GPU-accelerated DG methods at high order

Jesse Chan, T. Warburton

1Department of Mathematics
Virginia Tech

March 2, 2016
Rice Oil and Gas HPC 2016
Discretization of time-dependent wave problems

High order methods:

- Accurately represent acoustic and elastic waves.
- Give superior performance for equivalent resolution.
- Yield lower error per unknown.

Goals:

- Flexibility: approximation of solution over complex geometries.
- Accuracy: high order methods (mesh size $h$, error $\propto h^{N+1}$).

Figure courtesy of Axel Modave.
Discretization of time-dependent wave problems

High order methods:

- Accurately represent acoustic and elastic waves.
- Give superior performance for equivalent resolution.
- Yield lower error per unknown.

Goals:

- Flexibility: approximation of solution over complex geometries.
- Accuracy: high order methods (mesh size $h$, error $\propto h^{N+1}$).
High order methods:

- Accurately represent acoustic and elastic waves.
- Give superior performance for equivalent resolution.
- Yield lower error per unknown.

Goals:

- Flexibility: approximation of solution over complex geometries.
- Accuracy: high order methods (mesh size $h$, error $\propto h^{N+1}$).
Discretization of time-dependent wave problems

High order methods:
- Accurately represent acoustic and elastic waves.
- Give superior performance for equivalent resolution.
- Yield lower error per unknown.

Max errors vs. dofs.

Goals:
- Flexibility: approximation of solution over complex geometries.
- Accuracy: high order methods (mesh size $h$, error $\propto h^{N+1}$).
Discretization of time-dependent wave problems

High order methods:

- Accurately represent acoustic and elastic waves.
- Give superior performance for equivalent resolution.
- Yield lower error per unknown.

Goals:

- Flexibility: approximation of solution over complex geometries.
- Accuracy: high order methods (mesh size $h$, error $\propto h^{N+1}$).

Max errors vs. dofs.
Given initial condition $u(x, 0)$:

- Compute numerical flux at face nodes (non-local).
- Compute RHS of (local) ODE.
- Evolve (local) solution using explicit time integration (RK, AB, etc).

\[
\frac{du}{dt} = D_x u + \sum_{\text{faces}} L_f(\text{flux}), \quad L_f = M^{-1}M_f.
\]

Maps very well to GPU architectures!
Given initial condition $u(x, 0)$:

- Compute numerical flux at face nodes (non-local).
- Compute RHS of (local) ODE.
- Evolve (local) solution using explicit time integration (RK, AB, etc).

$$\frac{du}{dt} = D_x u + \sum_{\text{faces}} L_f \text{(flux)},$$

$\text{Volume kernel}$

$\text{Surface kernel}$

$L_f = M^{-1} M_f$.

Maps very well to GPU architectures!
Given initial condition $u(x, 0)$:

- Compute numerical flux at face nodes (non-local).
- Compute RHS of (local) ODE.
- Evolve (local) solution using explicit time integration (RK, AB, etc).

$$\frac{du}{dt} = D_x u + \sum_{\text{faces}} L_f (\text{flux}), \quad L_f = M^{-1} M_f.$$  

Maps very well to GPU architectures!
Given initial condition $u(x, 0)$:

- Compute numerical flux at face nodes (non-local).
- Compute RHS of (local) ODE.
- Evolve (local) solution using explicit time integration (RK, AB, etc).

\[
\frac{du}{dt} = D_x u + \sum_{\text{faces}} L_f (\text{flux}), \quad L_f = M^{-1} M_f.
\]

Maps very well to GPU architectures!
Computational costs at high orders of approximation

Problem: (tetrahedral) DG at high orders becomes very expensive!

Very high orders usually use tensor-product elements.

Spectral element methods often use order \( N > 10 \).

\( O(N^{d+1}) \) vs \( O(N^6) \) cost, but less geometric flexibility.

DG runtimes for 50 timesteps, 98304 elements.

1 DG methods on hybrid meshes

2 High order Bernstein-Bezier DG methods
Outline

1. DG methods on hybrid meshes

2. High order Bernstein-Bezier DG methods
DG methods on hybrid meshes

Hybrid meshes and transitional elements

- Tensor-product hexahedra more efficient than tetrahedra.
- Drawback: hexahedral meshes are less geometrically flexible.
- Solution: use transitional pyramid and wedge elements.

https://www.sharcnet.ca/Software/TGrid/html/tg/node41.htm

Hybrid meshes and transitional elements

Unstructured hex-	extit{dominant} meshes: efficiency + geometric flexibility.

Meshes courtesy of J.F. Remacle (gmsh)
For planar tetrahedra, mapping is affine — can store one reference lift matrix 
\( \hat{L}_f = \hat{M}^{-1}\hat{M}_f \) for all elements.

Unlike tets, hexes, wedges and pyramids are not affinely mapped — requires storage of many lift matrices 
\( \mathbf{L} = \mathbf{M}^{-1}\mathbf{M}_f \).

\[ O(N^3) \] solution storage vs \( O(N^6) \) matrix storage per element — need high order bases with diagonal \( \mathbf{M} \) for hexes, wedges, and pyramids.
High order, low-storage bases for hybrid meshes

- Hexahedra: Lagrange basis at quadrature points.

\[
M_{ij} = \int_{\hat{D}} \ell_j \ell_i J \approx \sum_{k=1}^{N_p} \ell_j(x_k) \ell_i(x_k) w_k J(x_k) = \delta_{ij} w_j J(x_j).
\]

- Wedge: low-storage curvilinear DG (all mass matrices identical). Requires rational basis; stability and high order tricky.

- Low-storage, high order “semi-nodal” basis on mapped pyramids.

---

Warburton 2013, A low-storage curvilinear discontinuous Galerkin method for wave problems (SISC).
Chan and Warburton 2015, Orthogonal bases for non-affine pyramidal finite elements (SISC).
DG methods on hybrid meshes

Verification: hybrid mesh, 3rd order multi-rate scheme

- Different quadrature choices for hexahedra.
- Gauss-Legendre-Lobatto (SEM) quadrature less accurate, more efficient.

Gauss-Legendre (GL) Spectral element (SEM)

<table>
<thead>
<tr>
<th>N</th>
<th>GL</th>
<th>SEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.92</td>
<td>1.94</td>
</tr>
<tr>
<td>2</td>
<td>2.85</td>
<td>2.97</td>
</tr>
<tr>
<td>3</td>
<td>3.92</td>
<td>3.92</td>
</tr>
</tbody>
</table>

Number of elements

Error
- Comparison with tetrahedra - standard DG meshes.
- Exploit tensor product for hexes/wedges, new quadrature-free operations for pyramid.
- Note: hexes more accurate per-dof than tetrahedra!

Optimized costs per-unknown relative to tetrahedra ($\approx 100,000$ elements, Nvidia 980 GTX).
Current work: reverse time migration using hybrid meshes

(a) Simple model
(b) RTM image
(c) Layer interfaces

Imaging of two-layer model, hybrid meshes for complex interfaces.

For more info, see Zheng Wang’s poster: “GPU Accelerated Discontinuous Galerkin Method on Hybrid Meshes: Applications in Seismic Imaging”.

Chan (VT) High order DG March 2, 2016 10 / 16
Outline

1. DG methods on hybrid meshes

2. High order Bernstein-Bezier DG methods
High order nodal DG on tetrahedral meshes

Local ODE given in terms of derivative and lift matrices.

\[ \frac{du}{dt} = D_x u + \sum_{\text{faces}} L_f (\text{flux}), \quad L_f = M^{-1} M_f. \]

- Nodal bases reduce the cost of computing numerical fluxes.
- No special structure in nodal derivative/lift matrices.
- \(O(N^3)\) unknowns in 3D; \(O(N^6)\) costs for applying dense matrices.

Derivative and lift matrices depend on the basis: can we choose one that is efficient (and numerically stable)?
Each function attains its maximum at an equispaced lattice point of a $d$-simplex.

- Simple expression in 1D
  \[ B_i^N(x) = x^i (1 - x)^{N-i}, \quad 0 \leq x \leq 1. \]
- Barycentric monomials on a $d$-simplex. For a tetrahedron,
  \[ B_{ijkl}^N(\lambda_0, \lambda_1, \lambda_2, \lambda_3) = \frac{N!}{i!j!k!l!} \lambda_0^i \lambda_1^j \lambda_2^k \lambda_3^l, \quad i + j + k + l = N. \]
- Similar structure to nodal basis (vertex, edge, face, interior functions).
Bernstein-Bezier derivative and lift matrices in 3D

- Bernstein-Bezier barycentric differentiation matrices very sparse.
- Bernstein-Bezier lift matrix displays some sparsity - can be improved.

(a) Derivative matrix w.r.t. first barycentric coordinate.

(b) Bernstein-Bezier lift matrix
Theorem (Chan, Warburton 2015)

The Bernstein-Bezier lift matrix $L$ admits a factorization of the form

$$L = E_L \begin{pmatrix} L_0 & L_0 \\ L_0 & L_0 \end{pmatrix}.$$
Numerical stability of Bernstein-Bezier DG

- Conditioning of derivative, lift matrices comparable to nodal basis.

\[ \kappa(A) = \frac{\sigma_1}{\sigma_r} \]

- Comparable long-time growth of (single precision) numerical error.

Condition numbers of matrices for nodal and Bernstein-Bezier bases.
Numerical stability of Bernstein-Beziers DG

- Conditioning of derivative, lift matrices comparable to nodal basis.

\[ \kappa(A) = \frac{\sigma_1}{\sigma_r} \]

- Comparable long-time growth of (single precision) numerical error.

Evolution of $L^2$ error (acoustics) for nodal and Bernstein-Bezier bases.
Bernstein-Bezier DG achieves $\approx 2 \times$ speedup at moderate orders, and up to $4 \times$ speedup at high orders.

$$\frac{du}{dt} = D_x u + \sum_{\text{faces}} L_f (\text{flux}), \quad L_f = M^{-1} M_f.$$
Bernstein-Bezier DG achieves $\approx 2\times$ speedup at moderate orders, and up to $4\times$ speedup at high orders.

\[ \frac{du}{dt} = D \times u + \sum_{\text{faces}} L_f(\text{flux}), \quad L_f = M^{-1}M_f. \]
Conclusions

- High order advantageous, but new algorithms required for efficiency.

- Hybrid meshes exploit efficiency of hexahedra at high order.

- Bernstein-Bezier DG: next-gen accelerated time-domain solvers.

Special thanks to TOTAL E&P for their support.
Additional slides
Bernstein-Bezier DG achieves \( \approx 4 \times \) speedup at low-moderate orders, and \( 1.5 - 2 \times \) speedup at high orders.

\[
\frac{\text{d} \mathbf{u}}{\text{d} t} = \underbrace{\mathbf{D} \times \mathbf{u}}_{\text{Volume kernel}} + \underbrace{\sum_{\text{faces}} \mathbf{L}_f \left( \text{flux} \right)}_{\text{Surface kernel}}, \quad \mathbf{L}_f = \mathbf{M}^{-1} \mathbf{M}_f.
\]
Bernstein-Bezizer compared to CUBLAS

Bernstein-Bezier DG achieves $\approx 4 \times$ speedup at low-moderate orders, and $1.5 - 2 \times$ speedup at high orders.

\[
\frac{du}{dt} = D_x u + \sum_{\text{flux}} L_f, \quad L_f = M^{-1} M_f.
\]
Performance comparisons of Bernstein-Bezier DG

![Graphs showing performance comparisons of Bernstein-Bezier DG methods for volume and surface computations at different degrees.](image)

<table>
<thead>
<tr>
<th>Degree</th>
<th>TFLOPS/s (Volume)</th>
<th>TFLOPS/s (Surface)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>7</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>8</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>9</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

High order Bernstein-Bezier DG methods
Performance comparisons of Bernstein-Bezier DG

Bandwidth (Nodal)

Bandwidth (Bernstein)
Arithmetic intensity: floating-point operations per byte of data.

Computational efficiency: ratio of observed/achievable performance.
Roofline model: estimating computational efficiency

- Arithmetic intensity: floating-point operations per byte of data.
- Computational efficiency: ratio of observed/achievable performance.

Williams, Waterman, Patterson 2009. Roofline: an insightful visual performance model for multicore architectures.
Roofline model: estimating computational efficiency

- Arithmetic intensity: floating-point operations per byte of data.
- Computational efficiency: ratio of observed/achievable performance.

\[
\text{Computational efficiency} = \frac{\text{Observed}}{\text{Achievable}}
\]

Williams, Waterman, Patterson 2009. Roofline: an insightful visual performance model for multicore architectures.
Bernstein-Bezier DG: standard implementation, sparse matrices.

\[
\frac{d\mathbf{u}}{dt} = \mathbf{D}_x \mathbf{u} + \sum_{\text{faces}} \mathbf{L}_f(\text{flux}), \quad \mathbf{L}_f = \mathbf{M}^{-1} \mathbf{M}_f.
\]

Efficiency (Standard)

Efficiency (Bernstein)
Bernstein-Bezier DG: standard implementation, sparse matrices.

\[ \frac{du}{dt} = D_xu + \sum_{\text{faces}} L_f(\text{flux}), \quad L_f = M^{-1}M_f. \]

**Performance comparisons of Bernstein-Bezier DG**

Efficiency (Blocked)

Efficiency (Bernstein)
Bernstein-Bezier methods on hybrid meshes

- Hex, wedge bases defined as tensor products of lines, triangle.

- Pyramid basis: defined on cube, mapped using Duffy transform.

\[
B_{ijk}^N(a, b, c) = B_{i}^{N-k}(a) B_{j}^{N-k}(b) B_{k}^{N}(c).
\]

- Preserves Bernstein-Bezier properties (positivity, partition of unity).

Can use Bernstein-Bezier pyramids in 3D exact geometry meshing (ongoing collaboration with CU Boulder).