Performance tuning of Newton-GMRES methods for discontinuous Galerkin discretization of the Navier-Stokes equations

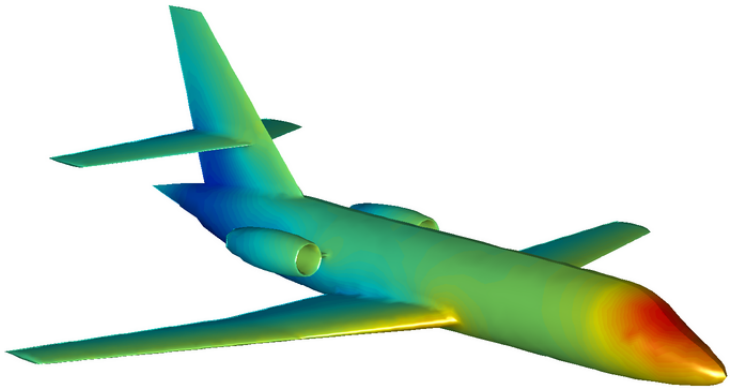
Matthew J. Zahr and Per-Olof Persson

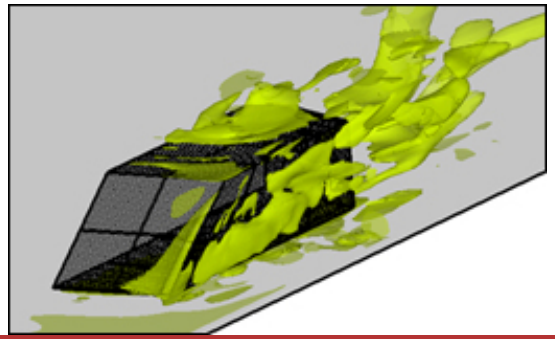
Stanford University
University of California, Berkeley
Lawrence Berkeley National Lab

25th June 2013
San Diego, CA
43rd AIAA Fluid Dynamics Conference and Exhibit

Motivation

- Low-order methods perform poorly for problems where high numerical accuracy is required
  - Wave propagation (e.g. aeroacoustics)
  - Turbulent flow (e.g. draw & transition prediction)
  - Non-linear interactions (e.g. fluid-structure coupling)
- High-order discontinuous Galerkin methods attractive options:
  - Low dissipation, stabilization, complex geometries
- Parallel computers required for realistic problems because of high computational and storage costs with DG





Motivation

- Fundamental properties of Discontinuous Galerkin (DG) methods:

	FVM	FDM	FEM	DG
1) High-order/Low dispersion	✗	✓	✓	✓
2) Unstructured meshes	✓	✗	✓	✓
3) Stability for conservation laws	✓	✓	✗	✓

- However, several problems to resolve:
  - High CPU/memory requirements (compared to FVM or H-O FDM)
  - Low tolerance to under-resolved features
  - High-order geometry representation and mesh generateion
- The challenge is to make DG competitive for real-world problems

Semi-discrete Equations

Discretization of the Navier-Stokes equations with DG-FEM

$$\mathbb{M}\dot{\mathbf{u}}(t) = \mathbf{r}(t, \mathbf{u}(t))$$

where

- $\mathbb{M} \in \mathbb{R}^{N \times N}$ is the block diagonal mass matrix,
- $\mathbf{u} \in \mathbb{R}^N$ is the time-dependent state vector arising from the DG-FEM discretization, and
- $\mathbf{r} : \mathbb{R}_+ \times \mathbb{R}^N \to \mathbb{R}^N$ is the spatially-discretized nonlinearity of the Navier-Stokes equations.

Implicit Time Integration

- Implicit solvers typically required because of CFL restrictions from viscous effects, low Mach numbers, and adaptive/anisotropic grids
  - Backward differentiation formulas
  - Runge-Kutta methods
- Jacobian matrices are large even at $p = 2$ or $p = 3$, however:
  - They are required for non-trivial preconditioners
  - They are very expensive to recompute
- Therefore, we consider matrix-based Newton-Krylov solvers

Backward Differentiation Formulas (BDF)

$$\mathbb{M}\mathbf{u}^{(n+1)} - \left(\sum_{i=0}^{n} \alpha_i \mathbb{M}\mathbf{u}^{(i)} + \kappa \Delta t \mathbf{r}(t_{n+1}, \mathbf{u}^{(n+1)}) \right) = 0$$

- BDF1 (Backward Euler)

$$\boldsymbol{\alpha}^1 = \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix} \qquad \kappa_1 = 1$$

- BDF2

$$\boldsymbol{\alpha}^2 = \begin{bmatrix} 0 & \cdots & 0 & -1/3 & 4/3 \end{bmatrix} \qquad \kappa_2 = 2/3$$

- BDF3

$$\boldsymbol{\alpha}^3 = \begin{bmatrix} 0 & \cdots & 0 & 2/11 & -9/11 & 18/11 \end{bmatrix} \qquad \kappa_3 = 6/11$$

- BDF23

$$\boldsymbol{\alpha}^{23} = \tau \boldsymbol{\alpha}^2 + (1-\tau)\boldsymbol{\alpha}^3 \qquad \kappa_{23} = \tau \kappa_2 + (1-\tau)\kappa_3$$

BDF23_3: 3rd Order, A-stable BDF

- Define $\mathbf{u}_{23}$ as

$$\mathbf{u}_{23} = \boldsymbol{\alpha}_n^{23}\mathbf{u}^{(n)} + \boldsymbol{\alpha}_{n-1}^{23}\mathbf{u}^{(n-1)} + \boldsymbol{\alpha}_{n-2}^{23}\mathbf{u}^{(n-2)}$$

- Solve the nonlinear Backward Cauchy-Euler (BCE) equation $\mathcal{R}(\mathbf{u}_i) = 0$, where

$$\mathcal{R}(\mathbf{u}_i) = \mathbb{M}\mathbf{u}_i - (\mathbb{M}\mathbf{u}_{23} + \kappa_{23}\Delta t\mathbf{r}(t_{n+1}, \mathbf{u}_i))$$

- Define $\mathbf{u}_{33}$ as

$$\mathbf{u}_{33} = \boldsymbol{\alpha}_n^{3}\mathbf{u}^{(n)} + \boldsymbol{\alpha}_{n-1}^{3}\mathbf{u}^{(n-1)} + \boldsymbol{\alpha}_{n-2}^{3}\mathbf{u}^{(n-2)} - \delta(\mathbf{u}_i - \mathbf{u}_{23})$$

- Solve the nonlinear BCE equation $\mathcal{R}(\mathbf{u}_{n+1}) = 0$, where

$$\mathcal{R}(\mathbf{u}_{n+1}) = \mathbb{M}\mathbf{u}^{(n+1)} - \left(\mathbb{M}\mathbf{u}_{33} + \kappa_{33}\Delta t\mathbf{r}(t_{n+1}, \mathbf{u}^{(n+1)})\right)$$

Diagonally-Implicit Runge Kutta (DIRK)

Standard formulation (k-form)

$$\mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} + \sum_{i=1}^{s} b_i \mathbf{k}_i$$

$$\mathbb{M}\mathbf{k}_i = \Delta t \mathbf{r}\left(t_n + c_i \Delta t, \mathbf{u}^{(n)} + \sum_{j=1}^{i} a_{ij} \mathbf{k}_j \right),$$

Alternate formulation (u-form)

$$\mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} + \Delta t \sum_{j=1}^{s} b_j \mathbb{M}^{-1} \mathbf{r}(t_n + c_j \Delta t, \bar{\mathbf{u}}_j)$$

$$\mathbb{M}\bar{\mathbf{u}}_i = \mathbb{M}\mathbf{u}^{(n)} + \Delta t \sum_{j=1}^{i} a_{ij} \mathbf{r}\left(t_n + c_j \Delta t, \bar{\mathbf{u}}_j\right).$$

Newton Prediction

Accurate predictions for Newton's method may result in fewer nonlinear iterations

- Extrapolation using Lagrangian polynomial
  - Construct polynomial of order p with $p+1$ points in solution history
  - Use polynomial to predict solution at next time step
  - Constant (LAG0), linear (LAG1), quadratic (LAG2)
- Extrapolation using Hermite polynomial
  - Construct polynomial of order $2p+1$ with p points in history of solution and derivative
  - Use polynomial to predict solution at next time step
  - Linear (HERM1), cubic (HERM2), quintic (HERM3)

Jacobian Recycling

- For matrix-based methods, every nonlinear iteration requires a Jacobian evaluation
  - Jacobian assembly at least 10× as expensive as residual evaluation
- Re-using Jacobians yield inexact Newton directions
  - May require more Newton iterations per time step
  - Enables re-use of preconditioner
  - Reduces number of Jacobian evaluations and preconditioner computations
- Recompute Jacobian when corresponding Newton step fails to reduce nonlinear residual

GMRES Tolerance

When using GMRES to solve

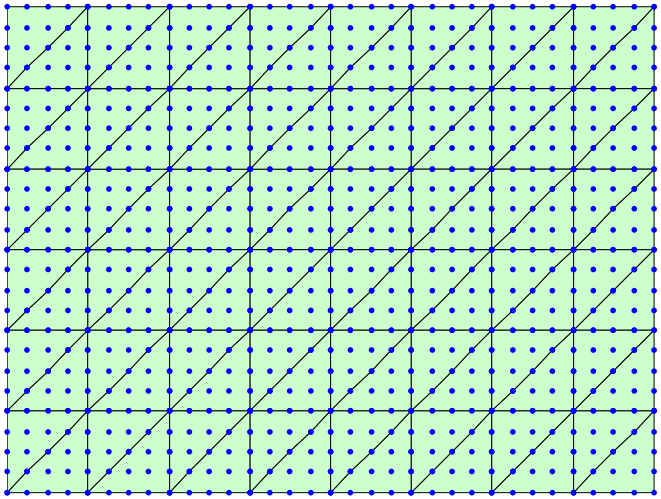
$$Ax = b,$$

common convergence criteria is

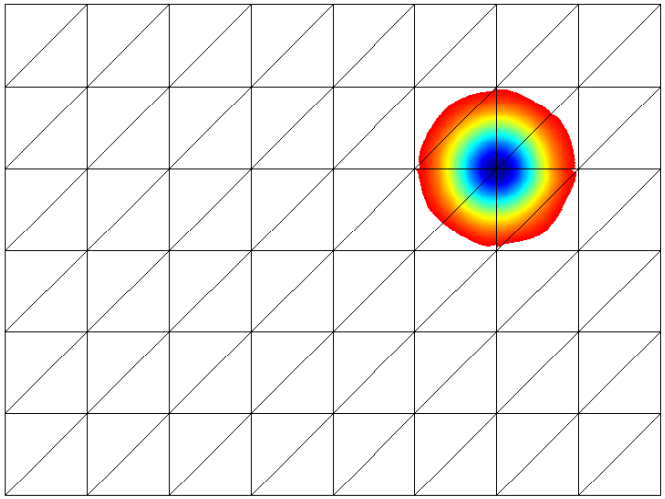
$$||Ax - b||_2 \leq Gtol\, ||b||_2$$

- Small GMRES tolerance $\rightarrow$ search directions "close" to Newton directions
  - More GMRES iterations per Newton step, fewer Newton iterations
- Large GMRES tolerance $\rightarrow$ search directions may be far from Newton directions
  - Fewer GMRES iterations per Newton step, more Newton iterations

Euler Vortex



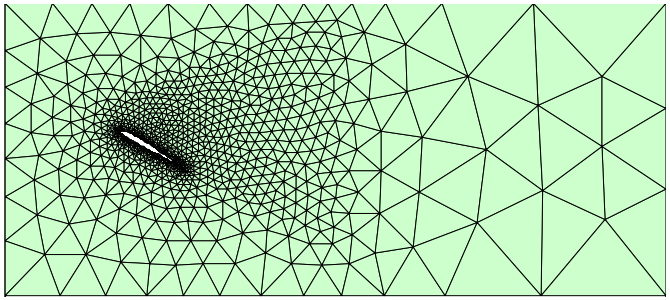
Euler vortex mesh, with degree $p = 4$



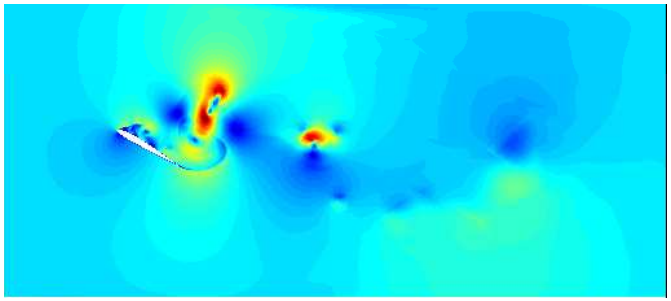
Solution (density)

Figure : Euler Vortex: Mesh and Solution at $t_0 = \sqrt{10^2 + 5^2}$

Viscous flow over NACA wing at high angle of attack

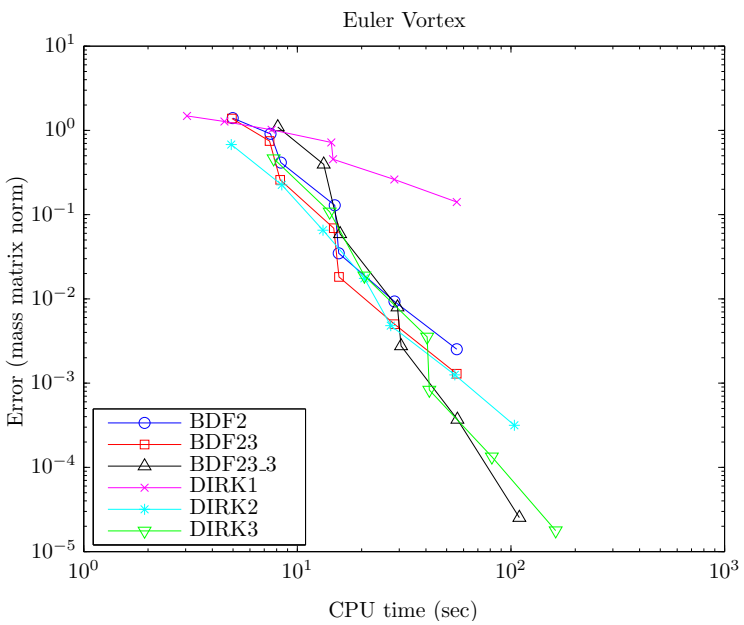


NACA mesh, with degree $p = 4$

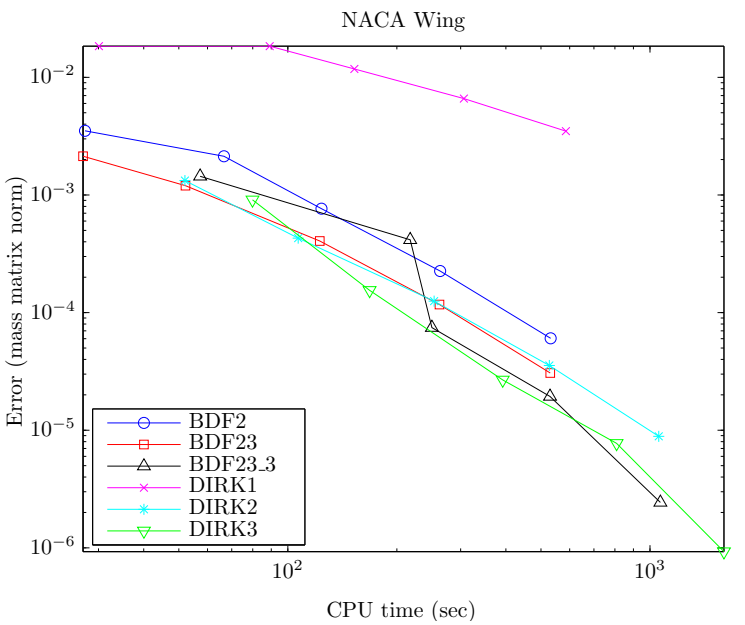


Solution (Mach)

Figure : NACA Wing: Mesh and Solution at $t_0 = 5.01$

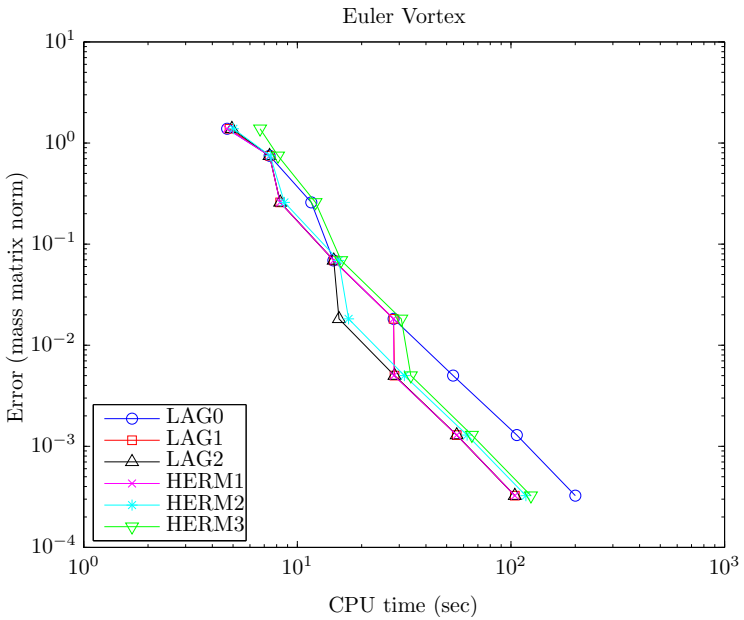


LAG2, Jacobian Recomputation, $Gtol = 10^{-5}$

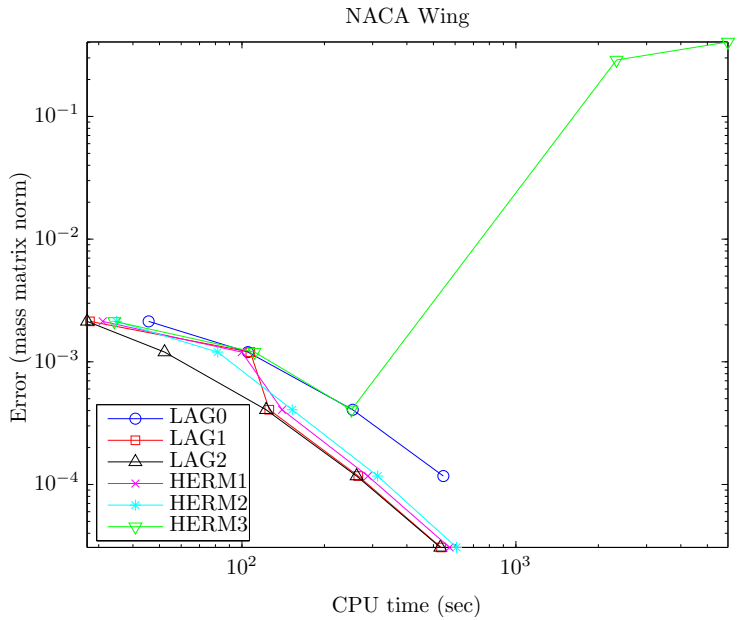


LAG2, Jacobian Recomputation, $Gtol = 10^{-5}$

- BDF23_3 cheaper than DIRK3 for high accuracy
- BDF23 has same slope but better offset than BDF2

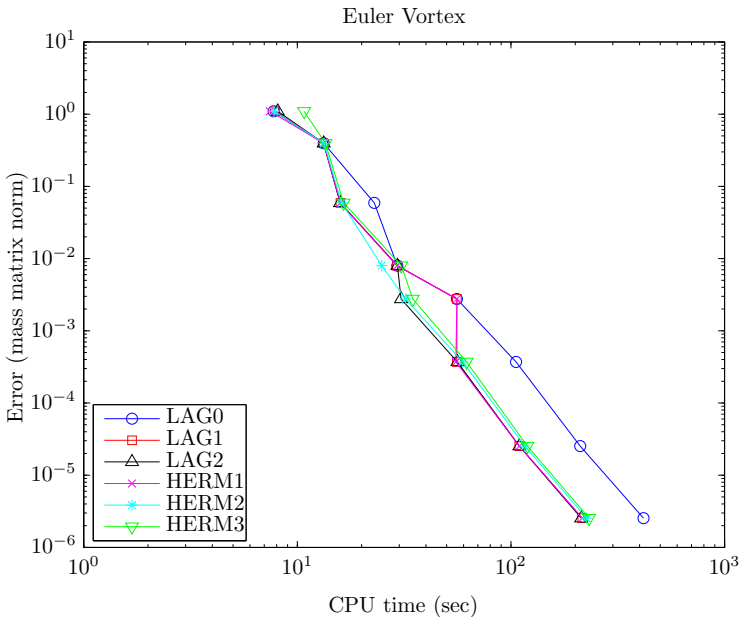


BDF23, Jacobian Recomputation, $Gtol = 10^{-5}$

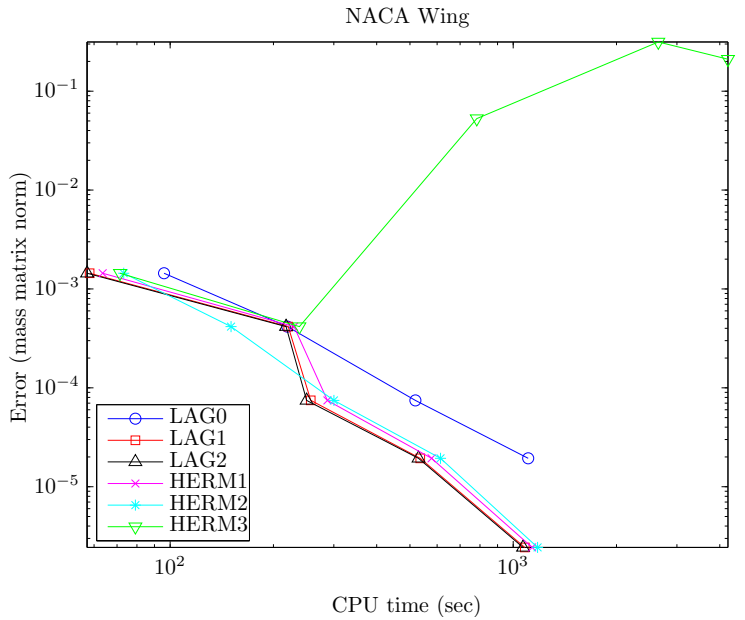


BDF23, Jacobian Recomputation, $Gtol = 10^{-5}$

- LAG0 is a poor predictor
- LAG1, LAG2, HERM1, HERM2 are comparable predictors
- LAG2 is a good predictor for all Δt considered
- High-order extrapolation may not be a good idea

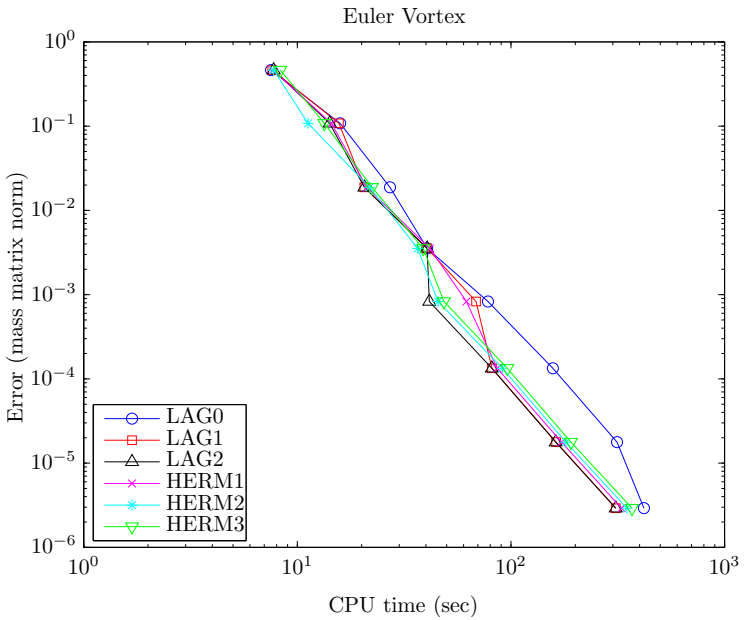


BDF23_3, Jacobian Recomputation, $Gtol = 10^{-5}$

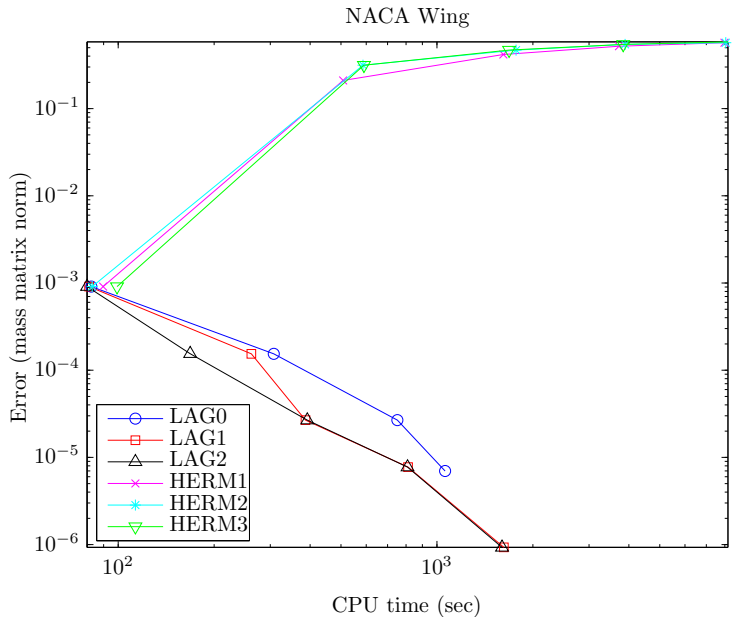


BDF23_3, Jacobian Recomputation, $Gtol = 10^{-5}$

- LAG0 is a poor predictor
- LAG1, LAG2, HERM1, HERM2 are comparable predictors
- LAG2 is a good predictor for all Δt considered
- High-order extrapolation may not be a good idea

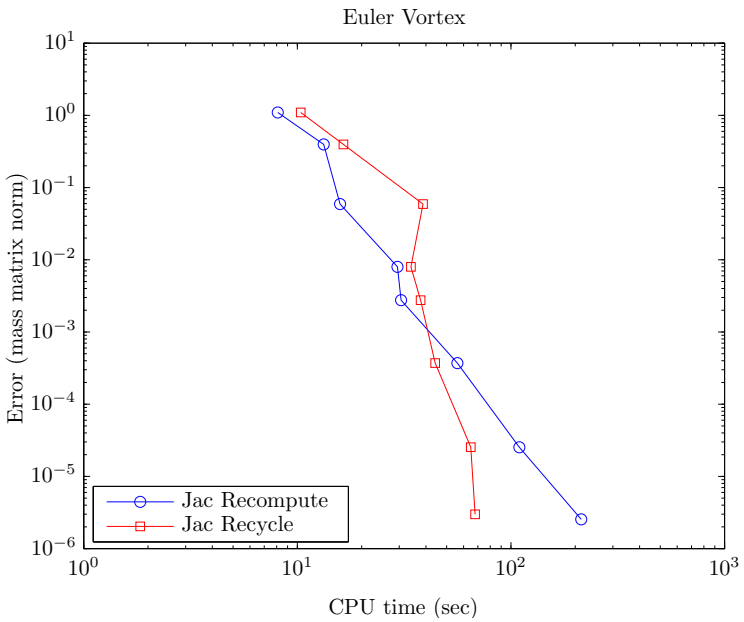


DIRK3, Jacobian Recomputation, $Gtol = 10^{-5}$

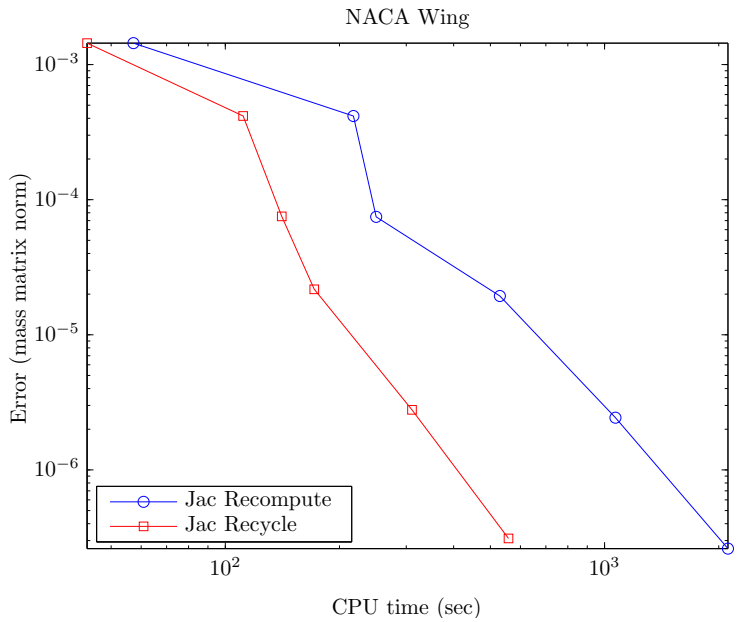


DIRK3, Jacobian Recomputation, $Gtol = 10^{-5}$

- LAG0 is a poor predictor
- LAG1, LAG2, HERM1, HERM2 are comparable predictors
- LAG2 is a good predictor for all Δt considered
- Hermite predictors not reliable

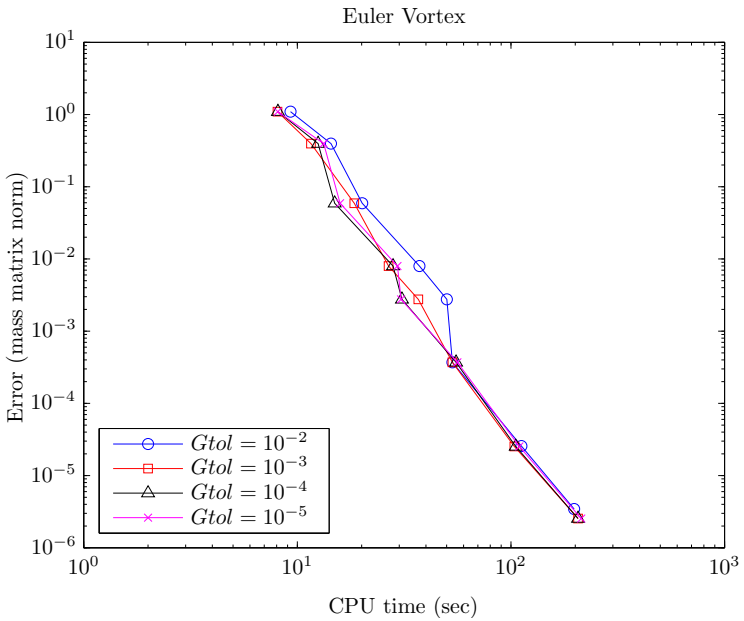


BDF23_3, LAG2, $Gtol = 10^{-5}$

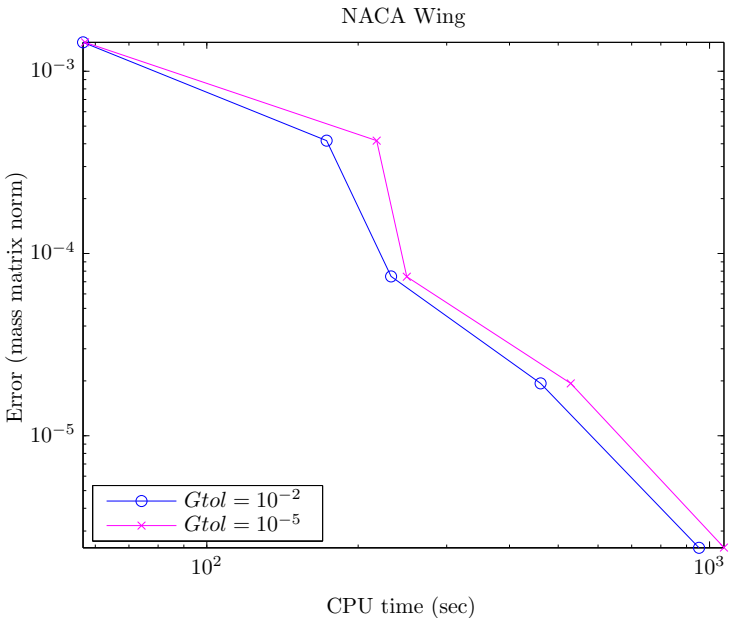


BDF23_3, LAG2, $Gtol = 10^{-5}$

- Jacobian recycling is beneficial most beneficial for small Δt
- More sophisticated recomputation strategies could make the differences more pronounced

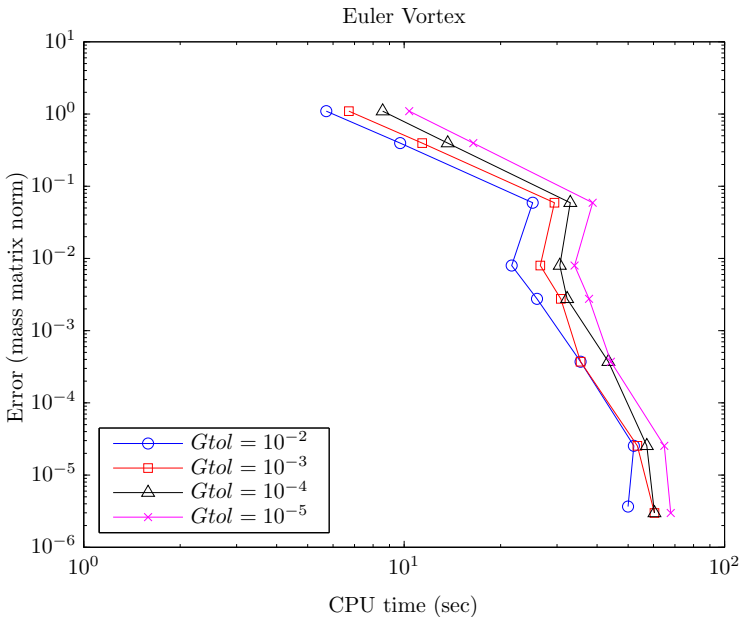


BDF23_3, LAG2, Jacobian Recomputation

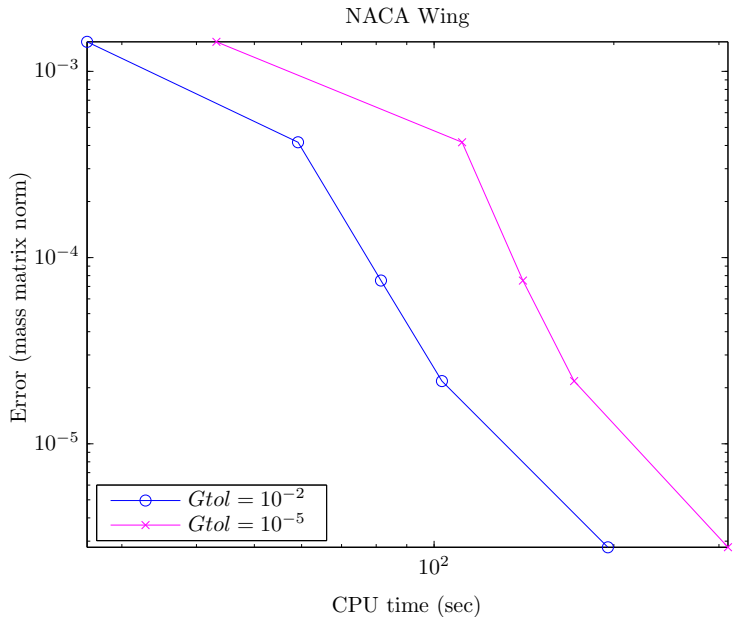


BDF23_3, LAG2, Jacobian Recomputation

- EV: Smaller *Gtol* better for range of Δt considered
- NACA: Larger *Gtol* better for range of Δt considered



BDF23_3, LAG2, Jacobian Recycling



BDF23_3, LAG2, Jacobian Recycling

- Larger *Gtol* better for range of Δt considered

Speedup Results - BDF23

	BDF23, LAG0, 10^{-5}	**BDF23, LAG2,** 10^{-5}
L2 Error	3.24×10^{-4}	3.24×10^{-4}
CPU Time (sec)	1.95×10^{4}	7.86×10^{3}
Speedup over Base	5.41	13.4

Table : Speedup/Error Results: Euler Vortex - BDF23, Jacobian Recycling

	BDF23, LAG0, 10^{-5}	**BDF23, LAG2,** 10^{-5}
L2 Error	6.34×10^{-5}	7.82×10^{-6}
CPU Time (sec)	1.14×10^{3}	6.20×10^{2}
Speedup over Base	6.24	11.5

Table : Speedup/Error Results: NACA Wing - BDF23, Jacobian Recycling

Base: DIRK3, LAG0, Jacobian Recomputation, $Gtol = 10^{-5}$

Speedup Results - BDF23_3

	BDF23_3, LAG0, 10^{-5}	**BDF23_3, LAG2,** 10^{-5}
L2 Error	2.95×10^{-6}	2.98×10^{-6}
CPU Time (sec)	4.48×10^{4}	1.71×10^{4}
Speedup over Base	2.35	6.17

Table : Speedup/Error Results: Euler Vortex - BDF23_3, Jacobian Recycling

	BDF23_3, LAG0, 10^{-5}	**BDF23_3, LAG2,** 10^{-5}
L2 Error	3.10×10^{-5}	3.11×10^{-7}
CPU Time (sec)	2.38×10^{3}	1.25×10^{3}
Speedup over Base	3.00	5.73

Table : Speedup/Error Results: NACA Wing - BDF23_3, Jacobian Recycling

Base: DIRK3, LAG0, Jacobian Recomputation, $Gtol = 10^{-5}$

Speedup Results - DIRK3

	DIRK3, LAG0, 10^{-5}	**DIRK3, LAG2,** 10^{-5}
L2 Error	2.92×10^{-6}	2.92×10^{-6}
CPU Time (sec)	4.80×10^{4}	4.13×10^{4}
Speedup over Base	2.20	2.55

Table : Speedup/Error Results: Euler Vortex - DIRK3, Jacobian Recycling

	DIRK3, LAG0, 10^{-5}	**DIRK3, LAG2,** 10^{-5}
L2 Error	1.64×10^{-7}	1.15×10^{-7}
CPU Time (sec)	3.65×10^{3}	3.59×10^{3}
Speedup over Base	1.96	1.99

Table : Speedup/Error Results: NACA Wing - DIRK3, Jacobian Recycling

Base: DIRK3, LAG0, Jacobian Recomputation, $Gtol = 10^{-5}$

Conclusions

- Two new BDF-type schemes introduced: BDF23, BDF23_3
- BDF23_3 attractive high-order alternative to DIRK3
- Quadratic Lagrange polynomial prediction significantly better than commonly used constant prediction
- Jacobian recycling speeds up computations by factor of $2-3$ for small Δt
- Larger GMRES tolerance provides speedup *particularly when Jacobians are recycled*
- BDF23 with LAG2 predictor is 11 - 14 times faster than DIRK3 with LAG0 prediction
- BDF23_3 with LAG2 predictor is about 6 times faster than DIRK3 with LAG0 prediction

Acknowledgments

- Department of Energy Computational Science Graduate Fellowship
- Per-Olof Persson