Interior penalty tensor-product preconditioners for high-order discontinuous Galerkin discretizations

Will Pazner & Per-Olof Persson
January 10, 2018
wp@berkeley.edu

Brown University & UC Berkeley
2018 AIAA SciTech Forum
Kissimmee, FL
Motivation

• Need higher fidelity predictions in computational mechanics
  • *e.g.* Turbulent flows, wave propagation, multiscale phenomena, non-linear interactions
• Goal: Develop **robust**, **efficient**, and **accurate** high-order methods based on fully unstructured meshes

*Example:* LES of turbulent flow over an airfoil
High-order methods are less dissipative and are able to resolve small-scale features
Spatial discretization

- DG discretization
- Second-order PDE in conservation form

\[
\frac{\partial u}{\partial t} + \nabla \cdot (F_i(u) + F_v(u, \nabla u)) = f
\]

- e.g. convection-diffusion, Navier-Stokes
- Nodal (tensor-product) basis functions
- Curved elements via isoparametric mapping
- Second-order terms: symmetric interior penalty method (more on this choice later)
Method-of-lines approach, solve the system of ODEs

\[
M \frac{\partial u}{\partial t} = r(u)
\]

- Implicit methods can prove beneficial because of restrictive CFL
  - Boundary layers and viscous effects
  - Low Mach numbers
  - Highly anisotropic elements
Time integration

- Additionally, polynomial degree $p$ gives (approximately):

$$\Delta t \leq C \frac{h}{p^2}$$

Cf. Krivodonova, Qin (2013)

- For diffusion-dominated, more restrictive CFL:

$$\Delta t \leq C \frac{h^2}{p^4}$$

Conclusion

Explicit time integration impractical for boundary layer meshes with very high polynomial degree $p$
Tensor-product structure

- Quadrilateral/hexahedral mesh
- Tensor-product nodal basis functions

\[ \Phi_{ij}(x, y) = \phi_i(x)\phi_j(y), \quad \Phi_{ijk}(x, y, z) = \phi_i(x)\phi_j(y)\phi_k(z) \]

- \((p + 1)^d\) functions per element
- Apply sum-factorization for faster residual evaluations, matrix-vector products
  - For example, 2D evaluation:

\[
\sum_{ij} u_{ij} \Phi_{ij}(x_\alpha, x_\beta) = \sum_i \sum_j u_{ij} \phi_i(x_\alpha)\phi_j(x_\beta)
\]

Cf. Orszag (1980)
Tensor-product structure

- Quadrilateral/hexahedral mesh
- Tensor-product nodal basis functions
  \[ \Phi_{ij}(x, y) = \phi_i(x)\phi_j(y), \quad \Phi_{ijk}(x, y, z) = \phi_i(x)\phi_j(y)\phi_k(z) \]
- \((p+1)^d\) functions per element
- Apply sum-factorization for faster residual evaluations, matrix-vector products
  - For example, 2D evaluation:  
    \[
    \sum_{ij} u_{ij} \Phi_{ij}(x_\alpha, x_\beta) = \sum_i \sum_j u_{ij} \phi_i(x_\alpha)\phi_j(x_\beta) \\
    = \sum_i \phi_i(x_\alpha) \sum_j u_{ij}\phi_j(x_\beta)
    \]  
    Cf. Orszag (1980)
Tensor-product structure

- Quadrilateral/hexahedral mesh
- Tensor-product nodal basis functions

\[ \Phi_{ij}(x, y) = \phi_i(x)\phi_j(y), \quad \Phi_{ijk}(x, y, z) = \phi_i(x)\phi_j(y)\phi_k(z) \]

- \((p + 1)^d\) functions per element
- Apply sum-factorization for faster residual evaluations, matrix-vector products

  - For example, 2D evaluation:

\[
\sum_{ij} u_{ij} \Phi_{ij}(x_\alpha, x_\beta) = \sum_i \sum_j u_{ij} \phi_i(x_\alpha)\phi_j(x_\beta)
\]

\[
= \sum_i \phi_i(x_\alpha) \sum_j u_{ij} \phi_j(x_\beta)
\]

\[\implies\text{residual evaluation in } \mathcal{O}(p^{d+1})\text{ complexity}\]
Explicit time integration:

- Residual evaluation: $O(p^{d+1})$
- Linear solve with mass matrix:
  - For straight-sided meshes, $O(p^{d+1})$ using direct solvers
  - For isoparametric meshes, efficiently solved using preconditioned CG iterations
- Overall: linear complexity in $p$ per degree of freedom
Computational complexities: implicit

Implicit time integration:

• Jacobian matrix: $\mathcal{O}(p^{2d})$ unknowns
• Matrix-vector products: $\mathcal{O}(p^{2d})$
• Element-wise direct solvers: $\mathcal{O}(p^{3d})$

Goal

Implicit time integration with *linear complexity in $p$ per DOF*, i.e. $\mathcal{O}(p^{d+1})$
Matrix-free implicit solvers

• Cannot explicitly form Jacobian matrices
• Instead, use matrix-free computation of matrix-vector products
• Sum factorize to obtain linear complexity per DOF

\[
\left( \frac{\partial f_{ij}}{\partial u_{k\ell}} \right) v_{k\ell} = \sum_{k=1}^{p+1} \sum_{\ell=1}^{p+1} \sum_{\alpha=1}^{\mu} \sum_{\beta=1}^{\mu} w_{\alpha} w_{\beta} \phi_k(x_{\alpha}) \phi_\ell(x_{\beta}) \frac{\partial F}{\partial u}(x_{\alpha}, x_{\beta}) \cdot \nabla \left( \phi_i(x_{\alpha}) \phi_j(x_{\beta}) \right) v_{k\ell}
\]

\[
- \sum_{k=1}^{p+1} \sum_{\ell=1}^{p+1} \sum_{\alpha=1}^{\mu} \sum_{\ell=1}^{p+1} \sum_{e \in \partial K} w_{\alpha} \phi_k(x_{\alpha}) \phi_\ell(y_{\alpha}) \frac{\partial \hat{F}}{\partial u_{-}}(x_{\alpha}, y_{\alpha}) \phi_i(x_{\alpha}) \phi_j(y_{\alpha}) v_{k\ell}
\]
Matrix-free implicit solvers

- Cannot explicitly form Jacobian matrices
- Instead, use matrix-free computation of matrix-vector products
- Sum factorize to obtain linear complexity per DOF

\[
\left( \frac{\partial f_{ij}}{\partial u_{k\ell}} \right) v_{k\ell} = \sum_{k=1}^{p+1} \sum_{\ell=1}^{p+1} \sum_{\alpha=1}^{\mu} \sum_{\beta=1}^{\mu} w_{\alpha} w_{\beta} \phi_k(x_{\alpha}) \phi_{\ell}(x_{\beta}) \frac{\partial F}{\partial u} (x_{\alpha}, x_{\beta}) \cdot \nabla (\phi_i(x_{\alpha}) \phi_j(x_{\beta})) v_{k\ell}
\]

\[
- \sum_{k=1}^{p+1} \sum_{\ell=1}^{p+1} \sum_{e \in \partial K} \sum_{\alpha=1}^{\mu} w_{\alpha} \phi_k(x_{\alpha}^e) \phi_{\ell}(y_{\alpha}^e) \frac{\partial \hat{F}}{\partial u} (x_{\alpha}^e, y_{\alpha}^e) \phi_i(x_{\alpha}^e) \phi_j(y_{\alpha}^e) v_{k\ell}
\]

\[
= \sum_{\alpha=1}^{\mu} w_{\alpha} \phi_i'(x_{\alpha}) \sum_{\beta=1}^{\mu} w_{\beta} \frac{\partial F_1}{\partial u} (x_{\alpha}, x_{\beta}) \phi_j(x_{\beta}) \sum_{\ell=1}^{p+1} \phi_{\ell}(x_{\beta}) \sum_{k=1}^{p+1} \phi_k(x_{\alpha}) v_{k\ell}
\]

\[
+ \sum_{\alpha=1}^{\mu} w_{\alpha} \phi_i(x_{\alpha}) \sum_{\beta=1}^{\mu} w_{\beta} \frac{\partial F_2}{\partial u} (x_{\alpha}, x_{\beta}) \phi_j'(x_{\beta}) \sum_{\ell=1}^{p+1} \phi_{\ell}(x_{\beta}) \sum_{k=1}^{p+1} \phi_k(x_{\alpha}) v_{k\ell}
\]

\[
+ \sum_{e \in \partial K} \sum_{\alpha=1}^{\mu} \frac{\partial \hat{F}}{\partial u} (x_{\alpha}^e, y_{\alpha}^e) \phi_i(x_{\alpha}^e) \phi_j(y_{\alpha}^e) \sum_{\ell=1}^{p+1} \phi_{\ell}(y_{\alpha}^e) \sum_{k=1}^{p+1} \phi_k(x_{\alpha}^e) v_{k\ell},
\]
Preconditioning strategies

• Solve the systems using Newton-Krylov iterations
• Preconditioning is essential for performance
• Standard methods:
  • Block Jacobi: $J \approx \text{blockdiag}(J)$
  • Block ILU: $J \approx \tilde{L}\tilde{U}$
• Parallelize through domain decomposition approach
• Combine with $p$-multigrid for improved performance
Tensor-product preconditioners

- Standard preconditioners require block inverses
  \[ \Rightarrow \text{Need explicit representation of the matrix} \]
- Find **optimal** approximations to block Jacobi preconditioner with tensor-product structure:

  2D: \[ P = A_1 \otimes B_1 + A_2 \otimes B_2 \]

  3D: \[ P = A_1 \otimes B_1 \otimes C_1 + A_1 \otimes B_2 \otimes C_2 \]

**Three key features**

1. Efficient to form via “KSVD” and Lanczos iterations
2. Fast inverses via Schur factorization
3. Exact representation for certain class of equations
Nearest Kronecker problem

Finding $A_1, B_1, A_2, B_2$ such that

$$A \approx A_1 \otimes B_1 + A_2 \otimes B_2$$

is known as the nearest Kronecker problem (NKP)

Optimal solution (in Frobenius norm) is given by Kronecker-product singular value decomposition (KSVD)

Low tensor-rank approximation of $A$ is equivalent to low-rank approximation of $\tilde{A}$, where

$$\tilde{A} = \begin{pmatrix} \tilde{A}_1 \\ \tilde{A}_2 \\ \vdots \\ \tilde{A}_{n_1} \end{pmatrix}, \text{ and } \tilde{A}_j \text{ is a block of rows given by } \tilde{A}_j = \begin{pmatrix} \text{vec}(A_{1,j})^T \\ \text{vec}(A_{2,j})^T \\ \vdots \\ \text{vec}(A_{m_1,j})^T \end{pmatrix}$$
Fast inverses via Schur factorization

• Similar to “matrix diagonalization” (spectral methods)
• Transform the system

\[(A_1 \otimes B_1 + A_2 \otimes B_2)x = b\]

\[(A_2^{-1}A_1 \otimes I + I \otimes B_2^{-1}B_2)x = (A_2^{-1} \otimes B_2^{-1})b\]

\[(Q_1 \otimes Q_2)(T_1 \otimes I + I \otimes T_2)(Q_1^T \otimes Q_2^T)x = (A_2^{-1} \otimes B_2^{-1})b\]

using Schur factorizations

\[A_2^{-1}A_1 = Q_1 T_1 Q_1^T, \quad B_2^{-1}B_2 = Q_2 T_2 Q_2^T\]

• (Similar approach in 3D)
Incorporation of second-order terms

Introduce discrete gradient $\sigma_h \approx \nabla u_h$

$$\sigma_h = \nabla (u_h) + r([u_h]) + \ell(\{u_h - \hat{u}\}).$$

for lifting operators:

$$\int_\Omega r(q) \cdot \tau \, dx = - \int_\Gamma q \cdot \{\tau\} \, ds,$$

$$\int_\Omega \ell(v) \cdot \tau \, dx = - \int_{\Gamma \setminus \partial\Omega} v[\tau] \, ds$$

Lifting operators equivalent to local inverse mass matrix

$\implies$ **not** amenable to sum factorization
Incorporation of second-order terms: primal form

- Locally eliminate the discrete gradient $\sigma_h$ element-wise to obtain the **primal form**
- Bilinear form given by

$$B(u_h, v_h) = \int_\Omega \nabla (u_h) \cdot \nabla v_h \, dx - \int_\Gamma [u_h] \cdot \{\nabla v_h\} \, ds$$

$$- \int_{\Gamma \setminus \partial \Omega} \{u_h - \hat{u}\} [\nabla v_h] \, ds - \int_\Gamma \hat{\sigma} \cdot [v_h] \, ds.$$

- LDG, BR2, SIPDG, CDG, ... methods determined by choice of numerical flux functions: $\hat{u}$ and $\hat{\sigma}$
- If $\hat{\sigma}$ depends on $\sigma$, must compute lifting operators: expensive, not amenable to sum factorization
The IP method sets fluxes

\[ \hat{u} = \{ u_h \}, \quad \hat{\sigma} = \{ \nabla u_h \} - \eta_e [ u_h ] \]

\( \hat{\sigma} \) has no dependence on \( \sigma_h \) \( \implies \) no need to calculate lifting operators

Primal form for general equations (with \( F_v(u, \nabla u) = G_{ij} \nabla u \))

\[
\int_{\Omega} \partial_t u_h \cdot v_h \, dx - \int_{\Omega} F_i(u_h) : \nabla v_h \, dx + \int_{\Gamma} \hat{F}_i(u_h^+, u_h^-) : [v_h] \, ds
\]

\[
- \int_{\Omega} \sum_{j=1}^{d} G_{ij}(u_h) \frac{\partial u_h}{\partial x_j} : \nabla v_h \, dx + \int_{\Gamma} \left\{ \sum_{j=1}^{d} G_{ij}(u_h) \frac{\partial u_h}{\partial x_j} \right\} : [v_h] \, ds
\]

\[
+ \int_{\Gamma} \eta [ u_h ] : [ v_h ] \, ds + \int_{\Gamma} [ u_h ] : \left\{ \sum_{i=1}^{d} G^T_{ij}(u_h) \frac{\partial v_h}{\partial x_i} \right\} \, ds = \int_{\Omega} f \cdot v_h
\]
Exact representations

Constant coefficient convection diffusion equation

\[ u_t + \nabla \cdot (\beta u) - \epsilon \Delta u = 0 \]

Cartesian mesh:

\[ G^T W G \otimes \left( -D^T W D - \eta_e G_0^T G_0 - \eta_e G_1^T G_1 - D_0^T G_0 - G_0^T D_0 + D_1^T G_1 + G_1^T D_1 \right) \]

+ \left( -D^T W D - \eta_e G_0^T G_0 - \eta_e G_1^T G_1 - D_0^T G_0 - G_0^T D_0 + D_1^T G_1 + G_1^T D_1 \right) \otimes G^T W G 

Constant velocity field:
Results: convection diffusion test case 1

Test case 1 (Mackenzie-Morton)

Prescribed velocity field $\mathbf{\beta}(x, y) = (2y(1 - x^2), -2x(1 - y^2))$

Diffusion coefficients: $\epsilon = 10^{-6}, 2 \times 10^{-3}, 10^{-2}, \text{ and } 10^{-1}$.

Quantity of interest: outflow profile
Results: convection diffusion test case 1

\[
\epsilon = 10^{-6}
\]

\[
\epsilon = 2 \times 10^{-3}
\]

\[
\epsilon = 10^{-1}
\]

\[
\epsilon = 10^{-1}
\]
Results: convection diffusion test case 1

\[ p = 1, \quad \epsilon = 10^{-6} \]
Results: convection diffusion test case 1

\[ p = 4, \quad \epsilon = 10^{-6} \]
Results: convection diffusion test case 1

\[ p = 9, \quad \epsilon = 10^{-6} \]
Results: convection diffusion test case 1

\[ p = 1, \quad \epsilon = 2 \times 10^{-3} \]

\begin{figure}
\centering
\includegraphics[width=\textwidth]{chart.png}
\caption{Comparison of iterations for different methods with varying \( \Delta t \).}
\end{figure}
Results: convection diffusion test case 1

\[ p = 4, \quad \epsilon = 2 \times 10^{-3} \]
Results: convection diffusion test case 1

\[ p = 9, \quad \epsilon = 2 \times 10^{-3} \]
Results: convection diffusion test case 1

\( p = 1, \quad \epsilon = 10^{-2} \)

- \( \Delta t \) vs. Iterations for:
  - Jacobi
  - KSVD-IP
  - KSVD
Results: convection diffusion test case 1

\[ p = 4, \quad \epsilon = 10^{-2} \]
Results: convection diffusion test case 1

\[ p = 9, \quad \epsilon = 10^{-2} \]

![](graph.png)
Results: convection diffusion test case 1

\[ p = 1, \quad \epsilon = 10^{-1} \]
Results: convection diffusion test case 1

\[ p = 4, \quad \epsilon = 10^{-1} \]
Results: convection diffusion test case 1

\[ p = 9, \quad \epsilon = 10^{-1} \]
Results: convection diffusion test case 2

\[ g_D = 100 \]

Homogeneous Dirichlet on other boundaries

Largest allowable time step for RK4

<table>
<thead>
<tr>
<th>( \epsilon )</th>
<th>( p = 1 )</th>
<th>( p = 4 )</th>
<th>( p = 9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 \times 10^{-6} )</td>
<td>( 1 \times 10^{-2} )</td>
<td>( 1 \times 10^{-2} )</td>
<td>( 4 \times 10^{-3} )</td>
</tr>
<tr>
<td>( 2 \times 10^{-3} )</td>
<td>( 2 \times 10^{-4} )</td>
<td>( 8 \times 10^{-5} )</td>
<td>( 2 \times 10^{-5} )</td>
</tr>
<tr>
<td>( 1 \times 10^{-2} )</td>
<td>( 6 \times 10^{-5} )</td>
<td>( 1 \times 10^{-5} )</td>
<td>( 4 \times 10^{-6} )</td>
</tr>
<tr>
<td>( 1 \times 10^{-1} )</td>
<td>( 6 \times 10^{-6} )</td>
<td>( 1 \times 10^{-6} )</td>
<td>( 4 \times 10^{-7} )</td>
</tr>
</tbody>
</table>
Results: convection diffusion test case 2

\( p = 1, \quad \epsilon = 10^{-6} \)

\( \Delta t \) vs. Iterations

- Jacobi
- KSVD-IP
- KSVD

\( 10^{-2} \) to \( 10^{-1} \) to \( \infty \)
Results: convection diffusion test case 2

\[ p = 4, \quad \epsilon = 10^{-6} \]
Results: convection diffusion test case 2

\[ p = 9, \quad \epsilon = 10^{-6} \]
Results: convection diffusion test case 2

\[ p = 1, \quad \epsilon = 2 \times 10^{-3} \]
Results: convection diffusion test case 2

$p = 4, \quad \epsilon = 2 \times 10^{-3}$
Results: convection diffusion test case 2

\[ p = 9, \quad \epsilon = 2 \times 10^{-3} \]

\[ \begin{array}{cccc}
\hline
\Delta t & \text{Iterations} \\
\hline
\infty & 320 \\
10^{-2} & 40 \\
10^{-1} & 80 \\
\infty & 160 \\
\hline
\end{array} \]
Results: convection diffusion test case 2

\[ p = 1, \quad \epsilon = 10^{-2} \]
Results: convection diffusion test case 2

\[ p = 4, \quad \epsilon = 10^{-2} \]
Results: convection diffusion test case 2

\[ p = 9, \quad \epsilon = 10^{-2} \]
Results: convection diffusion test case 2

\[ p = 1, \quad \epsilon = 10^{-1} \]

\[
\begin{array}{c|c|c|c|c}
\Delta t & \text{Iterations} & \text{Jacobi} & \text{KSVD-IP} & \text{KSVD} \\
\hline
10^{-2} & 80 & & & \\
10^{-1} & 160 & & & \\
\hline
& & \infty & & \\
\end{array}
\]
Results: convection diffusion test case 2

\[ p = 4, \quad \epsilon = 10^{-1} \]

\[ \begin{array}{c|cccc}
\Delta t & 10^{-2} & 10^{-1} & \infty \\
Jacobi & 80 & 160 & 320 \\
KSVD-IP & 80 & 160 & 320 \\
KSVD & 80 & 160 & 320 \\
\end{array} \]
Results: convection diffusion test case 2

\[ p = 9, \quad \epsilon = 10^{-1} \]
Flow over a circular cylinder

Three cases: $Re = 10$, $Re = 200$, $Re = 1000$

$\Delta t = 10^{-2}, 2 \times 10^{-2}, 4 \times 10^{-2}, \text{and } 8 \times 10^{-2}$

$p = 1, 4, 9$

# DOFS = 6400 / solution component
Results: Navier-Stokes test case

\[ p = 1, \quad Re = 1000 \]
Results: Navier-Stokes test case

\[ p = 4, \quad \text{Re} = 1000 \]
Results: Navier-Stokes test case

$p = 9, \quad \text{Re} = 1000$

\[
\begin{align*}
10^{-2} & \quad 2 \times 10^{-2} & \quad 4 \times 10^{-2} & \quad 8 \times 10^{-2} \\
\text{Jacobi} & \quad 1 \times 10^{-2} & \quad 2 \times 10^{-2} & \quad 3 \times 10^{-2} & \quad 4 \times 10^{-2} & \quad 5 \times 10^{-2} \\
\text{KSVD-IP} & \quad 1 \times 10^{-2} & \quad 2 \times 10^{-2} & \quad 3 \times 10^{-2} & \quad 4 \times 10^{-2} & \quad 5 \times 10^{-2} \\
\text{KSVD} & \quad 1 \times 10^{-2} & \quad 2 \times 10^{-2} & \quad 3 \times 10^{-2} & \quad 4 \times 10^{-2} & \quad 5 \times 10^{-2}
\end{align*}
\]
Results: Navier-Stokes test case

\[ p = 1, \quad \text{Re} = 200 \]
Results: Navier-Stokes test case

\( p = 4, \quad \text{Re} = 200 \)

\[
\begin{align*}
\Delta t & \quad 10^{-2} & \quad 2 \times 10^{-2} & \quad 4 \times 10^{-2} & \quad 8 \times 10^{-2} \\
\text{Iterations} & \quad 740 & \quad 320 & \quad 160 & \quad 80
\end{align*}
\]

- Jacobi
- KSVD-IP
- KSVD
Results: Navier-Stokes test case

$p = 9, \quad \text{Re} = 200$

The graph shows the number of iterations required for convergence of different methods as a function of the time step $\Delta t$. The methods compared are Jacobi, KSVD-IP, and KSVD.
Results: Navier-Stokes test case

\[ p = 1, \quad Re = 10 \]

\[ \Delta t \]

Iterations

Jacobi
KSVD-IP
KSVD
Results: Navier-Stokes test case

\[ p = 4, \quad \text{Re} = 10 \]
Results: Navier-Stokes test case

$p = 9, \quad Re = 10$

![Graph showing iterations vs. \(\Delta t\)]

- **Jacobi**
- **KSVD-IP**
- **KSVD**
Conclusions

• Construction of tensor-product approximations to block Jacobi preconditioner
• Efficient formation of preconditioner using a matrix-free Lanczos framework
• $O(p^3)$ complexity for implicit solvers in 2D
• Incorporation of second-order/viscous terms using an interior-penalty method
• Large improvement compared with previous tensor-product preconditioner which neglected viscous terms
• Competitive iteration counts compared with block Jacobi