High-Order Solvers + GPU Acceleration

Will Pazner

pazner@pdx.edu

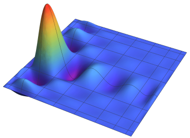
Joint with Tzanio Kolev, John Camier, and members of the MFEM team

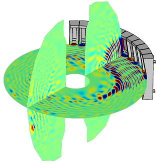
MFEM Community Workshop
October 25, 2022

Portland State
Fariborz Maseeh Department of
Mathematics & Statistics

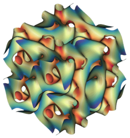
High order methods...

- Promise higher accuracy per DOF than low-order
- Have demonstrated success modeling under-resolved physics such as turbulence (e.g. large eddy simulation)
- Symmetry preservation, curved geometries, adaptivity, problems with singularities
- Better suited for modern architectures

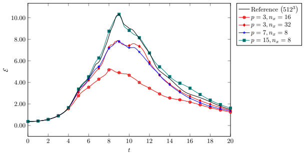




High-order wave propagation in magnetic fusion device



High-order incompressible Taylor-Green vortex



More accurate resolution of enstrophy for equal # DOFs with high-order methods

Solving high-order finite element problems remains challenging!

Inverting the resulting linear operators is expensive:

- Extremely ill-conditioned
- Expensive to assemble
- High memory cost to store

Solving high-order finite element problems remains challenging!

Inverting the resulting linear operators is expensive:

- Extremely ill-conditioned
- Expensive to assemble
- High memory cost to store

We would like to construct linear solvers that:

- Converge quickly
- Have low memory requirements
- Are applicable to different types of physics
- Support end-to-end GPU acceleration
- Are available and easy to use in MFEM

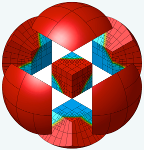
Solving high-order finite element problems remains challenging!

Inverting the resulting linear operators is expensive:

- Extremely ill-conditioned
- Expensive to assemble
- High memory cost to store

We would like to construct linear solvers that:

- Converge quickly
- Have low memory requirements
- Are applicable to different types of physics
- **Support end-to-end GPU acceleration**
- **Are available and easy to use in MFEM**



Matrix-free solvers for high-order methods

Memory usage and comput. complexity: scaling with p		3D
Number of DOFs	$\mathcal{O}(p^d)$	p^3
Matrix-based methods		
Nonzeros in system matrix	$\mathcal{O}(p^{2d})$	p^6
Traditional (naïve) assembly	$\mathcal{O}(p^{3d})$ ops	p^9
Sum-factorized assembly	$\mathcal{O}(p^{2d+1})$ ops	p^7
"*Matrix-free*" methods		
Optimal memory usage	$\mathcal{O}(p^d)$	p^3
Sum-factorized operator application	$\mathcal{O}(p^{d+1})$ ops	p^4

Matrix-free solvers for high-order methods

Memory usage and comput. complexity: scaling with p		3D
Number of DOFs	$\mathcal{O}(p^d)$	p^3
Matrix-based methods		
Nonzeros in system matrix	$\mathcal{O}(p^{2d})$	p^6
Traditional (naïve) assembly	$\mathcal{O}(p^{3d})$ ops	p^9
Sum-factorized assembly	$\mathcal{O}(p^{2d+1})$ ops	p^7
"*Matrix-free*" methods		
Optimal memory usage	$\mathcal{O}(p^d)$	p^3
Sum-factorized operator application	$\mathcal{O}(p^{d+1})$ ops	p^4

Goal: Iterative solvers with:

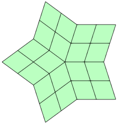
optimal $\mathcal{O}(p^d)$ memory | $\mathcal{O}(p^{d+1})$ operations | $\mathcal{O}(1)$ iterations

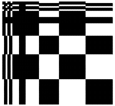
$\Longrightarrow$ Cannot assemble the matrix

$\Longrightarrow$ Must construct preconditioners without access to matrix entries

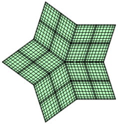
Low-order-refined preconditioning

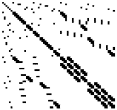
- High-order operator A_p
  - Matrix-free operator evaluation
- Low-order-refined operator A_h
  - Gauss–Lobatto refined mesh
  - A_h is **sparse:** $\mathcal{O}(1)$ nonzeros per row
  - $B_h \sim A_h^{-1}$ uniform preconditioner
- Use B_h as a preconditioner for A_p
- LOR spectral equivalence ("FEM–SEM equivalence")





A_p





A_h

Theorem [Canuto, Quarteroni]

A_p, A_h are H^1 discretizations of Poisson $\implies$ A_h is spectrally equivalent to A_p (constant independent of h and p).

$$\underset{\text{"nodes"}}{H^1} \xrightarrow{\nabla} \underset{\text{"edges"}}{\boldsymbol{H}(\mathbf{curl})} \xrightarrow{\nabla\times} \underset{\text{"faces"}}{\boldsymbol{H}(\mathrm{div})} \xrightarrow{\nabla\cdot} \underset{\text{"elements"}}{L^2}$$

Kinematics | MHD | Rad. diffusion | Rad. transfer

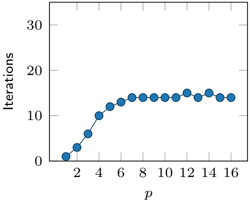
Theorem [Dohrmann, Kolev, P.]

Spectral equivalence (independent of h and p) extends to curl-curl problems in $\boldsymbol{H}(\mathbf{curl})$, grad-div problems in $\boldsymbol{H}(\mathrm{div})$, and DG diffusion problems in L^2 using the "interpolation–histopolation" basis.

Poisson

$-\nabla \cdot \nabla u = f$

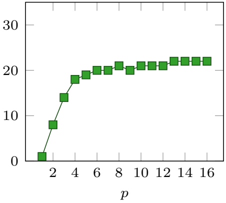
Lagrange H^1



Maxwell

$\boldsymbol{u} + \nabla \times \nabla \times \boldsymbol{u} = \boldsymbol{f}$

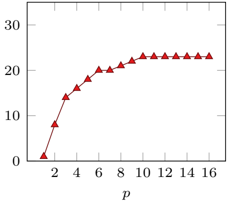
Nédélec $\boldsymbol{H}(\mathbf{curl})$



Grad-div

$\boldsymbol{u} - \nabla(\nabla \cdot \boldsymbol{u}) = \boldsymbol{f}$

Raviart–Thomas $\boldsymbol{H}(\mathrm{div})$



Solution Algorithm

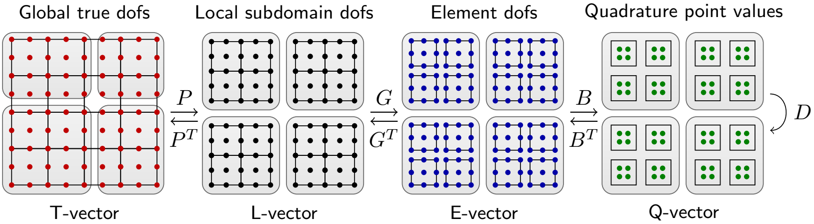
- Setup phase
  1. High-order operator setup
  2. Low-order-refined matrix assembly
  3. AMG setup
- Solve phase
  1. High-order operator evaluation
  2. AMG V-cycle

- Delegate the AMG setup and V-cycle to *hypre*
- LOR preconditioning $\implies$ can use **any** matrix-based preconditioner applied to the LOR system to precondition the HO problem

High-order operator setup and application

- Use MFEM's *partial assembly* approach

$$\boxed{A_p = P^T G^T B^T D B G P}$$

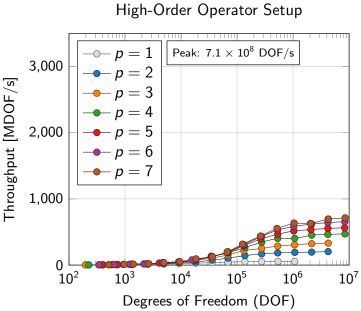


- Represent operator in *matrix-free* format
  - Nested product of linear operators
- Closely related to the CEED project and libCEED library

CEED
EXASCALE DISCRETIZATIONS

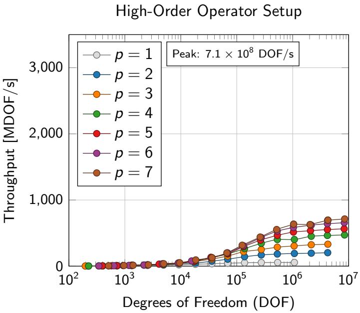
High-order operator setup and application

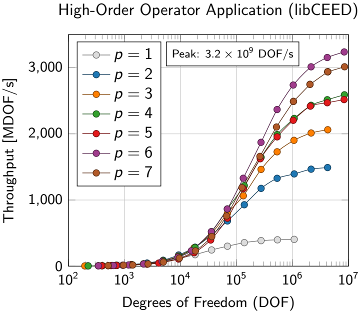
- Optimal $\mathcal{O}(p^d)$ memory requirements
- $\mathcal{O}(p^{d+1})$ computational complexity



High-order operator setup and application

- Optimal $\mathcal{O}(p^d)$ memory requirements
- $\mathcal{O}(p^{d+1})$ computational complexity



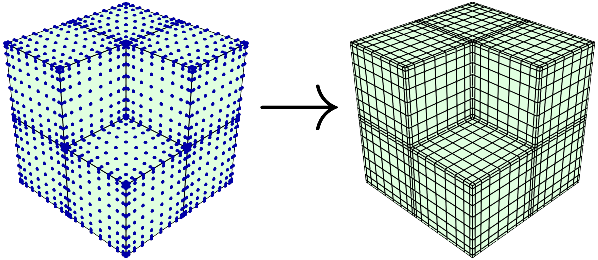


- Best performing kernels: MFEM's libCEED backend with `cuda-gen` kernel fusion/code generation
- Typical behavior of high-order methods on GPU:
  - Higher-order $\implies$ **faster** performance

Low-order-refined matrix assembly

Until now, this was a major bottleneck in LOR preconditioning

- Creation and "bookkeeping" for the low-order refined mesh induced significant overhead
- Actual matrix assembly either on host (`AssemblyLevel::LEGACY`)
- Or more recently on device (`AssemblyLevel::FULL`)

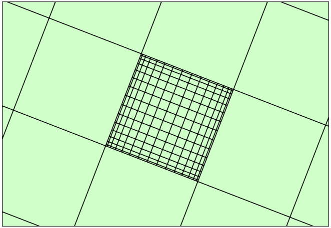


Low-order-refined matrix assembly on GPU

Macro-element batching strategy

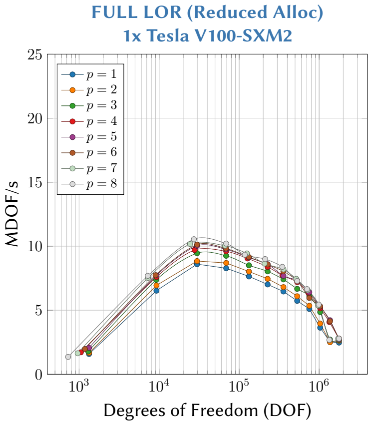
- Perform **all** work at the level of *macro-elements*
- **Avoid** generating LOR mesh
- **Reuse** all data structures and connectivity from high-order (coarse) mesh
- Make use of **local Cartesian structure**

- One block of threads per macro-element
- Thread over LOR "subelements"
- Assembly macro-elements into local **sparse** matrices with **fixed** sparsity
- Assemble into global (parallel) CSR format for use with AMG

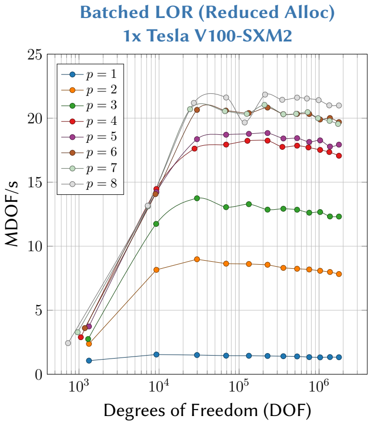


LOR assembly throughput

Before



After

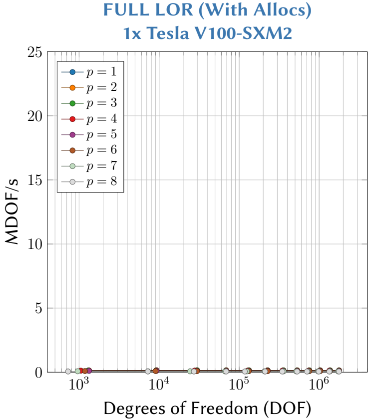


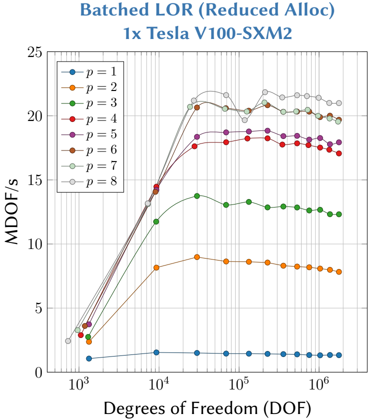
Including only assembly kernels (no pre-processing)

LOR assembly throughput

Before

After

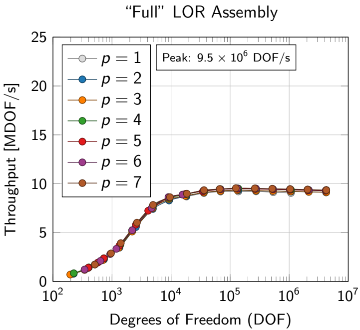




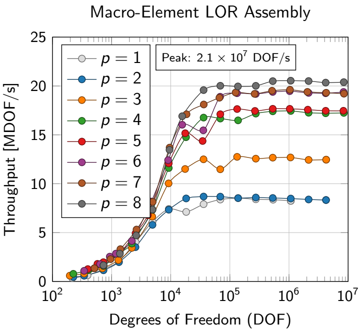
Including full assembly procedure (with pre-processing)

LOR assembly throughput

Before



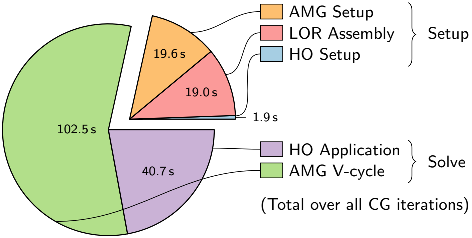
After



Macro-element strategy: higher order $\implies$ faster performance
(In constrast to "legacy" assembly algorithm)

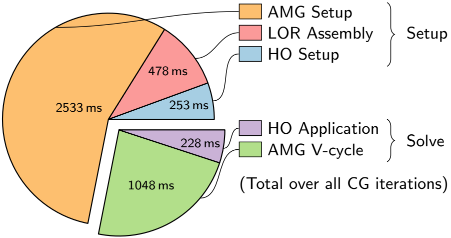
CPU and GPU comparisons

CPU:



Total: 181 s
3D Case, CPU

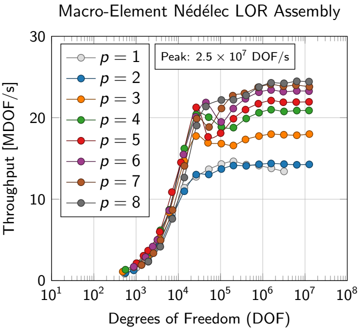
GPU:



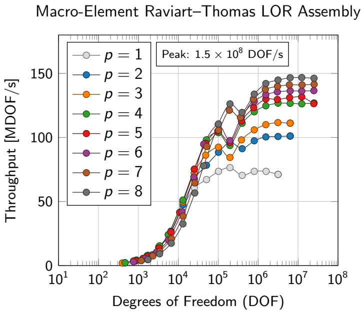
Total: 4540 ms
3D Case, GPU

Nédélec and Raviart–Thomas elements

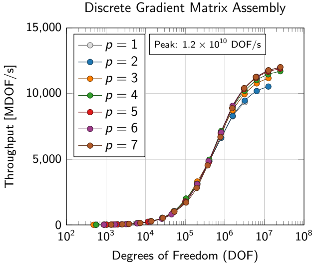
Nédélec

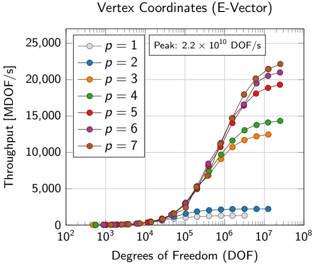


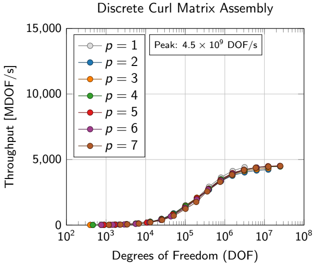
Raviart–Thomas

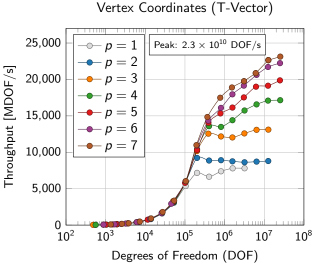


- To solve curl-curl (electromagnetic diff.) and grad-div (radiation diff.) problems, use *hypre*'s **AMS** and **ADS** auxiliary space preconditioners
- In addition to the **system matrix**, these solvers require:
  - vertex coordinates, discrete gradient matrix, discrete curl matrix



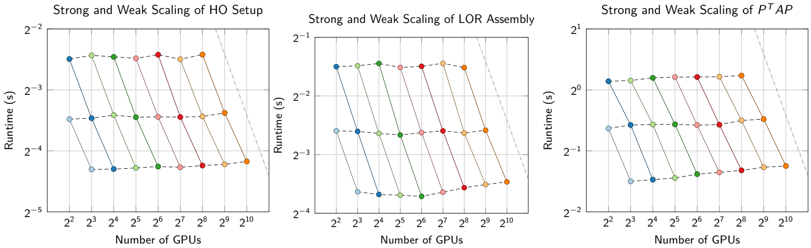






Parallel scalability

- Good strong and weak scalability (shown here up to 1024 GPUs)



Scaling Legend
Ideal Strong Scaling
Strong Scaling
Weak Scaling

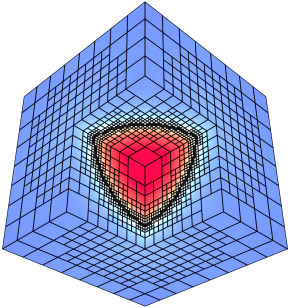
Problem Size (DOFs)		
8.4×10^6	1.7×10^7	3.4×10^7
6.7×10^7	1.4×10^8	2.7×10^8
5.4×10^8	1.1×10^9	

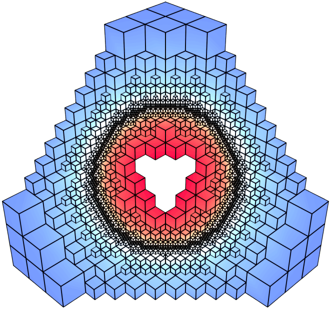
LOR AMR preconditioning

- New LOR preconditioning method based on variational restriction

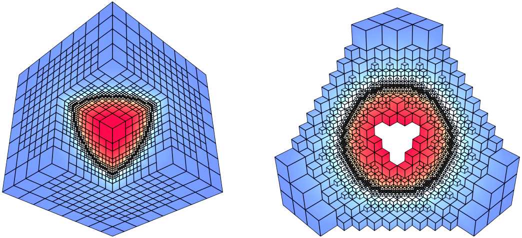
$$A_p = \Lambda_p^T \widehat{A}_p \Lambda_p, \qquad A_h = \Lambda_p^T \widehat{A}_h \Lambda_p$$

$$A_h \sim A_p \qquad \text{independent of } p$$





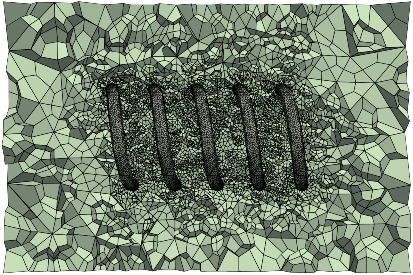
Results: AMR



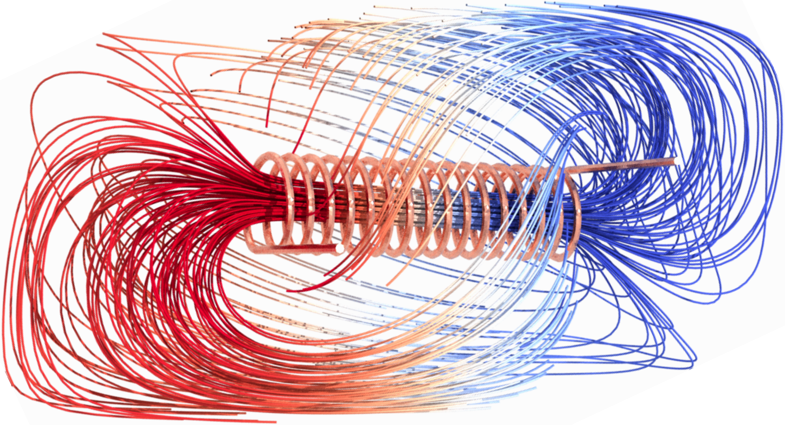
p	DOFs	NNZ	NNZ per row	Its.	GPU Runtime (s)
1	6.0×10^4	1.7×10^6	28	28	0.4
2	6.1×10^5	2.2×10^7	36	43	0.7
3	2.2×10^6	8.8×10^7	40	42	1.1
4	5.5×10^6	2.3×10^8	42	44	2.0
5	1.1×10^7	5.0×10^8	45	45	3.3
6	1.9×10^7	9.2×10^8	48	46	5.7
7	3.1×10^7	1.6×10^9	52	47	9.9

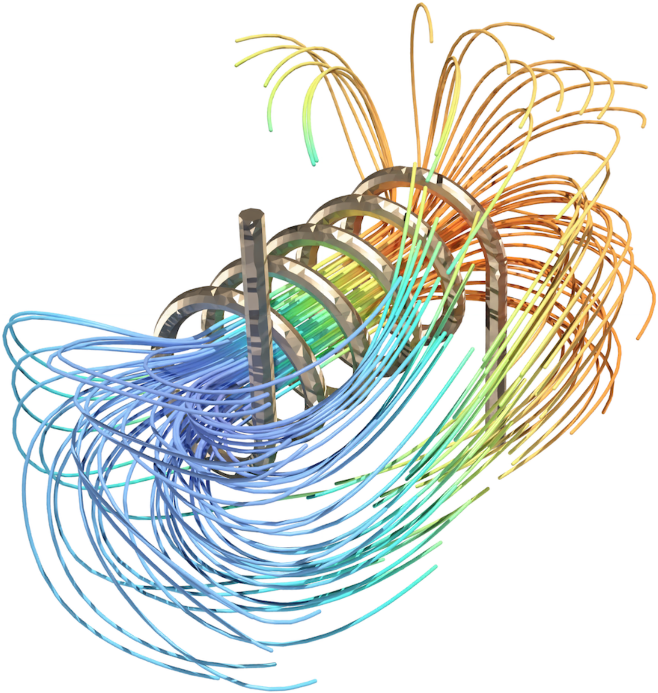
Electromagnetic diffusion

- Solve for magnetic field induced by electric current running through a coil
- Use A–ϕ formulation of magnetic diffusion
- Drive current by potential difference at two terminals
- Piecewise constant conductivity coefficient in two materials (copper and air)



- Solve for electric scalar potential ϕ — LOR + AMG
- Compute electric field with discrete gradient
- Solve for magnetic vector potential A — LOR + AMS in $\boldsymbol{H}(\mathbf{curl})$
- Compute magnetic field B in $\boldsymbol{H}(\mathrm{div})$ with discrete curl
- 1.5×10^6 hexahedral elements mesh
- 2.9×10^8 Nédélec DOFs with $p = 4$
- 45 CG iterations in H^1, 22 CG iterations in $\boldsymbol{H}(\mathbf{curl})$
- Wall clock runtime on 320 V100 GPUs 26 seconds





How can I use this?

- All of these methods are **available** and **easy to use** in MFEM
- **GPU acceleration** and **macro-element batching** are **automatically enabled** if applicable
- Creating LOR solvers is **one line of code**

```
// For any SolverType (AMG, direct solver, etc.), form the
// corresponding LOR preconditioner
LORSolver<SolverType> lor_solver(a, ess_dofs);
```

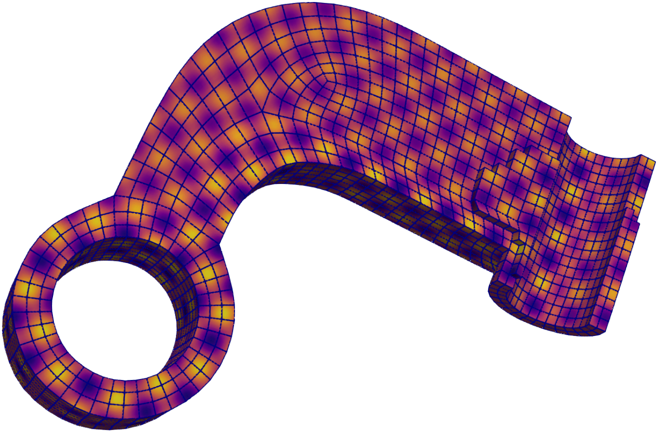
```
// For example:
// if 'a' is H1 diffusion...
LORSolver<HypreBoomerAMG> lor_amg(a, ess_dofs);
// if 'a' is ND curl-curl...
LORSolver<HypreAMS> lor_ams(a, ess_dofs);
// if 'a' is RT div-div...
LORSolver<HypreADS> lor_ads(a, ess_dofs);
```

Demo

- These methods are illustrated in the LOR solvers miniapp (included with MFEM)

Demo

- These methods are illustrated in the LOR solvers miniapp (included with MFEM)



Conclusions

- Matrix-free high-order solvers on the GPU
- MFEM supports **end-to-end** GPU acceleration of LOR preconditioners
- Preconditioners for all of the de Rham complex
  - H^1, $\boldsymbol{H}(\mathbf{curl})$, $\boldsymbol{H}(\mathrm{div})$ problems
- Convergence independent of mesh size and polynomial degree h
- Easy to use API: usually just one line of code
- Illustrated in bundled solvers miniapp

Pazner, Kolev, Dohrmann. *Low-order preconditioning for the high-order de Rham complex* (2022).

Pazner, Kolev, Camier. *End-to-end GPU acceleration of low-order-refined preconditioning for high-order finite element discretizations* (2022).