Performance-Portable Implicit Scale-Resolving Compressible Flow Using libCEED

SIAM CSE 2023

James Wright, Jed Brown, Kenneth Jansen, Leila Ghaffari

February 27, 2023

Ann and H.J. Smead Department of Aerospace Engineering Sciences

Smead Aerospace
UNIVERSITY OF COLORADO **BOULDER**

Outline

1. libCEED Overview
2. Compressible Fluid Equations in libCEED
3. Efficient Implicit Timestepping
4. Performance and Results of Flat Plate Boundary Layer Simulation

libCEED Overview

What is libCEED?

- C library for element-based discretizations
  - Bindings available for Fortran, Rust, Python, and Julia

What is libCEED?

- C library for element-based discretizations
  - Bindings available for Fortran, Rust, Python, and Julia
- Designed for matrix-free operator evaluation

What is libCEED?

- C library for element-based discretizations
  - Bindings available for Fortran, Rust, Python, and Julia
- Designed for matrix-free operator evaluation
- Portable to different hardware via computational backends
  - Code that runs on CPU also runs on GPU without changes
  - Computational backend selectable at runtime, using runtime compilation

What is libCEED?

- C library for element-based discretizations
  - Bindings available for Fortran, Rust, Python, and Julia
- Designed for matrix-free operator evaluation
- Portable to different hardware via computational backends
  - Code that runs on CPU also runs on GPU without changes
  - Computational backend selectable at runtime, using runtime compilation
- Geared toward high-order finite element discretizations

What is libCEED?

- C library for element-based discretizations
  - Bindings available for Fortran, Rust, Python, and Julia
- Designed for matrix-free operator evaluation
- Portable to different hardware via computational backends
  - Code that runs on CPU also runs on GPU without changes
  - Computational backend selectable at runtime, using runtime compilation
- Geared toward high-order finite element discretizations
- Performance demonstrated for solids in Brown *et al.* 2022[1]
  - Want to apply those methods and lessons-learned to fluids

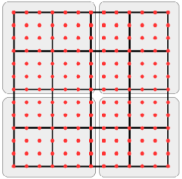
[1] *Performance Portable Solid Mechanics via Matrix-Free p-Multigrid*, Brown *et al.*, arXiv:2204.01722

Finite Element Operator Decomposition

Finite Element Operator Decomposition

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$

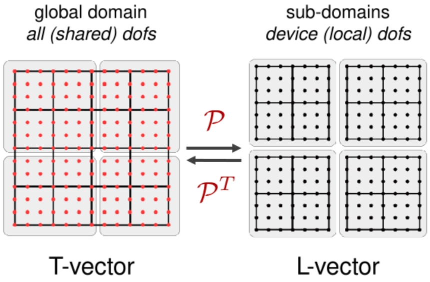
global domain
all (shared) dofs



T-vector

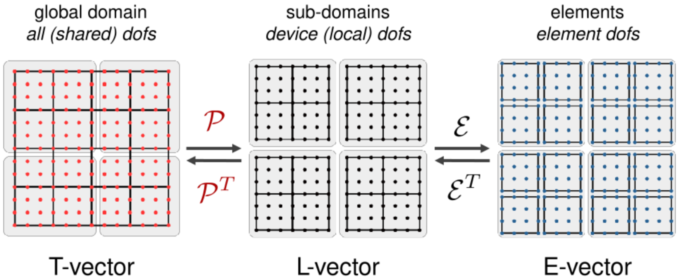
Finite Element Operator Decomposition

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



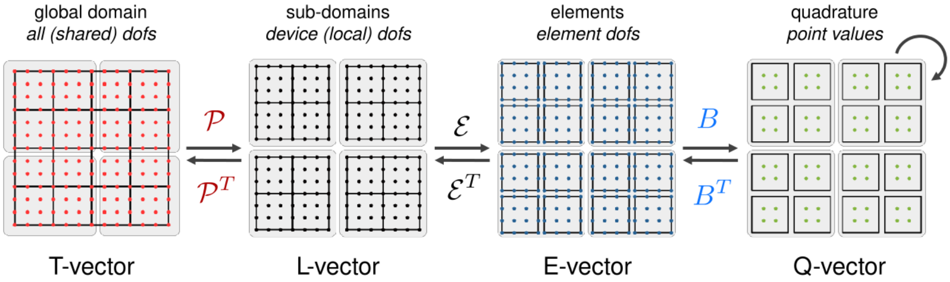
Finite Element Operator Decomposition

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



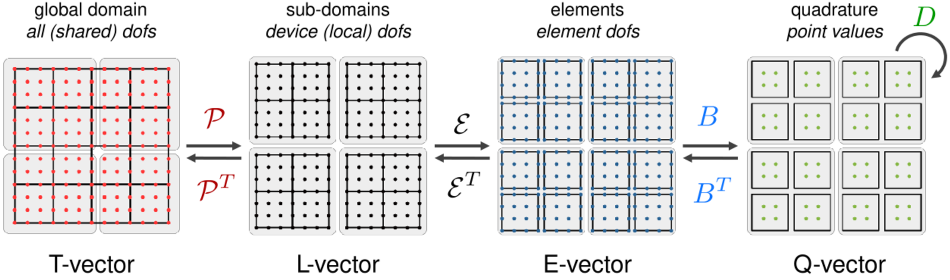
Finite Element Operator Decomposition

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



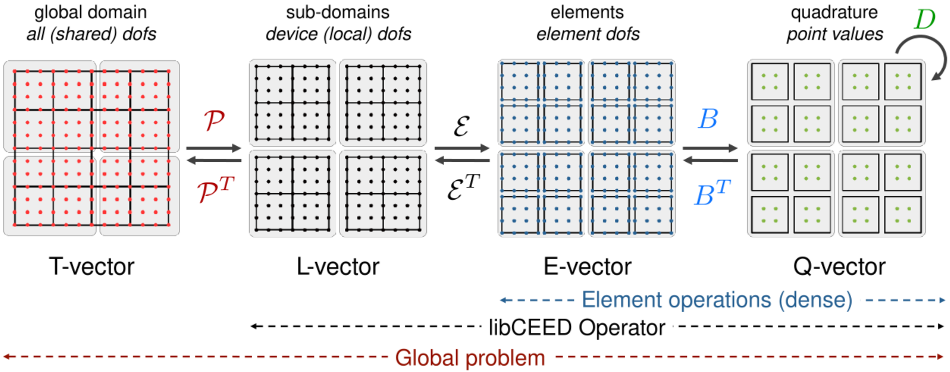
Finite Element Operator Decomposition

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



Finite Element Operator Decomposition

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



Compressible Fluid Equations in libCEED

Compressible Navier-Stokes

$$\boldsymbol{A_0} \boldsymbol{Y}_{,t} + \boldsymbol{F}_{i,i}(\boldsymbol{Y}) - S(\boldsymbol{Y}) = 0$$

for

$$\boldsymbol{A_0} \underbrace{\begin{bmatrix} p \\ u_i \\ T \end{bmatrix}}_{\boldsymbol{Y}} = \begin{bmatrix} \rho \\ \rho u_i \\ \rho e \end{bmatrix}, \quad \boldsymbol{F}_i(\boldsymbol{Y}) = \underbrace{\begin{pmatrix} \rho u_i \\ \rho u_i u_j + p\delta_{ij} \\ (\rho e + p) u_i \end{pmatrix}}_{\boldsymbol{F}_i^{\text{adv}}} + \underbrace{\begin{pmatrix} 0 \\ -\sigma_{ij} \\ -\rho u_j \sigma_{ij} - k T_{,i} \end{pmatrix}}_{\boldsymbol{F}_i^{\text{diff}}}, \quad S(\boldsymbol{Y}) = -\begin{pmatrix} 0 \\ \rho \boldsymbol{g} \\ 0 \end{pmatrix}$$

Compressible Navier-Stokes for Continuous-Galerkin FEM

Find $\boldsymbol{Y} \in \mathcal{S}^h$, $\forall \boldsymbol{v} \in \mathcal{V}^h$

$$\int_\Omega \boldsymbol{v} \cdot \left[\boldsymbol{A_0} \boldsymbol{Y}_{,t} - \boldsymbol{S}(\boldsymbol{Y})\right] \,\mathrm{d}\Omega + \int_\Omega \boldsymbol{v} \cdot \boldsymbol{F}_{i,i}(\boldsymbol{Y}) \,\mathrm{d}\Omega$$

Compressible Navier-Stokes for Continuous-Galerkin FEM

Find $\boldsymbol{Y} \in \mathcal{S}^h$, $\forall \boldsymbol{v} \in \mathcal{V}^h$

$$\int_\Omega \boldsymbol{v} \cdot \left[\boldsymbol{A_0} \boldsymbol{Y}_{,t} - \boldsymbol{S}(\boldsymbol{Y})\right] \,\mathrm{d}\Omega - \int_\Omega \boldsymbol{v}_{,i} \cdot \boldsymbol{F}_i(\boldsymbol{Y}) \,\mathrm{d}\Omega + \int_{\partial\Omega} \boldsymbol{v} \cdot \boldsymbol{F}_i(\boldsymbol{Y}) \cdot \widehat{\boldsymbol{n}}_i \,\mathrm{d}\partial\Omega$$

Compressible Navier-Stokes for Continuous-Galerkin FEM

Find $\boldsymbol{Y} \in \mathcal{S}^h$, $\forall \boldsymbol{v} \in \mathcal{V}^h$

$$\int_\Omega \boldsymbol{v} \cdot \left[\boldsymbol{A_0} \boldsymbol{Y}_{,t} - \boldsymbol{S}(\boldsymbol{Y})\right] \,\mathrm{d}\Omega - \int_\Omega \boldsymbol{v}_{,i} \cdot \boldsymbol{F}_i(\boldsymbol{Y}) \,\mathrm{d}\Omega + \int_{\partial\Omega} \boldsymbol{v} \cdot \boldsymbol{F}_i(\boldsymbol{Y}) \cdot \widehat{\boldsymbol{n}}_i \,\mathrm{d}\partial\Omega$$
$$\underbrace{+ \int_\Omega \mathscr{L}^{\mathrm{adv}}(\boldsymbol{v}) \boldsymbol{\tau} \left[\boldsymbol{A_0} \boldsymbol{Y}_{,t} + \boldsymbol{F}_{i,i}(\boldsymbol{Y}) - S(\boldsymbol{Y})\right] \,\mathrm{d}\Omega}_{\text{SUPG}} = 0$$

Compressible Navier-Stokes for Continuous-Galerkin FEM

Find $\boldsymbol{Y} \in \mathcal{S}^h$, $\forall \boldsymbol{v} \in \mathcal{V}^h$

$$\int_\Omega \boldsymbol{v} \cdot \left[\boldsymbol{A_0} \boldsymbol{Y}_{,t} - \boldsymbol{S}(\boldsymbol{Y})\right] \,\mathrm{d}\Omega - \int_\Omega \boldsymbol{v}_{,i} \cdot \boldsymbol{F}_i(\boldsymbol{Y}) \,\mathrm{d}\Omega + \int_{\partial\Omega} \boldsymbol{v} \cdot \boldsymbol{F}_i(\boldsymbol{Y}) \cdot \widehat{\boldsymbol{n}}_i \,\mathrm{d}\partial\Omega$$
$$\underbrace{+ \int_\Omega \mathscr{L}^{\mathrm{adv}}(\boldsymbol{v}) \boldsymbol{\tau} \left[\boldsymbol{A_0} \boldsymbol{Y}_{,t} + \boldsymbol{F}_{i,i}(\boldsymbol{Y}) - S(\boldsymbol{Y})\right] \,\mathrm{d}\Omega}_{\mathrm{SUPG}} = 0$$

Simplify into residual form:

$$\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y}) = 0$$

Compressible Navier-Stokes for Continuous-Galerkin FEM

Find $\boldsymbol{Y} \in \mathcal{S}^h$, $\forall \boldsymbol{v} \in \mathcal{V}^h$

$$\int_\Omega \boldsymbol{v} \cdot \left[\boldsymbol{A_0}\boldsymbol{Y}_{,t} - \boldsymbol{S}(\boldsymbol{Y})\right] \,\mathrm{d}\Omega - \int_\Omega \boldsymbol{v}_{,i} \cdot \boldsymbol{F}_i(\boldsymbol{Y}) \,\mathrm{d}\Omega + \int_{\partial\Omega} \boldsymbol{v} \cdot \boldsymbol{F}_i(\boldsymbol{Y}) \cdot \widehat{\boldsymbol{n}}_i \,\mathrm{d}\partial\Omega$$
$$\underbrace{+ \int_\Omega \mathscr{L}^{\mathrm{adv}}(\boldsymbol{v})\boldsymbol{\tau} \left[\boldsymbol{A_0}\boldsymbol{Y}_{,t} + \boldsymbol{F}_{i,i}(\boldsymbol{Y}) - S(\boldsymbol{Y})\right] \,\mathrm{d}\Omega}_{\mathrm{SUPG}} = 0$$

Simplify into residual form:

$$\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y}) = 0$$

$$\Rightarrow \quad {\color{red}\mathcal{P}^T}\mathcal{E}^T{\color{blue}B^T}{\color{green}G}{\color{blue}B}\mathcal{E}{\color{red}\mathcal{P}} \begin{bmatrix} \boldsymbol{Y}_{,t} \\ \boldsymbol{Y} \end{bmatrix} = 0$$

Efficient Implicit Timestepping

Implicit Timestepping

Implicit timestepping requires solving:

$$\frac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta \boldsymbol{Y} = -\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})$$

Implicit Timestepping

Implicit timestepping requires solving:

$$\frac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta \boldsymbol{Y} = -\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})$$

- System too large for direct solve $\longrightarrow$ iterative solve

Implicit Timestepping

Implicit timestepping requires solving:

$$\frac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta \boldsymbol{Y} = -\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})$$

- System too large for direct solve $\longrightarrow$ iterative solve
- Krylov subspace methods used most commonly

Implicit Timestepping

Implicit timestepping requires solving:

$$\frac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta \boldsymbol{Y} = -\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})$$

- System too large for direct solve $\longrightarrow$ iterative solve
- Krylov subspace methods used most commonly
- Krylov solvers form solution basis from $\operatorname{span}\left\{ \left[\frac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \right]^n \Delta \boldsymbol{Y} \right\}_{n=0}$

Implicit Timestepping

Implicit timestepping requires solving:

$$\frac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta \boldsymbol{Y} = -\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})$$

- System too large for direct solve $\longrightarrow$ iterative solve
- Krylov subspace methods used most commonly
- Krylov solvers form solution basis from $\mathrm{span}\left\{ \left[\frac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \right]^n \Delta \boldsymbol{Y} \right\}_{n=0}$

Bottom Line

Cost of $\frac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta \boldsymbol{Y}$ dominates implicit timestepping cost

Jacobian Matrix-Vector Multiply Options

How to compute $\dfrac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta\boldsymbol{Y}$?

Jacobian Matrix-Vector Multiply Options

How to compute $\dfrac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta \boldsymbol{Y}$?

- Store $\frac{\mathrm{d}\mathcal{G}}{\mathrm{d}\boldsymbol{Y}}$ directly (sparse matrix representation)
  - **Pros:** Opens up preconditioning options
  - **Cons:** Is large, expensive to store

Jacobian Matrix-Vector Multiply Options

How to compute $\dfrac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta \boldsymbol{Y}$?

- Store $\frac{\mathrm{d}\mathcal{G}}{\mathrm{d}\boldsymbol{Y}}$ directly (sparse matrix representation)
  - **Pros:** Opens up preconditioning options
  - **Cons:** Is large, expensive to store
- Finite difference matrix-free approximation:

$$\frac{\mathrm{d}\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\mathrm{d}\boldsymbol{Y}} \Delta \boldsymbol{Y} \approx \frac{\mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y} + \epsilon \Delta \boldsymbol{Y}) - \mathcal{G}(\boldsymbol{Y}_{,t}, \boldsymbol{Y})}{\epsilon}$$

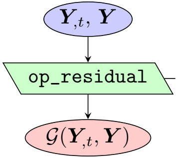
  - **Pros:** Just need a residual evaluation, cheap (in programming and computation)
  - **Cons:** Accuracy limited to $\sqrt{\epsilon_{\text{machine}}}$, preconditioning require partial assembly

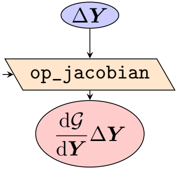
Exact Matrix-Free Jacobian via CeedOperator

$$
\begin{aligned}
\frac{\mathrm{d}\mathcal{G}}{\mathrm{d}\boldsymbol{Y}}\Delta\boldsymbol{Y} &= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{Y}} \overbrace{\left[\mathcal{P}^T \mathcal{E}^T B^T G B \mathcal{E} \mathcal{P}\right]}^{\mathcal{G}(\boldsymbol{Y}_{,t},\boldsymbol{Y})} \Delta\boldsymbol{Y} \\
&= \left[\mathcal{P}^T \mathcal{E}^T B^T \frac{\mathrm{d}G}{\mathrm{d}\boldsymbol{Y}} B \mathcal{E} \mathcal{P}\right] \Delta\boldsymbol{Y}
\end{aligned}
$$

Exact Matrix-Free Jacobian via CeedOperator

$$
\begin{aligned}
\frac{\mathrm{d}\mathcal{G}}{\mathrm{d}\boldsymbol{Y}}\Delta\boldsymbol{Y} &= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{Y}}\overbrace{\left[\mathcal{P}^T\mathcal{E}^T B^T G B \mathcal{E}\mathcal{P}\right]}^{\mathcal{G}(\boldsymbol{Y}_{,t},\boldsymbol{Y})}\Delta\boldsymbol{Y} \\
&= \left[\mathcal{P}^T\mathcal{E}^T B^T \frac{\mathrm{d}G}{\mathrm{d}\boldsymbol{Y}} B\mathcal{E}\mathcal{P}\right]\Delta\boldsymbol{Y}
\end{aligned}
$$

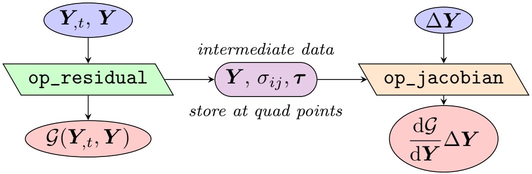




Exact Matrix-Free Jacobian via CeedOperator

$$\frac{\mathrm{d}\mathcal{G}}{\mathrm{d}\boldsymbol{Y}}\Delta\boldsymbol{Y} = \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{Y}}\overbrace{\left[\mathcal{P}^T\mathcal{E}^T B^T G B \mathcal{E}\mathcal{P}\right]}^{\mathcal{G}(\boldsymbol{Y}_{,t},\boldsymbol{Y})}\Delta\boldsymbol{Y}$$

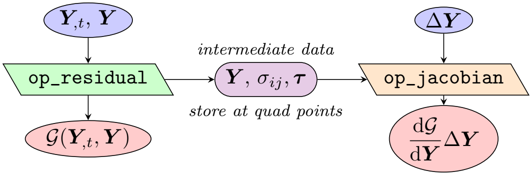
$$= \left[\mathcal{P}^T\mathcal{E}^T B^T \frac{\mathrm{d}G}{\mathrm{d}\boldsymbol{Y}} B \mathcal{E}\mathcal{P}\right]\Delta\boldsymbol{Y}$$



Exact Matrix-Free Jacobian via CeedOperator

$$\frac{\mathrm{d}\mathcal{G}}{\mathrm{d}\boldsymbol{Y}}\Delta\boldsymbol{Y} = \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{Y}}\overbrace{\left[\mathcal{P}^T\mathcal{E}^T B^T G B \mathcal{E}\mathcal{P}\right]}^{\mathcal{G}(\boldsymbol{Y}_{,t},\boldsymbol{Y})}\Delta\boldsymbol{Y}$$

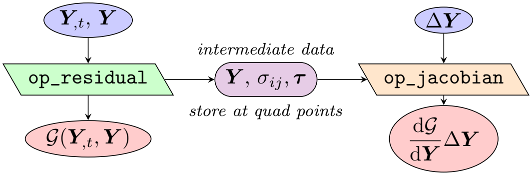
$$= \left[\mathcal{P}^T\mathcal{E}^T B^T \frac{\mathrm{d}G}{\mathrm{d}\boldsymbol{Y}} B \mathcal{E}\mathcal{P}\right]\Delta\boldsymbol{Y}$$



- **Pros:** Exact Jacobian matrix-vector product[2]
- **Cons:** Preconditioning requires partial assembly, requires coding Jacobian

Exact Matrix-Free Jacobian via CeedOperator

$$\begin{aligned}\frac{\mathrm{d}\mathcal{G}}{\mathrm{d}\boldsymbol{Y}}\Delta\boldsymbol{Y} &= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{Y}}\overbrace{\left[\mathcal{P}^T\mathcal{E}^T B^T G B \mathcal{E}\mathcal{P}\right]}^{\mathcal{G}(\boldsymbol{Y}_{,t},\boldsymbol{Y})}\Delta\boldsymbol{Y} \\ &= \left[\mathcal{P}^T\mathcal{E}^T B^T \frac{\mathrm{d}G}{\mathrm{d}\boldsymbol{Y}} B\mathcal{E}\mathcal{P}\right]\Delta\boldsymbol{Y}\end{aligned}$$



- **Pros:** Exact Jacobian matrix-vector product[2]
- **Cons:** Preconditioning requires partial assembly, requires coding Jacobian

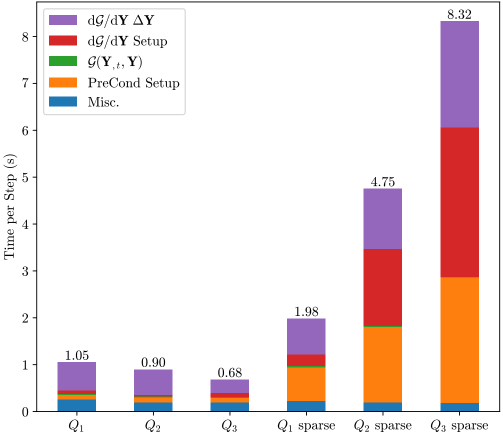
[2]Affect of specific terms may be ignored from the Jacobian. This is done for $\mathrm{d}\boldsymbol{\tau}/\mathrm{d}\boldsymbol{Y}$

Performance and Results of Flat Plate Boundary Layer Simulation

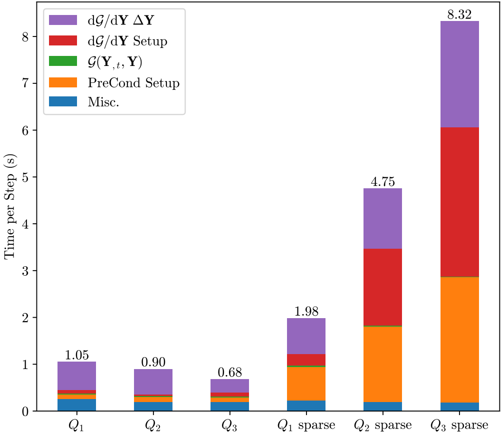
Problem Description

- Flat plate boundary layer with zero pressure gradient
  - $Re_\theta \approx 970$ boundary layer at inflow, $M \approx 0.1$
  - Synthetic turbulence generation (STG) used for inflow structures
  - Run at implicit large eddy simulation (ILES) resolution for linears (higher orders may be DNS level, tbd)
- Test 3 different order elements, Q_1, Q_2, Q_3 tensor-product hexes
- Maintain *DOF resolution* (DoFs per physical length/ global DoF count)
- Performance results shown for two nodes of ALCF's Polaris ($4\times$ NVIDIA A100 per node)

Exact Matrix-Free Jacobian vs Sparse

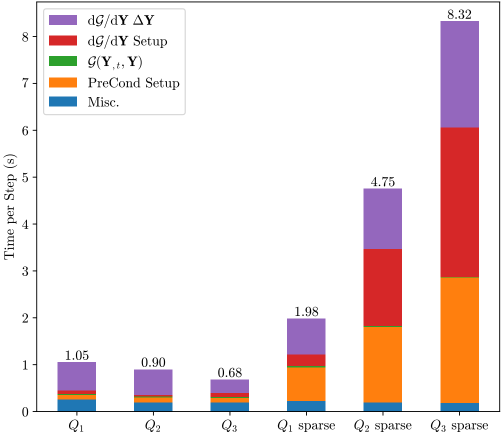


Exact Matrix-Free Jacobian vs Sparse



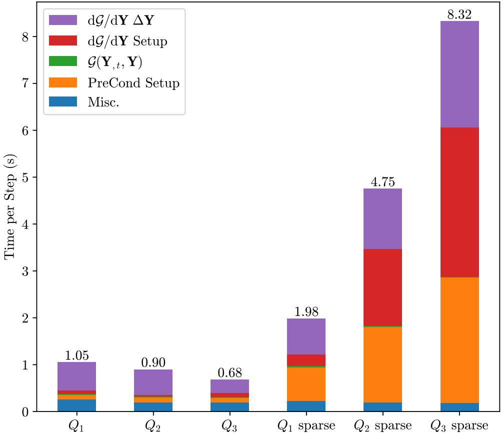
- Sparse $\mathrm{d}\mathcal{G}/\mathrm{d}\boldsymbol{Y}\Delta\boldsymbol{Y}$ significantly slower than matrix-free

Exact Matrix-Free Jacobian vs Sparse



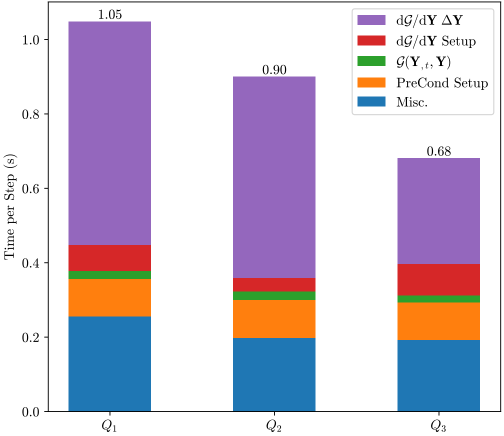
- Sparse $\mathrm{d}\mathcal{G}/\mathrm{d}\boldsymbol{Y}\,\Delta\boldsymbol{Y}$ significantly slower than matrix-free
- Time to assemble $\mathrm{d}\mathcal{G}/\mathrm{d}\boldsymbol{Y}$ quite large

Exact Matrix-Free Jacobian vs Sparse

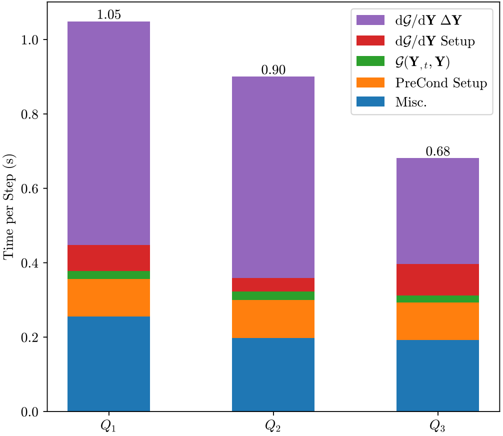


- Sparse $\mathrm{d}\mathcal{G}/\mathrm{d}\boldsymbol{Y}\Delta\boldsymbol{Y}$ significantly slower than matrix-free
- Time to assemble $\mathrm{d}\mathcal{G}/\mathrm{d}\boldsymbol{Y}$ quite large
- Associated costs rise with element order

Fluids Performance Analysis

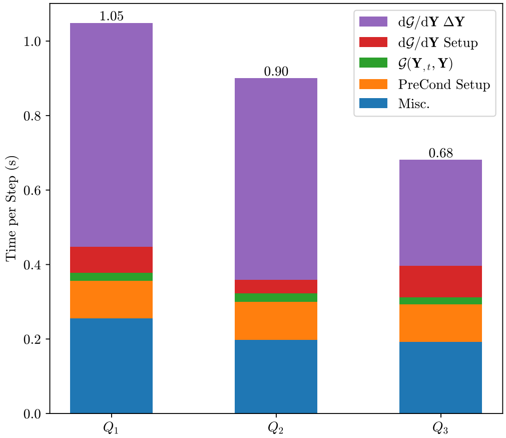


Fluids Performance Analysis



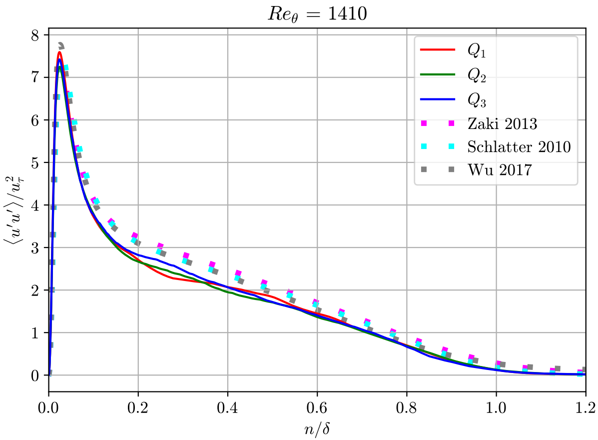
- Time of $\mathrm{d}\mathcal{G}/\mathrm{d}\boldsymbol{Y}\Delta\boldsymbol{Y}$ decreases as order increases

Fluids Performance Analysis



- Time of $\mathrm{d}\mathcal{G}/\mathrm{d}\boldsymbol{Y}\,\Delta\boldsymbol{Y}$ decreases as order increases
- $\mathrm{d}\mathcal{G}/\mathrm{d}\boldsymbol{Y}$ setup time increases with order
  - Dominant cost is partial matrix assembly for preconditioning

Results of Flat Plate Boundary Layer



- Spanwise statistics implemented to verify scale-resolving results
- Results *not* converged, but show realistic stress profiles

Zaki *et al.*, 2013, *From Streaks to Spots and on to Turbulence: Exploring the Dynamics of Boundary Layer Transition*
Schlatter *et al.*, 2010, *Assessment of direct numerical simulation data of turbulent boundary layers*
Wu *et al.*, 2017, *Transitional–turbulent spots and turbulent–turbulent spots in boundary layers*

Support and References

- Research supported by US Department of Energy through DE-SC0021411 FASTMath SciDAC Institute
- Argonne Leadership Computing Facility resources used for this research
- We thank PETSc and libCEED developers, especially Jeremy Thompson and Junchao Zhang among many others