



Development of Aerodynamic Shape Optimization Capability within NASA's LAVA Framework

Brandon M. Lowe, Chase P. Ashby, Timothy Chau, Joshua L. Anibal,
James R. L. Koch, David A. Craig Penner, Jeffrey A. Housman,
and Jared C. Duensing

*Computational Aerosciences Branch
NASA Ames Research Center*

Advanced Modeling & Simulation (AMS) Seminar Series
NASA Ames Research Center, December 12th, 2024

NASA's Aeronautics Research Mission Directorate (ARMD) is focused on the transformation of aviation through four key technological areas



Ultra-Efficient
Airliners



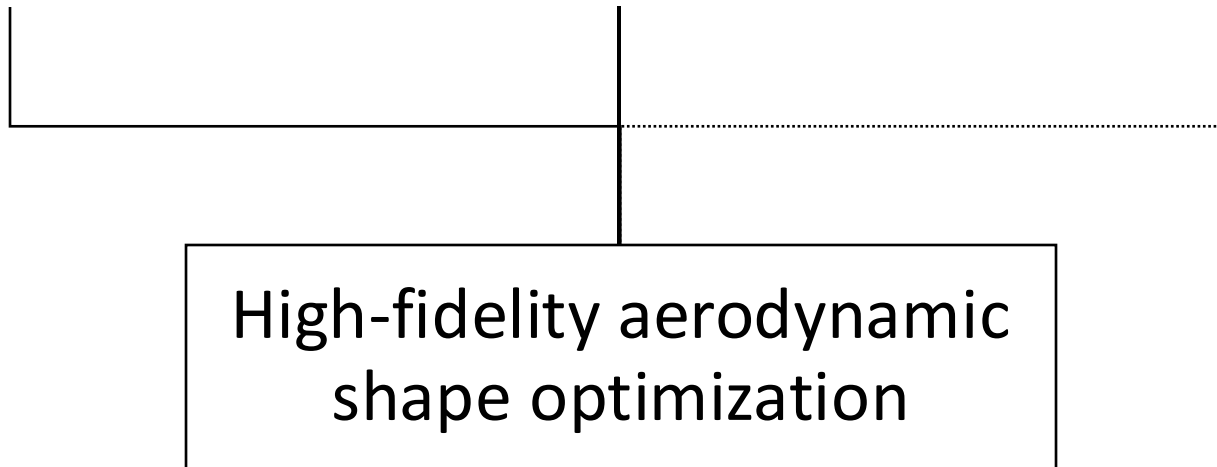
High-Speed
Commercial Flight



Advanced Air
Mobility

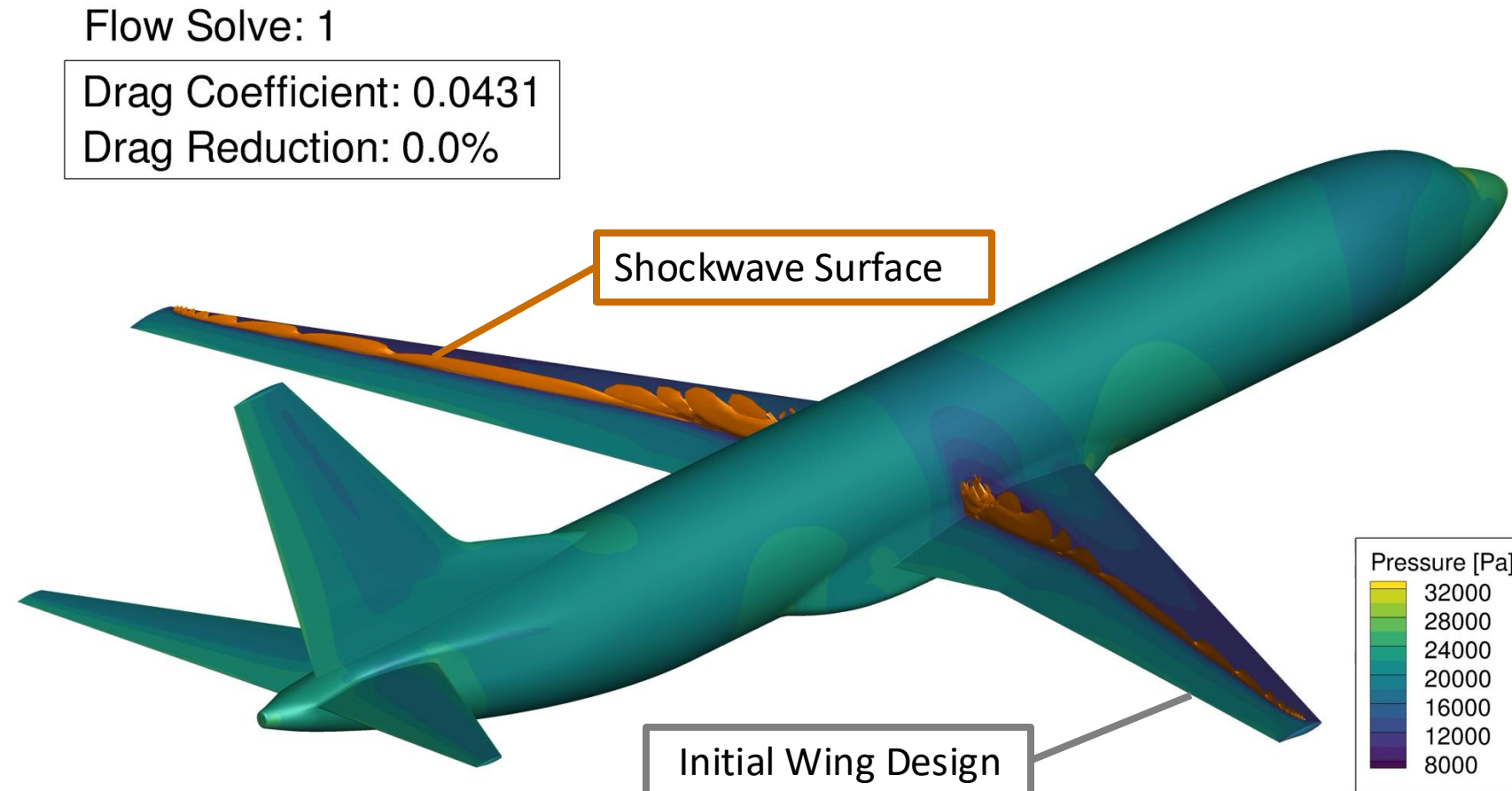


Future Airspace
and Safety



Aerodynamic shape optimization aims to modify the shape of a geometry to improve its performance

Shown is an example of designing the wing of a single-aisle aircraft for minimum drag





Our goal is to develop methods for accurate, robust, and efficient aerodynamic shape optimization

Accuracy

- Sufficient modeling fidelity for flow physics (viscous, compressible)

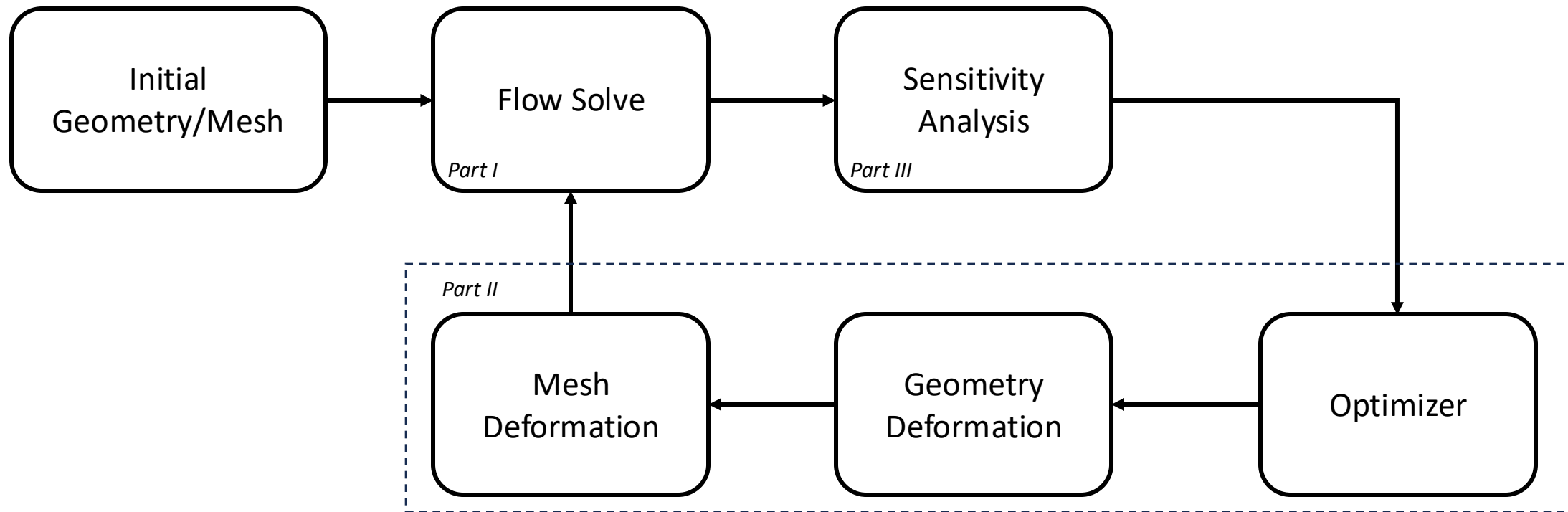
Robustness

- Robust flow solver capable solving on complex and unconventional geometries
- Robust surface and volume mesh deformations for component-based design

Efficiency

- Computationally efficient algorithms for flow analyses and gradient computations

This flow chart depicts the general approach for aerodynamic shape optimization





- Part I – LAVA Framework
- Part II – Geometry Parameterization, Mesh Deformation, Optimizer
- Part III – Gradient Calculations
- Part IV – Results
- Part V – Future Work



Part I – LAVA Framework

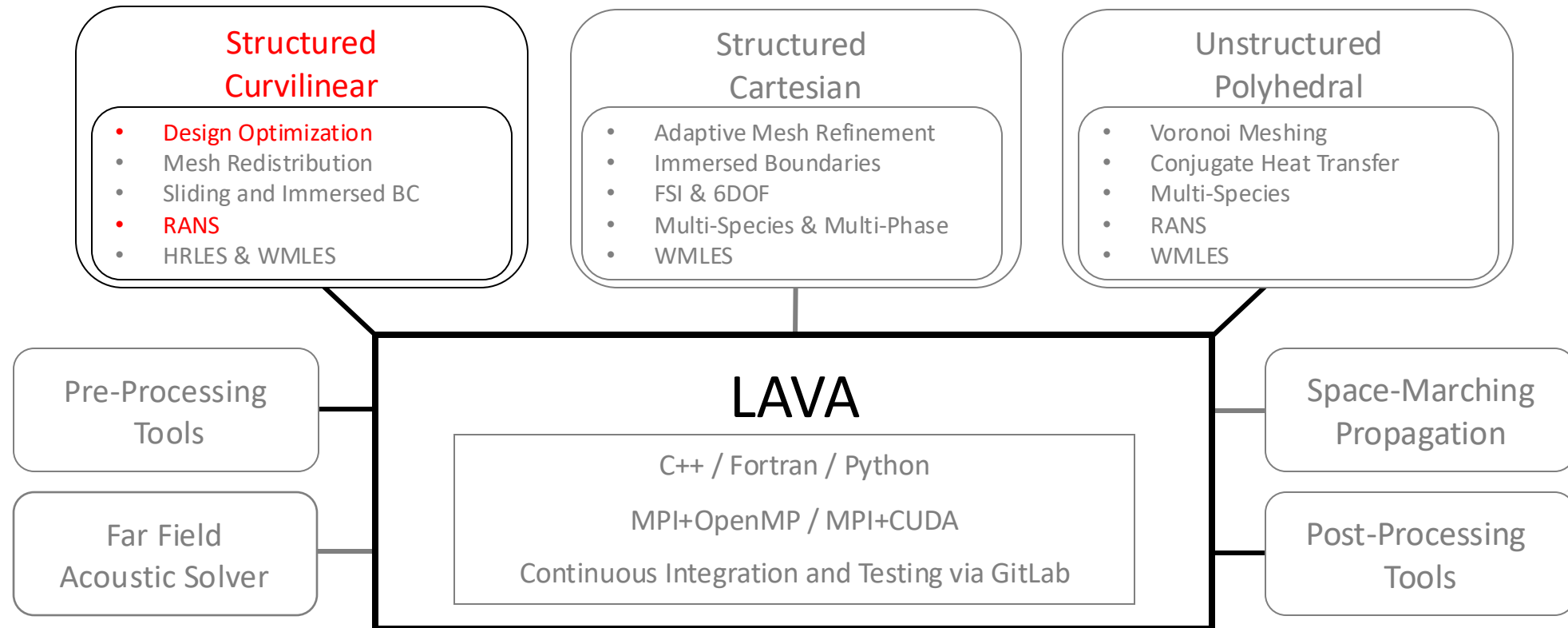
Part II – Geometry Parameterization, Mesh Deformation, Optimizer

Part III – Gradient Calculations

Part IV – Results

Part V – Future Work

The Launch, Ascent, and Vehicle Aerodynamics (LAVA) framework is a comprehensive computational tool





This work uses the LAVA curvilinear overset solver

Governing Flow Equations:

- Compressible Reynolds-averaged Navier-Stokes (RANS) equations in strong conservation form
- Spalart-Allmaras turbulence model (negative form) with rotation curvature (RC), compressibility correction (CC), and Quadratic Constitutive Relation, 2000 version (QCR2000)

Mesh topology: Curvilinear structured overset

Discretization: Second-order finite-difference approach

Solution Method: Approximate Newton-Krylov with preconditioned GMRES



Part I – LAVA Framework

Part II – Geometry Parameterization, Mesh Deformation, Optimizer

Part III – Gradient Calculations

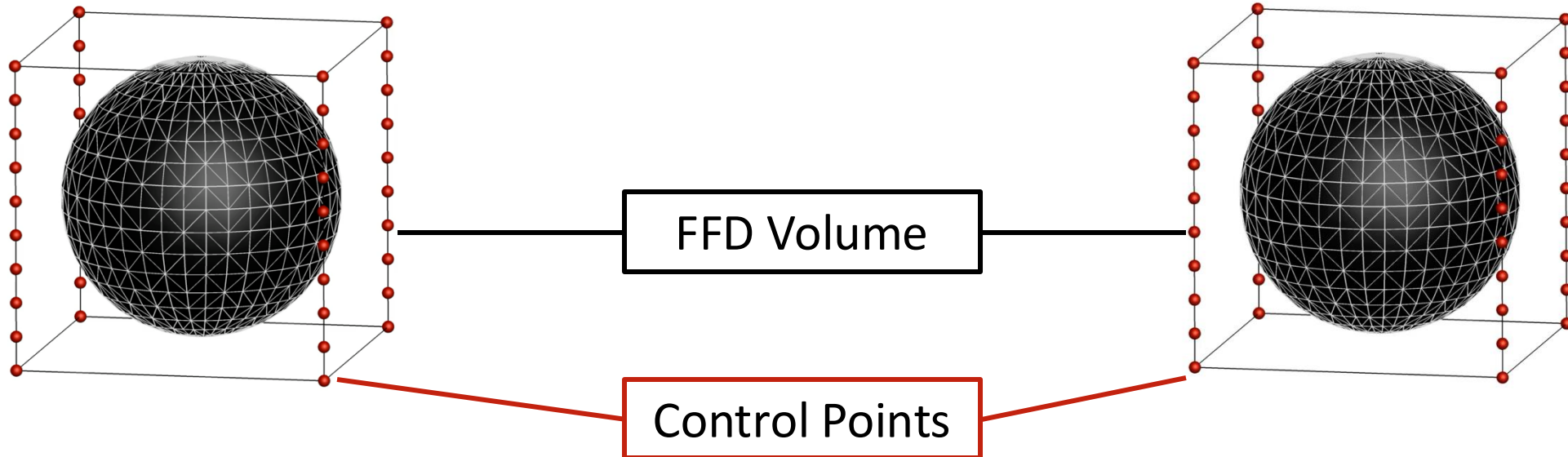
Part IV – Results

Part V – Future Work

Geometric parameterization is accomplished with Freeform Deformation (FFD) Volumes with the open-source package pyGeo¹

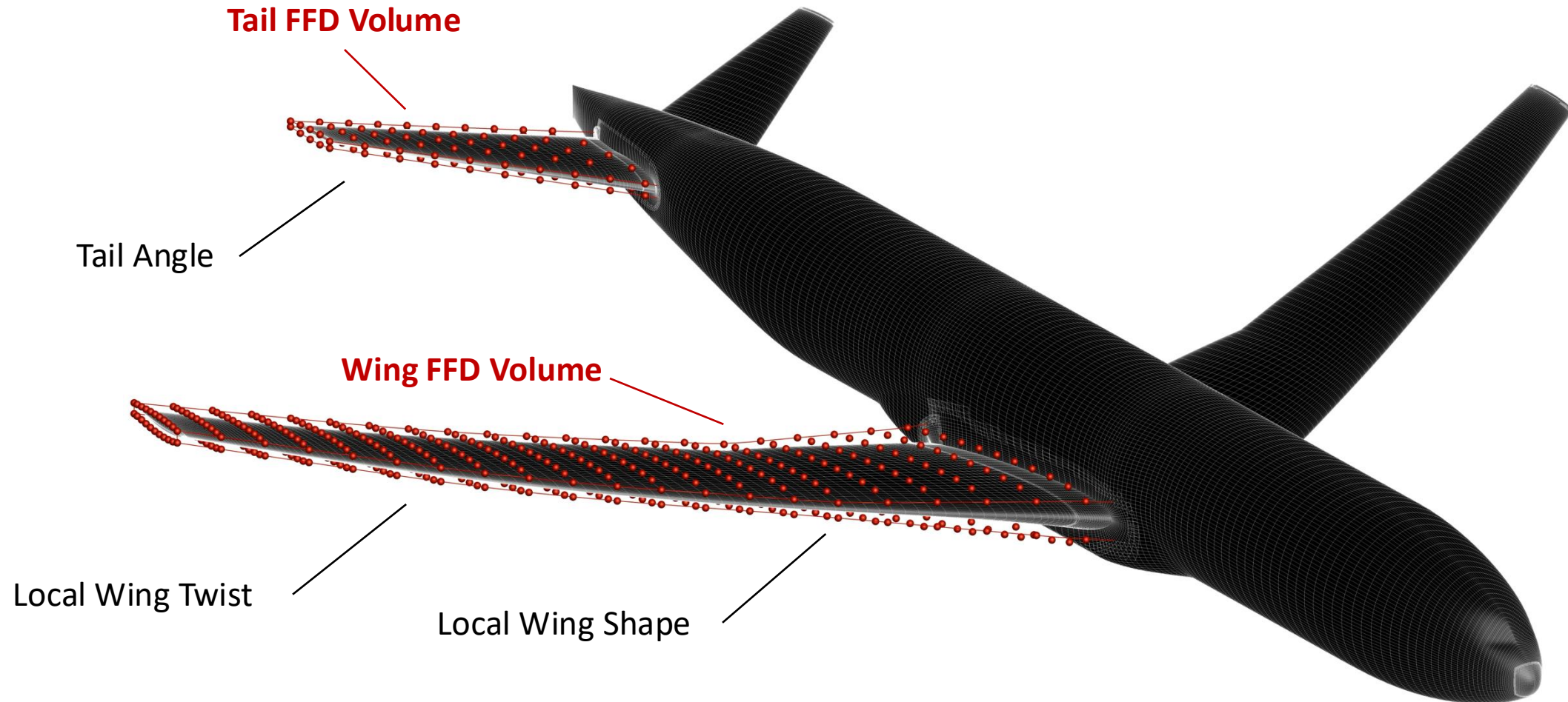
Local Shape Change

Global Shape Change



¹pyGeo: github.com/mdolab/pygeo

FFD volumes are placed around key components of the geometry

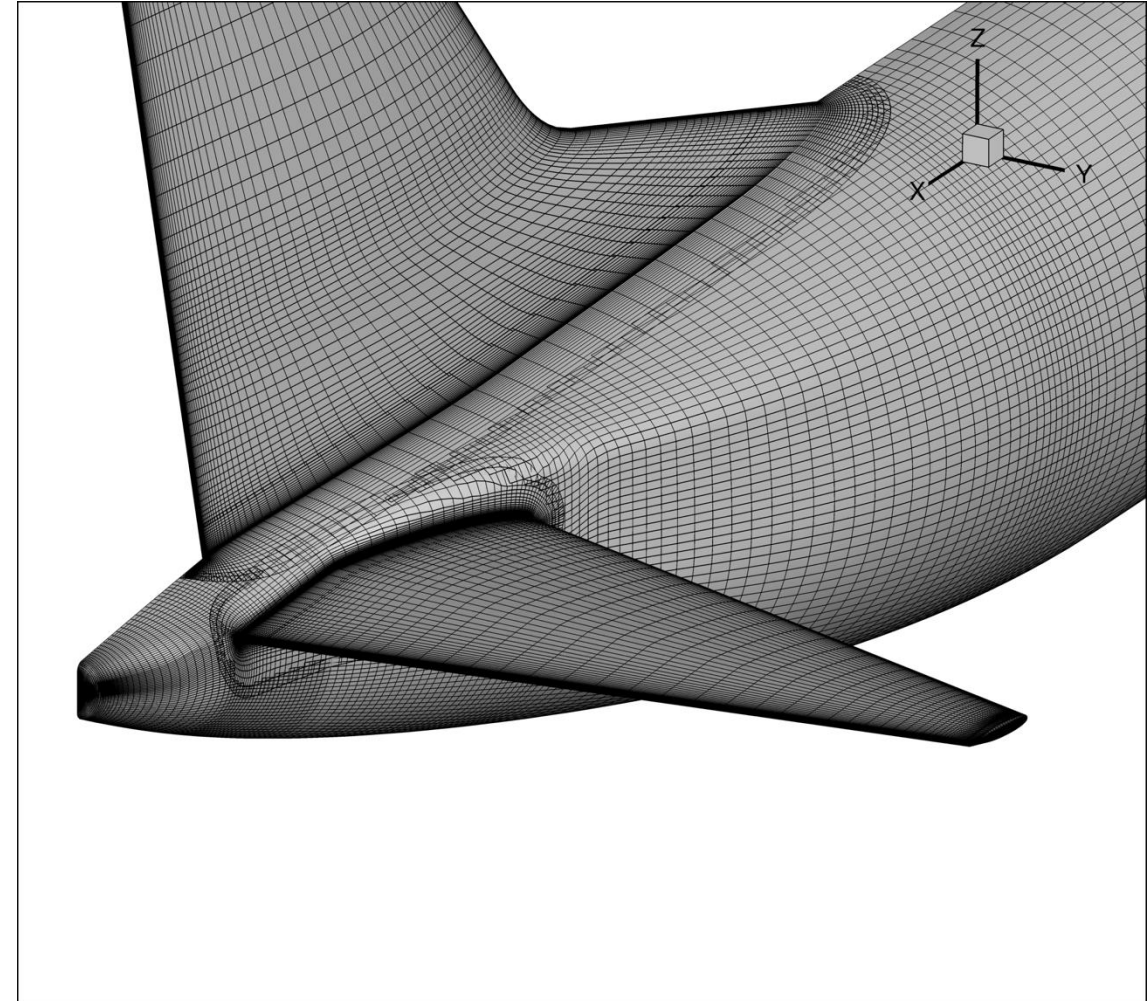


Surface deformation at component intersections is accomplished using pyGeo¹'s multi-component method

Deformation of the surface mesh at the intersections is challenging for body-fitted methods

We have found success in using pyGeo's multi-component capability:

- each component is parameterized in a separate FFD
- intersections are recomputed and used to deform the surface mesh



¹pyGeo: github.com/mdolab/pygeo

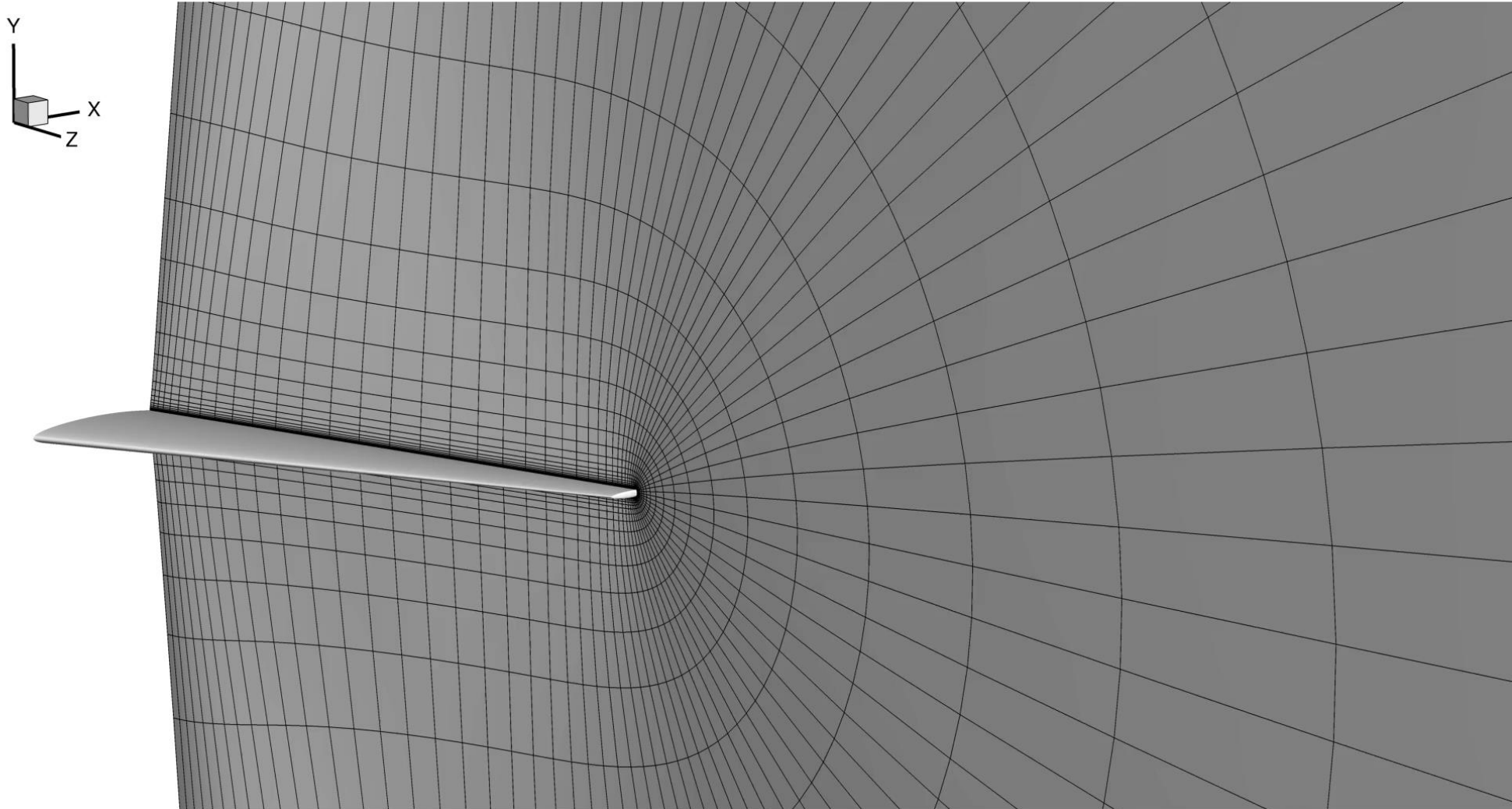


Mesh deformation is accomplished using an in-house implementation of the IDWarp¹ algorithm

- Surface mesh deformation is propagated to the volume mesh using an approximate inverse distance weighting method
- Computationally efficient: a k-dimensional (kd)-tree approximates deformations
 - Surface nodes are represented in a kd-tree structure
 - Surface node weights and induced deformations are condensed at each tree node
 - Volume deformation is approximated via tree traversal
- Highly parallelizable: volume node deformations depend only on surface node deformations

¹IDWarp: github.com/mdolab/idwarp

This mesh deformation scheme is sufficiently robust
for our purposes





Gradient-based optimization is used due to its efficiency

- Gradient-based methods require fewer function evaluations than gradient-free methods
- The Sparse Nonlinear OPTimizer (SNOPT¹) is used in this work
- SNOPT is an optimizer designed to solve large-scale constrained optimization problems with smooth nonlinear objective and constraint functions
- pyOptSparse² is used to interface with SNOPT through Python

¹SNOPT: ccom.ucsd.edu/~optimizers/solvers/snopt/

²pyOptSparse : github.com/mdolab/pyoptsparse



Part I – LAVA Framework

Part II – Geometry Parameterization, Mesh Deformation, Optimizer

Part III – Gradient Calculations

Part IV – Results

Part V – Future Work

Gradients are computed using the discrete-adjoint method

Primal (RANS + SA model) : $R(\mathbf{X}, \mathbf{Q}) = 0$

Discrete Adjoint : $\frac{\partial R^T}{\partial \mathbf{Q}} \psi = \frac{\partial J^T}{\partial \mathbf{Q}}$

Gradient : $\frac{dJ}{d\mathbf{X}} = \frac{\partial J}{\partial \mathbf{X}} - \psi^T \frac{\partial R}{\partial \mathbf{X}}$

where $\mathbf{X}$ = design variables

$\mathbf{Q}$ = flow states

ψ = adjoint vector

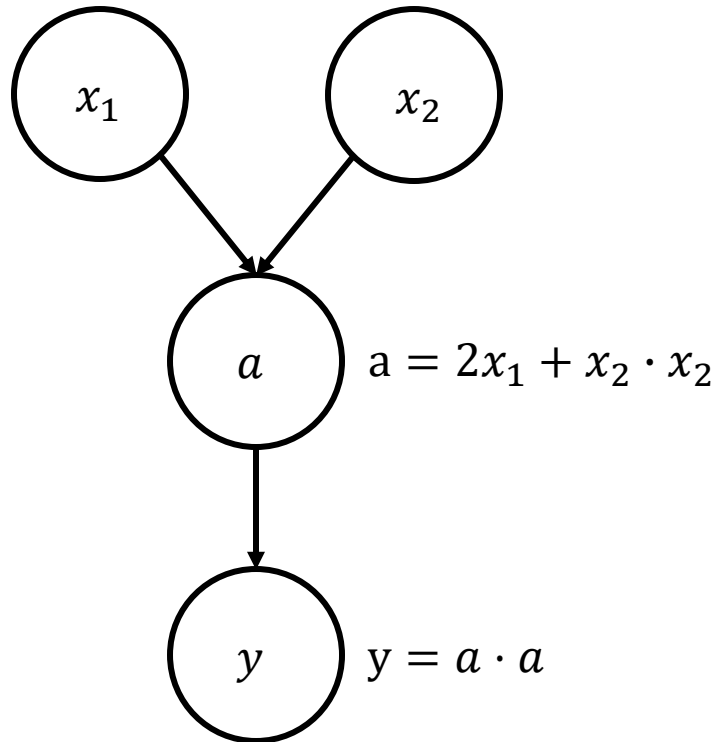
$J(\mathbf{X}, \mathbf{Q})$ = functional

Key Points:

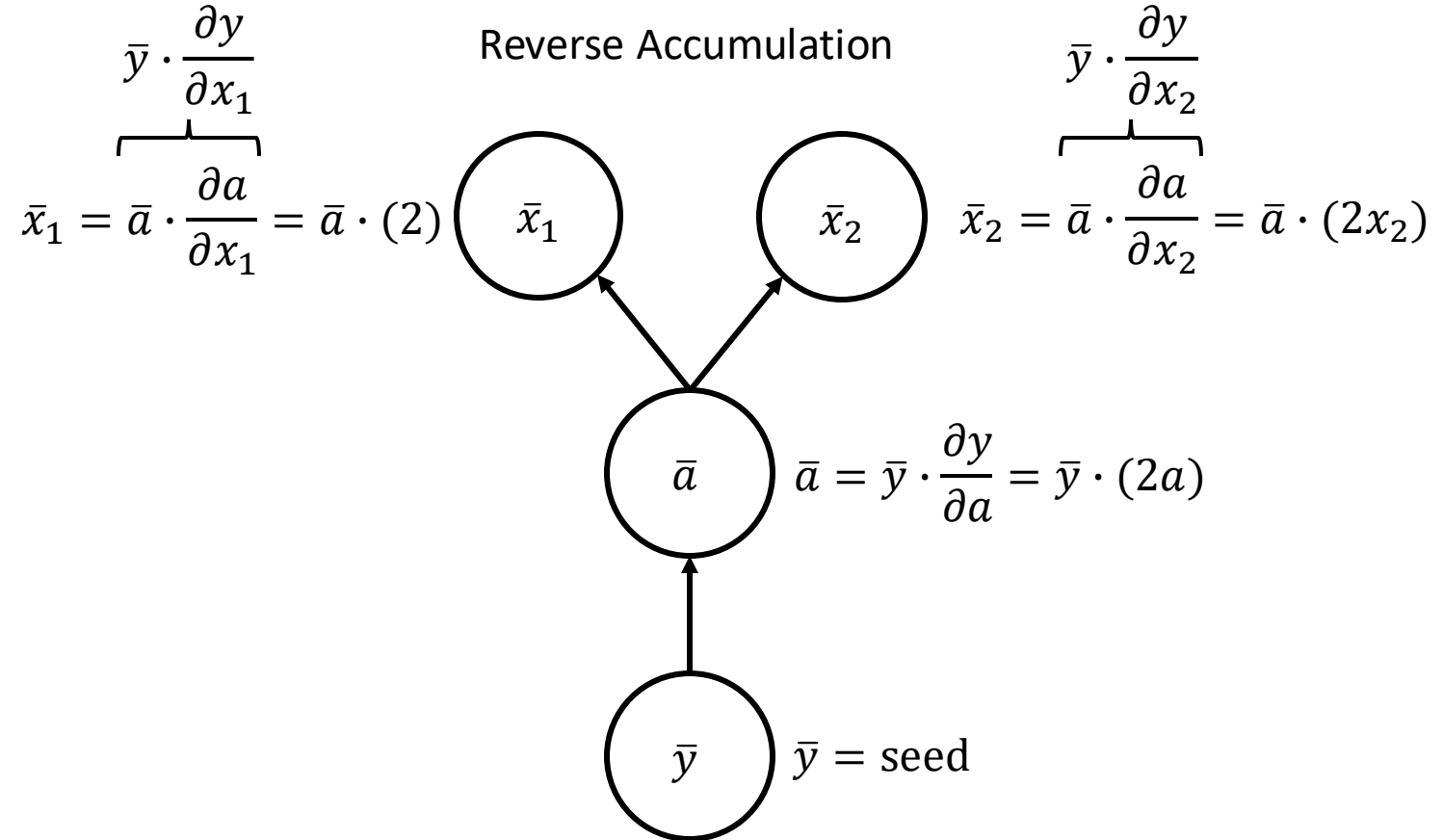
- Only implementation of partial derivatives is required
- If using a Krylov subspace method for the discrete adjoint, we only need matrix-vector products for the transposed residual Jacobian

LAVA computes partial derivatives using reverse accumulation differentiation

Forward Computation



Reverse Accumulation



LAVA utilizes both automatic- and hand-differentiation for its implementation of reverse accumulation

Automatic Differentiation

- Source code transformation is used to automatically differentiate single purpose subroutines
- We use the tool TAPENADE¹ for automatic-differentiation

Hand Differentiation

- The sequence of calls to the auto-differentiated routines are hand-coded into the larger code base
- MPI and OpenMP parallelization is also hand-coded to optimize communication and load distribution

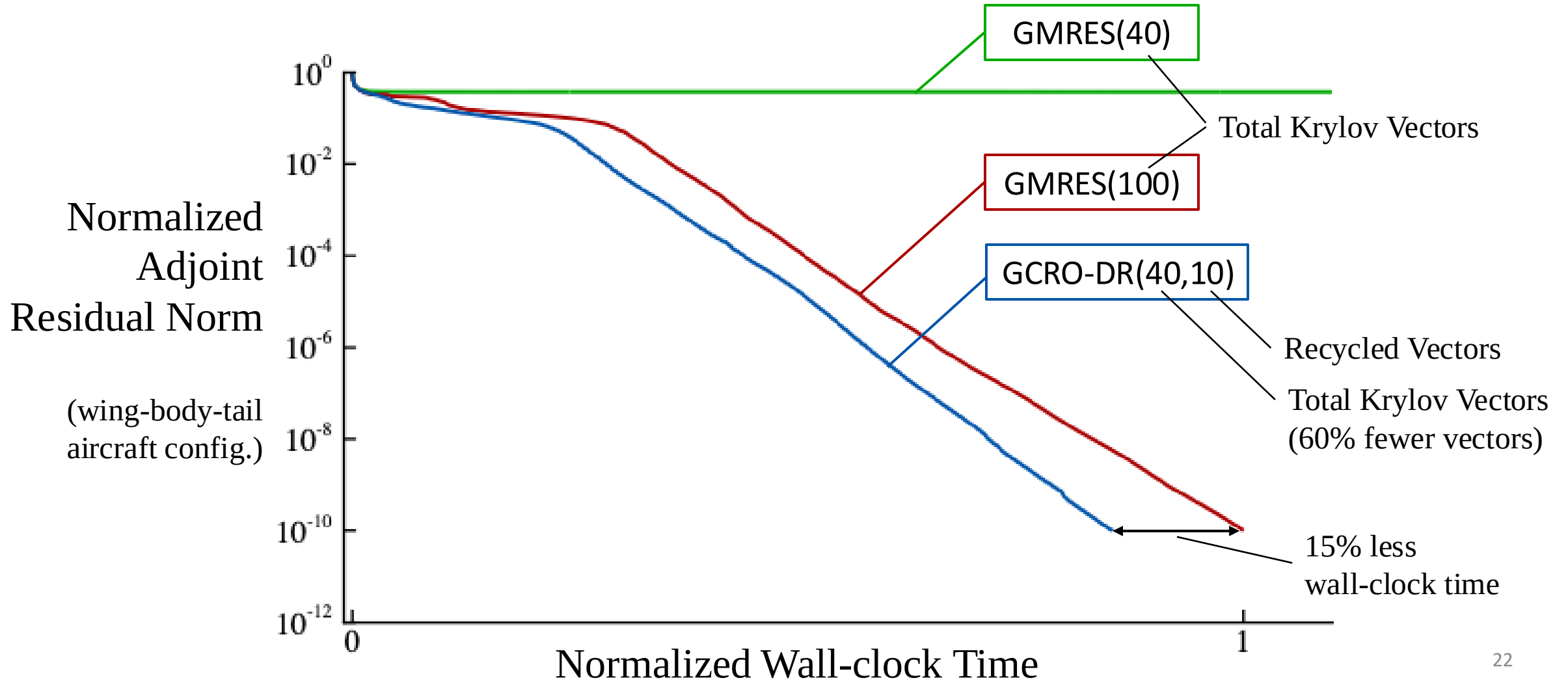
¹TAPENADE: team.inria.fr/ecuador/en/tapenade

The discrete-adjoint problem is solved using an in-house implementation of GCRO-DR

$$\text{Discrete Adjoint : } \frac{\partial R^T}{\partial \mathbf{Q}} \psi = \frac{\partial J^T}{\partial \mathbf{Q}}$$

- GCRO-DR is a Krylov subspace method that uses deflated restarting
- It recycles an approximate invariant subspace to deflate eigenvalues of smallest magnitude
- The preconditioner is obtained from the incomplete LU factorization of a transposed first-order residual Jacobian

For our purposes, GCRO-DR has been found to be faster, more memory efficient, and more robust than restarted GMRES

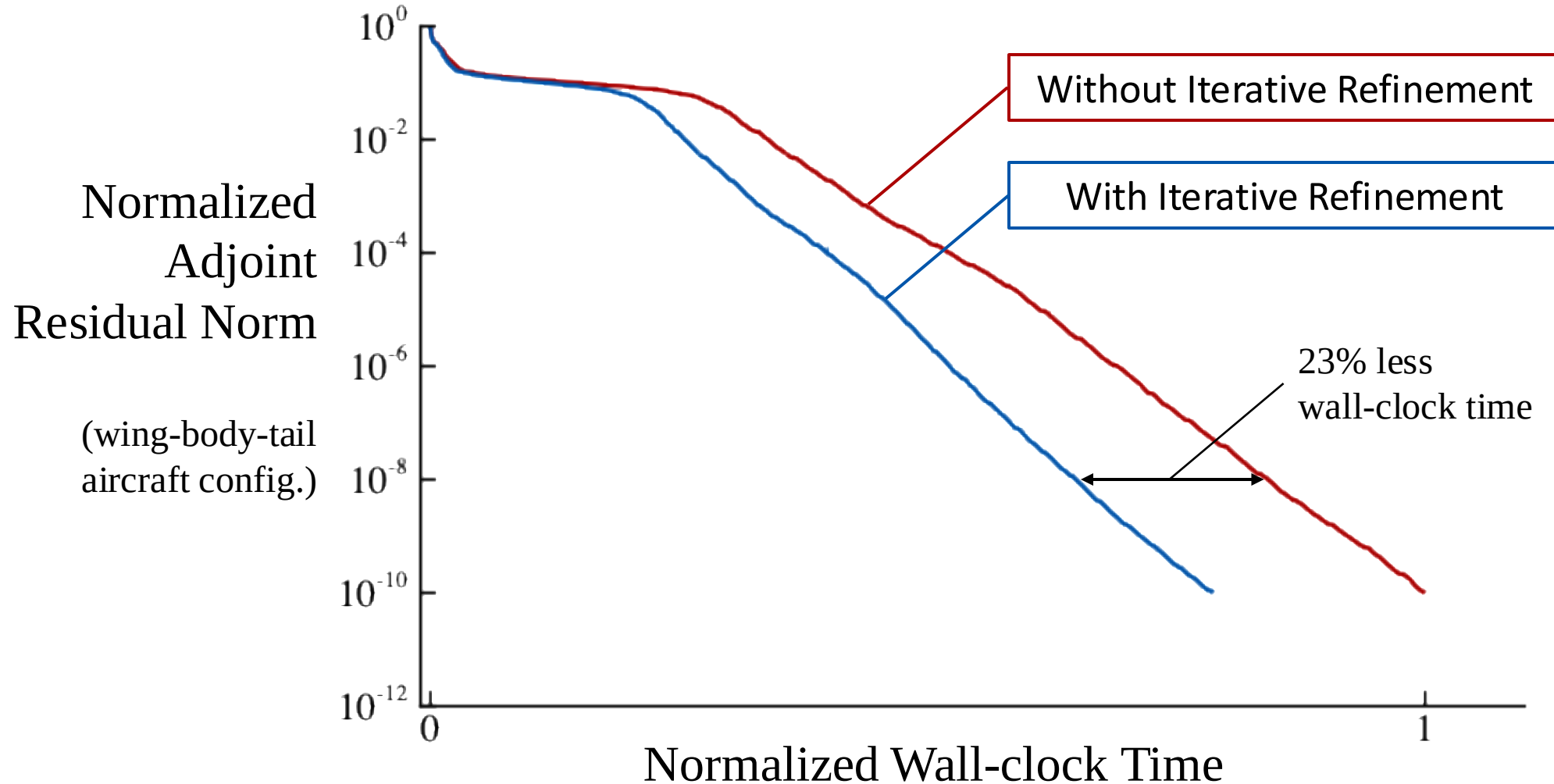


Iterative refinement is used to obtain a double precision adjoint solution with single precision linear solves

The following are the steps used to solve $Ax = b$:

- 1) Compute Residual: $r_i = b - Ax_i$ (double precision)
- 2) Solve: $A\Delta x = r_i$ (single precision)
- 3) Update: $x_{i+1} = x_i + \Delta x$ (double precision)

Iterative refinement has been found to provide around 20% computational cost savings



Wall-clock time comparisons between the flow and adjoint solvers



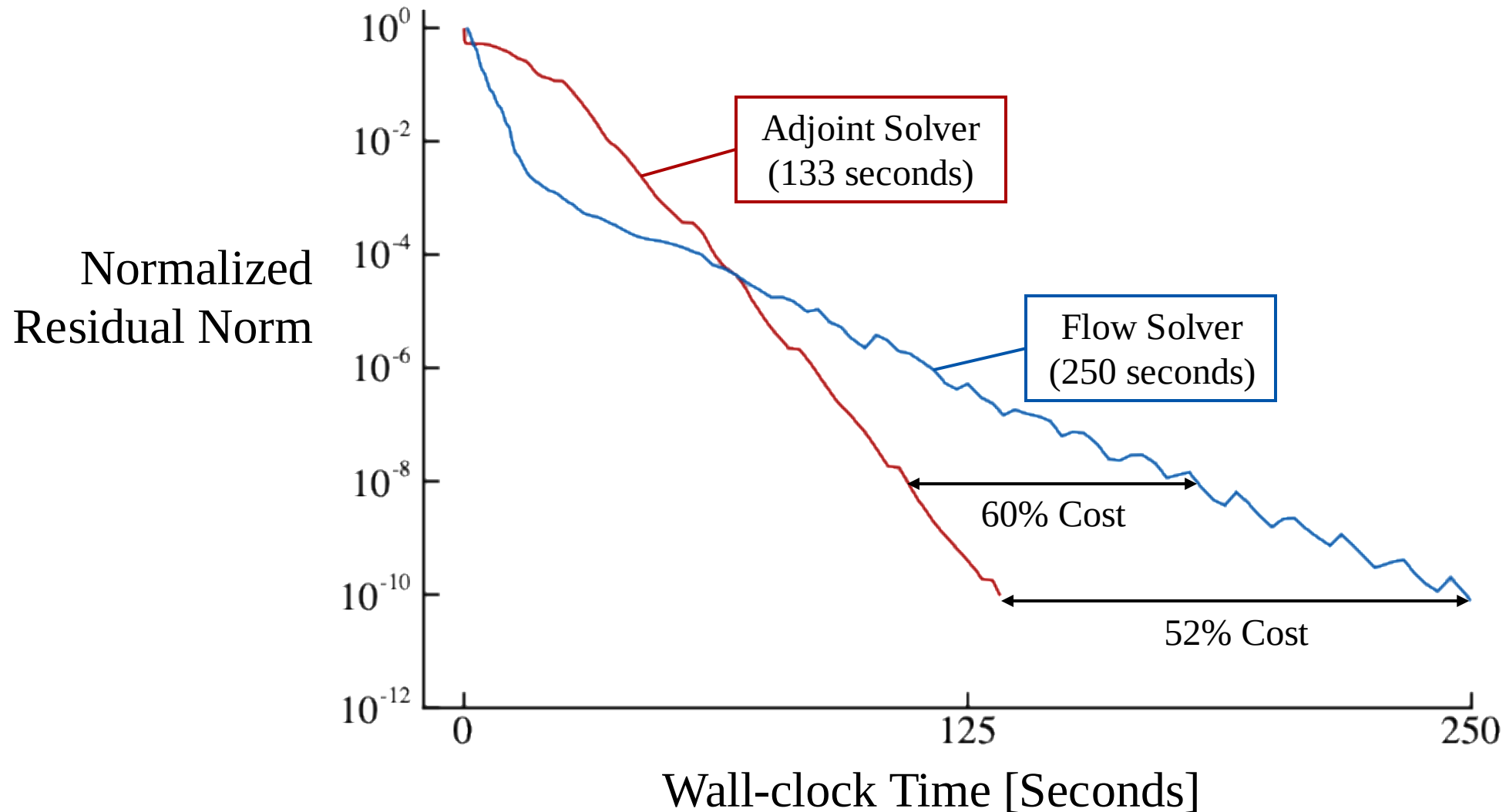
Case Details:

- Wing-only geometry
- 4 million node multi-block mesh
- 2 MPI rank, 64 OpenMP threads

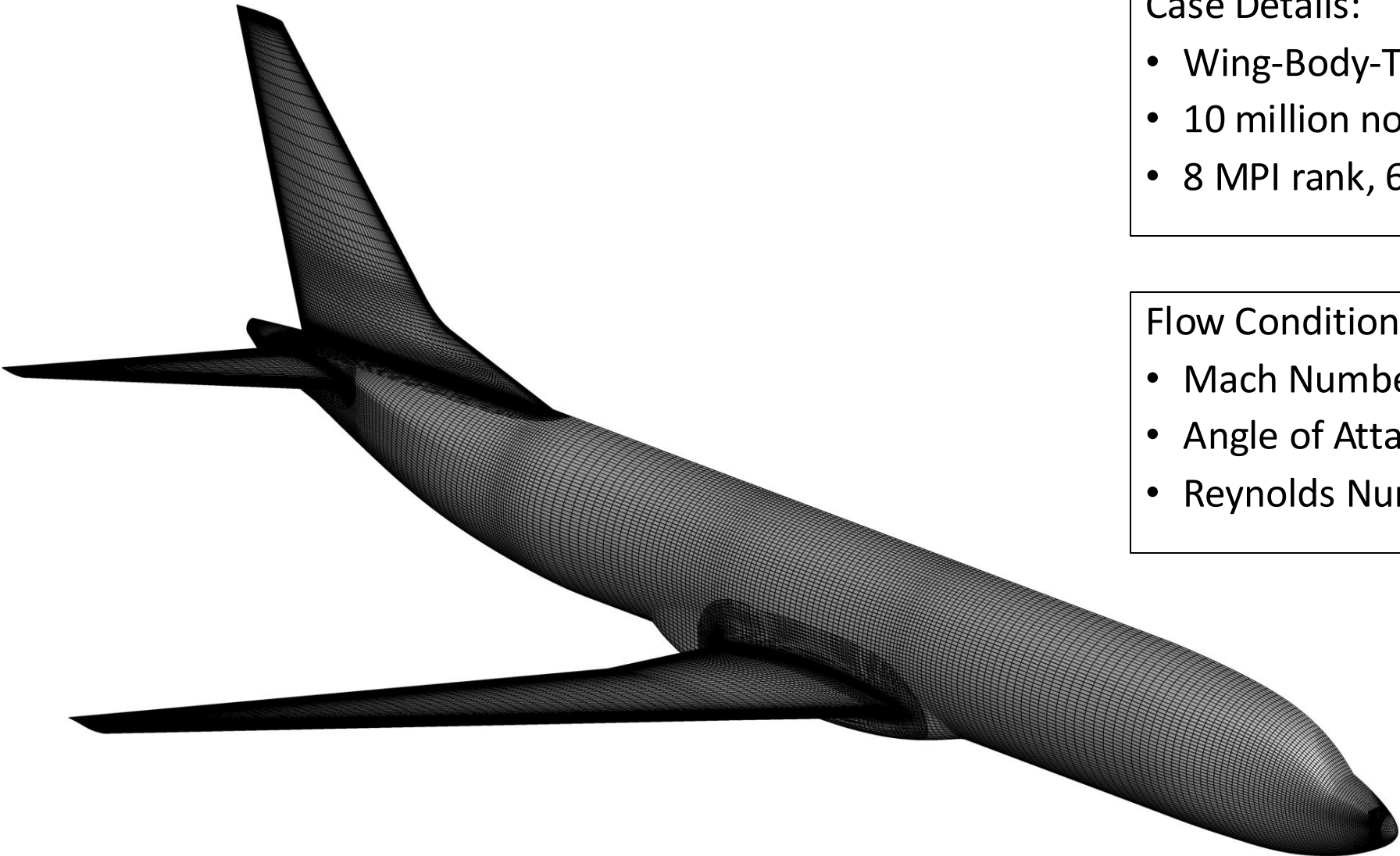
Flow Conditions:

- Mach Number: 0.85
- Angle of Attack: 2.3°
- Reynolds Number: 5 million

Wall-clock time comparisons between the flow and adjoint solvers



Wall-clock time comparisons between the flow and adjoint solvers



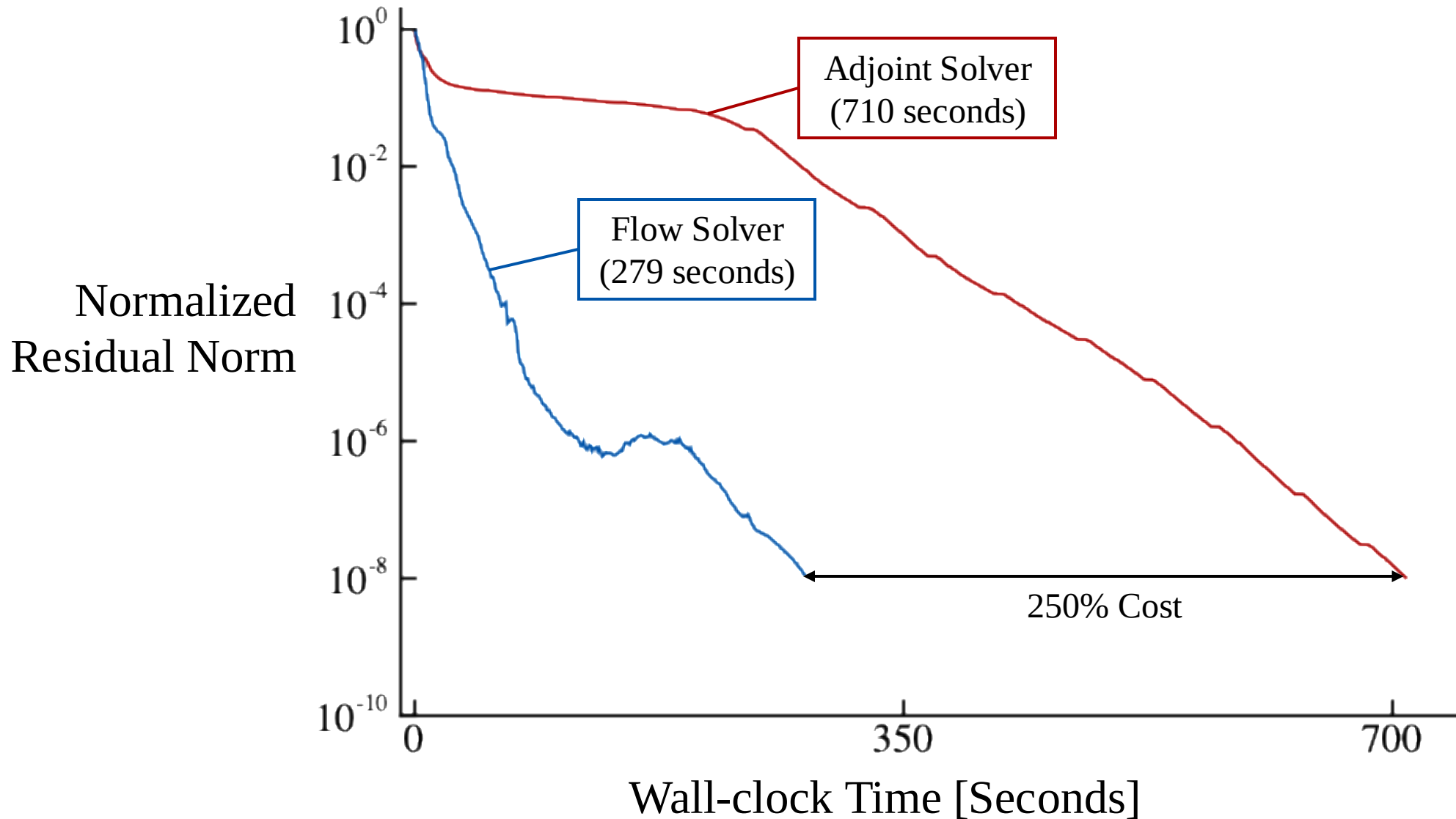
Case Details:

- Wing-Body-Tail geometry
- 10 million node overset mesh
- 8 MPI rank, 64 OpenMP threads

Flow Conditions:

- Mach Number: 0.78
- Angle of Attack: 2.0°
- Reynolds Number: 24.1 million

Wall-clock time comparisons between the flow and adjoint solvers





Part I – LAVA Framework

Part II – Geometry Parameterization, Mesh Deformation, Optimizer

Part III – Gradient Calculations

Part IV – Results

Part V – Future Work

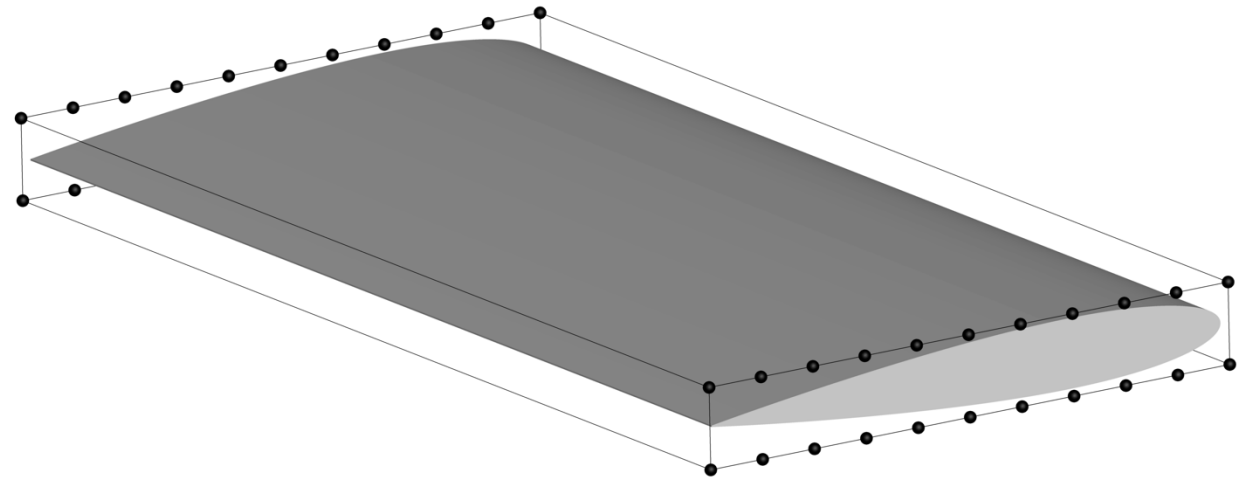
Inverse design of an approximate RAE 2822 airfoil based on a pressure distribution target

Objective Minimize $J = \sum_i \frac{(P_i - P_i^{target})^2}{P_\infty^2}$

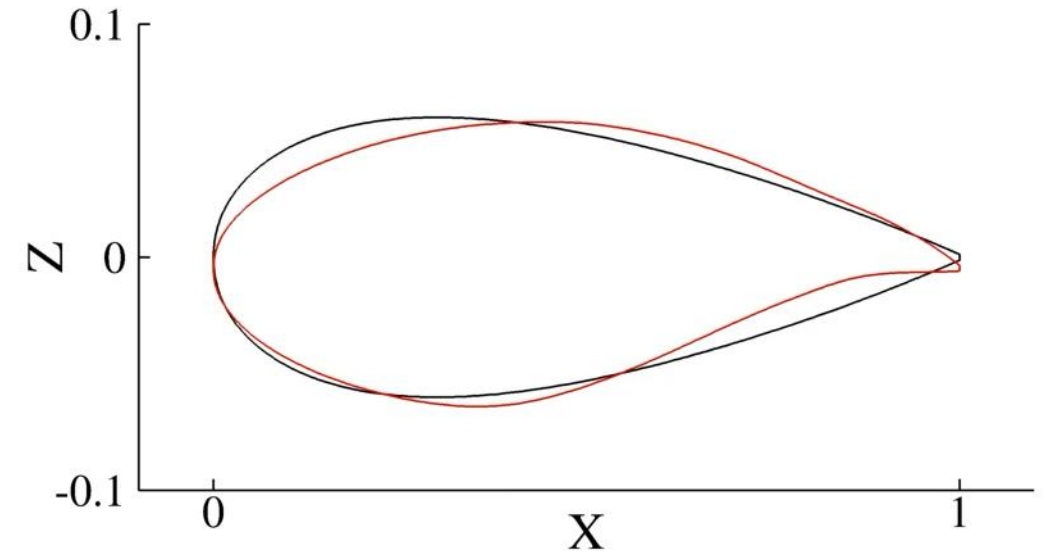
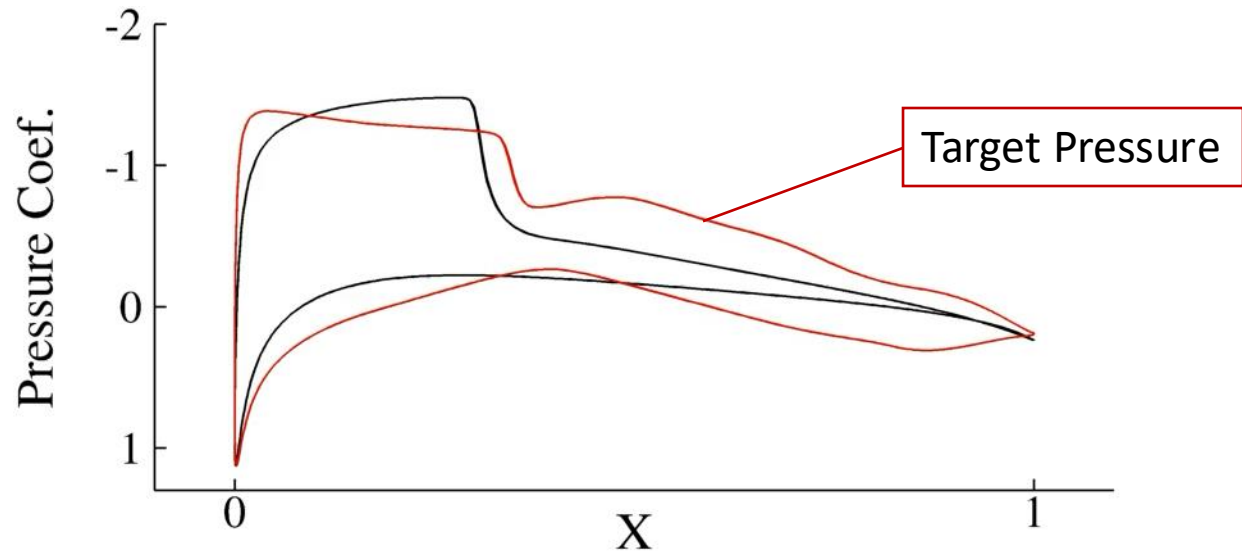
Flow Conditions

- Mach Number: 0.7
- Angle of attack : 3°
- Reynolds Number: 9 million

Design Variables z-displacement of FFD Volume Control Points



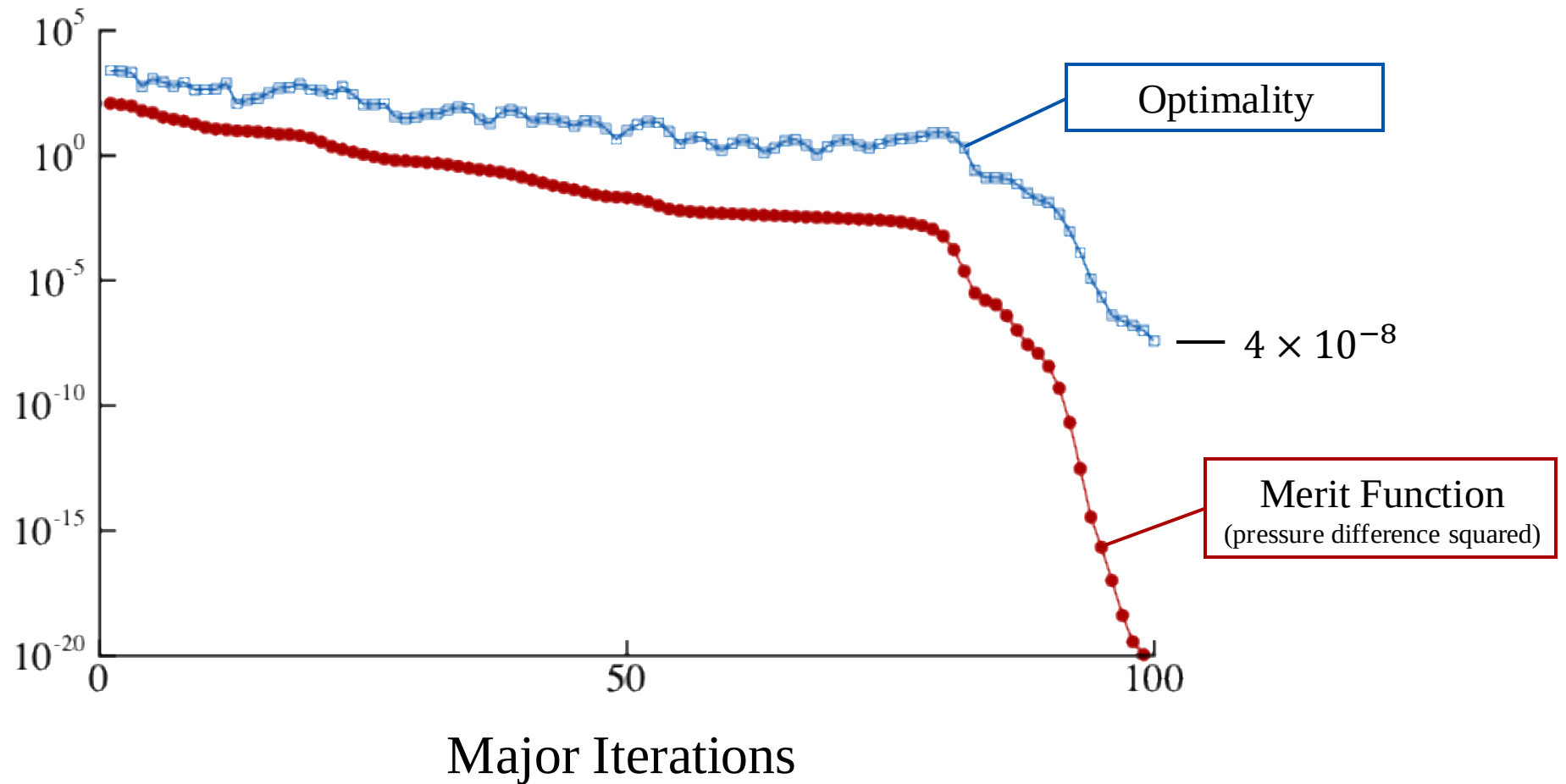
Inverse design of an approximate RAE 2822 airfoil based on a pressure distribution target



Function Evaluation : 1



Inverse design of an approximate RAE 2822 airfoil based on a pressure distribution target



Optimized twist distribution comparison with ADFlow¹

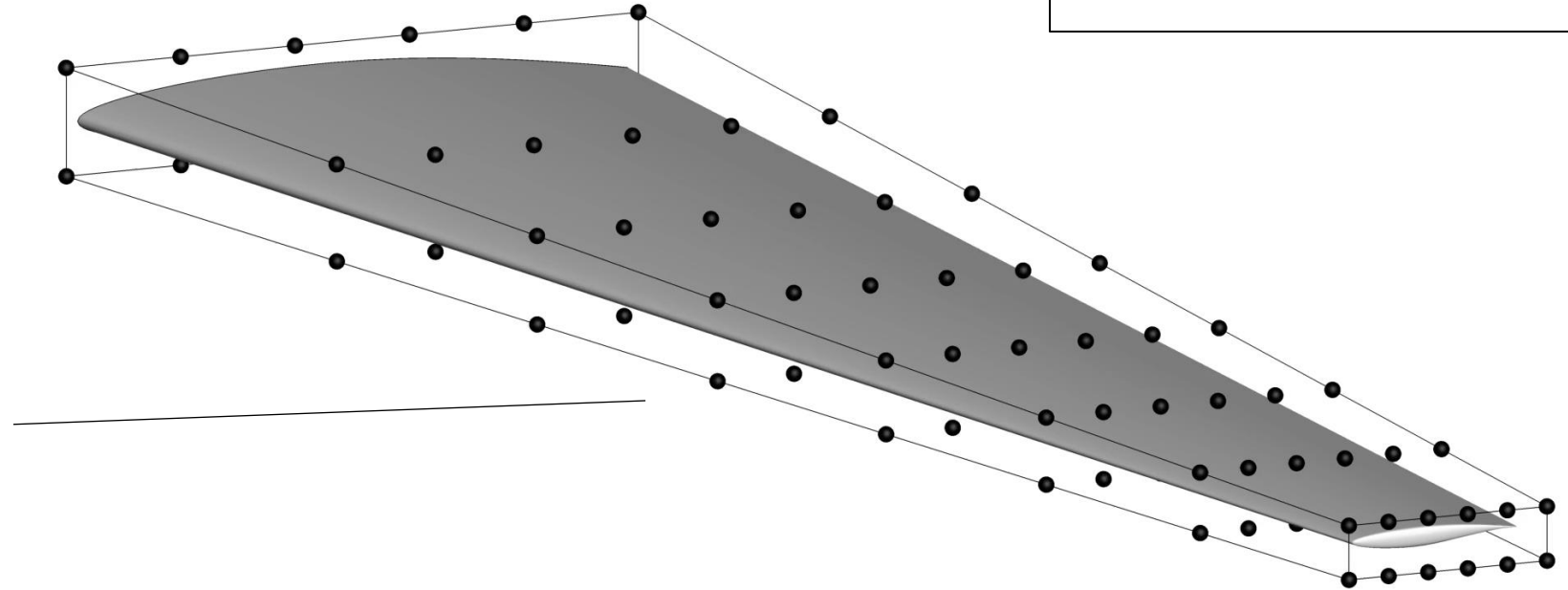
Flow Conditions

- Mach Number: 0.8
- Reynolds Number: 22.2 million

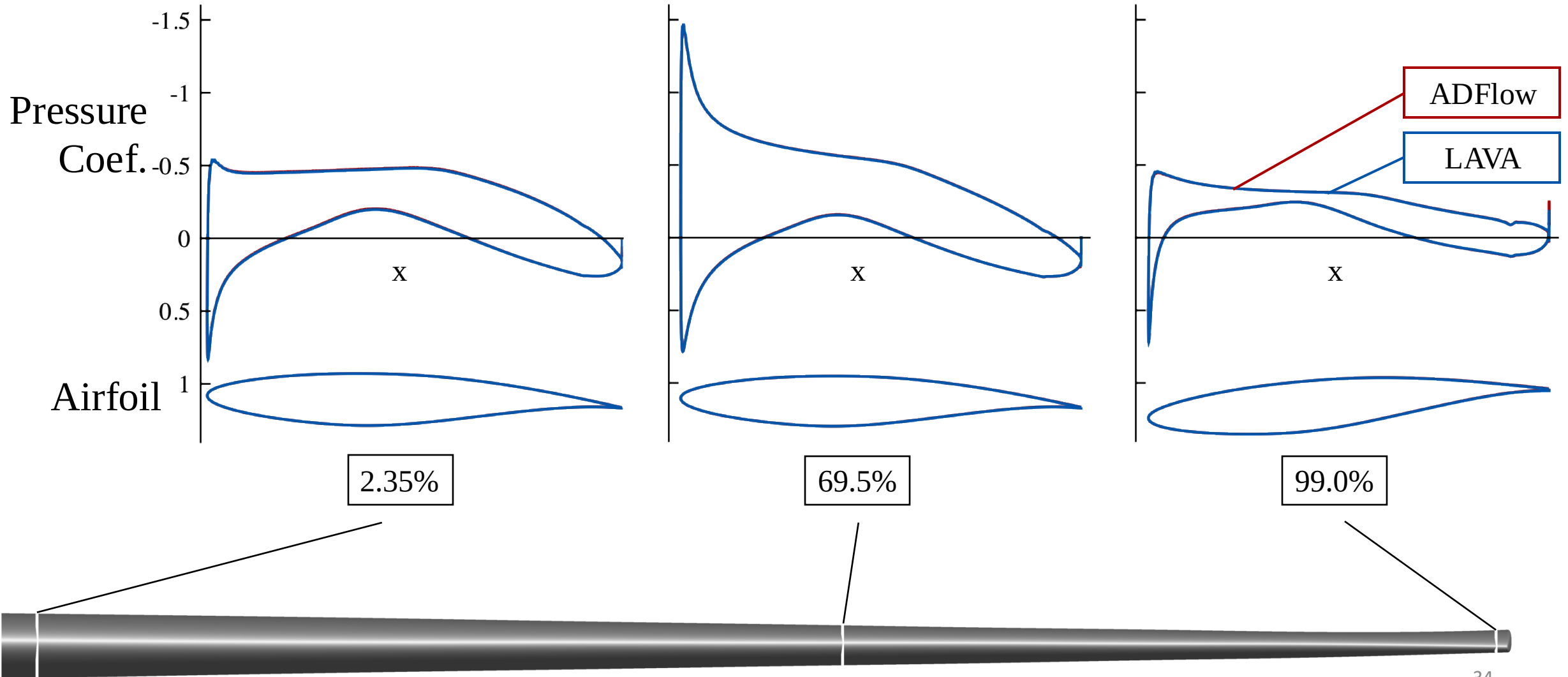
Objective Minimize C_D

Constraints $C_L = 0.5$

Design Variables Angle of attack
Local section twist



Optimized twist distribution comparison with ADFlow



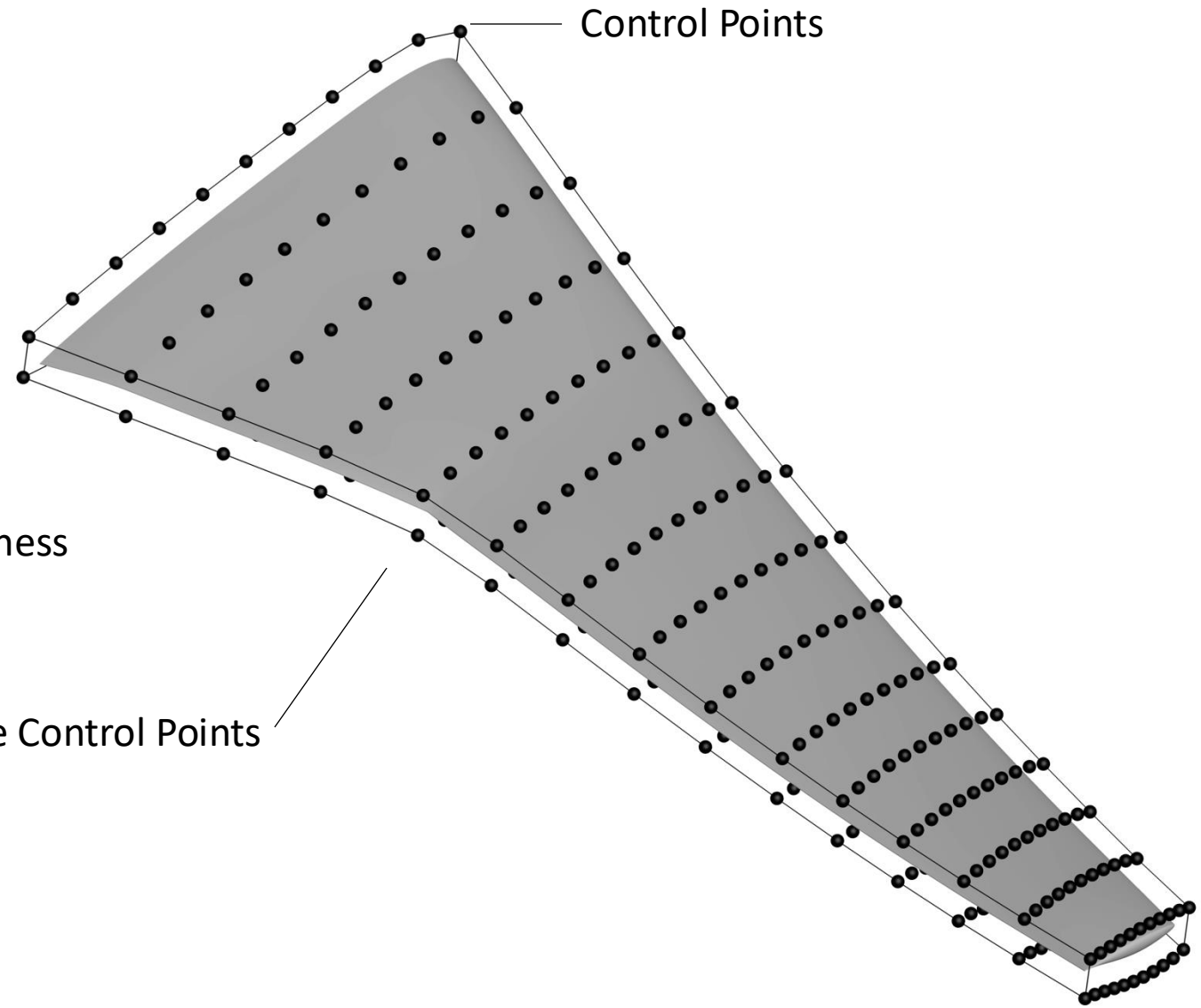
Aerodynamic Shape Optimization Test Case

CRM Wing

Objective Minimize C_D

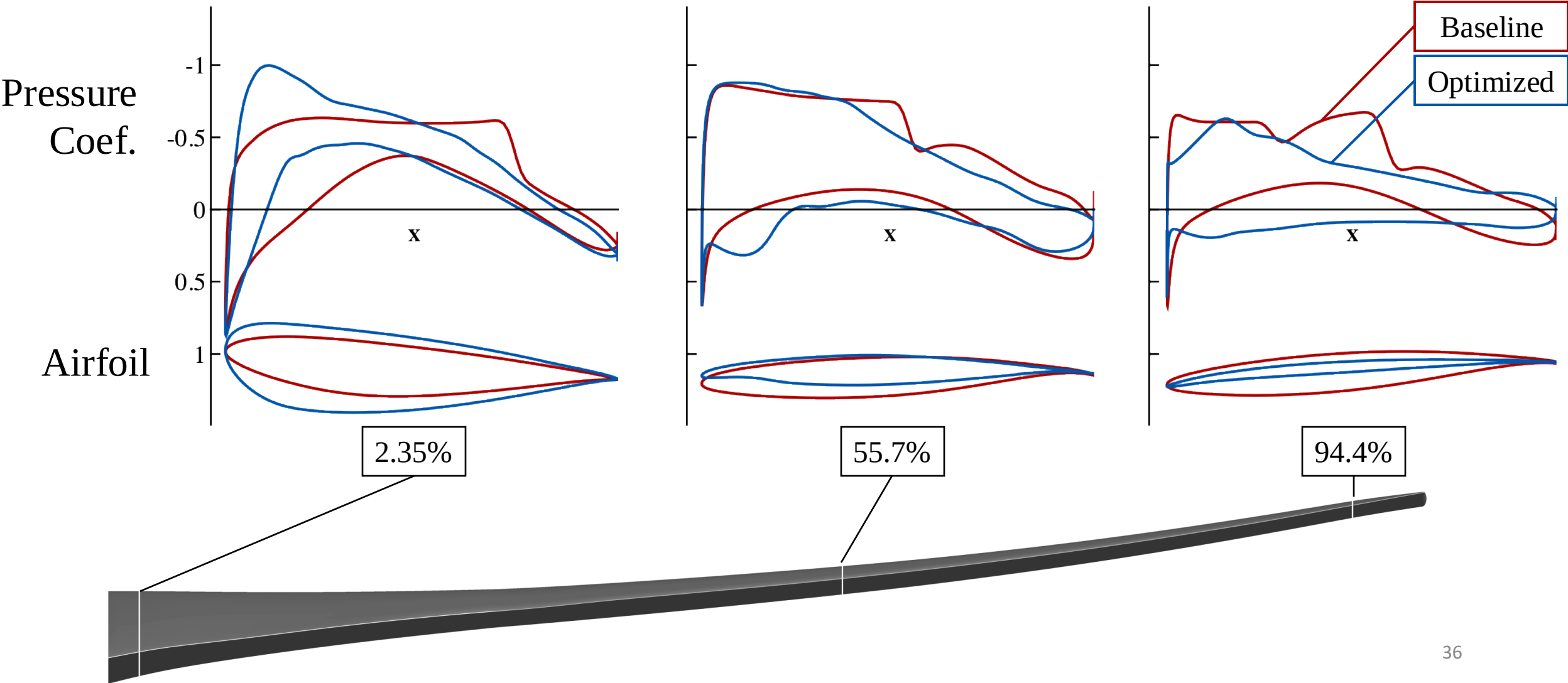
Constraints $C_L = 0.5$
 $C_{My} \geq -0.17$
Thickness $\geq 25\%$ Initial Thickness
Volume $\geq$ Initial Volume

Design Variables z-displacement of FFD Volume Control Points
Angle of Attack



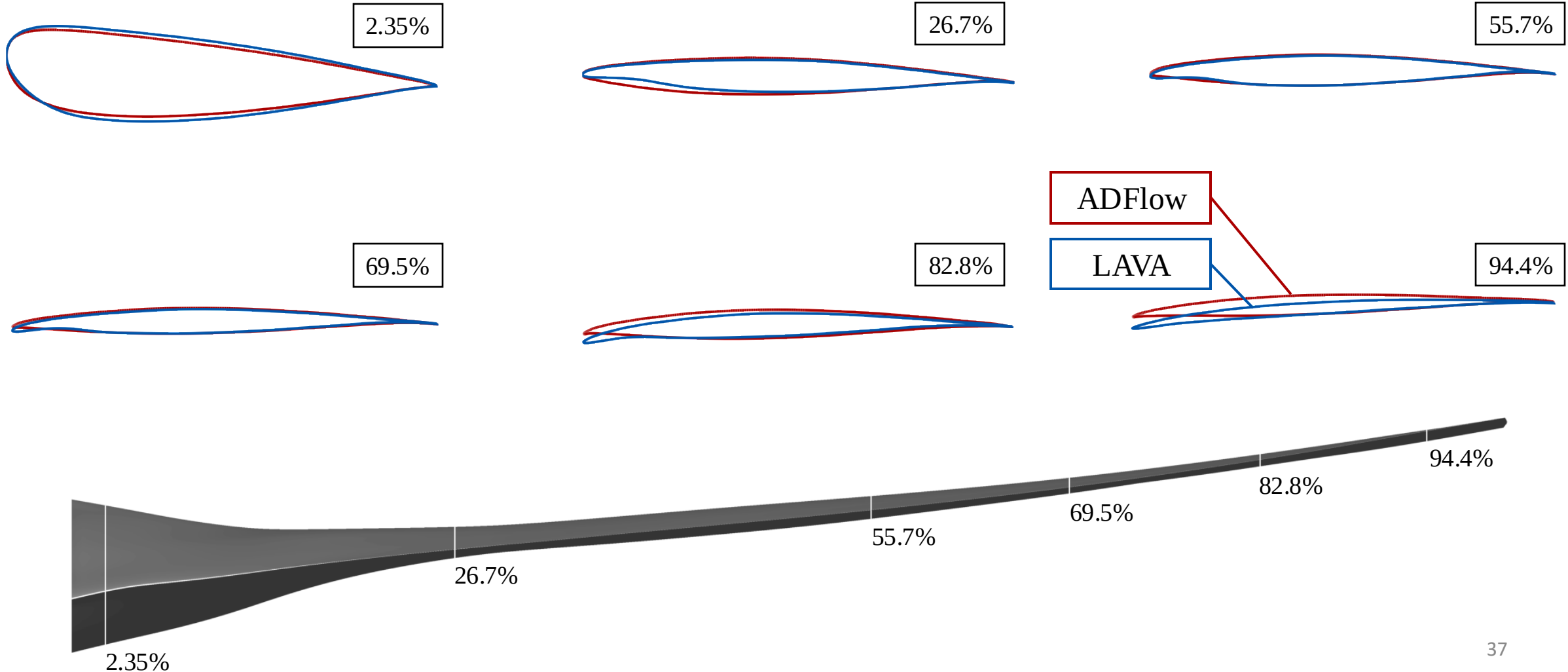
Aerodynamic Shape Optimization Test Case

CRM Wing

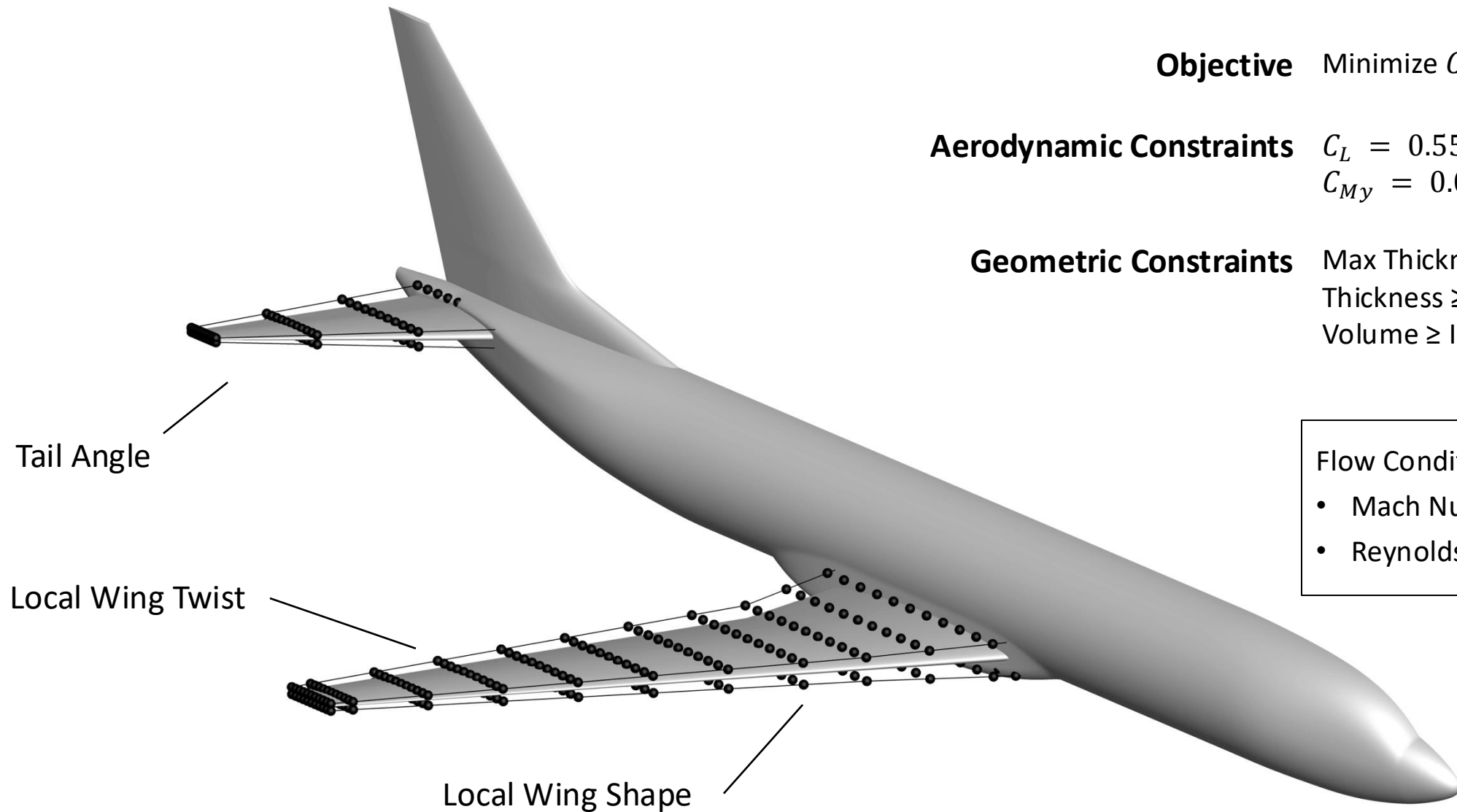


Aerodynamic Shape Optimization Test Case

CRM Wing



Optimization of a single-aisle conventional aircraft design



Objective Minimize C_D

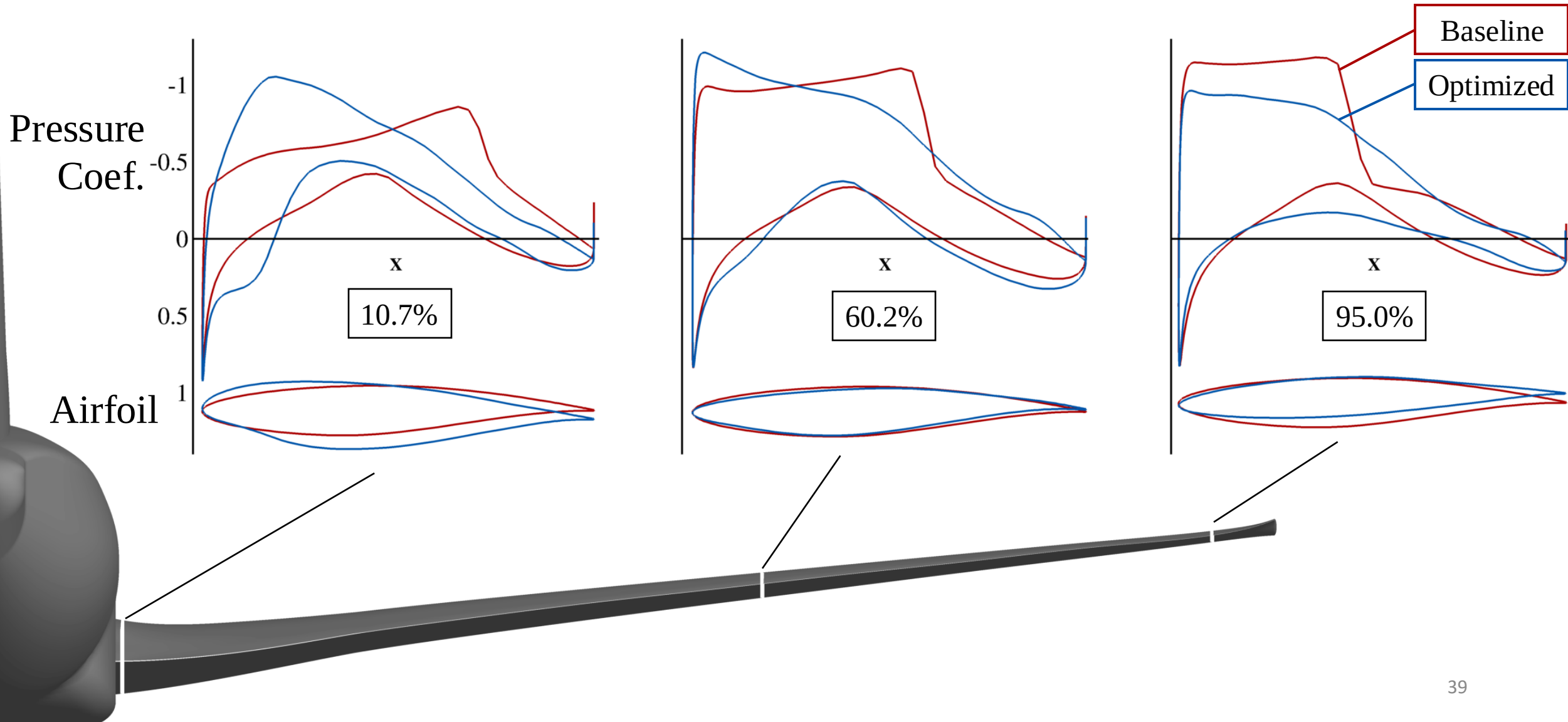
Aerodynamic Constraints $C_L = 0.556$
 $C_{My} = 0.0$

Geometric Constraints Max Thickness-to-Chord Distribution
 Thickness $\geq$ 80% Initial Thickness
 Volume $\geq$ Initial Volume

Flow Conditions

- Mach Number: 0.785
- Reynolds Number: 24.1 million

Optimization of a single-aisle conventional aircraft design



Optimization of a single-aisle conventional aircraft design

	Baseline	Optimized
Angle of Attack	3.41	3.00
Tail Angle	-3.27	-2.56
Lift Coef.	0.556	0.556
Pitching Moment Coef.	1×10^{-7}	7×10^{-8}
Drag Coef.	348 counts	280 Counts

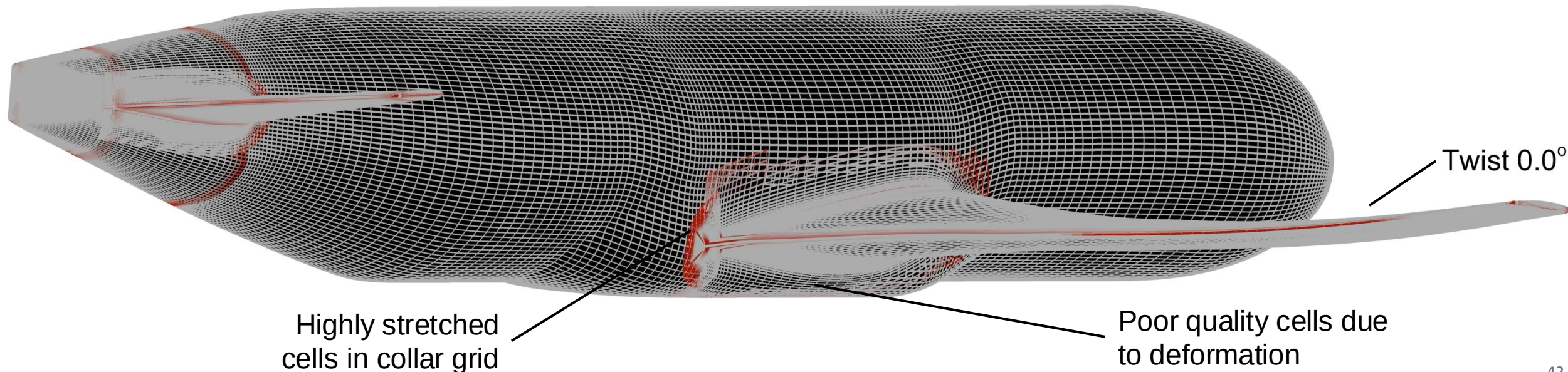
Processor Type	Number of processors	Function Evaluations	Total Wall Time	CPU Hours
Intel Broadwell (E5-2680v4)	1288 (46 nodes)	203	138 Hours	178,000



- Part I – LAVA Framework
- Part II – Geometry Parameterization, Mesh Deformation, Optimizer
- Part III – Gradient Calculations
- Part IV – Results
- Part V – Future Work**

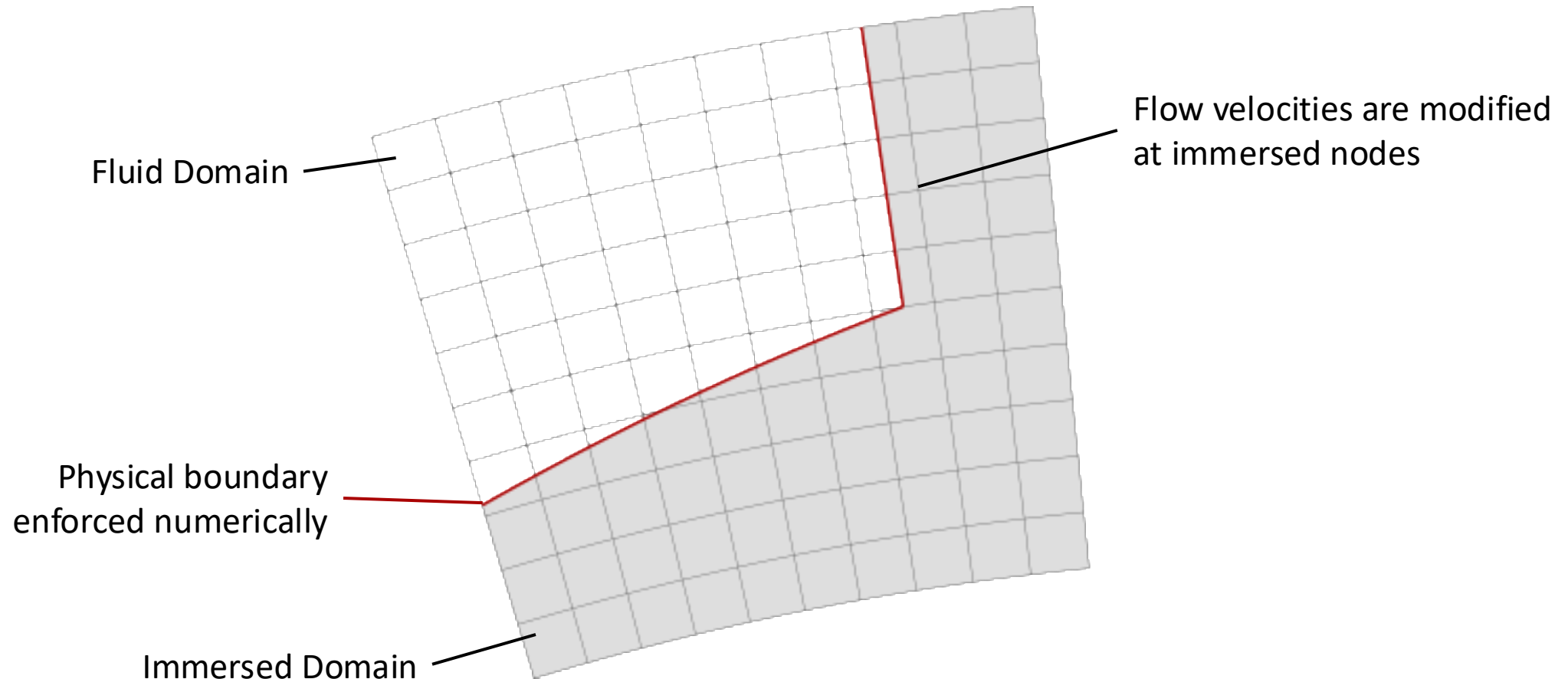
Body-fitted curvilinear overset grids can be problematic at component intersections

- Time consuming to generate collar grids at component intersections
- Collar grids often contain low quality cells
- Deformation of the geometry at component intersections can easily lead to poor grid quality or negative cell volumes

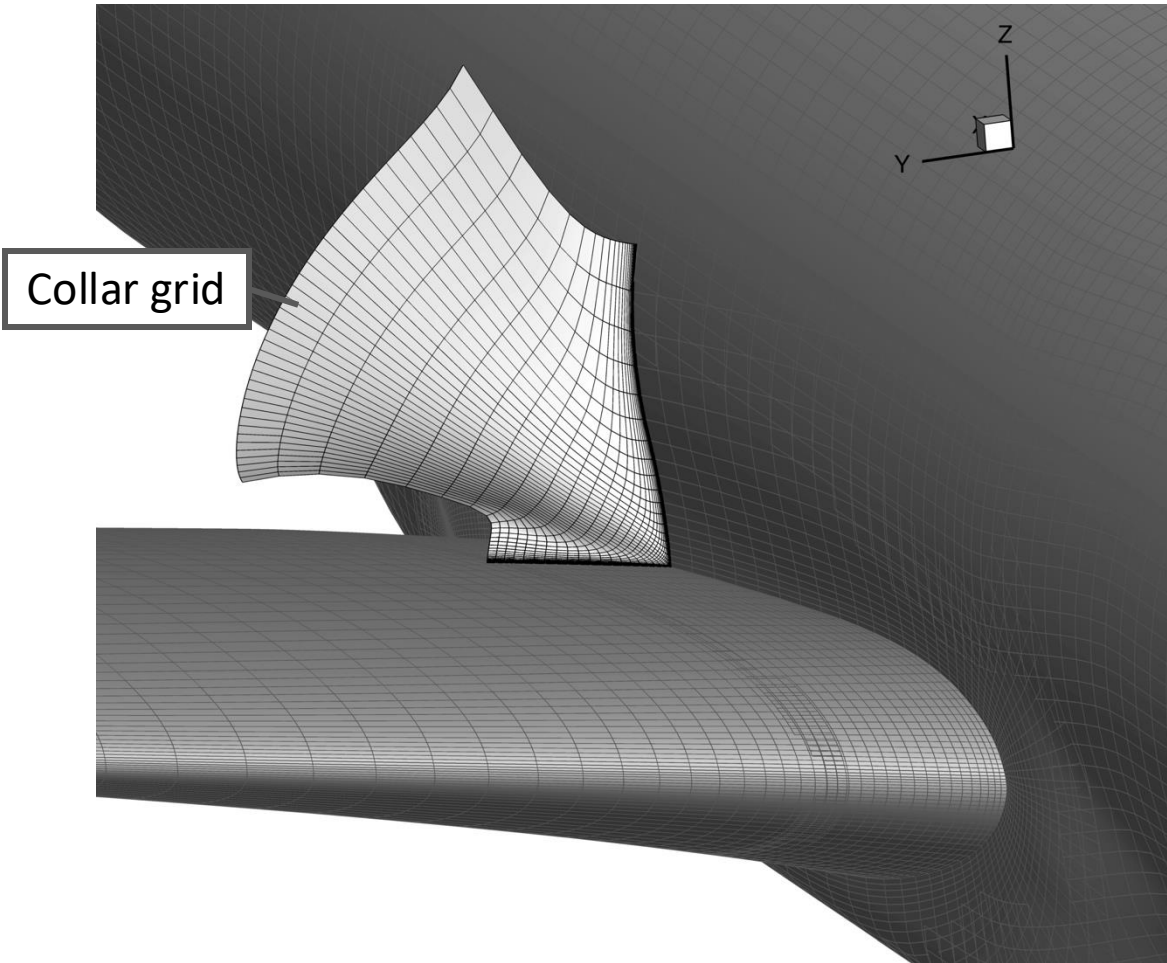


Can the application of an immersed boundary method remove the need for collar grids at component intersections?

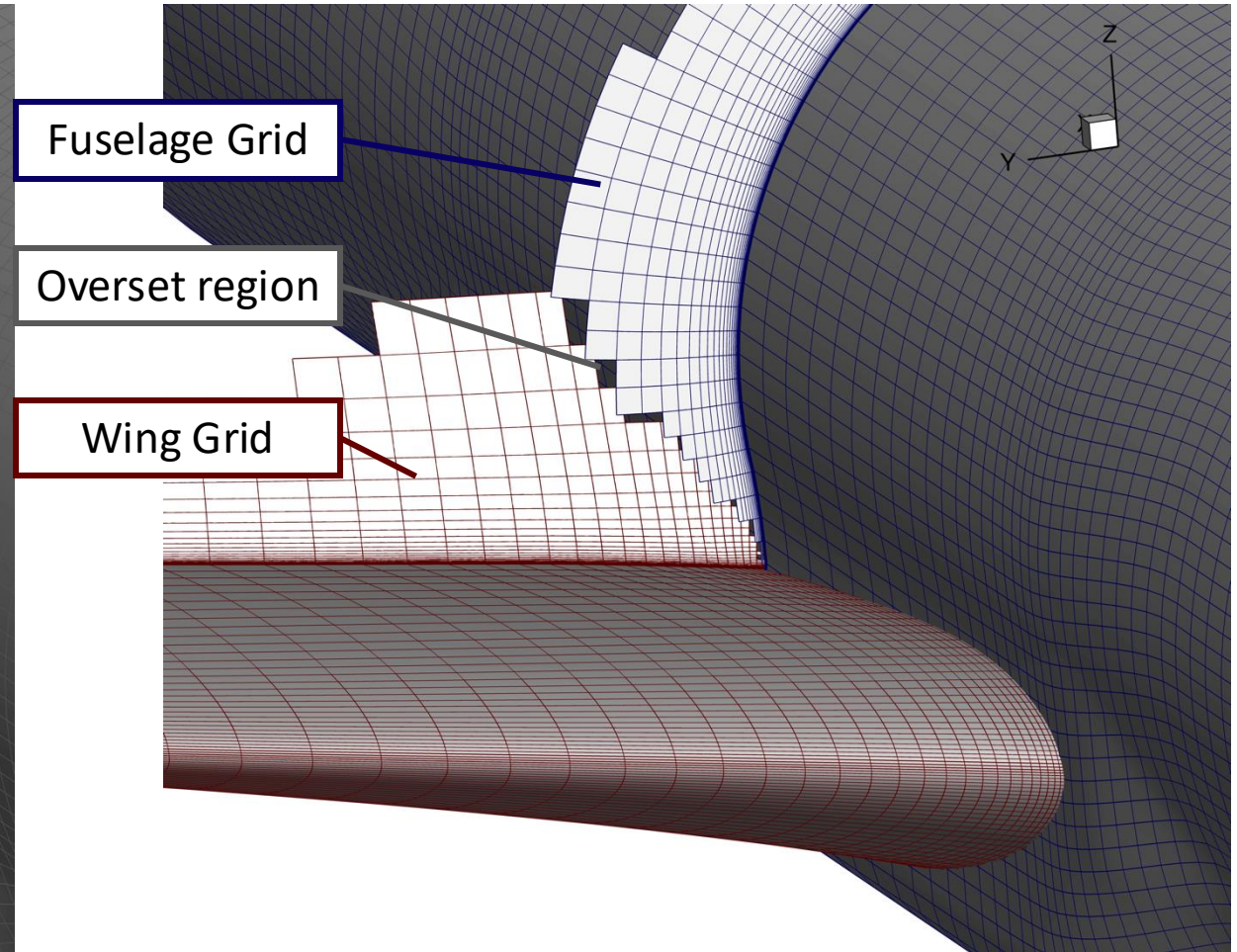
Immersed boundary methods aim to solve systems of equations on grids that do not conform with physical boundaries



With the application of an immersed boundary method, collar grids would be replaced with an overset region



Body-Fitted method



Immersed Boundary method

The **source-term** immersed boundary method imposes a penalty at nodes inside the immersed domain

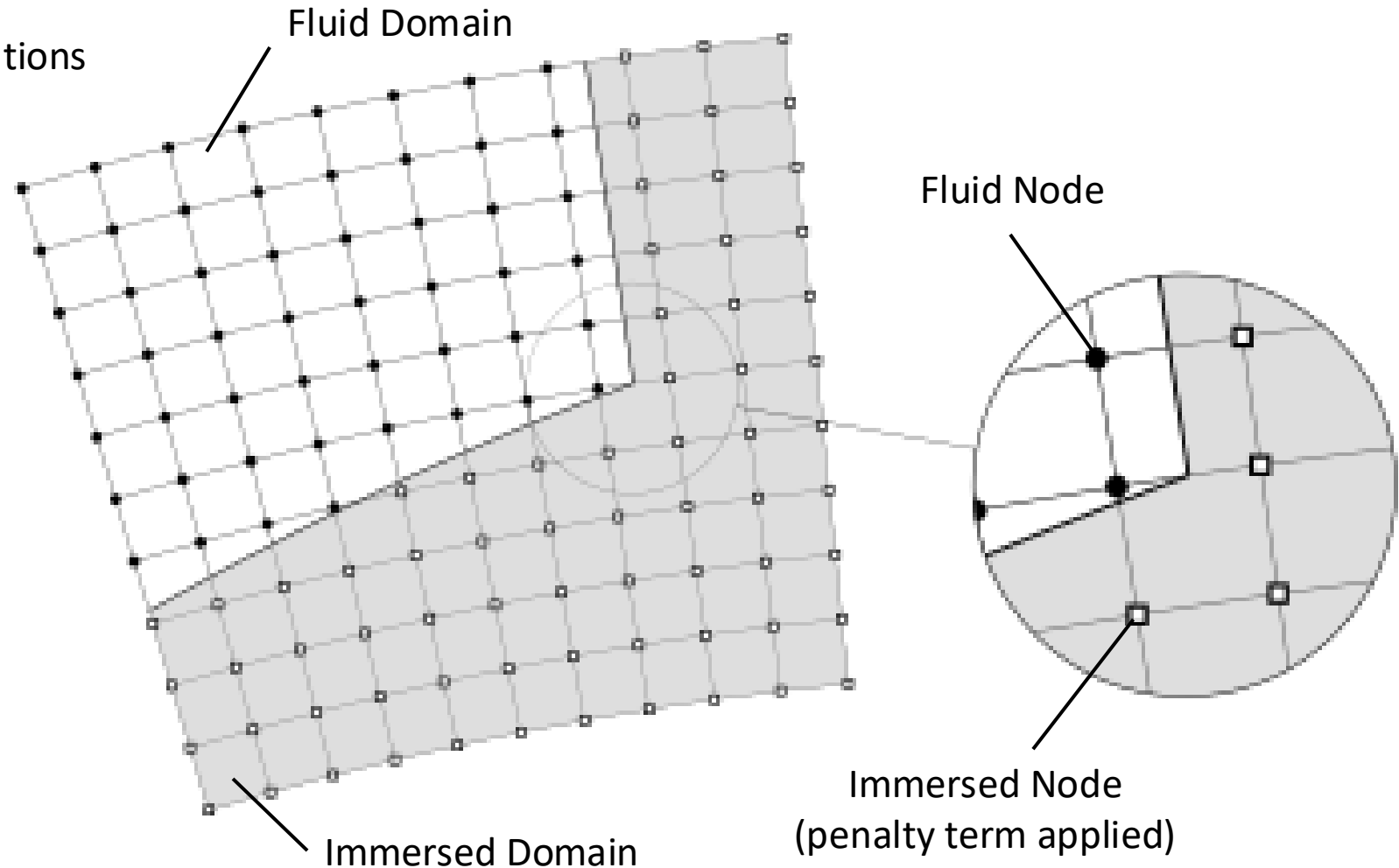
- A penalty term is added to the RANS equations

$$\mathbf{R}_{RANS} + \mathbf{F} = 0$$

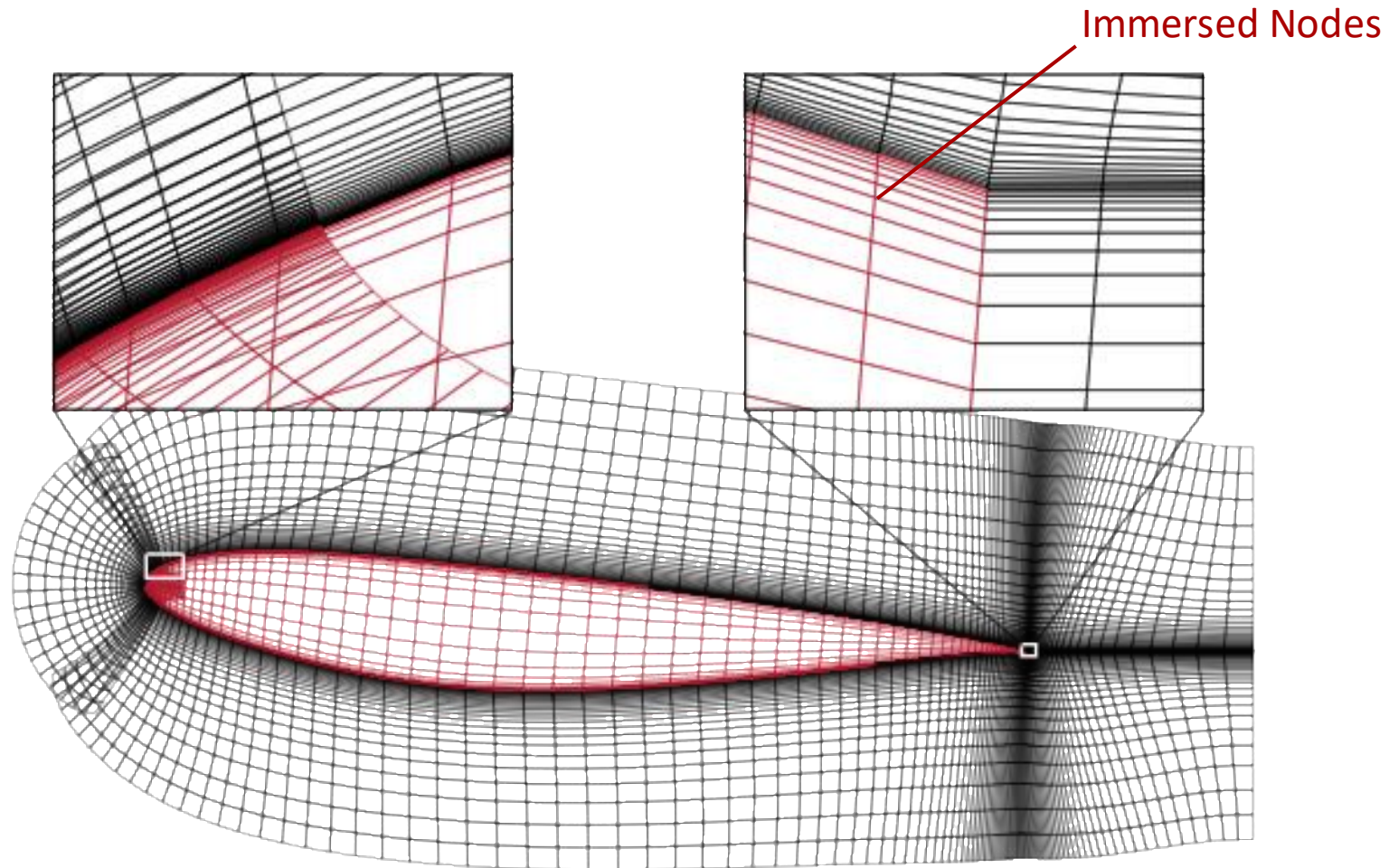
Flow Residual

Penalty Term

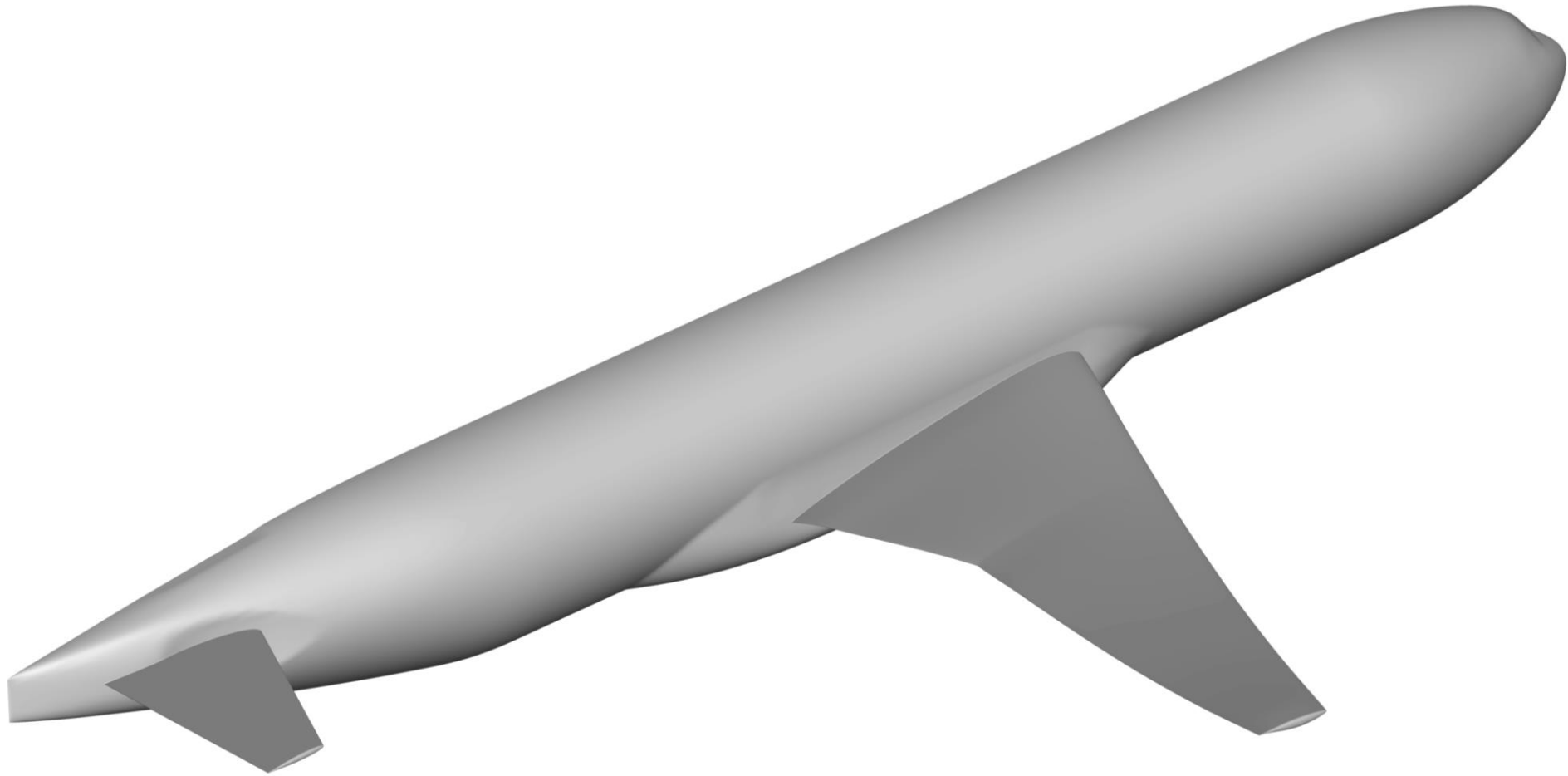
- The penalty term drives the flow velocities at the immersed nodes to zero



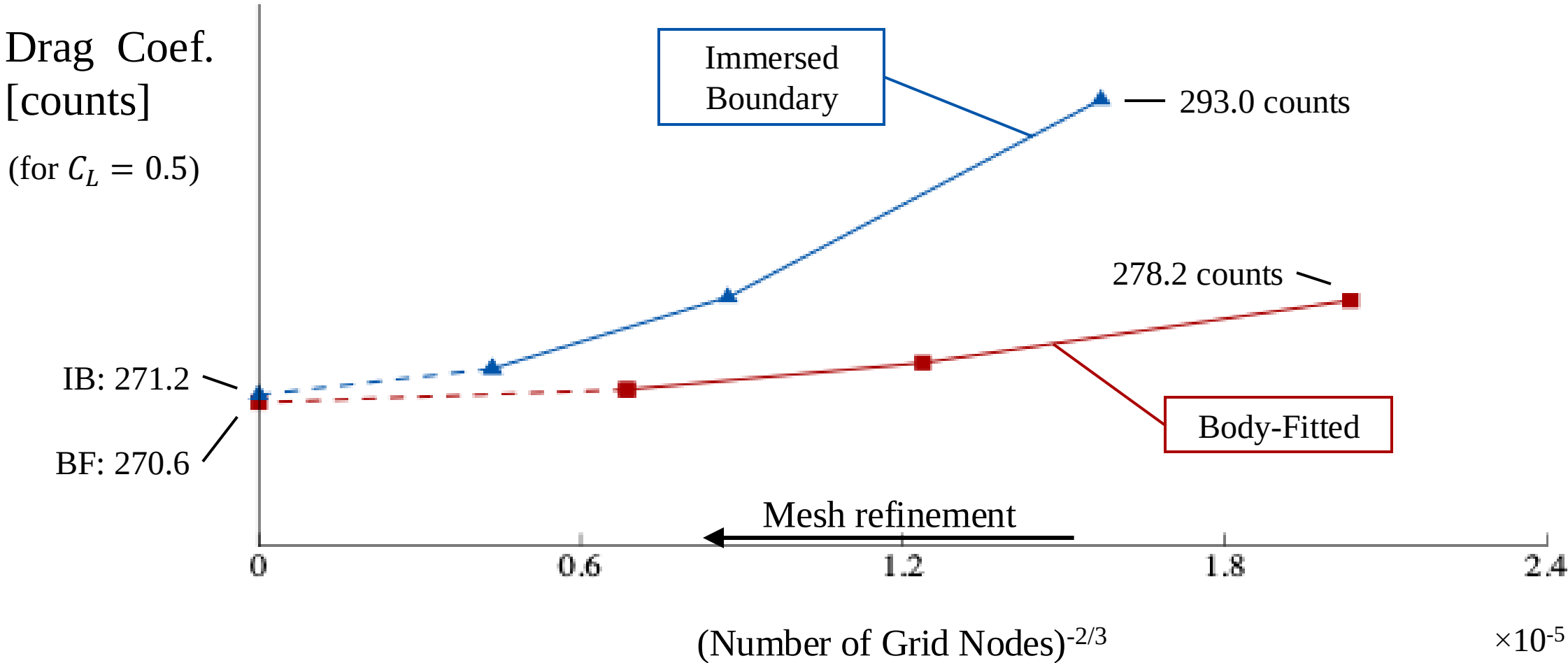
In the current approach, the entire domain must be discretized



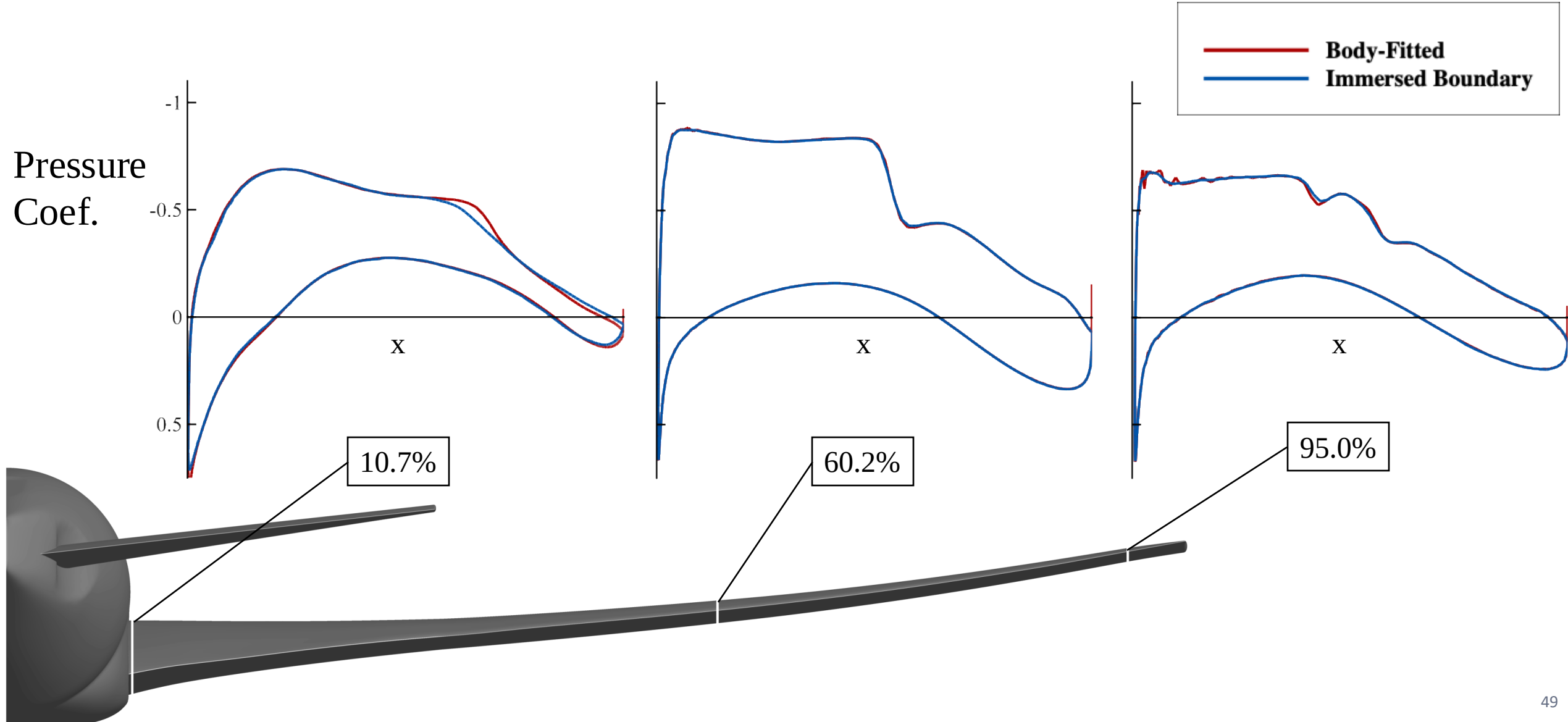
Following results are shown for the
CRM wing-body-tail geometry



The immersed boundary methods show higher values of drag compared to the body-fitted approach



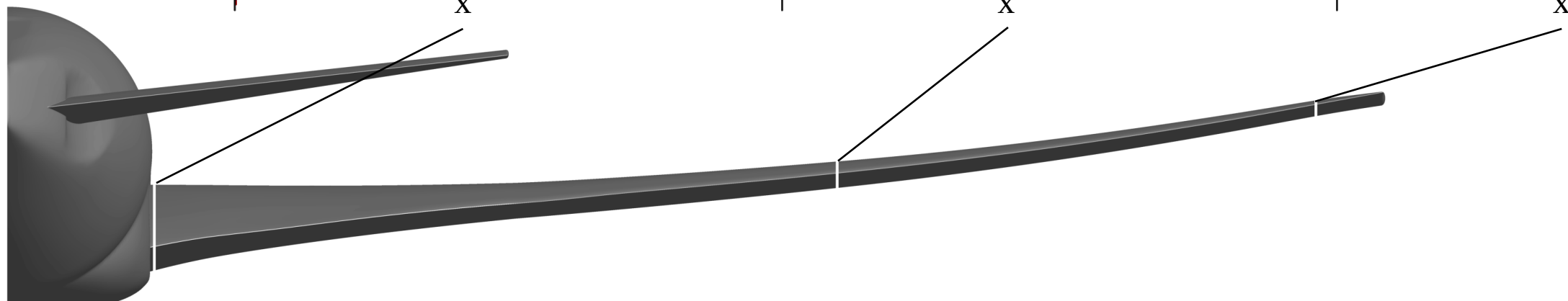
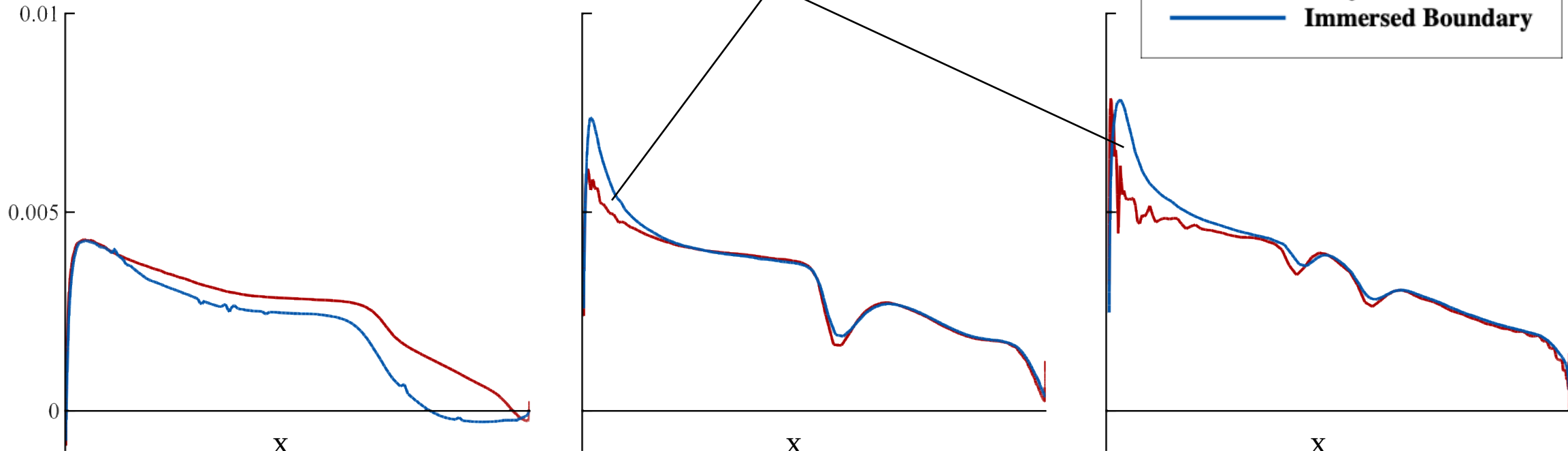
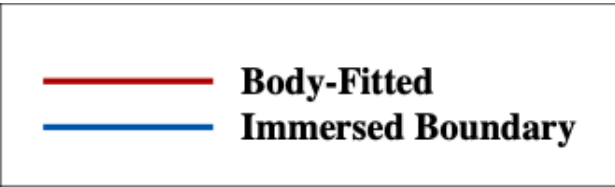
Pressure coefficient distributions generally agree well between all methods



Skin friction distributions show some disagreements between methods

Disagreement due to matching point for the wall model lying above the log layer

Skin Friction Coef. (Upper Surface)





LAVA's capabilities are being further extended for multi-disciplinary analysis and optimization

- The CFD Vision 2030 Study predicts that multidisciplinary simulations will become the norm due to the need to simulate complete aerospace systems
- LAVA's newly developed discrete adjoint and design optimization capabilities can be interfaced with other disciplines to achieve this goal
- An interface between LAVA and OpenMDAO/MPhys¹ is currently under development

¹MPHYS: github.com/OpenMDAO/mphys



Summary and Outlook



The LAVA Design Framework: Current Capabilities and Future Directions

Current Capabilities

- Fully implemented discrete adjoint solver and gradient calculations
- Capable of volume mesh deformation
- Interfaced with pyGeo and pyOptSparse for geometry parameterization and optimization

Future Directions

- Development of a hybrid body-fitted/immersed boundary method
- Integration with OpenMDAO/MPhys for multi-disciplinary analysis and optimization
- GPU acceleration for flow and adjoint solvers to increase computational efficiency



Acknowledgements

- This work was sponsored by the Transformational Tools & Technologies (TTT) project, the Convergent Aeronautics Solutions (CAS) project, and the Commercial Supersonic Technology (CST) project under NASA's Aeronautics Research Mission Directorate (ARMD)
- Computational resources provided by the NASA Advanced Supercomputing (NAS) facility at NASA Ames Research Center
- The authors would like to acknowledge the University of Michigan's MDO Lab for their open-source software
- The authors would also like to acknowledge all members of the LAVA group for their help and support



Thank You!

Brandon.Lowe@nasa.gov



Extra Slides

Example : use reverse accumulation to obtain the value of $\bar{y} \cdot \frac{\partial y}{\partial x}$

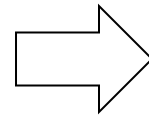
! Forward computation

! Inputs: x1, x2

! Output: y

a = 2*x1 + x2*x2 ! a = $2x_1 + x_2 \cdot x_2$

y = a * a ! y = $a \cdot a$



! Reverse accumulation

! Inputs : x1, x2, yb (seed = $\bar{y}$)

! Outputs : x1b ($= \bar{y} \cdot \frac{\partial y}{\partial x_1}$)

! x2b ($= \bar{y} \cdot \frac{\partial y}{\partial x_2}$)

a = 2*x1 + x2*x2

ab = yb*(2*a) ! $\bar{a} = \bar{y} \cdot \frac{\partial y}{\partial a} = \bar{y} \cdot (2a)$

x1b = ab*(2) ! $\bar{x}_1 = \bar{a} \cdot \frac{\partial a}{\partial x_1} = \bar{a} \cdot (2)$

x2b = ab*(2*x2) ! $\bar{x}_2 = \bar{a} \cdot \frac{\partial a}{\partial x_2} = \bar{a} \cdot (2x_2)$

Optimization convergence history for CRM Wing-Only optimization

