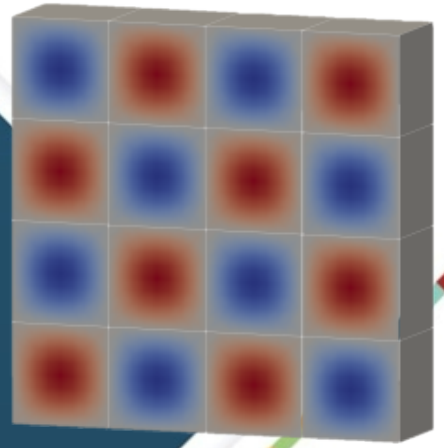
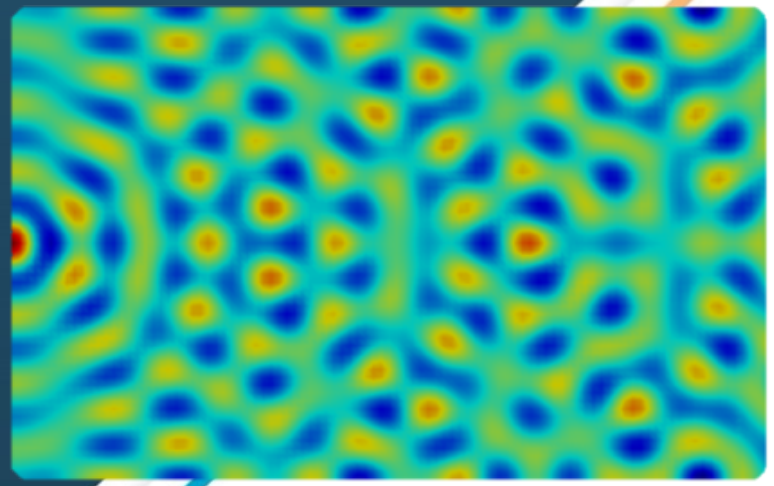
SAND2022-9180C

Sandia National Laboratories

Exceptional service in the national interest



Discretizations of high-frequency wave propagation problems on next-generation computing architectures



Julia A. Plews, Gregory Bunting, Jerry Rouse, Clark Dohrmann

Sandia National Laboratories, Albuquerque, New Mexico, USA

WCCM-APCOM Yokohama 2022

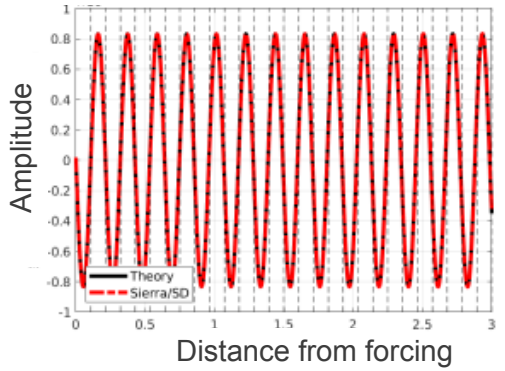
July 31 to August 5, 2022

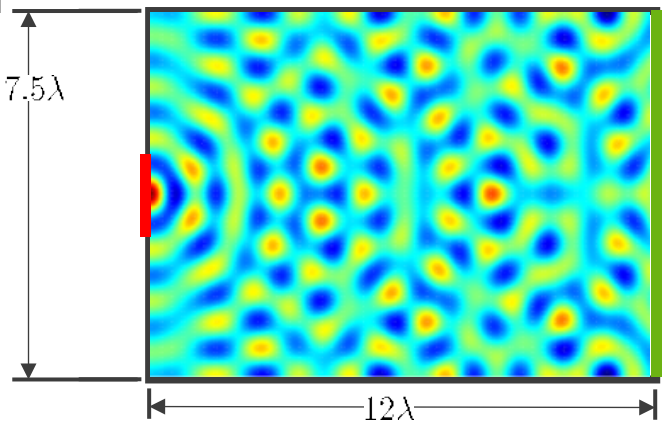
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Accurate modeling and simulation at ultra-high frequencies (UHF)

- **The goal:** *Predict the response of barrier materials to coupled multi-physics phenomena. Our focus? Mechanical wave propagation.*
- **Who cares**: DoD, DoE, engineers, designers, and analysts
- **Our team's objective**: Develop novel simulation approaches to overcome UHF limitations
- **Impacts**:
  - Rapid materials discovery
  - Multi-physics simulations

  } Model-based design & response prediction
- **Why SNL**: Leaders in engineering, ModSim codes, computing capability
  - Sierra: mature, massively parallel simulation codes
  - Trilinos: scalable numerical algorithms
  - HPC: state-of-the-art heterogeneous computing platforms





Modeling high-frequency wave propagation

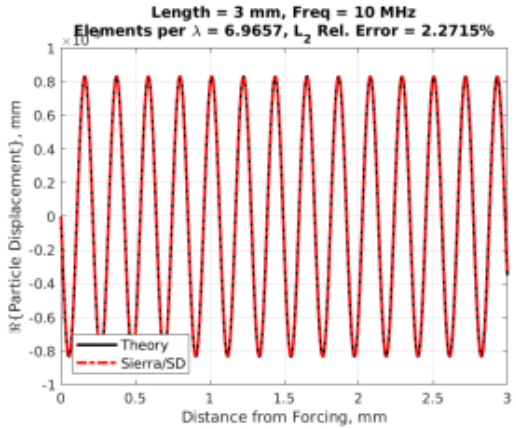
$$\frac{\partial^2 p(x)}{\partial x^2} + k^2 p(x) = 0$$ Helmholtz equation

Wavenumber: $k = \frac{\omega}{c}$

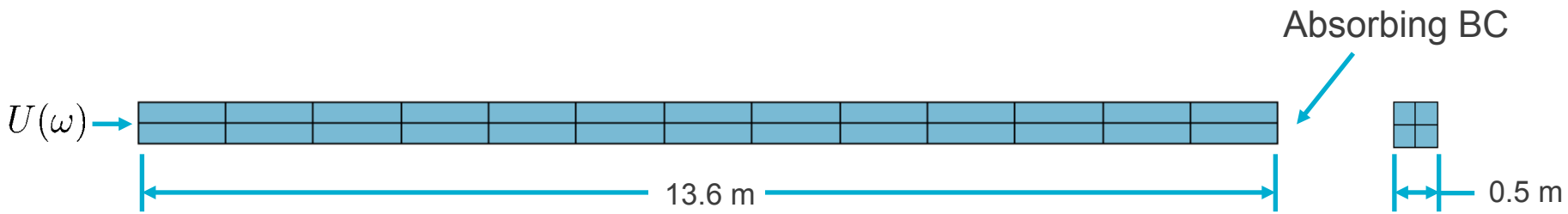
$$\left.\frac{\partial p}{\partial x}\right|_{x=0} = -i\omega\rho_o U$$

$$\left.\frac{\partial p}{\partial x}\right|_{x=1} = -ikp(1)$$

Boundary conditions



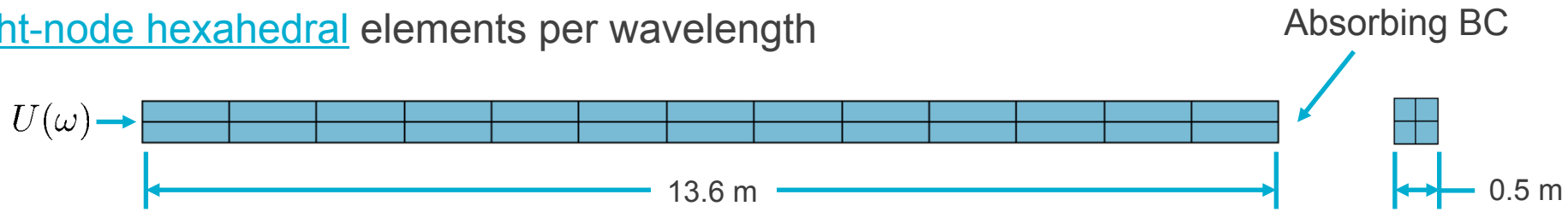
Example: Semi-infinite duct



Rouse, Walsh, ASA 2019

Modeling high-frequency wave propagation

- Excitation frequency: 20 Hz
- 16 eight-node hexahedral elements per wavelength

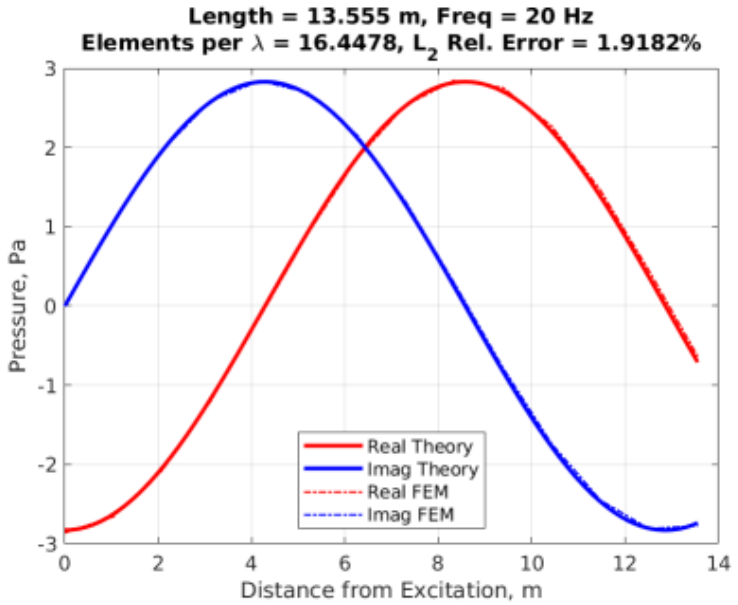


- Analytical solution: $p(z) = \rho_o c U(\omega) e^{-ikz}$

L_2 relative error = 1.9 %

$$\tilde{e}_2 = \frac{||p - p_h||_2}{||p||_2}$$

$$||u||_2 = \left[\int_0^L |u(z)|^2 dz \right]^{1/2}$$

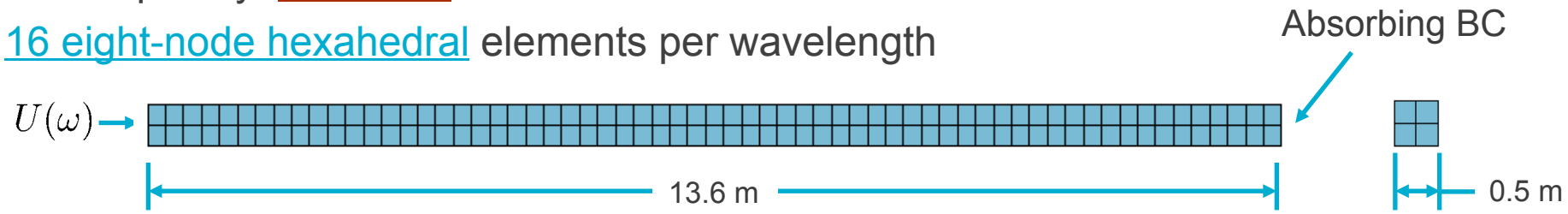


Percent error at end = 2.9 %

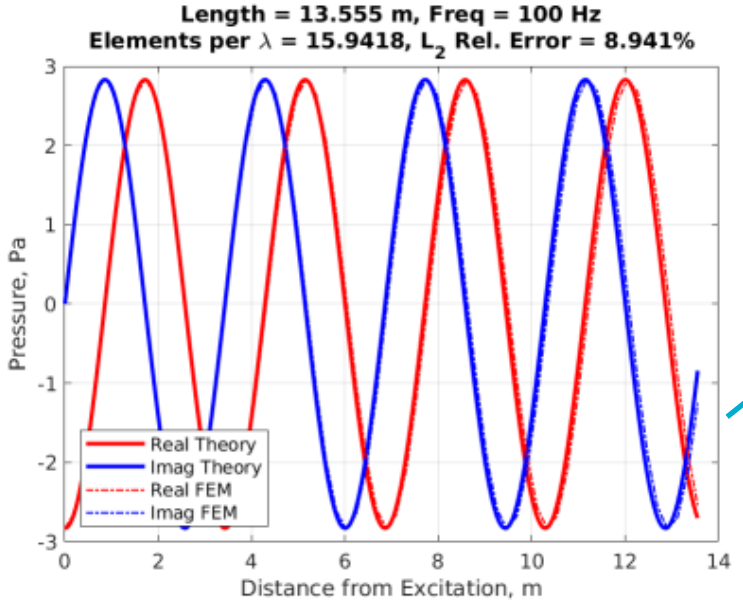
Rouse, Walsh, ASA 2019

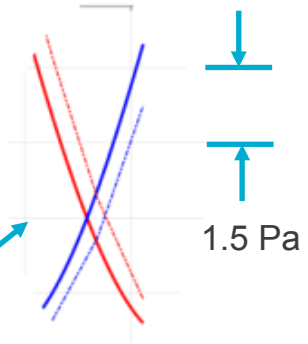
Modeling high-frequency wave propagation

- Excitation frequency: **100 Hz**
- Same 16 eight-node hexahedral elements per wavelength



L_2 relative error = 8.9 %





Error at end = **15.5 %**

Rouse, Walsh, ASA 2019

”Pollution error” creeps in at higher frequencies

Pollution error in high-frequency problems

- Relative error for classical finite element solution to the Helmholtz equation:

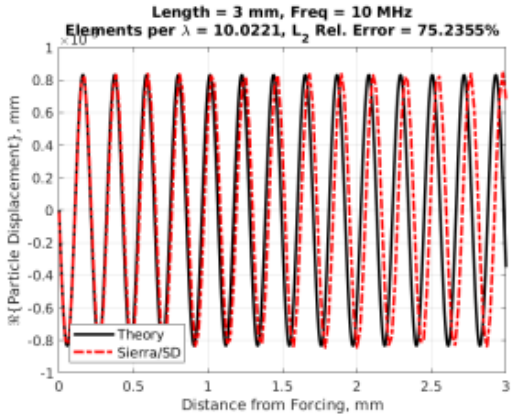
Element size: *h*
Polynomial order: *p*
Structure length: *L*

$$\frac{|u-u_h|_1}{|u|_1} \leq C_1\left(\frac{kh}{p}\right)^p + C_2 Lk\left(\frac{kh}{p}\right)^{2p}, \quad kh < 1$$

Interpolation error: how well shape functions approximate the solution

*Babuska & Sauter, 1997

'Pollution effect':* Accumulation of error over the length of the structure



Addressing pollution is the key to achieving accuracy at UHF

Addressing pollution in high-frequency problems

- Brute force! Domain decomposition, massively parallel **continuous Galerkin** finite elements (CGFE)
  - Pushed as far as possible with classical finite elements and traditional element formulations
  - Sierra Mechanics implementation is the state of the art!
- Higher-order CGFE
  - Going beyond quadratic approximations with *p*-elements drastically improves efficiency
  - Enriched methods
- Others in the literature…
- One that stands out: Discontinuous Petrov-Galerkin
  - Developed by Prof. Demkowicz at UT Austin

Discontinuous Petrov-Galerkin (DPG) method

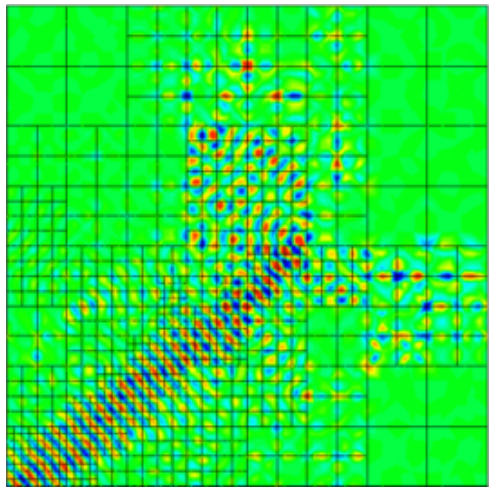
Like classical finite elements, but…

Trial functions are continuous, test functions are discontinuous
- Compute optimal test functions on the fly
- Built-in error indicator

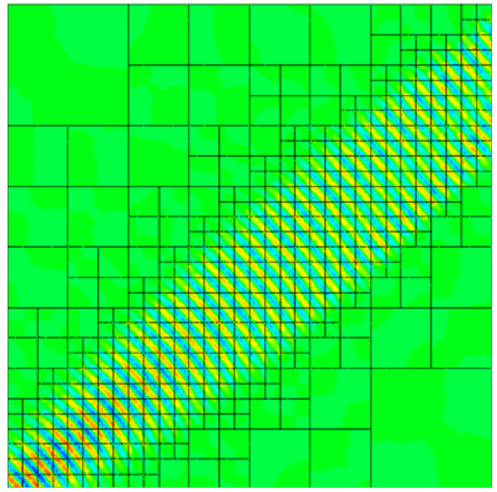
Ultraweak formulation
- Positive definite matrices

Pollution is diffusive rather than dispersive
- Pollution free in 1D

Unconditionally stable
- Allows for hp-adaptivity



CGFE



DPG

L. Demkowicz and J. Gopalakrishnan. *ICES REPORT 15-20, The University of Texas at Austin*, 2015.

J. Zitelli and I. Muga and L. Demkowicz and J. Gopalakrishnan and D. Pardo and V. M. Calo. *Journal of Computational Physics*, 230(7): 2406-2432, 2011.

S. Petrides. *PhD Dissertation, The University of Texas at Austin*, 2019.

Discontinuous Petrov-Galerkin discretizations

- Simplest case: Poisson's equation

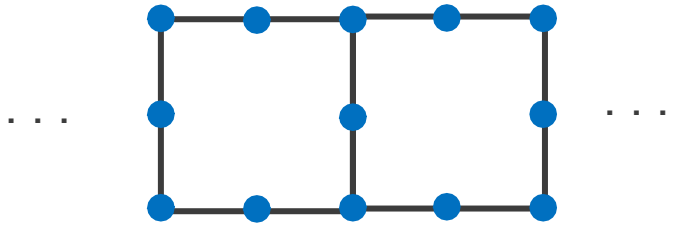
$$\nabla^2 \phi = f$$

$$\vec{\psi} = \vec{\nabla}\phi$$

CGFE formulation:

$$-\int_\Omega \vec{\nabla}\phi \cdot \vec{\nabla} v \, d\Omega + \int_\Gamma v(\vec{\nabla}\phi \cdot \hat{n}) \, d\Gamma = \int_\Omega f v \, d\Omega$$

Expressed for entire domain



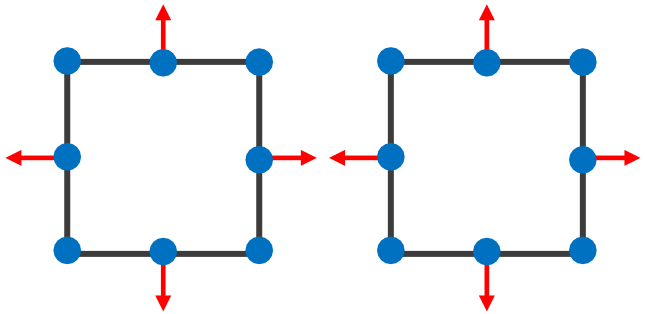
DPG 'ultraweak' formulation:

$$-\int_{\Omega_k} \phi(\vec{\nabla} \cdot \vec{q}) \, d\Omega_k - \int_{\Omega_k} \vec{\psi} \cdot \vec{q} \, d\Omega_k +$$

$$\int_{\Gamma_k} \widehat{\phi}(\vec{q} \cdot \hat{n}) \, d\Gamma_k + \int_{\Gamma_k} \widehat{\psi}_n v \, d\Gamma_k = \int_{\Omega_k} f v \, d\Omega_k$$

Trace — Flux — For each element *k*

DPG is complex and challenging in its implementation!



`mini_dpg`: A toolkit for FE and DPG discretizations

- `mini_dpg` mimics established, high-performance Sierra apps like **Structural Dynamics** and **Solid Mechanics**
  - Three-dimensional, MPI-parallel, heterogeneous (GPU-accelerated)
  - …but with advanced discretizations in mind

- Uses open source tools in the Sandia-developed **Trilinos library**
  - Intrepid2: **high-order finite elements** package
  - STK mesh: **massively parallel data structures** that Sierra is based on
  - Kokkos **GPU-portable** parallelism, batch dense linear algebra, data management
  - Tpetra, Amesos2, Ifpack2, and Belos **sparse, parallel linear solvers**
    - Also, *p*-preconditioning strategies, work by Clark Dohrmann et al.

Leveraging Sandia-developed, open-source tools allowed rapid prototyping and a strong “focus on the math”

Numerical results: Verification

p-convergence of DPG vs. CGFE formulation of Poisson's equation

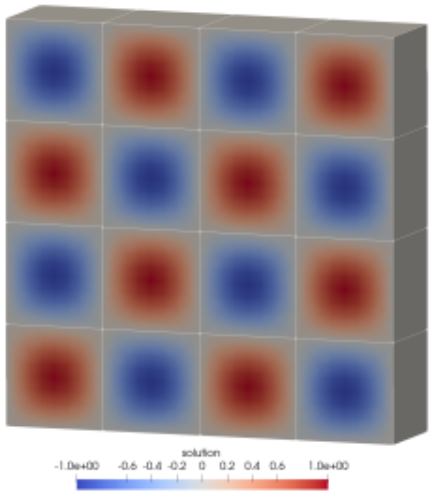
- 4x4x1 hex grid

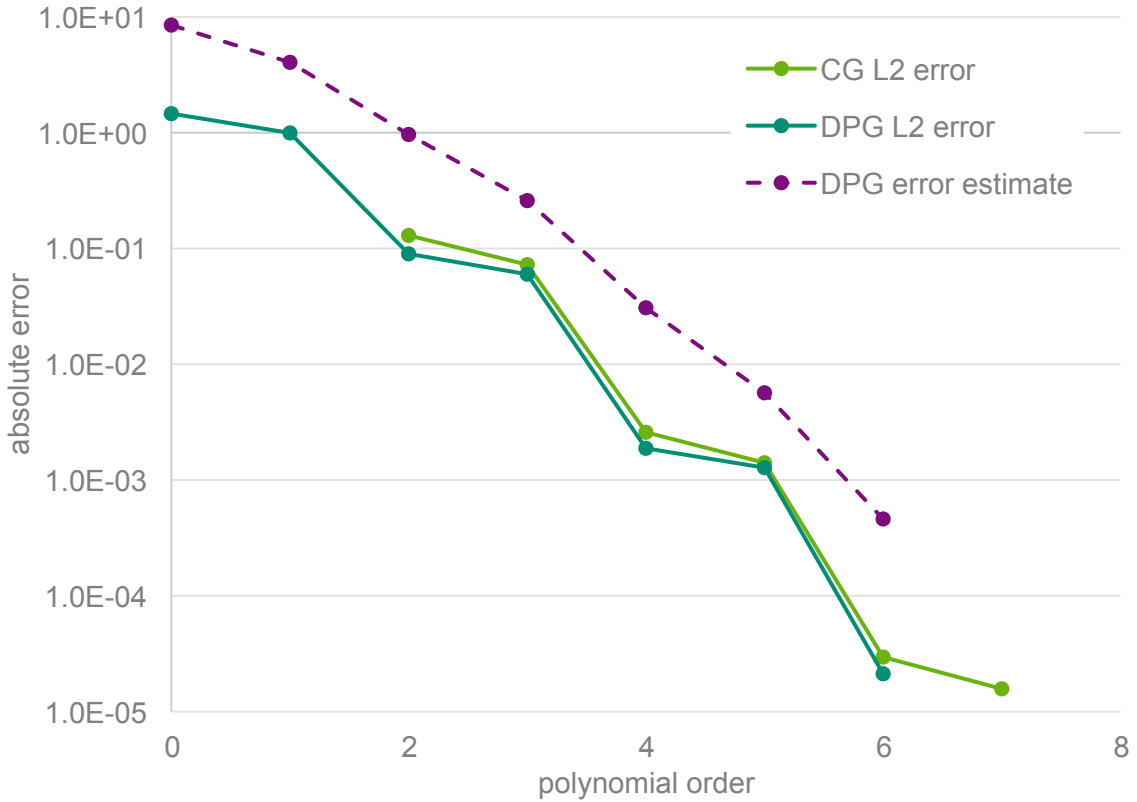
Known analytic solution for testing and verification:

$$\nabla^2 \phi = f$$

$$\phi = \sin(\pi x) sin(\pi y) \sin(\pi z)$$

$$\rightarrow f = -3\pi^2 \sin(\pi x) sin(\pi y) \sin(\pi z)$$

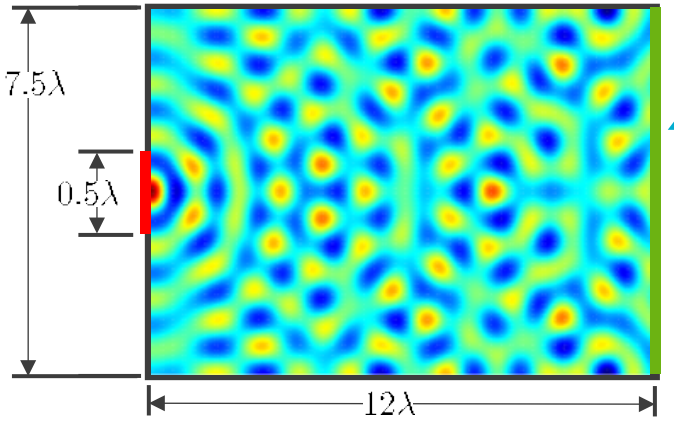




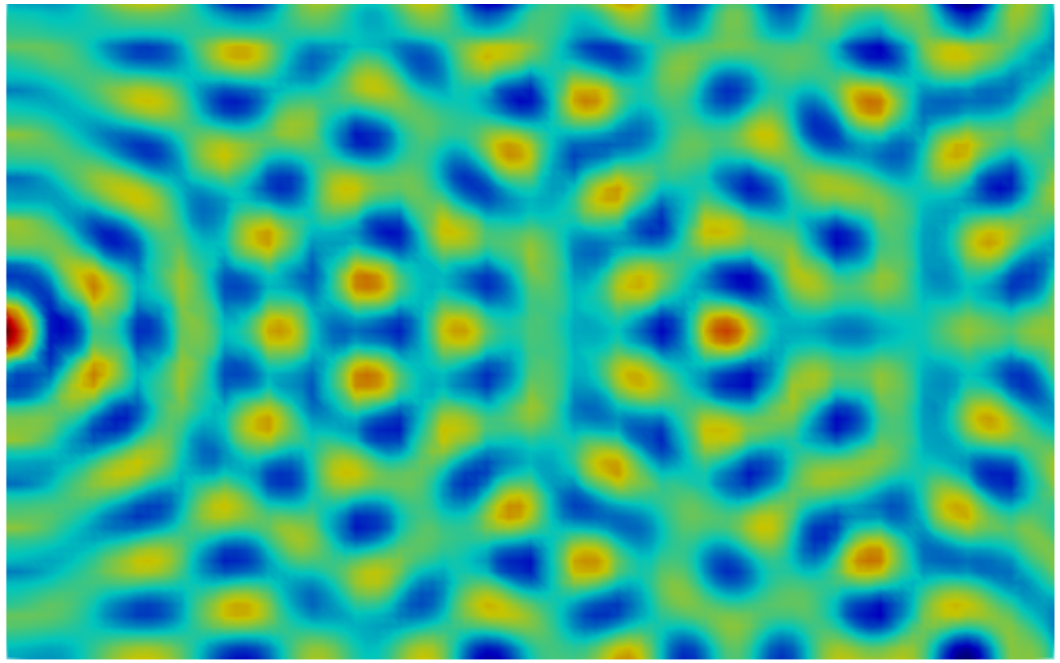
Numerical results: High-order Helmholtz solutions

- Custom-developed **high-order visualization & output**
- Strategy: *h*-adapt "viz" mesh and interpolate to points based on desired error
- Example: 2D acoustic duct (solved in 3D)
  1 element per wavelength

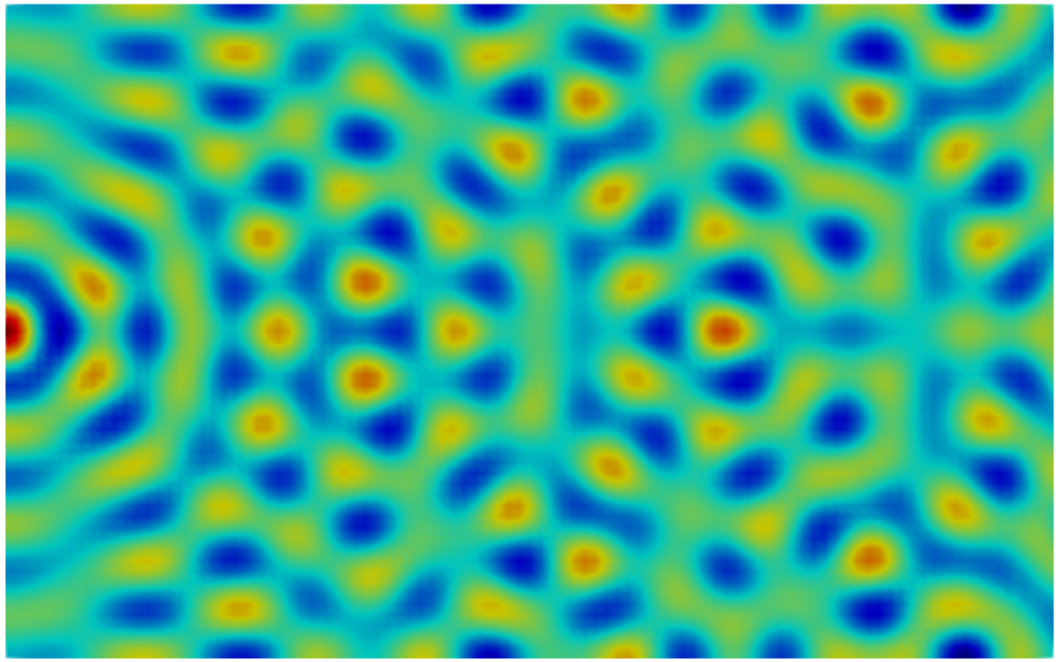
Exact solution
Re(pressure)



p=6 CGFE Re(pressure)

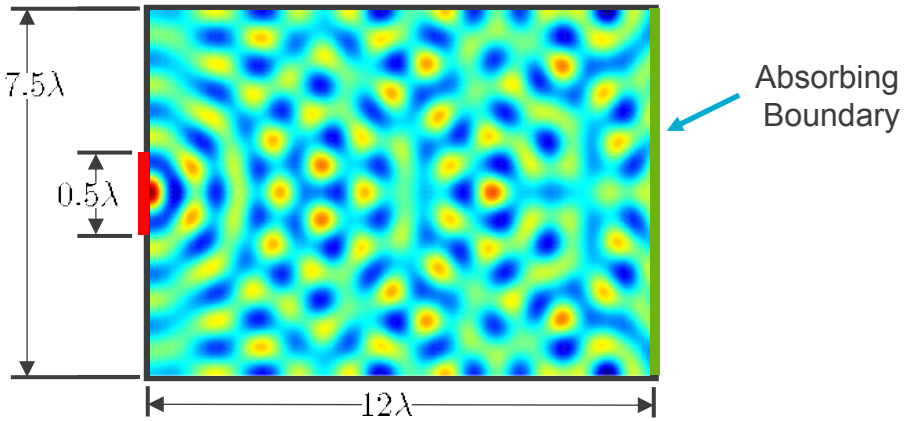


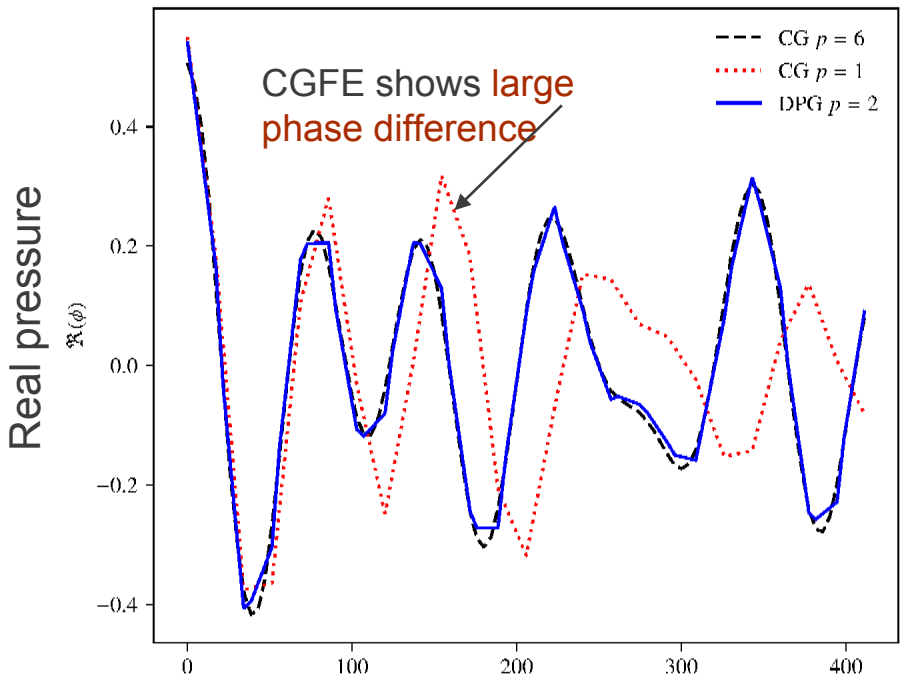
p=5 DPG Re(pressure)

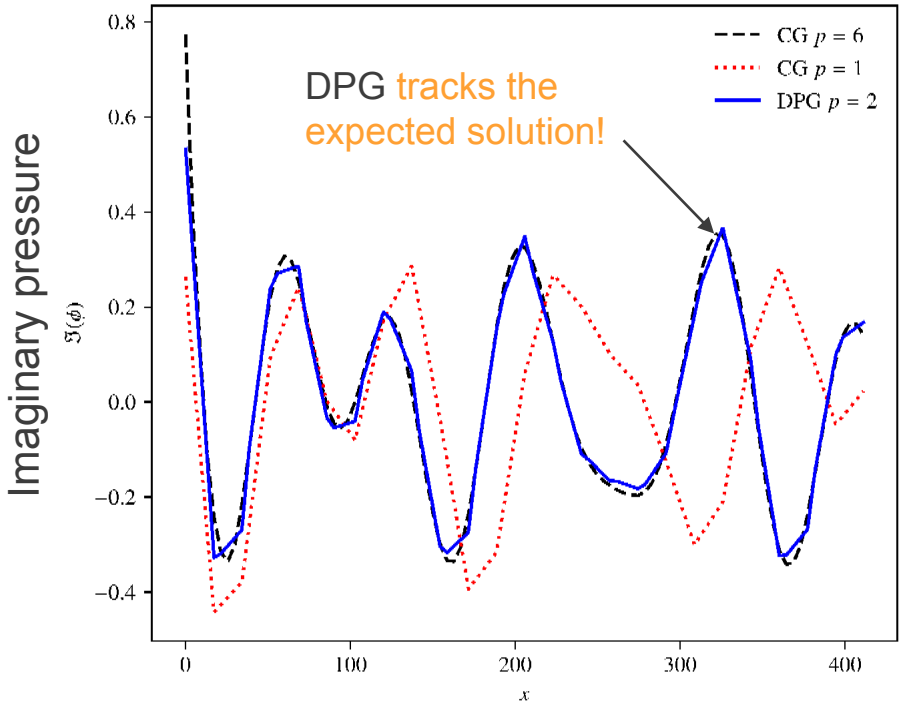


Numerical results: Low-order Helmholtz solutions

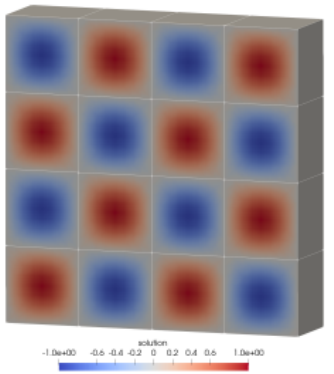
- Have we adequately addressed the dreaded pollution effect?
  - *Low-order CGFE, DPG vs. high-order reference solution*



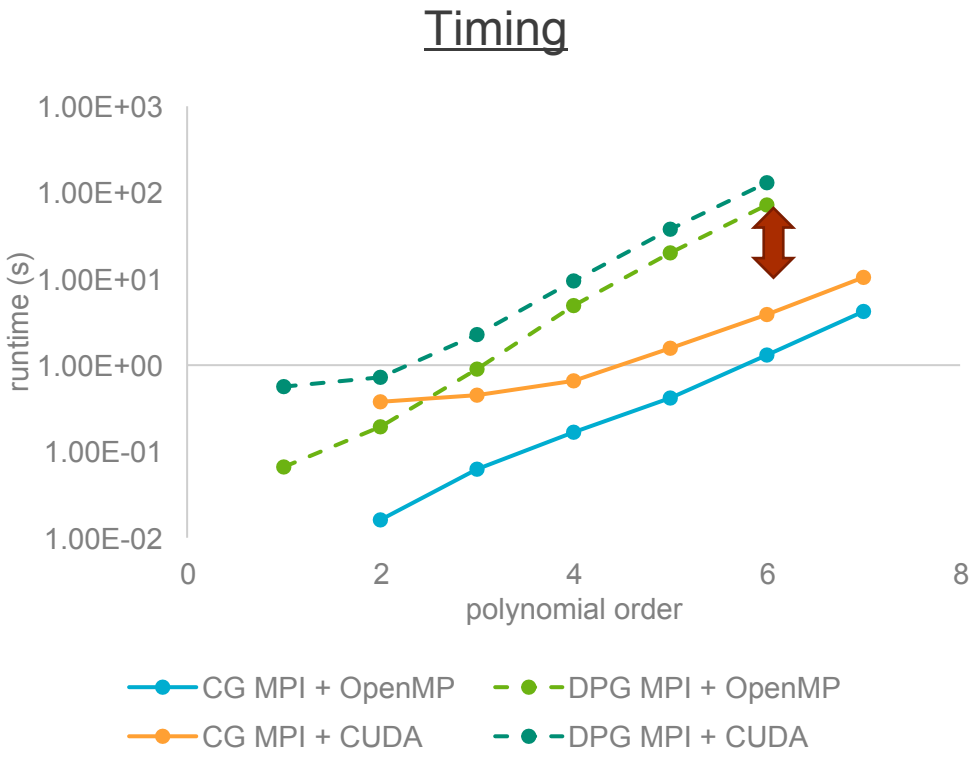


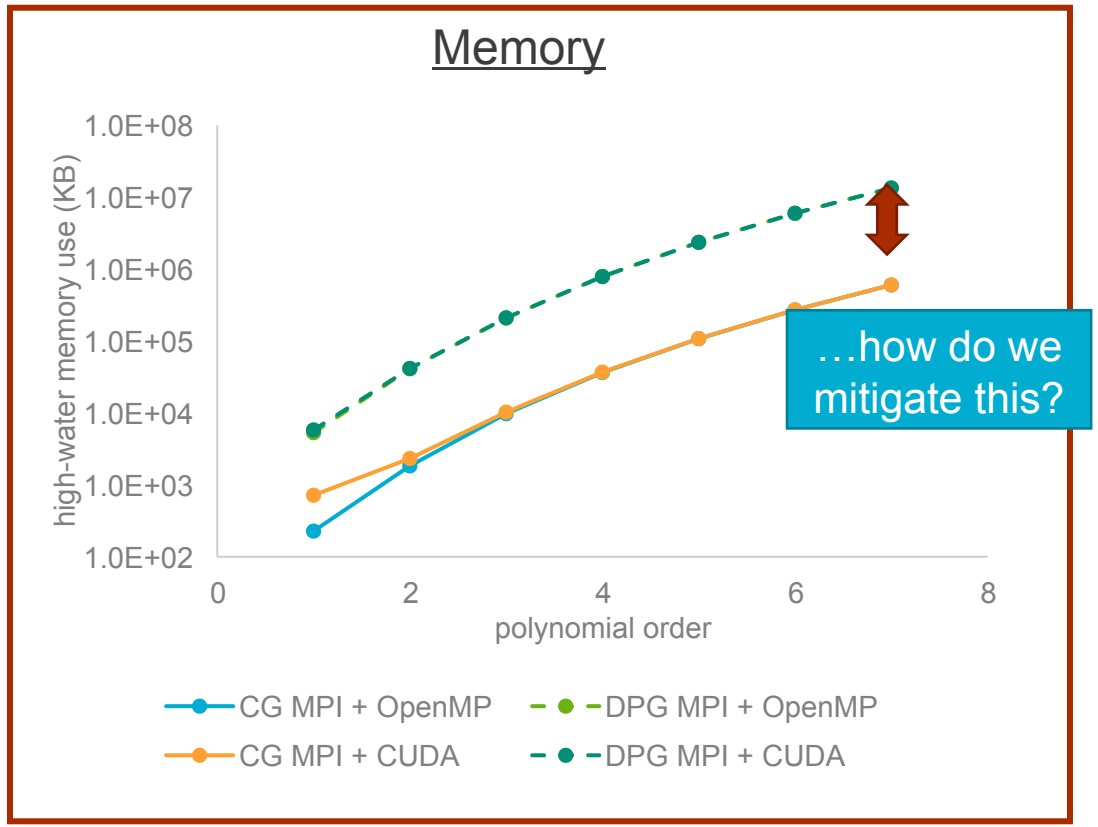


Numerical results: Performance in practice



- Memory usage in DPG is a concern…
  - Up to 2 orders of magnitude higher cost/memory use than FE





MPI + OpenMP: 1 MPI rank x 8 threads per rank
MPI + CUDA: 1 MPI rank + 1 Nvidia Volta GPU

Discretizations based on Trilinos/Intrepid2

- Example: Discretizing Poisson's equation
  - Intrepid2-based assembly of PDE terms is simple for continuous Galerkin…

```
intrepid2_basis::Basis hex = create_p_hierarchical_hex_basis("hierarchical", "hgrad", 1);

auto fs = intrepid2_basis::FunctionSpace::create(iface, hex, hexNodes, hexNodeGids);

auto transformDShape = fs->compute_transformed_derivatives();

// stiffness
fs->compute_weighted_integral(stiffness, transformDShape, transformDShape);
```

Discretizations based on Trilinos/Intrepid2

- Example: Discretizing Poisson's equation
  - DPG discretizations incur more complexity and bookkeeping
  - Introduce collections of function spaces and bases

Multiple bases

Function spaces corresponding to trace and element spaces

```
BasisCollection b;
short pInteg = 2 * porderTest;
b.add_basis(Basis("hex8", "dg_hierarchical", "hgrad", porderTest, pInteg), false);
b.add_basis(Basis("hex8", "dg_hierarchical", "hdiv", porderTest, pInteg), false);
b.add_basis(Basis("hex8", "hierarchical", "l2", porder - 1, pInteg), true);
b.add_basis(Basis("hex8", "hierarchical", "hdiv", porder, pInteg), false);
b.add_basis(Basis("hex8", "hierarchical", "hgrad", porder, pInteg), false);

FunctionSpaceCollection fc(cellCoords, cellVertexGids, bases);
fc.add_function_space_instance("v", "dg_hierarchical", "hgrad", 1);
fc.add_function_space_instance("q", "dg_hierarchical", "hdiv", 1);
fc.add_function_space_instance("phi", "hierarchical", "l2", 1);
fc.add_function_space_instance("psi", "hierarchical", "l2", 3);
fc.add_trace_function_space_instance("phiHat", "hierarchical", "hgrad", 1);
fc.add_trace_function_space_instance("psiHatN", "hierarchical", "hdiv", 1);
```

Discretizations based on Trilinos/Intrepid2

- Room for improvement: Return values, handling array sizes, avoiding excess memory use and repeat allocations

Each allocates a Kokkos -based multi-dimensional array—convenient for auto sizing

```
auto transformShapeFluxCgHdiv = fsFluxCgHdiv->compute_transformed_values();

auto hexFacePoints = fsFluxCgHdiv->get_integration_rule().points;
auto hexFaceNormals = hex.element_face_normals(elemCoords, hexFacePoints, iface);

// q dot n = transformShapeHDiv . cellSideNormals
auto psiHatDotN = compute_values_dotted_with_normals(transformShapeFluxCgHdiv, hexFaceNormals);
```

- Memory overhead due to arrays
- Strategies for array allocation and reuse?
- Opportunities for more C++ expression templates?

Discretizations based on Trilinos/Intrepid2

- Room for improvement: Opportunities to exploit parallel asynchrony to maximize concurrency

Operations on non-overlapping arrays could be asynchronous and non-blocking

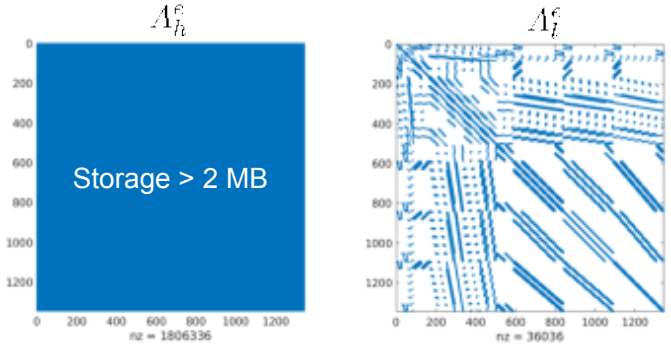
```
// B
fsHvol->compute_weighted_integral(stiffPhiQ, transformShapeHvol, transformDerivHdiv, -1.0);
fsHvol->compute_weighted_integral(stiffPsiQ, transformShapeHvolVec, transformShapeHdiv, -1.0);
fsHvol->compute_weighted_integral(stiffPsiV, transformShapeHvolVec, transformDerivHgrad, -1.0);

for (unsigned iface = 0; iface < hex.num_faces(); ++iface)
{
  ...

  // Bhat
  fsTraceCgHgrad->compute_weighted_integral(stiffPhiHatQ, transformShapeTraceCgHgrad, qDotN);
  fsFluxCgHdiv->compute_weighted_integral(stiffPsiHatNV, psiHatDotN, transformShapeHgradOnFace);
}
```

Performance mitigation: Matrix-free and lightweight preconditioners

- Computational cost and memory use grow exponentially in high-order numerical methods
  - Avoid allocating large matrices
  - Use novel solver preconditioning strategies

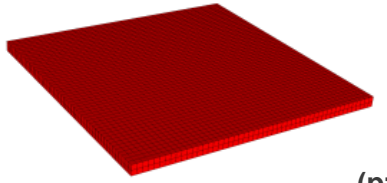


High-order system → low-order preconditioner

Simple plate problem

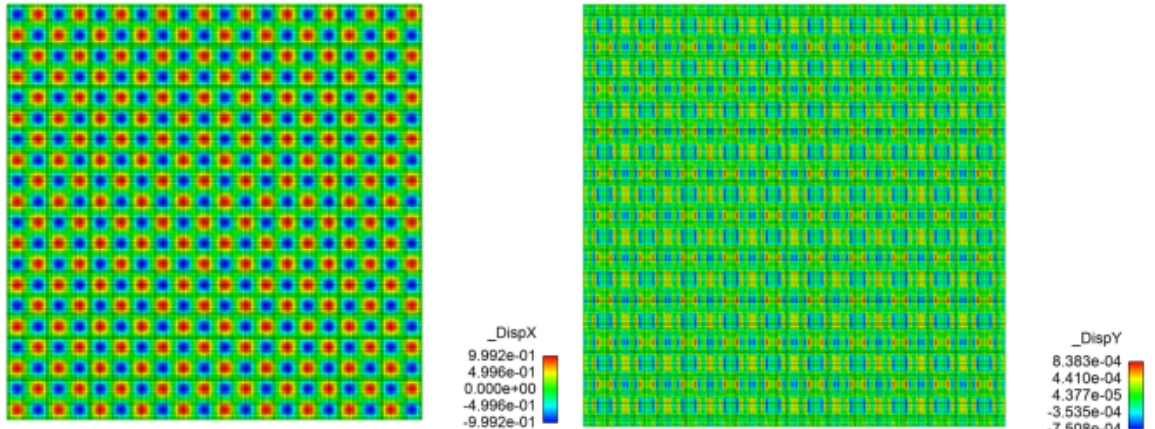
- 1 x 1 x 0.04
- Static load, elastic material
- 5000 uniform hexahedral elements
- Known analytic solution:

$$u_x = \sin(20\pi x)\sin(20\pi y)\sin(25\pi z)$$
$$u_y = 0$$
$$u_z = 0$$



Matrix-free solution

(p=3, 159,067 degrees of freedom)

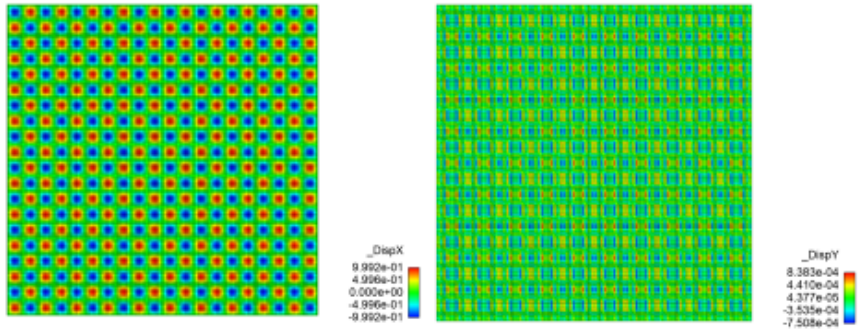


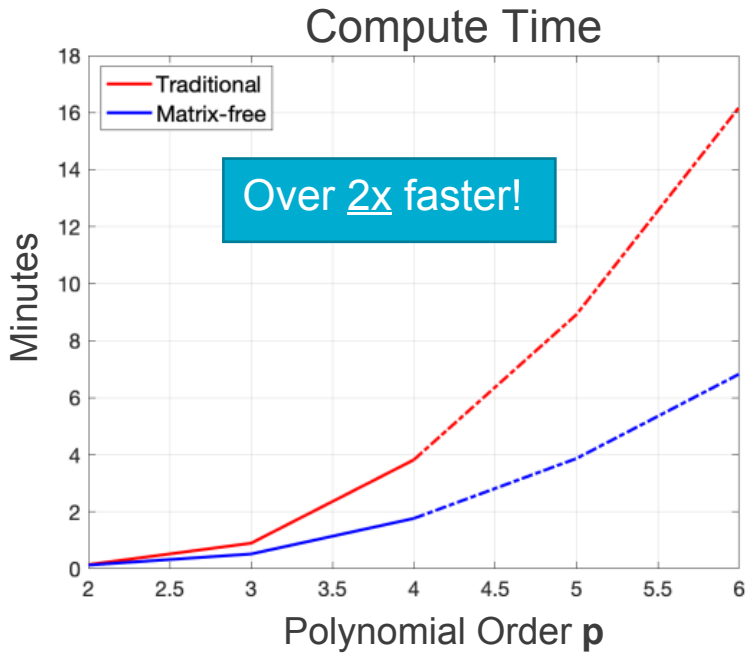
X-displacement Y-displacement

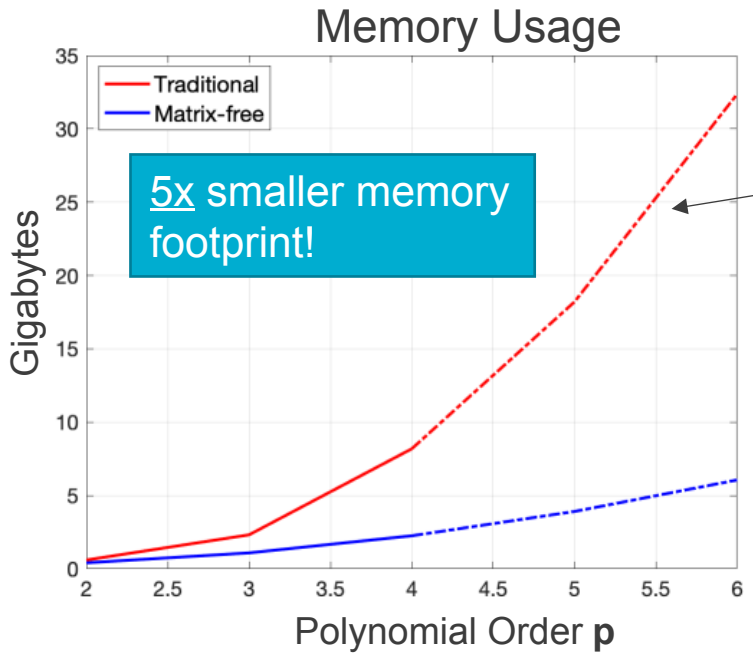
Work by C. Dohrmann et al.

Performance mitigation: Matrix-free and lightweight preconditioners

Computational cost and memory usage are **drastically reduced relative to traditional code implementations**







Sierra Structural Dynamics (massively parallel FE)

Work by C. Dohrmann et al.

Takeaways and future prospects

- Leveraging Trilinos packages enables high-order, advanced discretizations of PDEs of interest
  - More general than expected thanks to flexibility in Intrepid2
  - Elastodynamics implementation pending

- Threading and GPU acceleration in Intrepid2 discretization and Kokkos batched linear algebra prove valuable at higher polynomial orders
  - Work to do to prove performance on realistic problems

- Memory high-water mark, usability, and performance require mitigation for tractability
  - Matrix-free computations and efficient preconditioners
  - Avoid repeated memory allocations…especially on GPUs
  - Asynchronous, nonblocking kernels are essential to maximize concurrency

References & acknowledgments

- **Talks & publications**
  - C. Dohrmann: Spectral equivalence of low-order discretizations for high-order H(curl) and H(div) spaces. SIAM Journal on Scientific Computing, 43(6):A3992-A4014, 2021.
  - C. Dohrmann et al: Low-order preconditioning for the high-order finite element de Rham complex, submitted to SIAM Journal on Scientific Computing.
  - J. Plews: “Developing a GPU-enabled DPG Toolkit: Experiences with Intrepid2,” Trilinos User Group (TUG) Meeting, December 2021.

- **Thanks**
  - Computing Research at Sandia (Trilinos User Group, Kokkos Kernels)
  - DPG short course with Prof. Leszek Demkowicz (UT Austin)
  - Lawrence Livermore National Lab (C. Dohrmann)