Preconditioners for Discontinuous Galerkin Discretizations of 3D viscous compressible flows

Philipp Birken
Mark Haas
Gregor Gassner, Claus-Dieter Munz

University of Kassel (SFB/TRR 30)
University of Stuttgart (Mark now at Bosch GmbH)

October 9th 2013
Efficient solution of large systems of non-linear PDEs in science
Outline

1. The problem
2. Time integration schemes
3. Specific DG scheme and solver
4. Numerical Results
Outline

1. The problem
2. Time integration schemes
3. Specific DG scheme and solver
4. Numerical Results
Figure: Hurricane Catrina, NASA, PD (left), Lillgrund Offshore windfarm, Mariusz Padziora, CC-by-sa 3.0 (right); Gas Quenching, Steinhoff
Compressible Navier Stokes equations

Second order system of conservation laws (mass, momentum, energy) modeling viscous compressible flow:

\[
\begin{align*}
\partial_t \rho + \nabla \cdot \mathbf{m} &= 0, \\
\partial_t m_i + \sum_{j=1}^{3} \partial_{x_j} (m_i v_j + p \delta_{ij}) &= \frac{1}{Re} \sum_{j=1}^{3} \partial_{x_j} S_{ij}, & i = 1, 2, 3 \\
\partial_t (\rho E) + \nabla \cdot (H\mathbf{m}) &= \frac{1}{Re} \sum_{j=1}^{3} \partial_{x_j} \left( \sum_{i=1}^{3} S_{ij} v_i - \frac{1}{Pr} W_j \right),
\end{align*}
\]

Equation of state: \( p = (\gamma - 1) \rho e \).

\[
\begin{align*}
u_t + \nabla \cdot f(u, \nabla u) &= 0.
\end{align*}
\]
Consider \textit{unsteady 3D compressible viscous flow problems}

- Important: Turbulence, Boundary Layer, Mach number
- Discretization in space via FV or DG
- Leads to initial value problem in time
  \[\rightarrow \text{huge number of unknowns, problem is typically stiff.}\]
  \[\rightarrow \text{implicit time integration necessary.}\]
- Goal: \textit{Fast implicit 3D solver in the context of DG!}

Parallel Solvers

Form of equation is independent of precise time integration scheme

\[ u = \alpha \Delta t f(u) + \psi. \]

→ Need for fast low storage parallel scaling solvers

- Preconditioned inexact Jacobian-Free Newton-Krylov (JFNK)
- FAS (multigrid) with appropriate smoothers

Figure: Cray Hermit in Stuttgart; Bild: ThE cRaCkEr, CC-by-sa 3.0, via Wikimedia Commons
Why high order methods?

- Standard for Finite Elements, Finite Volume is 2. order
- Higher order for FV methods in 3D problematic, since a lot of neighboring cells necessary (ENO/WENO)
- Popular approach is DG: Localize high order in cell, but use discontinuous ansatz functions for stability for convection
- Alternative: Use FE with high order and stabilization
- For many turbulent flows necessary to resolve eddies
- Direct numerical simulation (DNS): $O(Re^3)$ unknowns
- Large Eddy Simulation (LES): $O(Re^{2.5})$ unknowns
- Efficient LES only imaginable using high order methods
- Goal: Efficient DG method for LES
Finite Volume schemes (FVM)

- Standard schemes in computational fluid dynamics.
- Developed in last 50 years.
- In most simple form first order.
- Higher order in space via linear reconstruction and limiter.
- More than second order not practical.
- For explicit Euler scheme stable for $CFL < 1$.
- Implicit methods necessary for large number of problems and part of 3D production solvers.
- Implicit methods magnitudes faster than explicit ones.
Fast implicit Discontinuous Galerkin solvers?

- For first order identical to FVM.
- Much higher orders $p$ possible using piecewise polynomials.
- Huge number of different approaches.
- For explicit scheme stable for $CFL = O\left(\frac{1}{2p-1}\right)$.
- Implicit methods thus even more necessary.
- In 2D no fast implicit method available.
- In 3D additional difficulties from memory requirements.
- Is it possible to get a fast implicit DG scheme in 3D?
- Determines applicability of DG in industry!

How do different time integration schemes compare for DG?
The story so far

In the implicit context...

- ...quite a number of papers on steady Euler and NS

Less so for the unsteady case:

- Klaij, van der Vegt, van der Wen 06, 2D-NS
- Wang, Mavriplis 07, 2D-Euler
- Kanevsky, Carpenter, Gottlieb, Hesthaven 07, 2D-NS
- Persson, Peraire 08, 2D-NS
- St.-Cyr, Neckels 09, 2D-Euler
- Dolejsi, Holik, Hozman 11, 2D-NS
- Uranga, Persson, Drega, Peraire 11, 3D-NS
- Birken, Gassner, Haas, Munz 13, 3D-NS

Implicit methods for unsteady 3D NS still need some work!
Comparing time integration schemes

In this talk, I’ll focus on comparing time integration schemes.

Need to fix

- Specific DG scheme
- Specific solver for systems in implicit time integration
- Find a fair way of comparing schemes
Outline

1. The problem
2. Time integration schemes
3. Specific DG scheme and solver
4. Numerical Results
Eigenvalues

Figure: Re=100, 4th degree polynomial
Implicit time integration

- Large stability region required: **A-stable** methods!
- Unsteady problem: **Higher order and time adaptivity.**
- BDF often used, but does not fit the profile.
- Bijl et al (01,02): ESDIRK methods competitive!
- One explicit and subsequent backward Euler steps.
- Time adaptivity easy.
explicit schemes

- **LSERK4**: 5-stage explicit RK, 4th order, large stability region
  
  (Carpenter, Kennedy)

- **RKCK**: local time stepping scheme using CERK methods to predict solution locally in each cell, then correct according to a higher order
  Space-time expansion via the Cauchy-Kovalevskaya procedure
  
  (Gassner, Dumbser, Hindenlang, Munz 11).

implicit schemes

- **ESDIRK4, ESDIRK3, SDIRK3**

- **ROS34PW2**, Rosenbrock method, linearized SDIRK3 (Rang, Angermann 05).
Tolerance scaling and time adaptivity

- Given TOL, determine new time step based on
  \[ d_i = TOL|u_i^n| + TOL \]
  \[ \Delta t_{\text{new}} = \Delta t_n \cdot \left\| \frac{\hat{I}}{d} \right\|^{-1/(\hat{p}+1)} \].

- We can prove that \( \|e\| \to 0 \) for \( \text{TOL} \to 0 \).

- For \( \text{TOL} \) towards zero, we observe
  \[ \|e\| = \tau \cdot \text{TOL}^\alpha \]
  with \( \alpha < 1 \) method dependent.

- Tolerance scaling (Söderlind, Wang ’06): Rescaling via
  \[ \text{TOL}' = \text{TOL}_0^{(\alpha-1/\alpha)} \cdot \text{TOL}^{1/\alpha} \]
  such that for one value \( \text{TOL}_0 = \text{TOL}' = \text{TOL} \).

- DIRK schemes: \( \alpha = 0.9 \), ROS34PW2: \( \alpha = 0.8 \).
Outline

1. The problem
2. Time integration schemes
3. Specific DG scheme and solver
4. Numerical Results
DG method: Weak form

- Chose grid and element types (teds, quadrilaterals, prisms,...)
- Start with:
  \[
  \tilde{u}_t + \nabla \cdot \tilde{f}(\tilde{u}, \nabla \tilde{u}) = 0.
  \]
- Determine solution using Galerkin condition
  \[
  (\tilde{u}_t, \phi) + (\nabla \cdot \tilde{f}, \phi) = 0
  \]
- Integration by parts:
  \[
  (\tilde{u}_t, \phi) + \int_{\partial \Omega} \tilde{f} \cdot n dS - (\tilde{f}, \nabla \phi) = 0
  \]
- Transform every cell to unit cell $E$:
  \[
  (u_t, \phi) + \int_{\partial E} f \cdot n dS - (f, \nabla \phi) = 0
  \]
DG: Polynomial Ansatz

- Use polynomials of degree $p - 1$ for trial and solution space:
  \[ u^P(x, t) = \sum_{j} u_j(t) \phi_j(x). \]

- Two types of basis for polynomial space possible
  1. Lagrange-type, thus based on nodes (nodal)
  2. Monomial-type, thus based on monomials (modal)

- Solution at cell boundary discontinuous by construction.

- Approximate boundary integrals using numerical fluxes

- Takes ideas from FE and FV world
Computation of the fluxes

- Convective terms using HLLC.
- Diffusive terms more difficult.
  - Standard averaging procedure from FVM unstable for DG.
  - Use dGRP (Gassner, Lörcher, Munz 06).
- Use Gaussian quadrature for scalar products, obtain

\[ \mathbf{M} \mathbf{u}' + \sum_{i=1}^{n\text{Faces}} \mathbf{M}_i^S \mathbf{g}_i - \sum_{k=1}^{d} \mathbf{S}_k f_k = 0. \]

- Matrices and vectors depend on basis, quadrature rule, grid and fluxes.
Polymorphic Modal-Nodal method

- Here: Polymorphic modal method with nodal integration (Gassner, Lörcher, Munz, Hesthaven 09)
- Unstructured cells with curved boundaries (Tets, Quads,...).
- Hierarchical orthonormalized monomial basis on reference cells
  \[ x_1^0 x_2^0 x_3^0, \quad x_1^1 x_2^0 x_3^0, \quad x_1^0 x_2^1 x_3^0, \quad x_1^0 x_2^0 x_3^1, \quad \ldots \]
- Use different (nodal) basis for integration for fast scheme!
- Convective terms using HLLC.
- Diffusive terms with dGRP (Gassner, Lörcher, Munz 06) (small stencil).
Newton and Multigrid

Steady States
- Multigrid fast
- Newton slow

Unsteady problems
- Here we expect significantly faster convergence
- Multigrid marginally, if at all, faster, not fast on unstructured grids
- Newton significantly faster than for steady state

Still: Existing solvers for unsteady problems not fast!
Improving the existing solvers

Multigrid

- Dual time stepping typically implies reusing the steady state algorithm without changes
- Thus less than optimal multigrid convergence
- Redesign multigrid (see B., Optimizing Runge-Kutta smoothers for unsteady flow problems, ETNA, 2012)

Newton

- Very good scheme: Jacobian-Free Newton-GMRES with good parallel preconditioner
- Multigrid candidate for preconditioner (see above)

Inexact Jacobian-free Newton-Krylov method

Solve nonlinear systems using Newton’s method.

- Iterate until relative TOL/5 satisfied.
- Linear equation systems solved using GMRES.
- Tolerances in GMRES by forcing terms of Eisenstat/Walker.
- GMRES does not need Jacobian, only matrix vector products.
- Approximate $\frac{\partial F}{\partial u} q$ in GMRES by finite differences:

$$Aq = \frac{\partial F(u^{(k)})}{\partial u} q \approx \frac{F(u^{(k)} + \epsilon q) - F(u^{(k)})}{\epsilon}.$$ 

- Immense flexibility
- Method is quadratically convergent in large radius!
Feedback-Loops for DIRK scheme

Given error tolerances $\tau$, initial time $t_0$ and time step size $\Delta t_0$

- For $i = 1, \ldots, s$
  - For $k = 0, 1, \ldots$ until termination criterion with tolerance $\tau/5$ is satisfied or MAX_NEWTON_ITER has been reached
    - Determine Eisenstat-Walker relative tolerance
    - Solve linear system using preconditioned GMRES

- If MAX_NEWTON_ITER has been reached, but the tolerance test has not been passed, repeat time step with $\Delta t_n = \Delta t_n/4$

- Estimate local error and compute new time step size $\Delta t_{n+1}$

- $t_{n+1} = t_n + \Delta t_n$

Note: Puts additional bound on time step via nonlinear solver
The linear system: 2D is not 3D, FV is not DG!

- System matrix: \( A = \left[ \frac{1}{\Delta t} - \frac{\partial \hat{f}(u)}{\partial u} \right]|_{u(k)} \).
- Sparse, nonnormal, not diagonally dominant and ill conditioned for reasonable \( \Delta t \).
- Generally unstructured, but symmetric block-sparsity pattern.
- In FVM context: block sizes of \( 5 \times 5 \) in 3D.
- Here: Block sizes depend on degree \( N \) and dimension:
  \((d + 2) \cdot (N + d)!/(N!d!)
- This makes the design of efficient implicit DG schemes in 3D so difficult, since blocks are huge! Degree 5: \( 280 \times 280 \!
- Worse for DG-SEM case \((d + 2) \cdot (N + 1)^d\), degree 5: \( 1080 \times 1080 \)
Preconditioning

- Use right preconditioning, since residual is unchanged (inexact JFNK)
  \[ APy = b, \quad y = P^{-1} \Delta u \]

- In a JFNK scheme, we do not compute the matrix a priori.
- Need to compute all parts of matrix needed for preconditioner.
- Interesting: Two-level ILU from Persson, Peraire 08. Uses a coarse scale correction via Jacobi (ILU-CSC)
- Jacobi:
  \[ P = D^{-1} \]
  Parallel, low storage, not very accurate
- Gauß-Seidel:
  \[ P = (D + L)D^{-1}(D + U) \]
  Sequential, stores full matrix, accurate
Alternative: ROBO-SGS!

- Common in FV: Compute first order matrix only
- Here: Make use of hierarchical basis
- **Compute offdiagonal blocks of lower order only in SGS!**
- Reduced Off-diagonal-Block-Order-SGS
- Saves setup time and application cost!
- In parallel, no communication between blocks for preconditioner.
- Birken, Gassner, Haas, Munz, JCP 2013
- Note: Idea works for any high order method employing a hierarchical basis
ROBO-SGS: Off-Block-Sparsity patterns

(a) $p = 0$ variant        (b) $p = 1$ variant

Figure: Reduced versions of the off-block Jacobians, $p = 0$ and $p = 1$ variants
Outline

1. The problem
2. Time integration schemes
3. Specific DG scheme and solver
4. Numerical Results
Flow around a cylinder, Ma=0.3, Re=1000

Figure: Initial (top) and final (bottom) velocity magnitude for cylinder problem. 10,400 hexahedral cells, order 6, 2,912,000 unknowns.
Comparison of preconditioners

<table>
<thead>
<tr>
<th>Preconditioner</th>
<th>Iter.</th>
<th>CPU [s]</th>
<th>Comparison to Jacobi [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>No preconditioner</td>
<td>8,797</td>
<td>2,194</td>
<td>36.0</td>
</tr>
<tr>
<td>Jacobi</td>
<td>3,712</td>
<td>1,613</td>
<td>0.0</td>
</tr>
<tr>
<td>ROBO-SGS-0</td>
<td>3,338</td>
<td>1,538</td>
<td>-4.6</td>
</tr>
<tr>
<td>ROBO-SGS-1</td>
<td>2,824</td>
<td>1,429</td>
<td>-11.4</td>
</tr>
<tr>
<td>ROBO-SGS-2</td>
<td>2,656</td>
<td>1,485</td>
<td>-7.9</td>
</tr>
<tr>
<td>ROBO-SGS-3</td>
<td>2,641</td>
<td>1,679</td>
<td>4.1</td>
</tr>
<tr>
<td>ROBO-SGS-4</td>
<td>2,645</td>
<td>1,989</td>
<td>23.3</td>
</tr>
<tr>
<td>ROBO-SGS-5</td>
<td>2,640</td>
<td>2,427</td>
<td>50.5</td>
</tr>
<tr>
<td>ILU(0)</td>
<td>2,641</td>
<td>2,467</td>
<td>52.9</td>
</tr>
<tr>
<td>ILU(0)-CSC</td>
<td>2,640</td>
<td>2,994</td>
<td>85.6</td>
</tr>
</tbody>
</table>

Table: Computations on 64 cores of the CRAY XE6 cluster *Hermit*.
### Parallel scaling

**Table:** Parallel scaling for cylinder test case on Cray Hermit.

<table>
<thead>
<tr>
<th>Cores</th>
<th>Unkn./core</th>
<th>Iter.</th>
<th>Time</th>
<th>Scaling</th>
<th>Iter.</th>
<th>Time</th>
<th>Scalling</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>45,500</td>
<td>2,824</td>
<td>1,429</td>
<td>-</td>
<td>3,712</td>
<td>1,613</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>22,750</td>
<td>2,926</td>
<td>750</td>
<td>95%</td>
<td>3,712</td>
<td>833</td>
<td>97%</td>
</tr>
<tr>
<td>256</td>
<td>11,375</td>
<td>3,031</td>
<td>395</td>
<td>90%</td>
<td>3,712</td>
<td>432</td>
<td>93%</td>
</tr>
<tr>
<td>512</td>
<td>5,688</td>
<td>3,479</td>
<td>230</td>
<td>78%</td>
<td>3,712</td>
<td>231</td>
<td>87%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cores</th>
<th>Unkn./core</th>
<th>Iter.</th>
<th>Time</th>
<th>Scaling</th>
<th>Iter.</th>
<th>Time</th>
<th>Scalling</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>45,500</td>
<td>2,641</td>
<td>2,467</td>
<td>-</td>
<td>2,640</td>
<td>2,994</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>22,750</td>
<td>2,641</td>
<td>1,196</td>
<td>103%</td>
<td>2,640</td>
<td>1,477</td>
<td>101%</td>
</tr>
<tr>
<td>256</td>
<td>11,375</td>
<td>2,647</td>
<td>576</td>
<td>107%</td>
<td>2,640</td>
<td>728</td>
<td>103%</td>
</tr>
<tr>
<td>512</td>
<td>5,688</td>
<td>2,713</td>
<td>287</td>
<td>107%</td>
<td>2,679</td>
<td>383</td>
<td>98%</td>
</tr>
</tbody>
</table>

- Scaling very good even for very small number of unknowns per core
- Iteration number almost constant
Flow around sphere, Re=1000, Ma=0.3

Figure: Isosurfaces of $\lambda_2 = -10^{-4}$ for $t = 30s$. Grid has 21,128 elements, 739,480 unknowns, $TOL = 10^{-3}$
Comparison of time integration schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Iter.</th>
<th>CPU in s</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSERK4</td>
<td>-</td>
<td>346,745</td>
</tr>
<tr>
<td>RKCK</td>
<td>-</td>
<td>80,889</td>
</tr>
<tr>
<td>SDIRK3</td>
<td>22,223</td>
<td>110,844</td>
</tr>
<tr>
<td>ESDIRK3</td>
<td>14,724</td>
<td>73,798</td>
</tr>
<tr>
<td>ESDIRK4</td>
<td>14,639</td>
<td>66,051</td>
</tr>
<tr>
<td>ROS34PW2</td>
<td>60,449</td>
<td>239,869</td>
</tr>
</tbody>
</table>

Table: Efficiency for flow around sphere, ROBO-SGS-1.
Implicit modal DG scheme for unsteady 3D viscous flows.

FV is not DG! 2D is not 3D!

Solver: JFNK with ROBO-SGS preconditioner.

ROBO-SGS interesting alternative to Jacobi for moderate degree of parallelism

Use tolerance scaling to make time integration schemes comparable

LSERK4 and ROS34PW2 not competitive

RKCK and ESDIRK schemes winners

This test case: ESDIRK4 fastest.