Sandia National Laboratories

SAND2020-6806C

Pressio: A Computational Framework Enabling Projection-Based Model Reduction for Large-Scale Nonlinear Dynamical Systems

PRESENTED BY

Patrick Blonigan

Collaborators: Francesco Rizzi, Eric Parish, and Kevin Carlberg

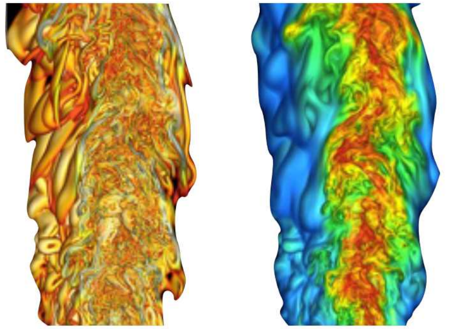
SAND2020-XXXXX C

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

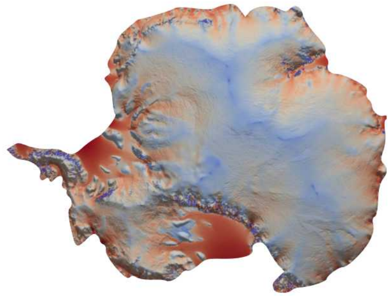
High-fidelity simulations are crucial, but often too costly for rigorous use in engineering and scientific applications

- **High-fidelity simulation:**
  - Extreme-scale nonlinear computational models,
  - Indispensable for engineering and scientific applications.
  - Example: captive carry aerodynamics simulation
    - Extreme-scale: 100 million cells, 200,000 time steps.
    - High cost: 6 weeks on 5000 cores.

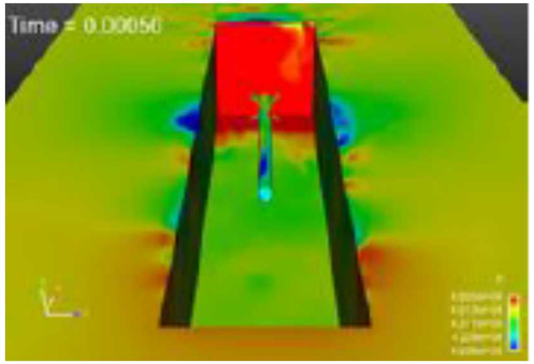
- **High-fidelity for time-critical and/or many-query**
  - Parameter estimation
    - Material property estimation
    - Matching field experiments
    - Parameters for digital twins
  - Uncertainty Quantification
    - Qualification of uncertainty in normal and abnormal environments
    - Quantification of Margins
  - Design optimization
    - Rapid iteration of conceptual designs
    - Shape and material optimization



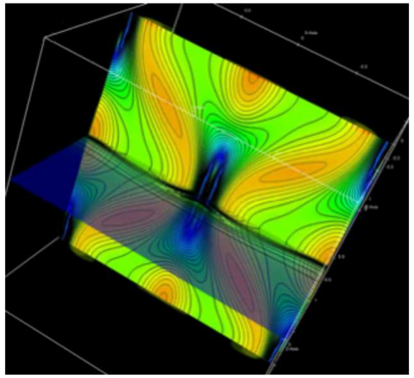
Turbulent reacting flows
courtesy J. Chen



Antarctic ice sheet modeling
courtesy R. Tuminaro



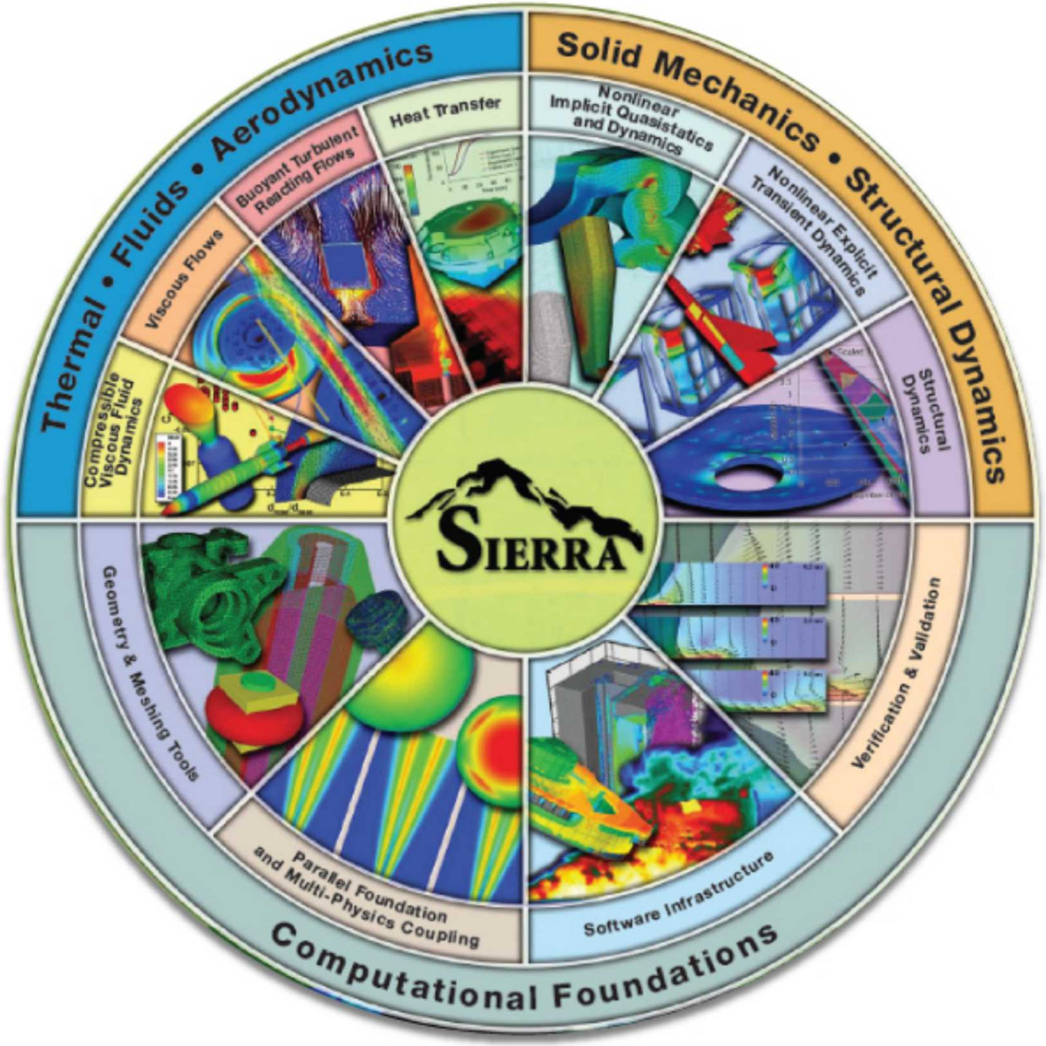
Captive carry aerodynamics
courtesy M. Barone



Magnetohydrodynamics
courtesy J. Shadid

Projection-based reduced-order models (ROMs)

- **Why ROMs?**
  - Directly tied to a ``full-order model”
    - Allows us **to leverage Sandia’s suite** of application codes
  - ROMs are “physics-based” surrogates
    - Results are **explainable**
  - Compatible with *a priori* and *a posteriori* error bounds
    - **Quantifying the uncertainty of the ROM is critical** for Sandia’s missions
  - Enables full-field predictions
    - Useful for engineering design and analysis



Mathematical setting

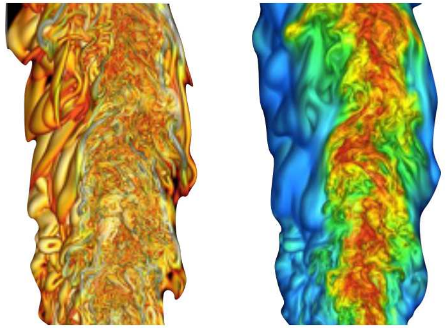
- We focus on dynamical systems emerging from **spatially** discretized PDEs

$$\dot{\boldsymbol{x}}(t;\boldsymbol{\mu}) = \boldsymbol{f}(\boldsymbol{x}(t;\boldsymbol{\mu}), t, \boldsymbol{\mu})$$
$$\boldsymbol{x} : [0, T] \times \mathcal{D} \to \mathbb{R}^N$$
$$\boldsymbol{\mu} \in \mathcal{D}$$

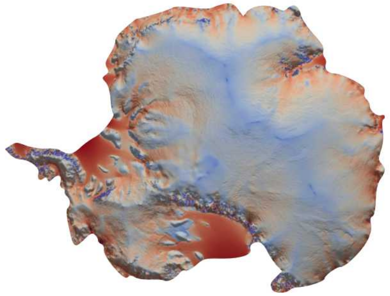
$\boldsymbol{x}$: state vector $\quad$ $\boldsymbol{\mu}$: system parameters

- Why semi-discrete?
  - **Formulation is versatile:** encompasses finite volume, finite difference, and finite element models
  - Encompasses the majority of Sandia's application codes
  - Steady systems are encompassed

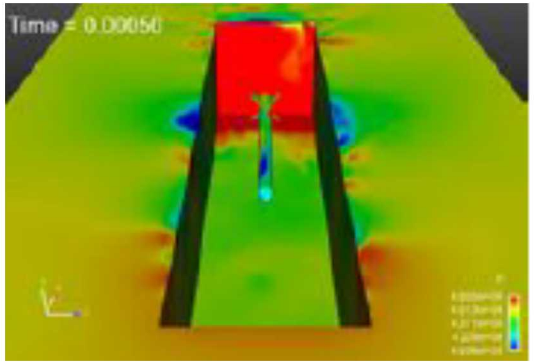
- Solving these systems are **computationally expensive**
  - **Motivates the need for ROMs**



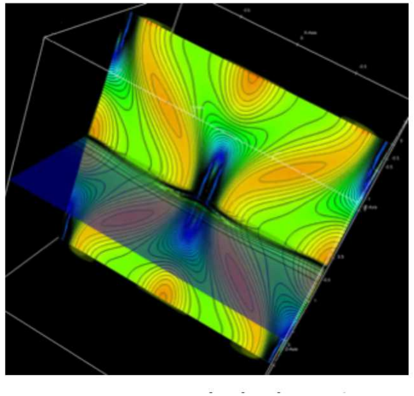
Turbulent reacting flows
courtesy J. Chen



Antarctic ice sheet modeling
courtesy R. Tuminaro



Captive carry aerodynamics
courtesy M. Barone

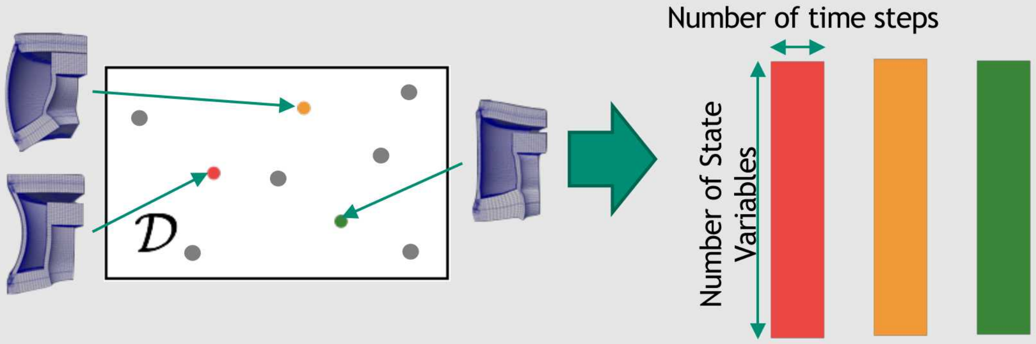


Magnetohydrodynamics
courtesy J. Shadid

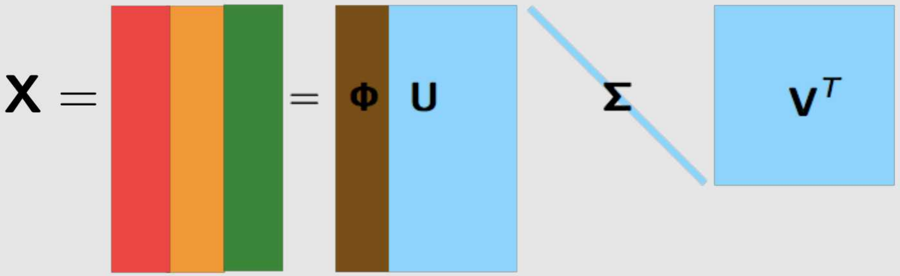
ROMs leverage an offline—online paradigm

Offline

- Execute solves of the FOM for "training" parameter instances



- Identify low-dimensional structure in data (POD)



- Approximate state in low-dimensional vector space

$$\boldsymbol{x}(t;\boldsymbol{\mu}) \approx \tilde{\boldsymbol{x}}(t;\boldsymbol{\mu}) = \boldsymbol{\Phi}\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})$$

Online

- Generate approximate solutions
- **We favor minimum residual formulations**
  - LSPG:
    - Minimize the time-discrete residual at each time step

$$\hat{\mathbf{x}}^{n+1} = \underset{\hat{\mathbf{v}}}{\arg\min} \left\| \mathbf{A}\mathbf{r}(\boldsymbol{\Phi}\hat{\mathbf{v}};\boldsymbol{\mu}) \right\|_2^2$$

  - Space—time LSPG:
    - Minimize time-discrete residual over the entire time domain
  - Windowed least-squares:
    - Minimize the time-continuous residual over windows

- **Why minimum residual?**
  - Robust for nonsymmetric systems
  - Straightforward to equip with constraints (e.g., conservation)

Outstanding challenges and ongoing ROM work at Sandia

- We are actively addressing the following ROM challenges at Sandia
  - Stability and accuracy for nonlinear, nonsymmetric, and noncoercive problems
  - Kolmogorov *n*-width limitations
  - Domain decomposition
  - Quantifying the "ROM model-form" uncertainty
  - **Portability**
  - Demonstrations on engineering applications
- **Portability of reduced-order models**
  - ROMs are traditionally viewed as an **intrusive** surrogate model
    - Requires modifications to the source code

  - Sandia maintains a heterogeneous set of high-performance application codes
    - Different data structures and data types
    - Different types of parallelization (OpenMP, MPI, GPUs)
    - Adding ROM capabilities to each code is not achievable

- Motivates ***Pressio***

What is Pressio?

- Open-source computational framework enabling projection-based model reduction for large-scale nonlinear dynamical systems.

- Applicable to a general ODE systems: Pressio provides ROM capabilities that are applicable to any system expressible as a parameterized system of ordinary differential equations (ODEs) as:

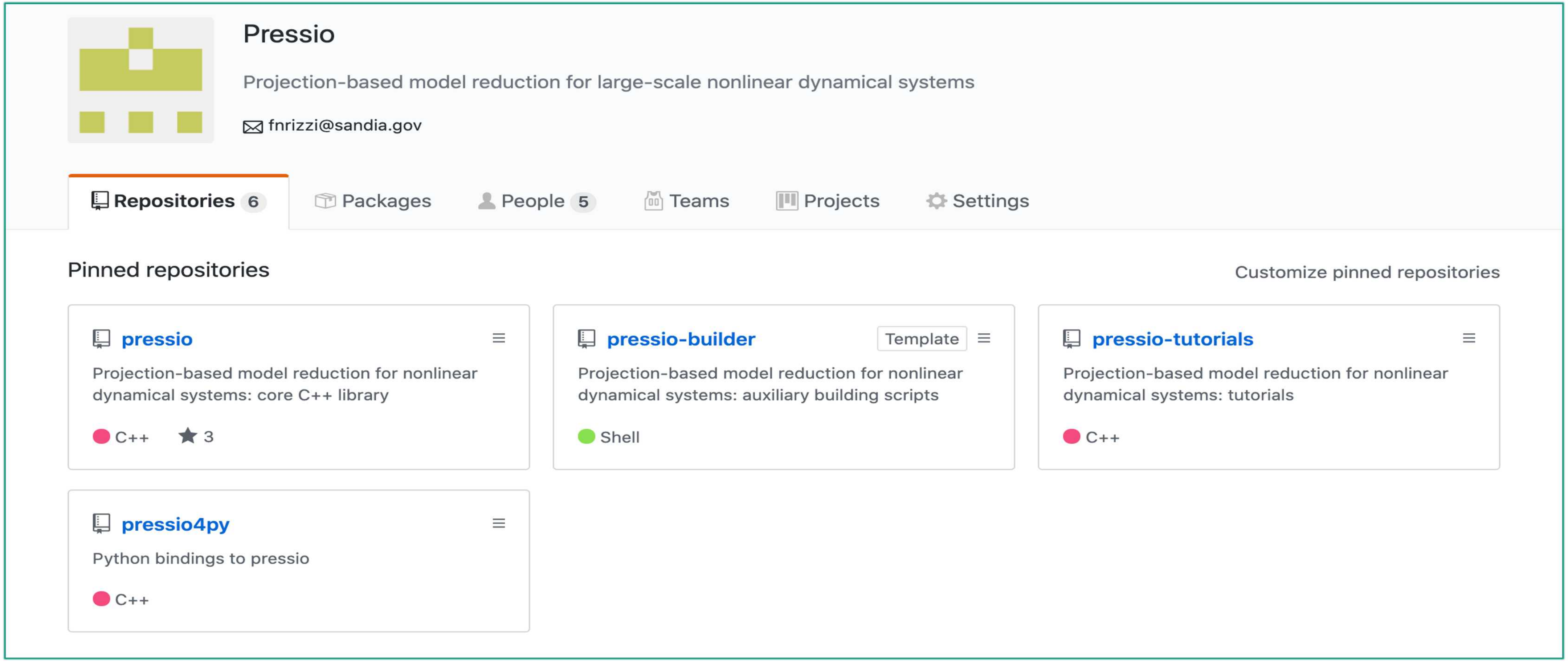
$$\dot{\boldsymbol{x}}(t;\boldsymbol{\mu}) = \boldsymbol{f}(\boldsymbol{x}(t;\boldsymbol{\mu}), t, \boldsymbol{\mu}), \qquad \boldsymbol{x}(0;\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu})$$

- Provides model reduction techniques for both spatial and temporal degrees of freedom.

https://github.com/Pressio

Github project

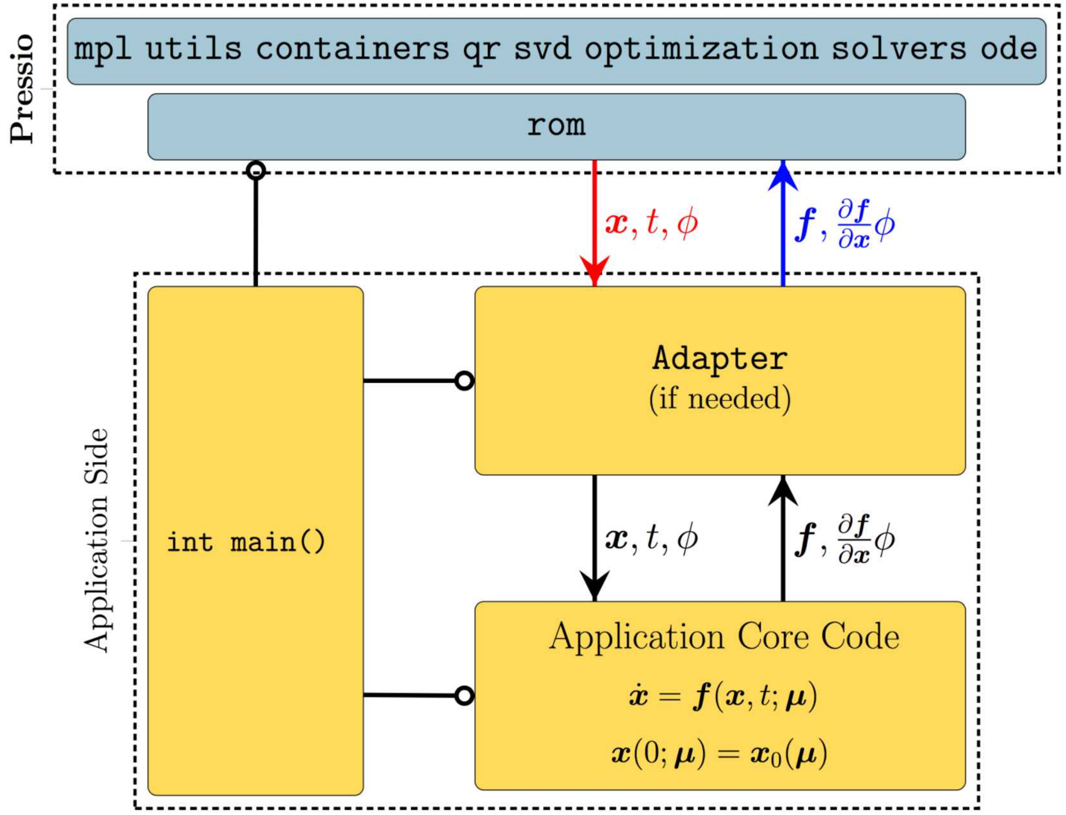
https://github.com/Pressio



Interfacing with simulation codes

- Previous ROM methods were implemented directly in multiple application codes
  - ✗ **Highly intrusive**: major changes to application code
  - ✗ **Not extensible**: individual ROM implementation for each application
  - ✗ **Access requirements**: developers need direct access to application

- Pressio: computational framework addressing all these issues:
  - ✓ Minimally intrusive API
  - ✓ Leverages modern software engineering practices (e.g. C++ template-metaprogramming)
    - ➢ Portable implementation that works on different architectures, including GPUs
    - ➢ Restricted to practices used by mission application partners
  - ✓ Facilitates contributions from external partners
    - ➢ Undergoing open source copyright assertion
  - ✓ Clear separation between methods and application
    - ➢ Enables methods work without access to restricted applications



Schematic of Pressio software workflow

High-level features

Header-only library, no need to be compiled and packaged

- Benefits portability

Modular structure

- Packages are designed to be self-contained with minimal inter dependencies
- Benefits the development cycle and extensibility

Relies on modern C++11 and metaprogramming for type detection and compile-time dispatching

Support for state-of-the-art HPC programming models (e.g. Kokkos)

- Seamless support for GPU computing via Kokkos

Unit and regression tests with continuous integration (growing feature)

Supports a basic Python API to expose the C++ ROM functionalities

- Enables Python users to use Pressio

Minimal API that is natural for ODE systems

We leverage the ODE expression

$$\dot{\boldsymbol{x}}(t;\boldsymbol{\mu}) = \boldsymbol{f}(\boldsymbol{x}(t;\boldsymbol{\mu}), t, \boldsymbol{\mu}), \quad \boldsymbol{x}(0;\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu})$$

as a pivotal design choice to enable a minimal API

```python
class Adapter:
  def __init__(self, *args):
    # initialize (if needed)
    # create velocity vector, f, and Jacobian matrix, Jac

  # compute velocity, f(x,t;...), for a given state, x
  def velocity(self, x, t):
    # compute f (here f is a member of the class)
    return self.f

  # given current state x(t):
  # 1. compute the spatial Jacobian, df/dx
  # 2. compute A=df/dx*B, B is typically a skinny dense matrix
  def applyJacobian(self, x, B, t):
    # compute Jac = df/dx (here Jac is a member of the class)
    # Jac is typically sparse, so we use Jac.dot(B)
    # When Jac is dense, use np.matmul(Jac,B)
    return self.Jac.dot(B)
```

Python adapter API

```cpp
class SampleAdapterClass{
  //...
public:
  /* C++11 type aliasing declarations that Pressio detects */
  /* this is equivalent to doing: typedef ... scalar_type  */
  using scalar_type       = /*application's scalar type       */;
  using state_type        = /*              state type        */;
  using velocity_type     = /*              velocity type     */;
  using dense_matrix_type = /*              dense matrix type */;
  //...

  // compute velocity, f(x,t;...), for a given state, x(t)
  void velocity(const state_type  & x,
                const scalar_type & t,
                velocity_type     & f) const;

  // given current state x(t):
  // 1. compute the spatial Jacobian, df/dx
  // 2. compute A=df/dx*B, B is typically a skinny dense matrix
  void applyJacobian(const state_type        & x,
                     const dense_matrix_type & B,
                     const scalar_type       & t,
                     dense_matrix_type       & A) const;

  // overload called once to construct an initial object
  velocity_type velocity(const state_type  & x,
                         const scalar_type & t) const;

  // overload called once to construct an initial object
  dense_matrix_type applyJacobian(const state_type        & x,
                                  const dense_matrix_type & B,
                                  const scalar_type & t) const;
```

C++ adapter API

Currently Supported ROM Features

- ROM formulations
  - Galerkin Projection
  - LSPG (steady and unsteady)
  - WLS
- Time Stepping Schemes
  - Forward/Backward Euler
  - BDF2
  - RK4
- Nonlinear solvers
  - Gauss-Newton
  - Levenberg-Marquardt
- Miniapps
  - 1D Burger's equation
  - 2D Advection-Diffusion
  - 2D Advection-Diffusion-Reaction

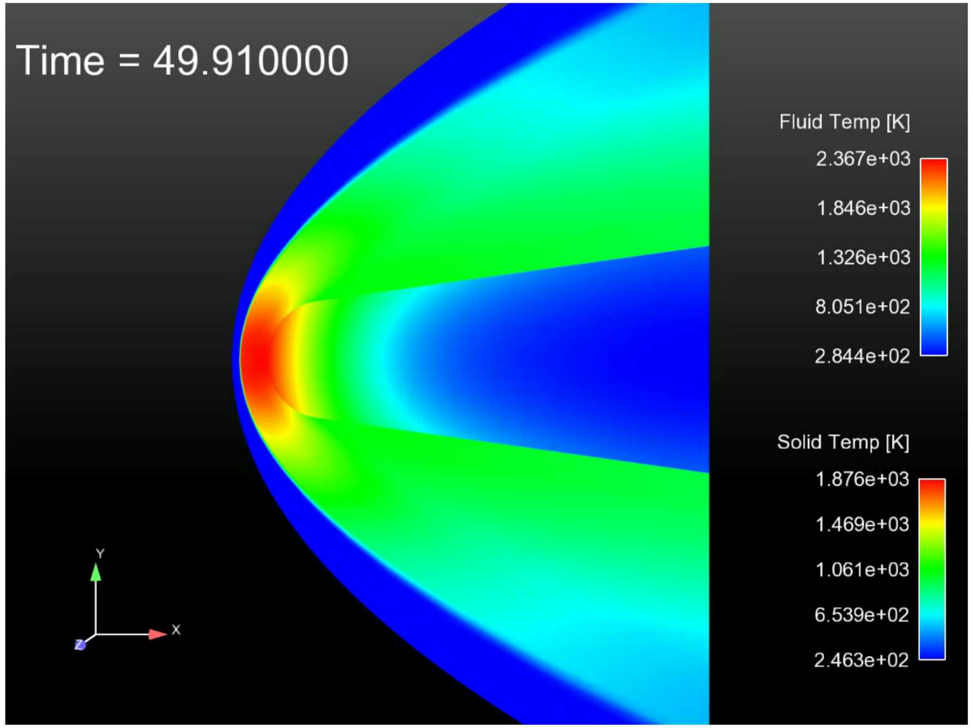
Applications Currently Interfaced with Pressio

- SPARC: Sandia Parallel Aerodynamics and Reentry Code
  - **Finite Volume compressible flow RANS solver**
  - Finite element thermal/ablation solver
- ARIA: Sandia proprietary multiphysics package
  - Incompressible flow solver
  - Thermal/chemical solver
- OpenFOAM: joint work with Samuel Majors and Karen Willcox (UT Austin)
  - Thermal conduction (In progress)
  - Compressible flow solver (In progress)

Sandia Parallel Aerodyanmics and Reentry Code (SPARC)

- Compressible CFD code focused on aerodynamics and aerothermodynamics in the Transonic and Hypersonic regimes
  - Being developed to run on today's leadership-class supercomputers and exascale machines.
  - Performance portability: SPARC leverages Kokkos to run on multiple machines with different architectures (e.g. CPU vs. CPU/GPU)

- Physics Capabilities include:
  - Navier—Stokes, cell-centered finite volume method
  - **Reynolds-Averaged Navier—Stokes (RANS) , cell-centered finite volume method**
  - Transient Heat Equation, Galerkin finite element method.
  - Decomposing and non-decomposing ablation equations, Galerkin finite element method.
  - One and two-way coupling between ablation, heat equation, RANS.



Temperature of a slender body in hypersonic flow simulated with SPARC

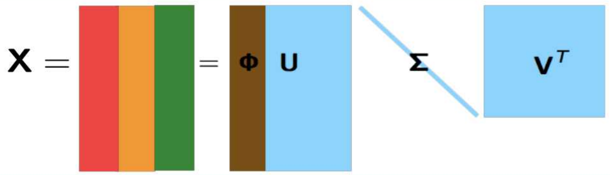
Implementing ROMs with Pressio and SPARC

1. Expose the functionalities required by the Pressio API
   - SPARC's modular design made this straightforward.
   - Routines needed for "velocity" were readily available.
   - "applyJacobian" leveraged existing spatial Jacobian.

velocity: $\boldsymbol{w} = \boldsymbol{f}(\boldsymbol{x}(t;\boldsymbol{\mu}), t, \boldsymbol{\mu})$

applyJacobian: $\boldsymbol{W} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}(t;\boldsymbol{\mu})} \boldsymbol{V}$

2. Implement main needed to drive the ROM computations.
   - e.g. reading snapshots, computing POD modes.



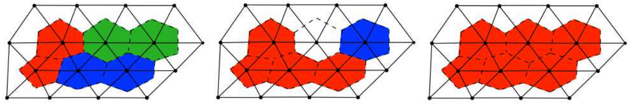
3. Hyper-reduction
   - Implemented sample mesh for an unstructured mesh format. Less intrusive than structured mesh format!
   - Algebraic hyper-reduction implemented for validation purposes.

4. Special ROM Features:
   - LSPG with conservation constraints was implemented with a custom nonlinear solver.
   - Clipper to eliminate non-physical flow phenomena in ROM state-reconstruction was implemented in SPARC.

$$\underset{\hat{\mathbf{v}}}{\text{minimize}} \; \|\mathbf{A}\mathbf{r}(\boldsymbol{\Phi}\hat{\mathbf{v}};\boldsymbol{\mu})\|_2^2$$

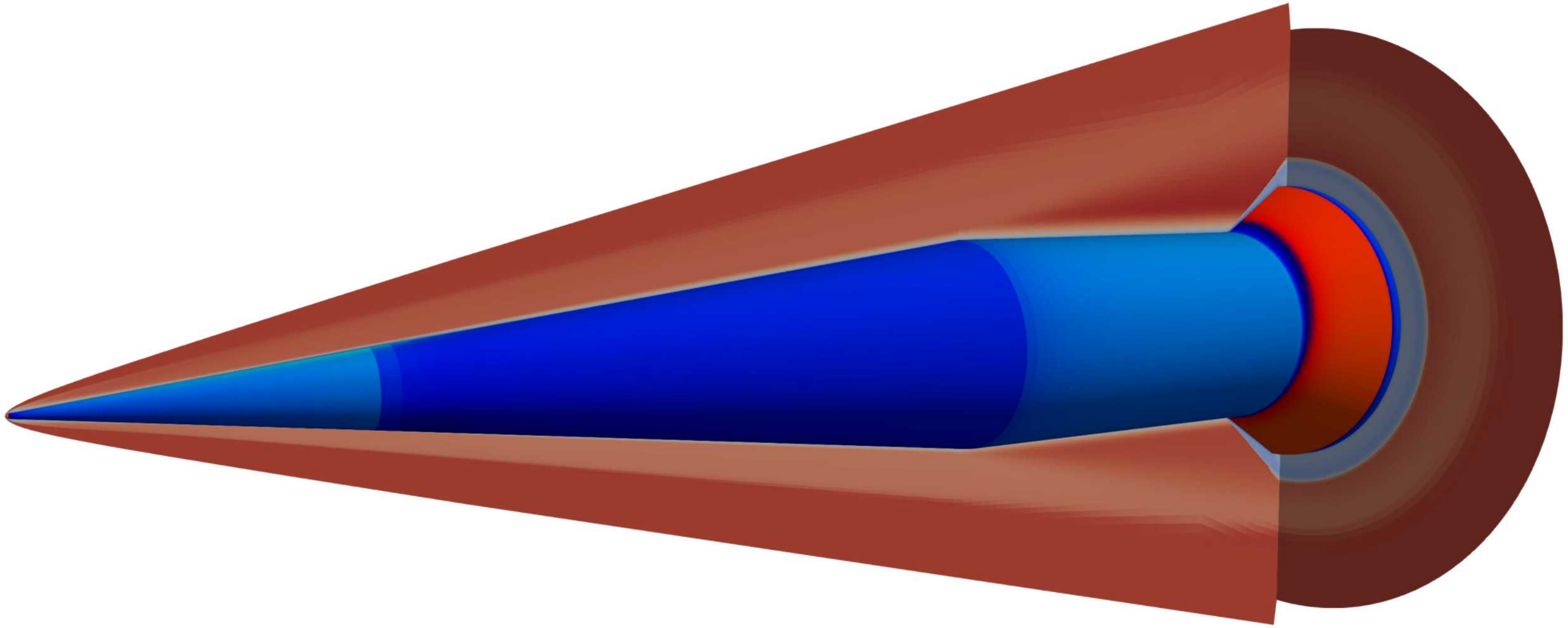
$$\text{s.t. } \mathbf{C}\mathbf{r}(\boldsymbol{\Phi}\hat{\mathbf{v}};\boldsymbol{\mu}) = \mathbf{0}$$

Enforce conservation over subdomains:

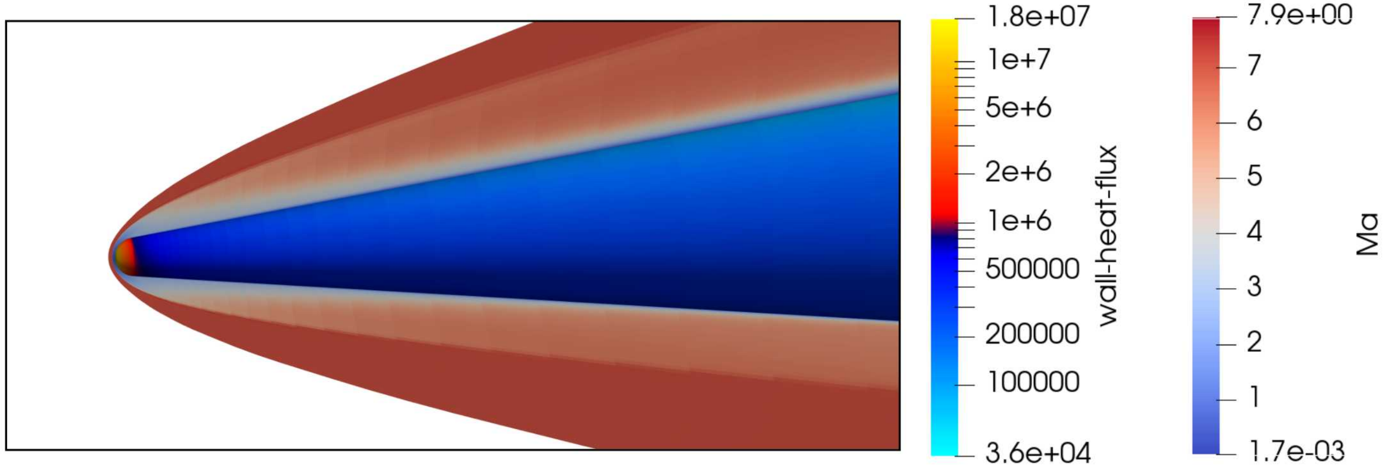


Test Case: HIFiRE-1 flight vehicle

- Flow field:
  - Free stream Mach No. = 7.1
  - Reynolds No. = 10.0 million/meter
  - Angle of Attack = 2 degrees
  - Boundary layer transitions to turbulence (use Spalart-Allmaras with specified transition location)
- Spatial discretization:
  - 2nd-order finite volume
  - 2,031,616 cells
  - $y^+ < 1$ near wall
- Solver:
  - Pseudo time stepping with backward Euler, CFL schedule.



Close up of nose:

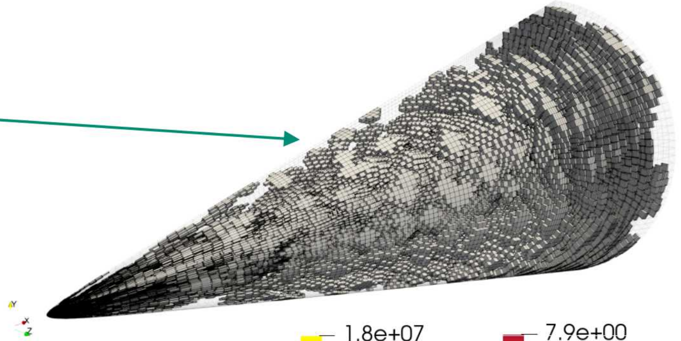


LSPG ROM applied to steady hypersonics using Pressio & SPARC

- HiFIRE-1 experiment. Baseline case: $Re=10^7$, Ma=7.1, AoA=2° .
- Training data set: 24 simulation results sampled over a range of freestream conditions:
  - Density: 0.056 to 0.070 kg/m^3
  - Free stream Mach number: 5.7 to 7.1
- Initial guess computed by inverse distance interpolation.

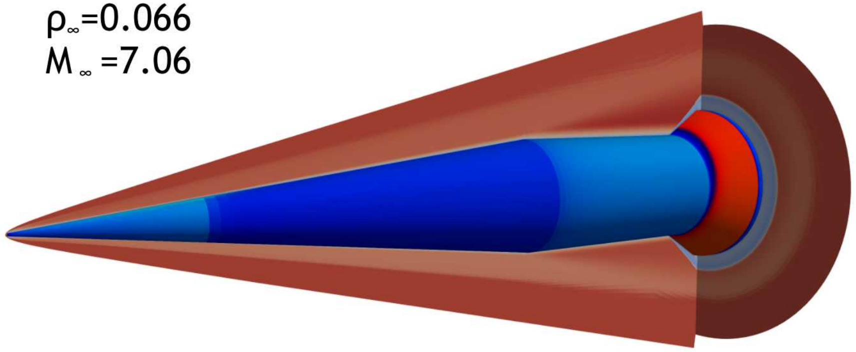
High-fidelity:
- Mesh: 2,031,616 cells
- Dofs = 12,189,696
- 128 MPI ranks, ~2,500-5,000 seconds

LSPG ROM:
- Sample mesh: 20,316 cells
- Dofs = 121,896
- 16 MPI ranks, ~30-55 seconds



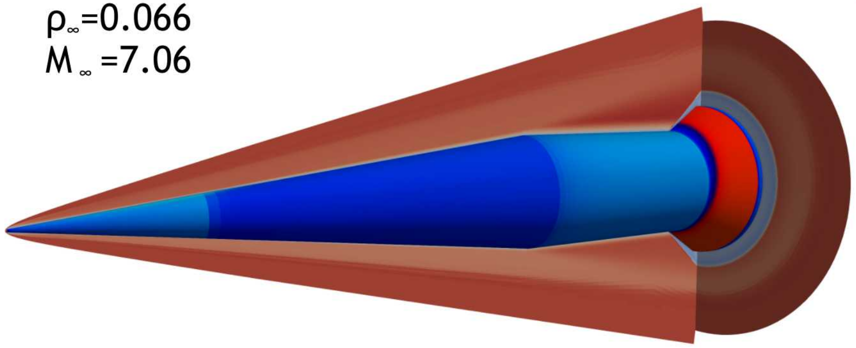
High-fidelity

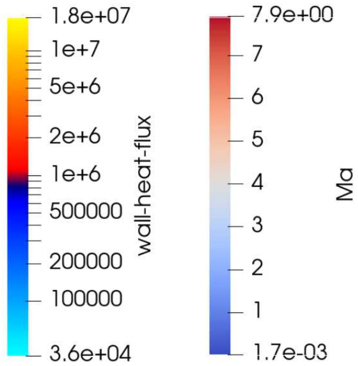
ρ_∞=0.066
M_∞ =7.06



LSPG ROM

ρ_∞=0.066
M_∞ =7.06





~300-1,000x savings in core–hours
< 1% error in density, momentum, and energy fields
~ 1-2% error in integrated wall heat flux

For more results, see AIAA 2020-0104 [Blonigan, Carlberg, Rizzi, Howard, Fike, 2020]

Conclusions

- High-fidelity simulations are computationally too expensive for time-critical or many-query analyses.
- Projection-based ROMs provide a rigorous surrogate for high-fidelity models
- ROMs are traditionally highly intrusive to implement
- **Pressio** leverages generic programming to enable the implementation of ROMs with a minimally intrusive APIs
- **Pressio** has been already interfaced with production-level application codes, including SPARC
  - In this case, computational speed-ups of 300-1,000x were obtained, with QoI errors of only 1-2%

Outlook

Ongoing:

- Increasing ROM robustness
  - stronger nonlinear solvers
  - preconditioning strategies
  - formulations better suited for shocks.
- OpenFOAM interface (with Samuel Majors, Karen Willcox)
  - Almost ready!

Future:

- ROMs for larger cases
  - larger meshes
  - multiple time scales
  - Coupled multi-physics
- Integration of machine learning
  - Error estimation
  - State dimension reduction
  - ROM deployment/management
- New programming models
  - Task-based programming
- **Collaborations enabled by Pressio**

Key references

- **F. Rizzi, P. Blonigan, and K. Carlberg. PRESSIO: Enabling Projection-based model reduction for lare-scale nonlinear dynamical systems. Submitted to SIAM Journal on Scientific Computing, Feb. 2020.**
- P. Blonigan, K. Carlberg, F. Rizzi, M. Howard, and J. Fike. Model reduction for hypersonic aerodynamics via conservative LSPG projection and hyper-reduction. AIAA Scitech 2020, AIAA 2020-0104.
- **E. Parish and K. Carlberg. Windowed least-squares model reduction for dynamical systems. arXiv e-print, (1910.11388), 2019.**
- **K. Lee and K. Carlberg. Deep Conservation: A latent dynamics model for exact satisfaction of physical conservation laws. arXiv e-print, (1909.09754), 2019.**
- E. Parish and K. Carlberg. Time-series machine-learning error models for approximate solutions to parameterized dynamical systems. Submitted to Computer Methods for Applied Mechanics in Engineering, 7/2019. arXiv e-print: 1907.11822
- P. Etter and K. Carlberg. Online adaptive basis refinement and compression for reduced order models. Submitted to Computer Methods for Applied Mechanics in Engineering, 2019. arXiv e-print: 1902.10659
- S. Pagani, A. Manzoni, and K. Carlberg. Statistical closure modeling for reduced-order models of stationary systems by the ROMES method. Submitted to SIAM Journal on Uncertainty Quantification, 2019. arXiv e-print: 1901.02792
- K. Lee and K. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. Journal of Computational Physics, 404:108973 (2020).
- M. Zahr, K. Carlberg, and D. Kouri. An efficient, globally convergent method for optimization under uncertainty using adaptive model reduction and sparse grids. SIAM/ASA Journal on Uncertainty Quantification, Vol. 7, No. 3, p.877–912 (2019).

https://github.com/Pressio

Backup Slides

Increasing ROM robustness with nonlinear mapping of POD basis

- Failed cases shown earlier are due to small regions of negative temperature.
- States with non-physical features are encountered by ROM solver more often as basis is made smaller and/or parameter space is increased in size.
- **Solution**: nonlinear mapping of POD modes to remove non-physical features from approx. state vector:

$$\underset{\hat{\mathbf{v}}}{\text{minimize}} \; \|\mathbf{A}\mathbf{r}(\boldsymbol{\Phi}\hat{\mathbf{v}};\boldsymbol{\mu})\|_2^2 \;\longrightarrow\; \underset{\hat{\mathbf{v}}}{\text{minimize}} \; \|\mathbf{A}\mathbf{r}(\mathbf{g}(\boldsymbol{\Phi}\hat{\mathbf{v}});\boldsymbol{\mu})\|_2^2$$

Where **g** transforms the conserved quantities in each cell as follows:

$$\tilde{\tilde{u}}_1 = \max(\epsilon_1, \tilde{u}_1)$$
$$\tilde{\tilde{u}}_2 = \tilde{u}_2$$
$$\tilde{\tilde{u}}_3 = \tilde{u}_3$$
$$\tilde{\tilde{u}}_4 = \tilde{u}_4$$
$$\tilde{\tilde{u}}_5 = \max\left(\epsilon_5 + \frac{1}{2\tilde{\tilde{u}}_1}\left[\tilde{u}_2^2 + \tilde{u}_3^2 + \tilde{u}_4^2\right]\;,\; \tilde{u}_5\right)$$

We do hyper-reduction with collocation to keep offline costs down

- Collocation has been used in past studies of CFD model reduction [Washabaugh, 2016]:

LSPG: $\underset{\hat{\mathbf{v}}}{\text{minimize}} \; \|\mathbf{A}\mathbf{r}(\boldsymbol{\Phi}\hat{\mathbf{v}};\boldsymbol{\mu})\|_2^2$

$\mathbf{A} =$ 

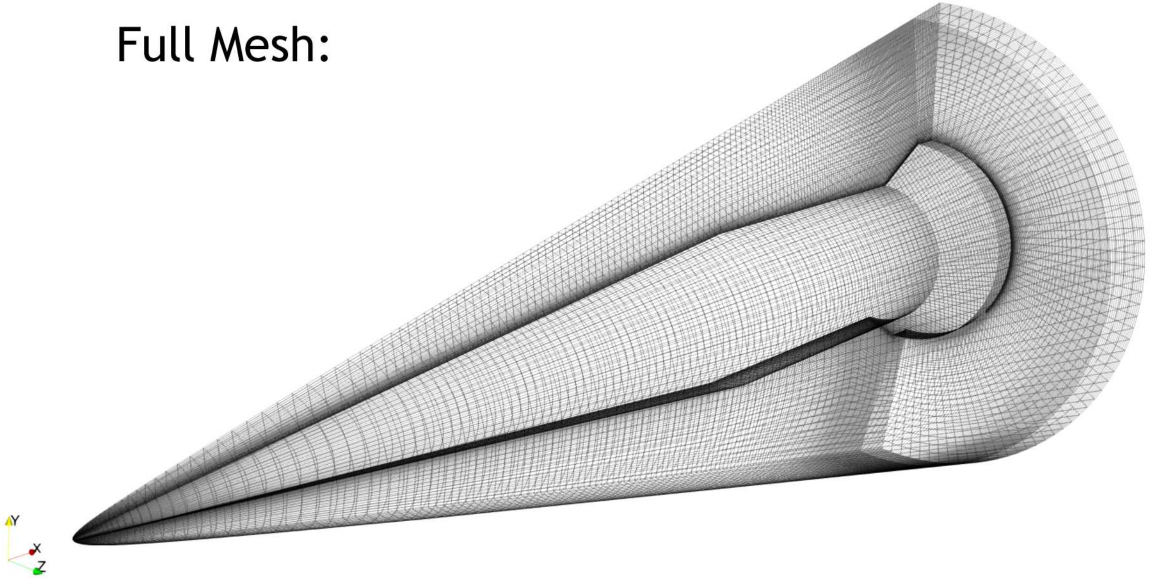
Collocation = choose rows of **A** from identity matrix

$$\begin{matrix} (1 & 0 & 0 & 0 & \dots & 0) \\ (0 & 0 & 1 & 0 & \dots & 0) \\ (0 & \dots & 0 & 1 & 0 & 0) \end{matrix}$$

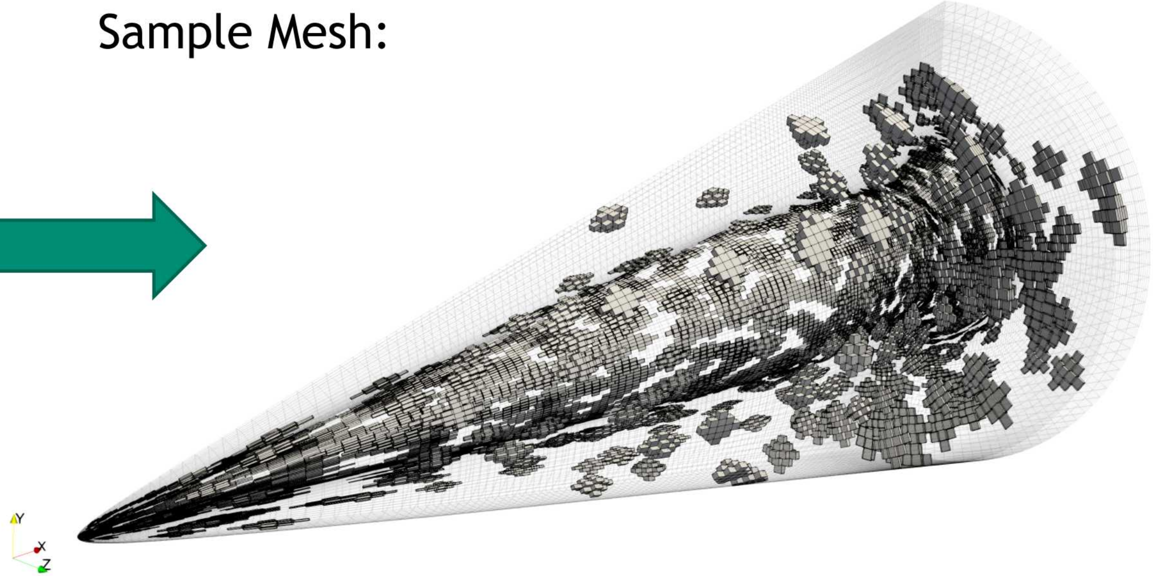
  - ➢Inexpensive compared to DEIM and GNAT.

- Sample mesh: subset of cells required to compute residual
- We consider random sampling of cells in this study.

Full Mesh:



Sample Mesh:



Training Data and Model details

- Samples:
  - Varied freestream density and velocity
  - Training set: 24 sample Latin hypercube
  - Test set: 12 sample Latin hypercube
- POD basis:
  - Mean flow subtracted from each snapshot.
  - Each conserved quantity scaled by its maximum over all FOM solutions.
  - 2,4, and 8 mode basis were considered.
- ROM: LSPG solved with Gauss-Newton iteration
  - Initial guess obtained via inverse-distance interpolation of POD modes.
  - Full mesh, two sample meshes considered