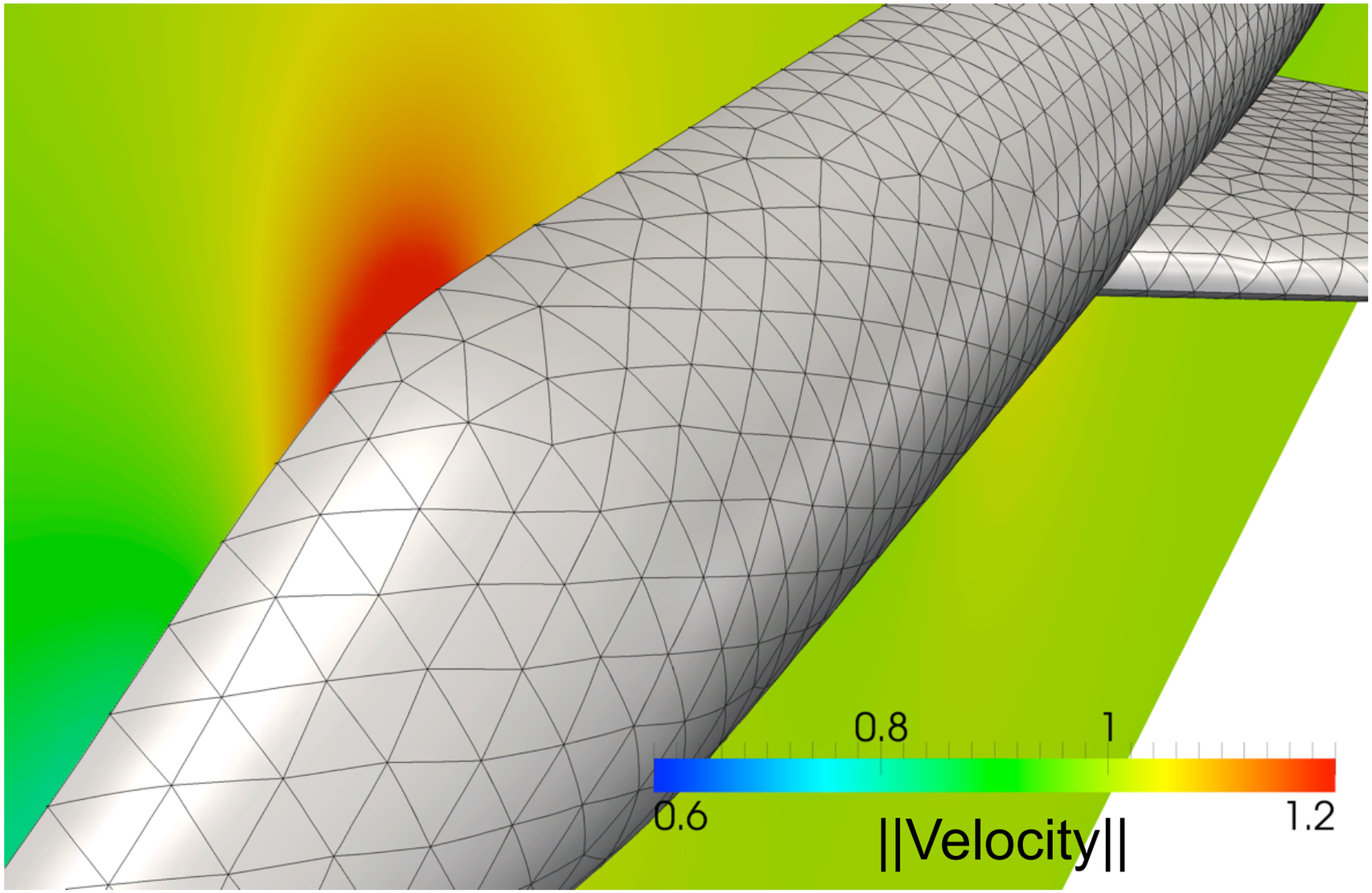
- p = 4, curved, **converges**

- p = 4, straight-sided, **does not converge**

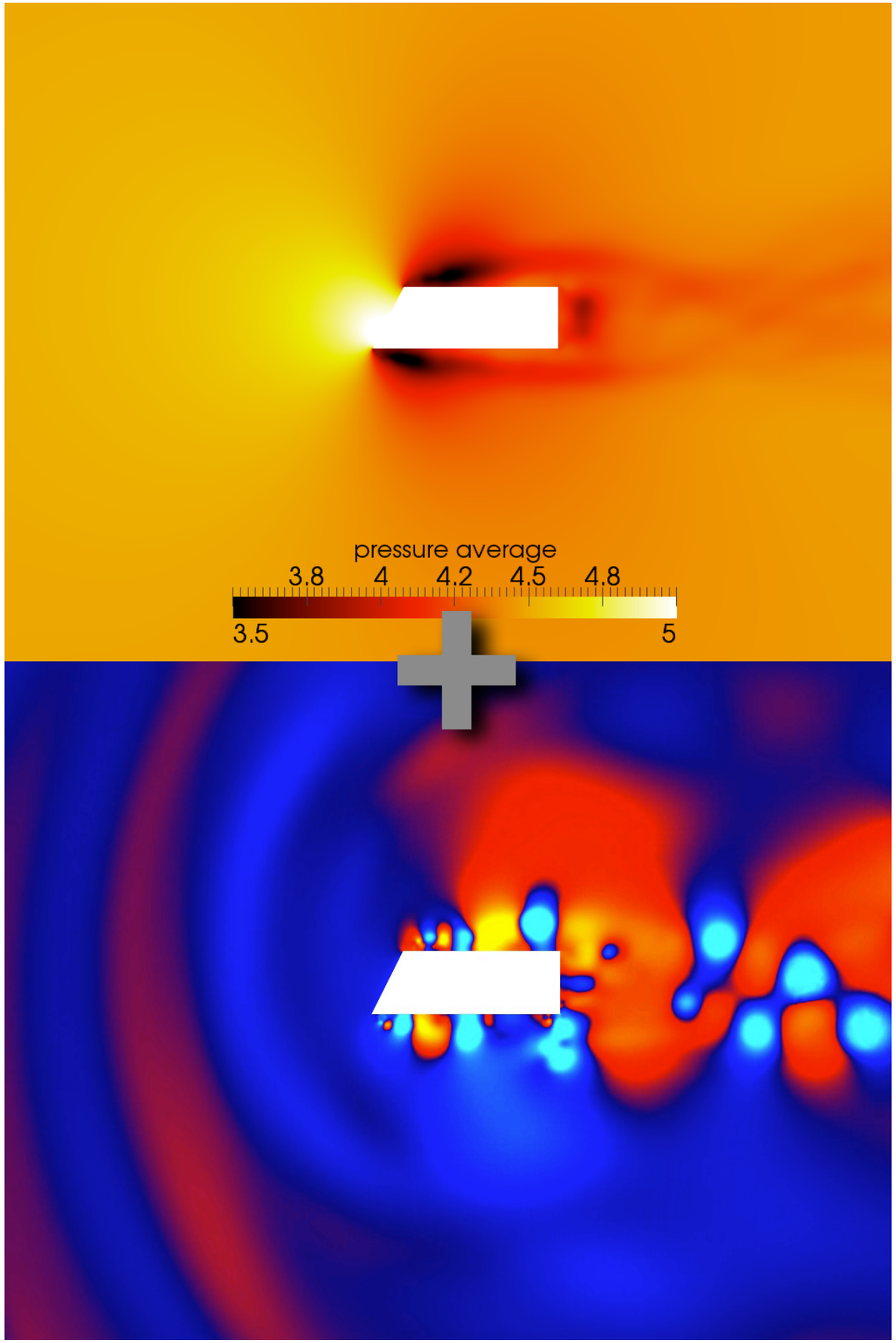
Mesh:
64992 elements
129984 faces

14M DOFS
u: 11.3M
û: 2.7M

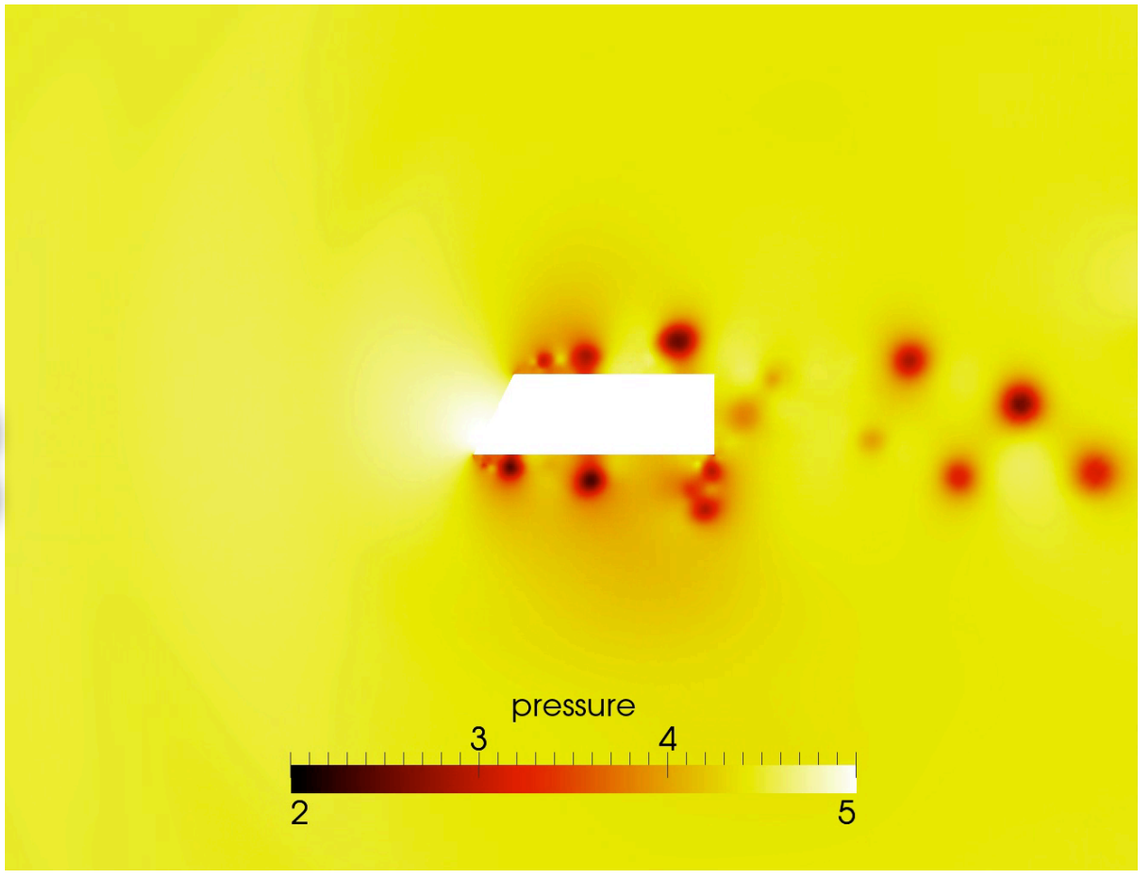
128 processors

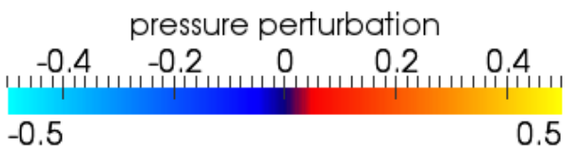


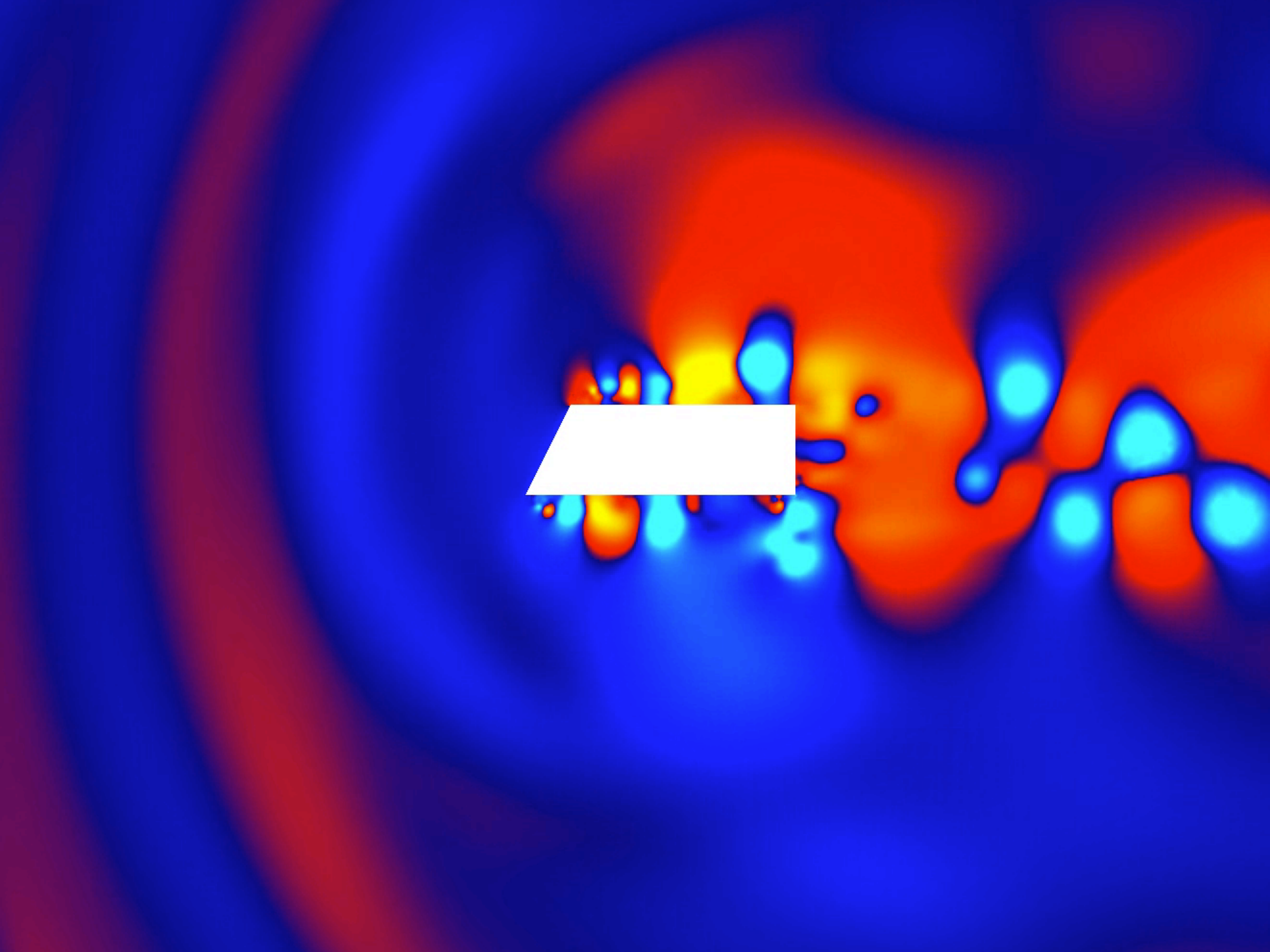
Compressible flow: Re = 20K, M = 0.4, p = 4, dt = 0.06, DIRK(3,3)



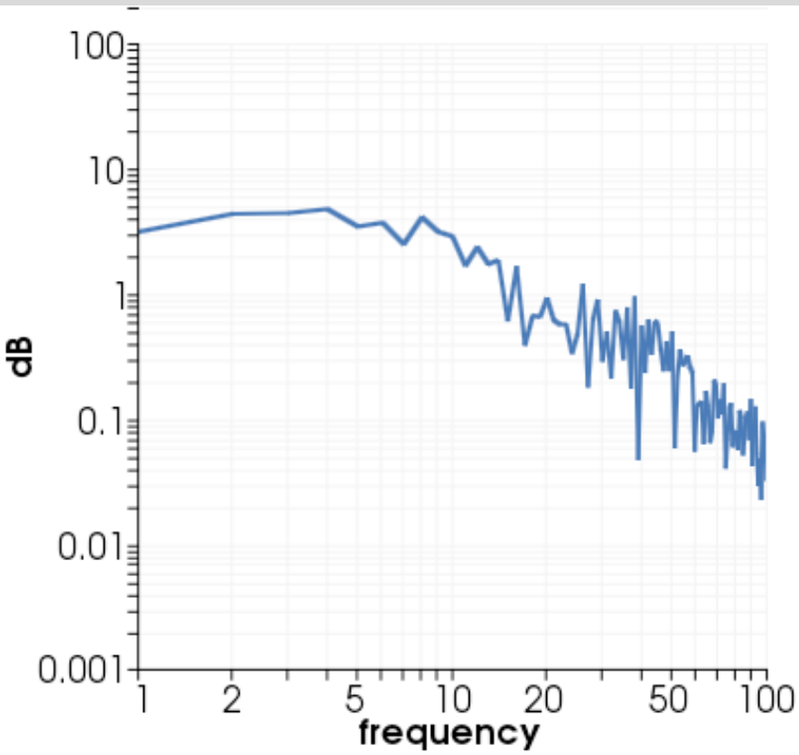
- **Same space-time resolution:**
  - **high-order** captures sound
  - **low-order** dissipates sound





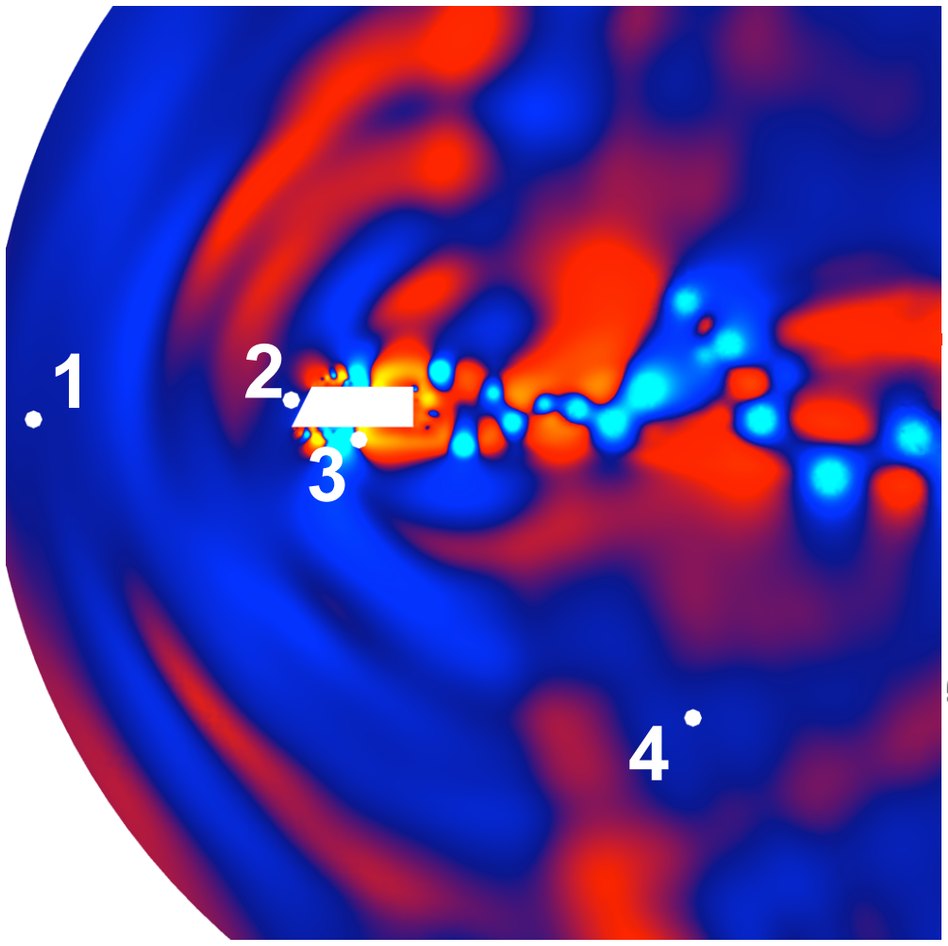


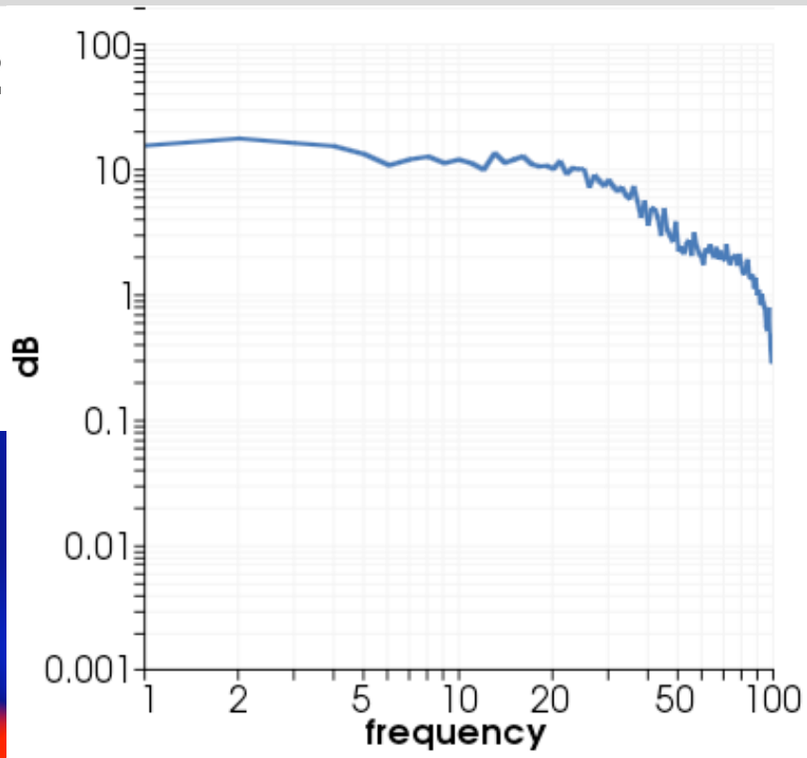
Acoustic pressure spectrum

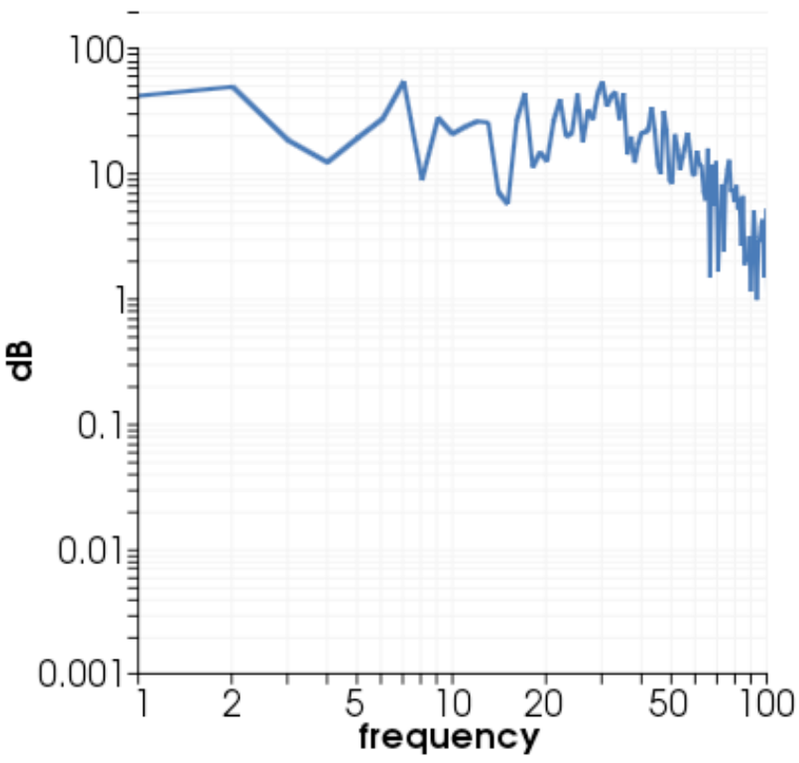


Probe 1

Probe 2



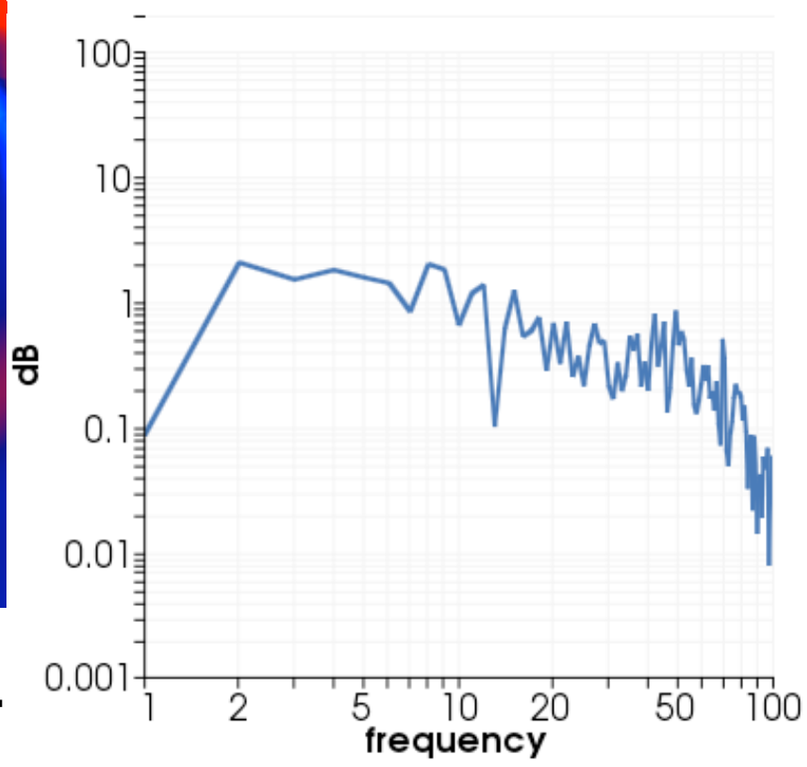




Probe 3

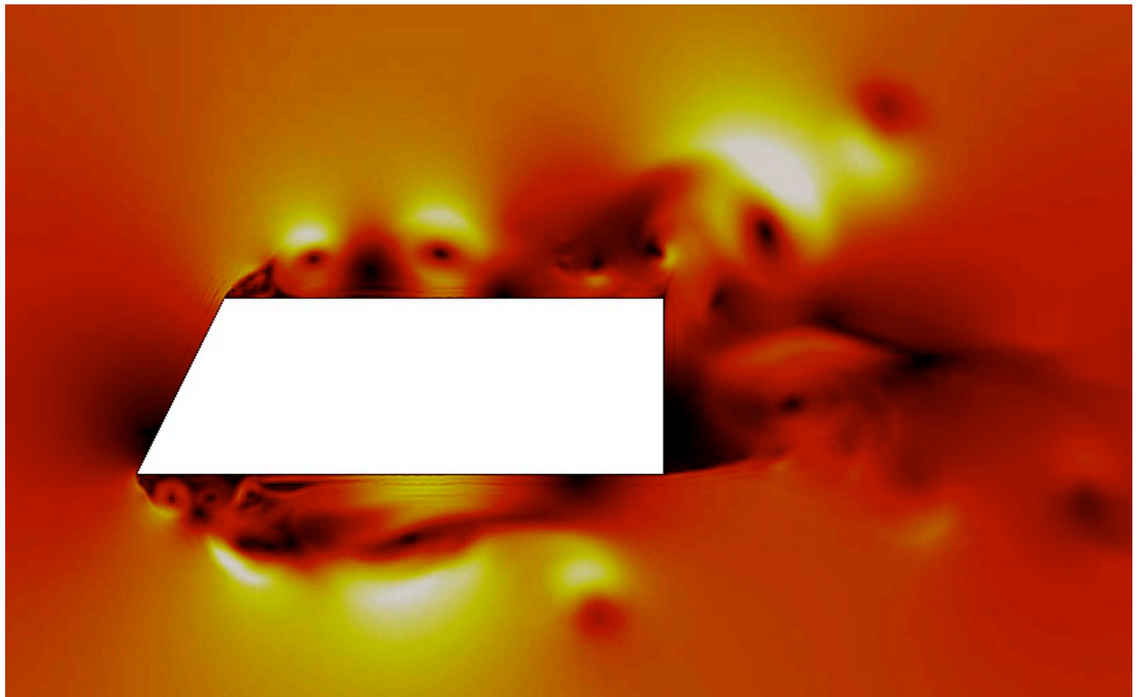
Pressure perturbation

Probe 4

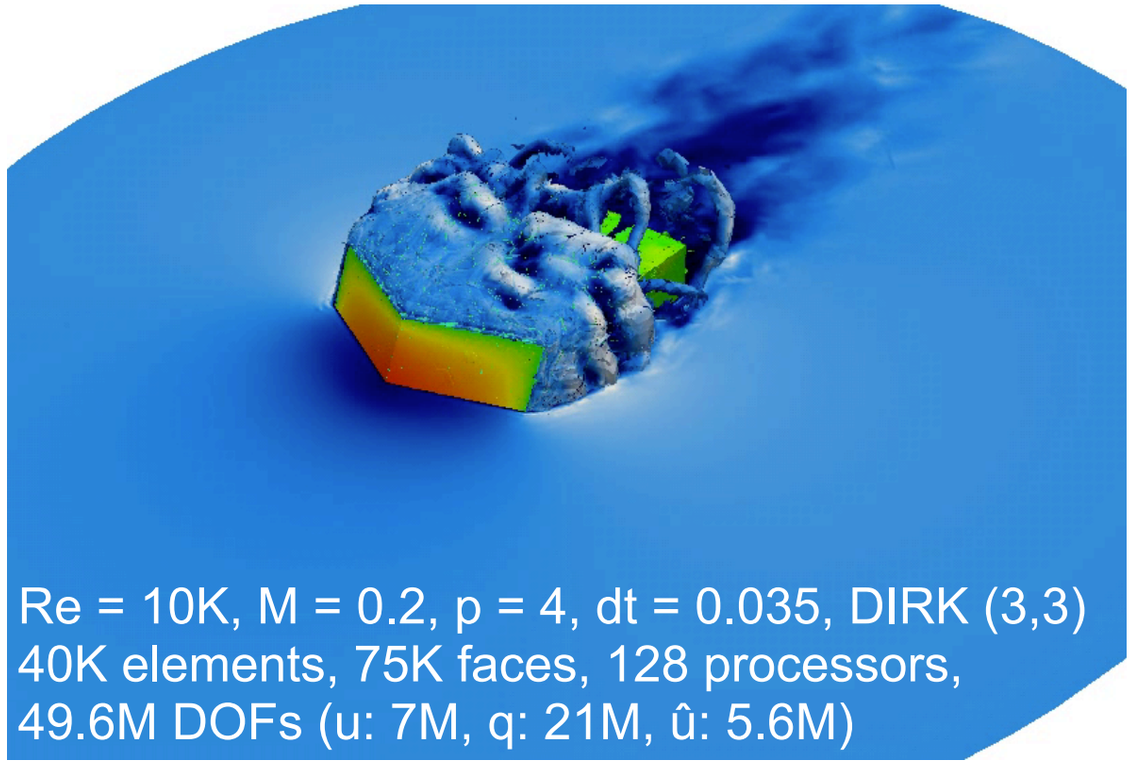


Predict sound spectrum: boundary layer meshes

with N.C. Nguyen & J. Peraire

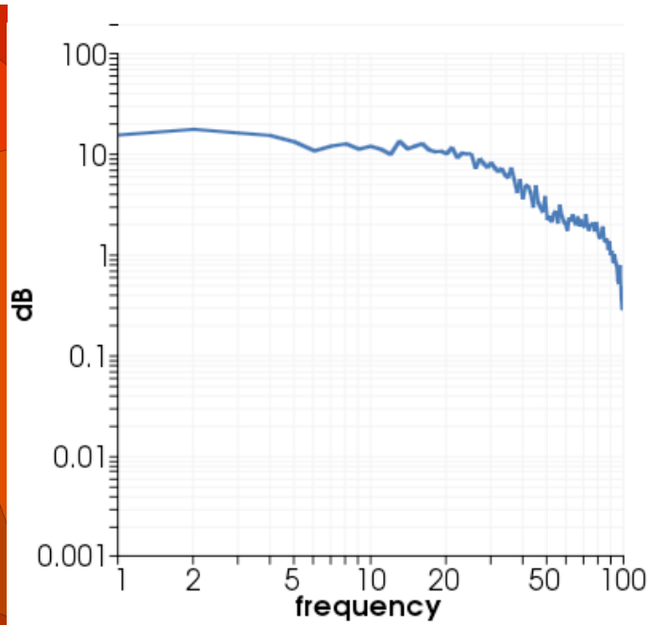


Non-resolved boundary layer: artificial recirculation !!

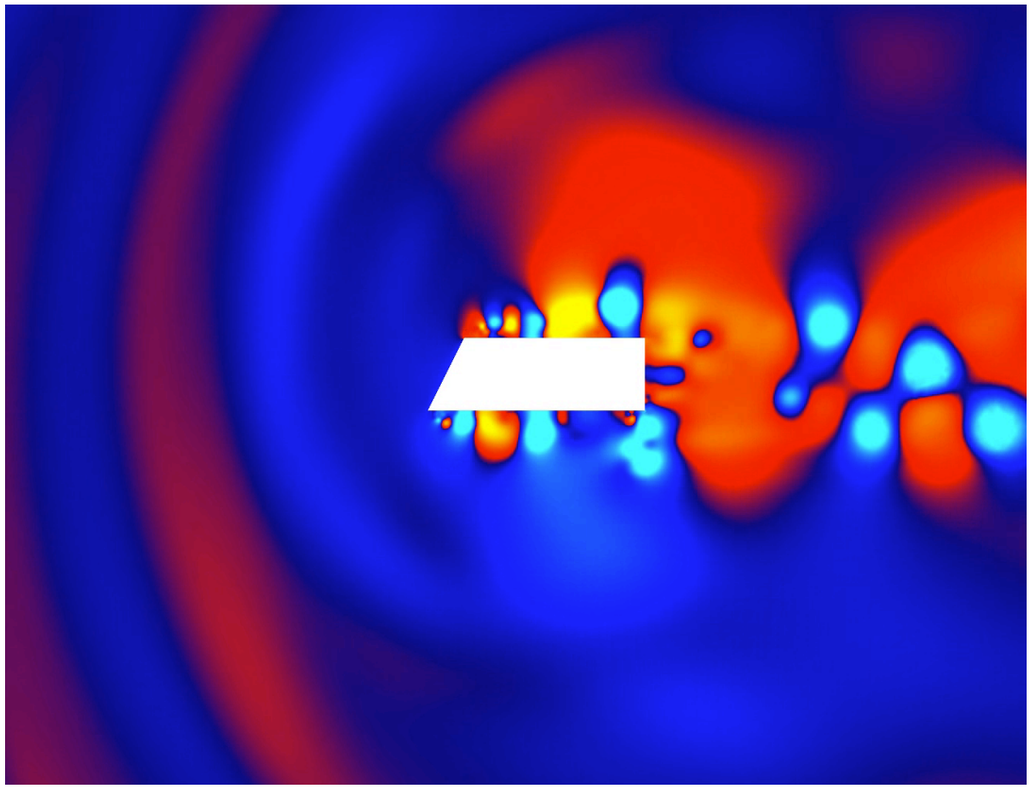


Pressure on the panels and density iso-surface

Curved boundary layers: all-acute-tetrahedra



Sound spectrum



High-order ILES: captures pressure perturbations

Summary

- Motivation: high-fidelity simulation
  - low dissipation and dispersion, cost, and curved geometries
- The HDG method: suitability to parallelization
  - Discontinuous, trace, inner products, local and global DOFs
- Parallel computing overview
  - Distributed memory, explicit and implicit, and libraries
- HDG linear systems:
  - local and global problems, and structure

Summary

- Parallel HDG solver
  - parallel pre-conditioner, iterative solver, distribution
  - partition, overlap and profiling
- Numerical examples
  - Viscous flow, inviscid flow, sound spectrums and curved and boundary layer meshes

Thank you

xevi.roca@bsc.es

References

- Y. Saad, *Iterative methods for sparse linear systems,* Society for Industrial and Applied Mathematics, 2003
- A. George, *Nested Dissection of a regular finite-element mesh*, SIAM J. Numer. Anal., 10(2), 1973
- T.A., Davis, *Algorithm 832: UMFPACK V4. 3 - an unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software, 30(2), 2004
- O. Schenk, K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Future Generation Computer Systems, 20(3), 2004
- P-O Persson, J. Peraire, *Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations*, SIAM J. Sci. Comp., 30(6), 2008

- **References to HDG by B. Cockburn, J. Peraire, N.C. Nguyen, and other authors, are covered in the other lectures**

Our references

- **Computational cost indicators**:
  - Angeloski, A; Roca, X; Peraire, J; Huerta, A; *Computational cost of simplices versus parallelotopes for Galerkin methods*, 21st International Meshing Roundtable, 2012
  - Angeloski, Aleksandar; Peraire, Jaime; Roca, Xevi; *Are High-order and Hybridizable Discontinuous Galerkin methods competitive?*, Oberwolfach reports; 1, 9, 2012
  - Huerta, Antonio; Angeloski, Aleksandar; Roca, Xevi; Peraire, Jaime; *Efficiency of high−order elements for continuous and discontinuous Galerkin methods*, International Journal for Numerical Methods in Engineering, 96(9), 529-560, 2013

- **Parallelization of the hybridizable discontinuous Galerkin method**:
  - Roca, Xevi; Nguyen, Ngoc Cuong; Peraire, Jaime; *GPU-accelerated sparse matrix-vector product for a hybridizable discontinuous Galerkin method,* Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, 2011
  - Roca, Xevi; Nguyen, Ngoc C; Peraire, Jaime; *Scalable parallelization of the hybridized discontinuous Galerkin method for compressible flow*, 21st AIAA Computational Fluid Dynamics Conference, 2013
  - Fernandez, Pablo; Nguyen, Ngoc-Cuong; Roca, Xevi; Peraire, Jaime; *Implicit large-eddy simulation of compressible flows using the Interior Embedded Discontinuous Galerkin method*, Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, 2016

Our references

- **Curved meshing**:
  - Roca, Xevi; Gargallo-Peiró, Abel; Sarrate, Josep; *Defining quality measures for high-order planar triangles and curved mesh generation*, Proceedings of the 20th International Meshing Roundtable, 365-383, 2011
  - Gargallo-Peiró, Abel; Roca, Xevi; Sarrate, Josep; Peraire, Jaime; *Inserting curved boundary layers for viscous flow simulation with high-order tetrahedra*, 22nd International Meshing Roundtable, Orlando, Florida, 2013
  - Gargallo-Peiró, Abel; Roca, Xevi; Sarrate, Josep; *A surface mesh smoothing and untangling method independent of the CAD parameterization* Computational mechanics, 53, 4, 587-609, 2014
  - Gargallo−Peiró, Abel; Roca, X; Peraire, J; Sarrate, J; *Optimization of a regularized distortion measure to generate curved high−order unstructured tetrahedral meshes*, International Journal for Numerical Methods in Engineering, 103, 5, 342-363, 2015
  - Gargallo−Peiró, Abel; Roca, Xevi; Peraire, Jaume; Sarrate, Josep; *A distortion measure to validate and generate curved high−order meshes on CAD surfaces with independence of parameterization*, International Journal for Numerical Methods in Engineering, 106, 13, 1100-1130, 2016
  - Ruiz-Gironés, Eloi; Roca, Xevi; Sarrate, Jose; *High-order mesh curving by distortion minimization with boundary nodes free to slide on a 3D CAD representation*, Computer-Aided Design, 72, 52-64, 2016