MFEM: Recent Developments

MFEM Workshop 2023

October 26, 2023, Virtual Meeting

Veselin Dobrev and the MFEM team

LLNL-PRES-857212

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

CASC

Lawrence Livermore National Laboratory

New developments in MFEM

- Two new releases since last year: v4.5.2 (Mar ’23) and v4.6 (Sep ’23)
- SubMesh and ParSubMesh have been extended to support the transfer of Nedelec and Raviart-Thomas finite element spaces, see the new Examples 34 and 35
  - Solving different physics on different subdomains or boundaries
  - Transferring solutions between parent and child meshes
- More TMOP metrics (asymptotically-balanced), auto-balancing of compound metrics; new tool, tmop-metric-magnitude
- New miniapp for interface and boundary fitting to surfaces defined via level-set functions
- New DPG miniapps: diffusion, advection-diffusion, acoustics, Maxwell

New developments in MFEM (cont.)

- Improved lambda body debugging: use mfem::forall* functions instead of MFEM_FORALL* macros
- HIP support in the SUNDIALS and PETSc integrations
- Added support for pyramids in non-conforming meshes and import from Gmsh
- Added a fast normalization-based distance solver, see the Distance miniapp
- Added KDTree class for 2D/3D set of points, the new nodal-transfer miniapp
- Added a new GPU-enabled H(div) solver miniapps, see miniapps/hdiv-linear-solver
- Updated the MUMPS interface, added interface to MKL Pardiso

New developments in MFEM (cont.)

- Several NURBS meshing improvements: support for free connectivity of NURBS patches (C-meshes), methods to set and get attributes on NURBS patches and patch boundaries, curve interpolation method for NURBS

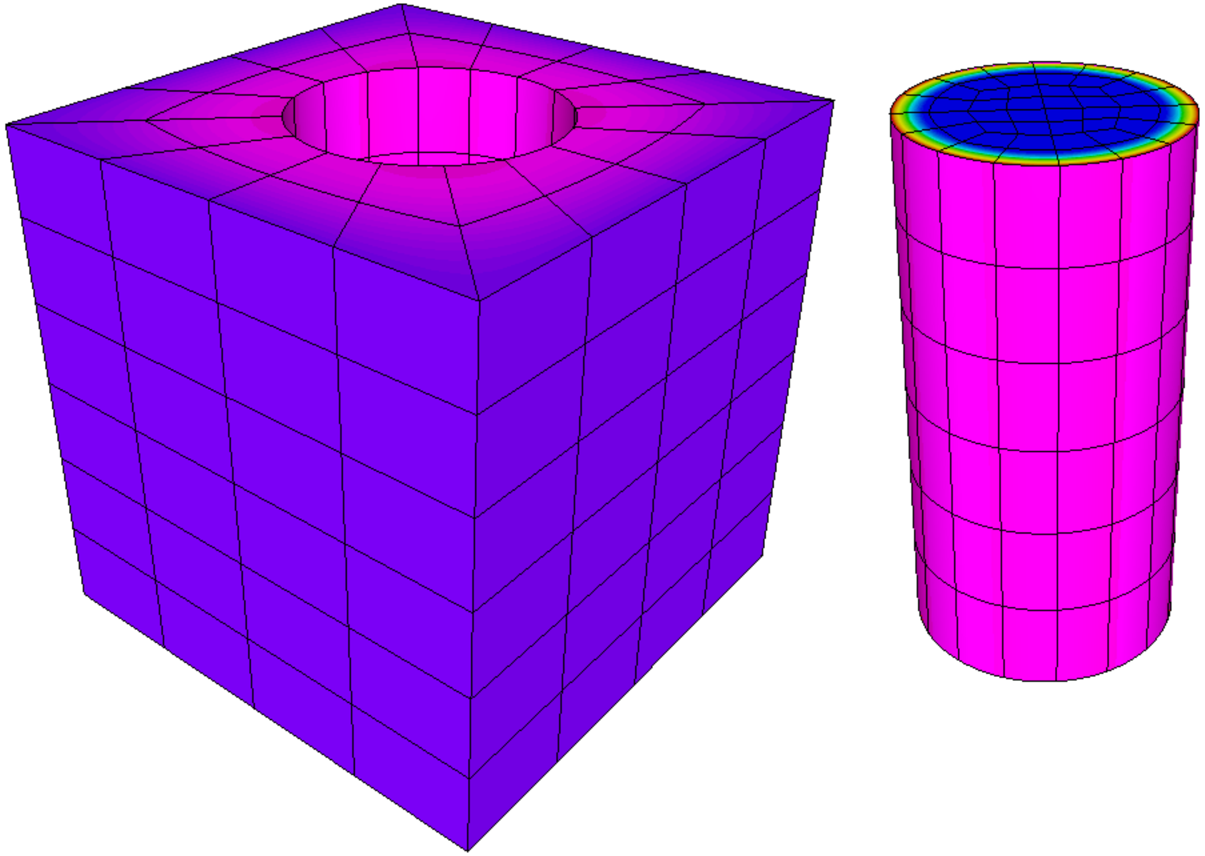
- Added support for partial assembly on NURBS patches, and NURBS-patch sparse matrix assembly

- Four new examples:
  - Example 34/34p solves a simple magnetostatic problem where source terms and boundary conditions are transferred with SubMesh objects.
  - Example 35p implements H1, H(curl) and H(div) variants of a damped harmonic oscillator with field transfer using SubMesh objects
  - Example 36/36p demonstrates the solution of the obstacle problem with a new finite element method (proximal Galerkin)
  - Example 37/37p demonstrates topology optimization with MFEM

- Many more …

Multi-domain multi-physics coupling via SubMesh

- All domains are meshed together in a conforming global mesh
- Each sub-domain, volume or surface, is extracted as a SubMesh
- SubMesh inherits from Mesh, so PDE discretization on it can be done as usual
- Due to conformity, solutions can be transferred without approximation errors between parent and child mesh
- In parallel, ParSubMesh inherits the partitioning from the parent
- For an example, see the multidomain miniapp



Simple example of coupling a heat equation (left) with convection-diffusion (right)

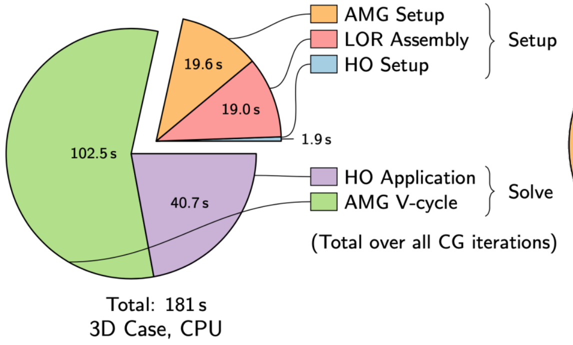
Low-Order-Refined GPU Solvers

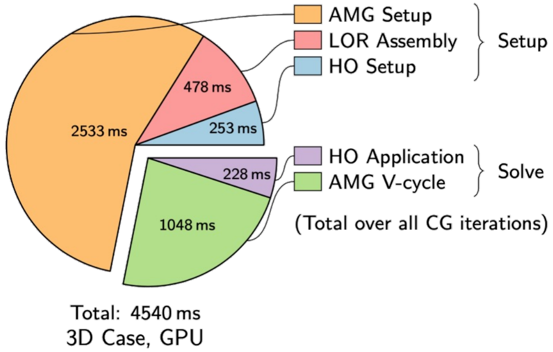
LOR solvers and GPU performance

- Using LOR + *hypre*'s AMG, AMS and ADS solvers in MFEM on the GPU is **one line of code**
  - MFEM is the FE interface to *hypre* for many apps

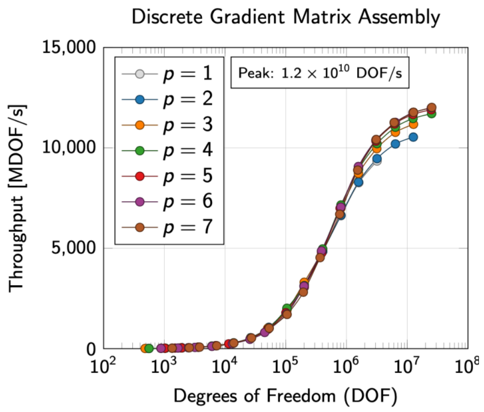
```
// For example:
// if 'a' is H1 diffusion...
LORSolver<HypreBoomerAMG> lor_amg(a, ess_dofs);
// if 'a' is ND curl-curl...
LORSolver<HypreAMS> lor_ams(a, ess_dofs);
// if 'a' is RT div-div...
LORSolver<HypreADS> lor_ads(a, ess_dofs);
```

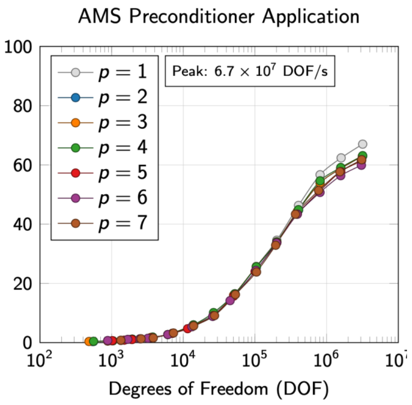
- We have performed **end-to-end GPU acceleration** of the entire solution algorithm
  - Assembly, preconditioner setup, solve phase
  - Details and performance metrics in *End-to-end GPU acceleration of low-order-refined preconditioning for high-order finite element discretizations, IJHPCA, submitted*





- **Flexibility:** solvers perform well
  - For H^1, H(curl), H(div)
  - With high-order elements
  - On AMR meshes, etc.
- Excellent **strong** and **weak scalability:**
  - Benchmarked up to 1024 GPUs, 1.1 billion DOFs



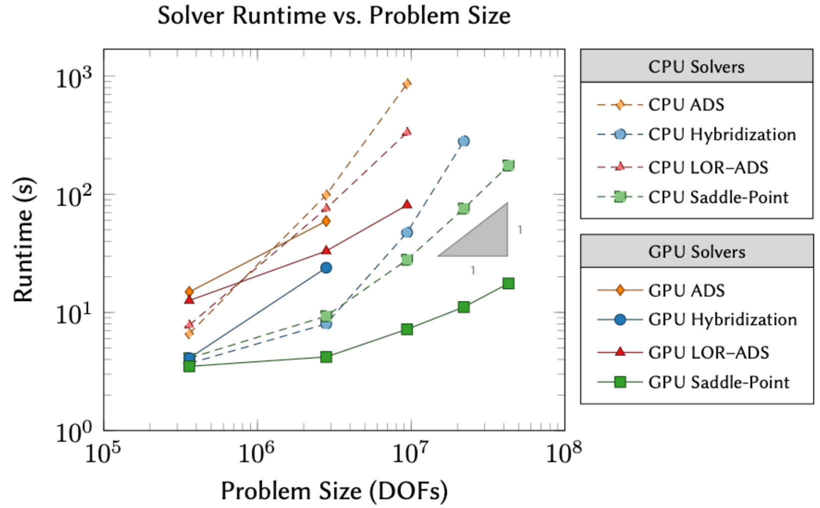


New GPU Solvers for Radiation Diffusion

Saddle-point formulation using LOR

- Radiation diffusion problems give rise to challenging linear systems (after linearization)
- Can be formulated as an H(div) problem, solved using ADS
  - Works for high-order + GPU using LOR solvers
  - But: not as fast as hybridization
- **State of the art:** algebraic hybridization solvers [SISC, 2019]
  - Main solver in MARBL
  - Not suitable for high-order, GPU acceleration
  - Requires fully assembled matrices
- **New approach:**
  - Works directly on the saddle-point system
  - Fast DG mass inverse kernels
  - Sparse approximate Schur complement
  - Scalable, high-order, GPU-accelerated solvers for H(div) and radiation diffusion problems

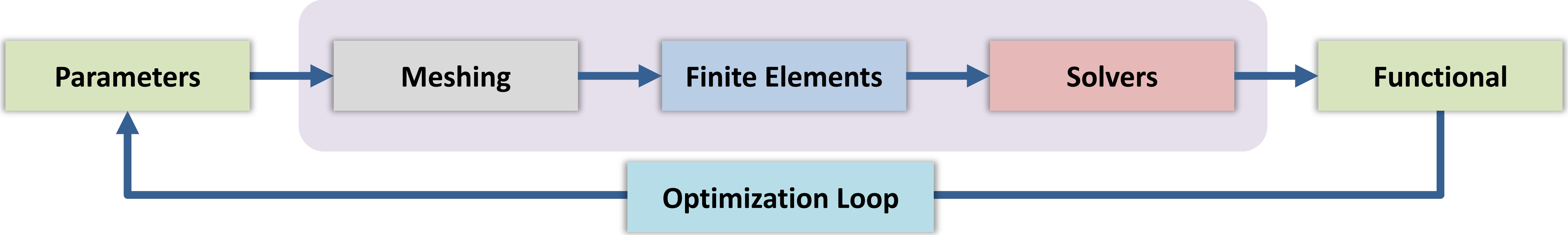
$$\partial\mathcal{N}(\mathbf{k}) = \begin{bmatrix} \ddots & & \mathbf{0} & \vdots & \vdots \\ & L_{\rho_k} + \partial H_k & & -c\Delta t L_{\sigma_k} & 0 \\ \mathbf{0} & & \ddots & \vdots & \vdots \\ \cdots & -\partial H_k & \cdots & L + c\Delta t \sum_k L_{\sigma_k} & D \\ \cdots & 0 & \cdots & -\frac{1}{3}\Delta t D^T & \frac{1}{c}R_\sigma + \frac{1}{3}R_n \end{bmatrix}$$



$$\begin{pmatrix} \tilde{L}^{-1} + \mathrm{D}\tilde{R}^{-1}\mathrm{D}^T & 0 \\ 0 & \tilde{R} \end{pmatrix}^{-1} \begin{pmatrix} -L^{-1} & \mathrm{D} \\ \mathrm{D}^T & R \end{pmatrix}$$

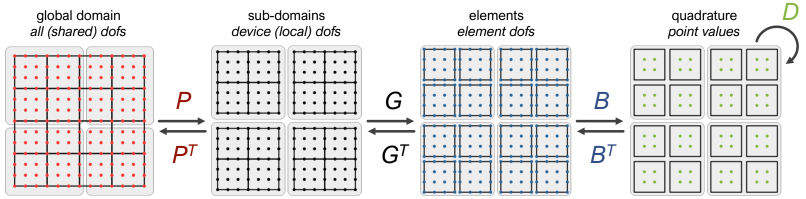
Automatic Differentiation with Partial Assembly

Jacobians and derivatives of FEM operators in a user-friendly way



- FEM decomposition

$$A = P^T G^T B^T D B G P$$



- Parameters $\hat{\rho} = B_\rho G_\rho P_\rho \, \rho$
- Parametric nonlinear operator

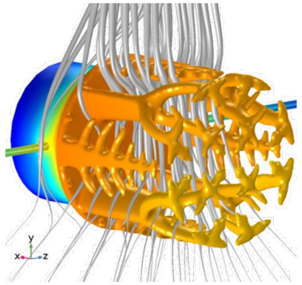
$$A(u;\rho) = P^T G^T B^T \, D(\hat{u},\hat{\rho})$$

- Need to differentiate at Q-points only!

$$\nabla_u A(u;\rho) = P^T G^T B^T \, \nabla_{\hat{u}} D(\hat{u},\hat{\rho}) \, B \, G \, P$$

(Jacobian is FEM decomposed linear operator)

- Differentiate the Q-function *D* with Enzyme!
  - AD at LLVM level, *after* compiler optimization
  - Can mix code from different languages
  - Differentiate across function calls (e.g. EOS)
  - Many parallel small ADs instead of 1 big one
  - Differentiate only what is necessary



Topology-optimized LED heat sink

MFEM + Enzyme

High-Order Mesh Optimization (TMOP)

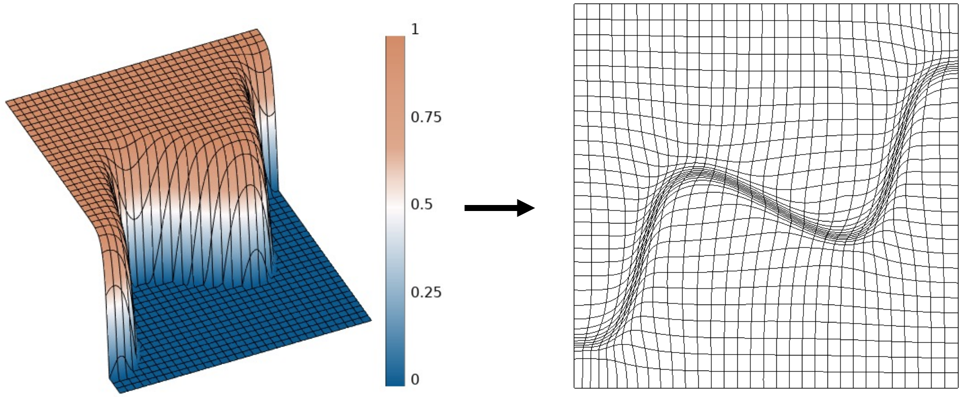
MFEM provides both geometric and simulation-driven adaptivity

- **User-controlled specification** of the target mesh
  - Control over: size/skew/aspect ratio/rotation
  - Can be based on discrete dynamic simulation fields
- **Variational minimization** through FE operations
  - Generality: dimension/element type/mesh order
  - Facilitates matrix-free formulations and GPU porting

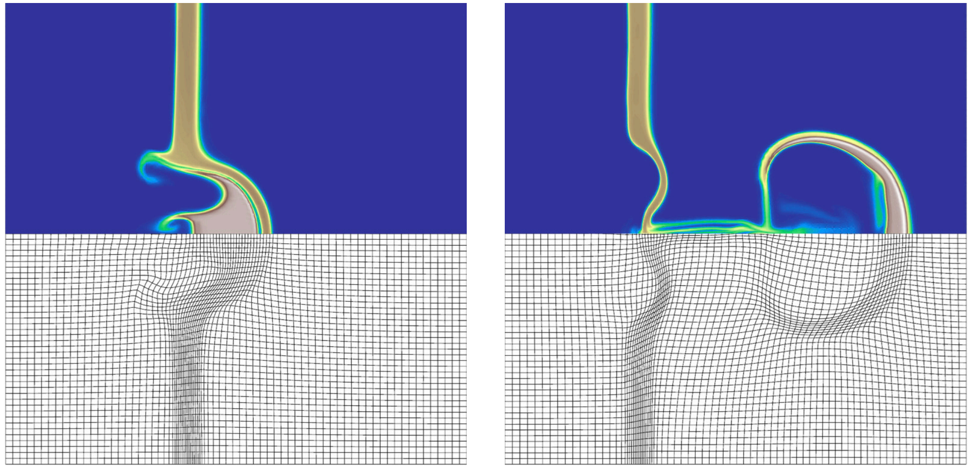
Minimizes $$\sum_{E \in \mathcal{M}} \int_{E_t} \mu(T(x_t))$$ mesh quality metric computed at q-points

- Numerous metric options in 2D and 3D; explicit combos
- Capability for ***hr*-optimization** of HO meshes
- Active ongoing theoretical research

The Target-Matrix Optimization Paradigm for high-order meshes, SISC, 2019
Simulation-driven optimization of high-order meshes in ALE hydrodynamics, Comput. Fluids, 2020
HR-adaptivity for nonconforming high-order meshes with TMOP, Eng. Comp, 2021
A target construction methodology for mesh quality improvement, Eng. Comp, 2022



Adaptivity of shape and size to a discrete interface

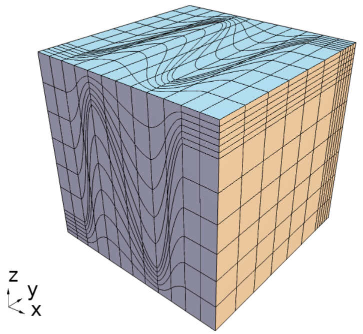


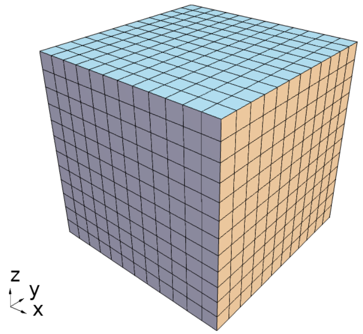
Adaptivity in a high-velocity impact simulation in MARBL

High-Order Mesh Adaptivity in MFEM

GPU Acceleration

- TMOP-based high-order mesh optimization:
  - Objective function based on user-defined target Jacobian and mesh quality metric is minimized for *r*-adaptivity.
  - Recent developments leverage GPU acceleration using partial assembly and matrix-free implementations.
  - Kershaw benchmark and multi-material ALE problem show significant (20-40x) speedup.



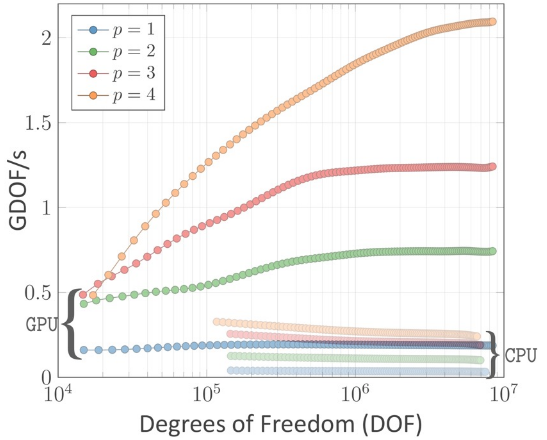


TMOP-based mesh optimization.

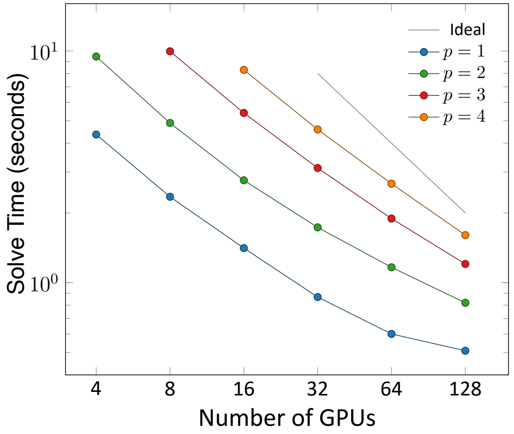
	Time to solution (sec)			
	$p = 1$	$p = 2$	$p = 3$	$p = 4$
CPU	18.8	43.0	129.6	224.3
GPU	0.4	1.0	3.9	7.5

	Speedup (GPU vs CPU)			
	47×	**43**×	**33**×	**30**×

Kershaw benchmark on 36 IBM Power9 CPU cores versus 4 CPU cores with 1 V100 GPU per core shows 30x speed-up.

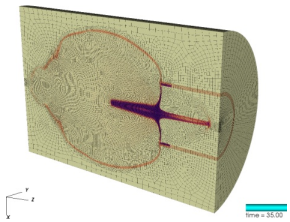


Throughput for the action of the second derivative operator on CPU versus GPU.



Strong scaling on GPUs for the full TMOP problem.

	Time ($p = 2$)	Speedup
CPU^{PA} Full TMOP problem	4730.830	-
CPU^{PA} 2nd Derivative	4713.426	-
GPU^{PA} Full TMOP problem	288.842	**16.37**×
GPU^{PA} 2nd Derivative	209.430	**22.51**×



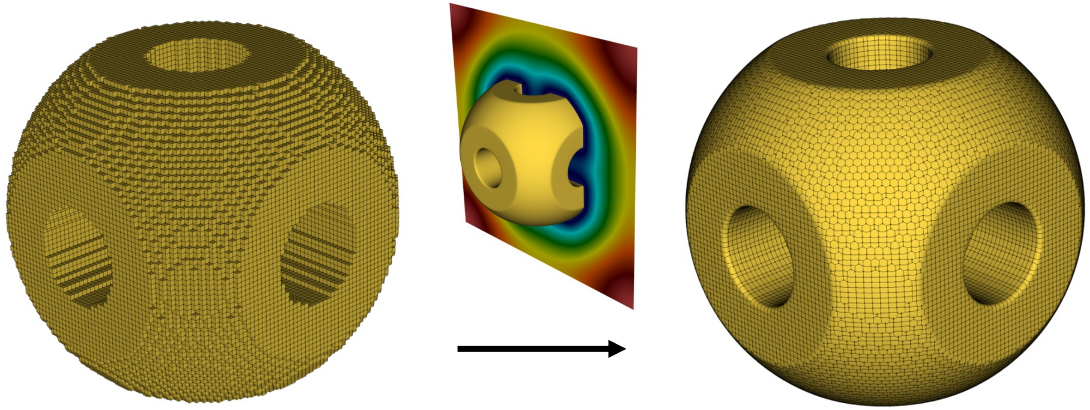
ALE problem in BLAST on 80 IBM Power9 CPU cores versus 8 CPU cores with 1 V100 GPU per core shows 20x speed-up.

Camier et al., *Accelerating high-order mesh optimization using finite element partial assembly on GPUs.* Journal of Computational Physics.

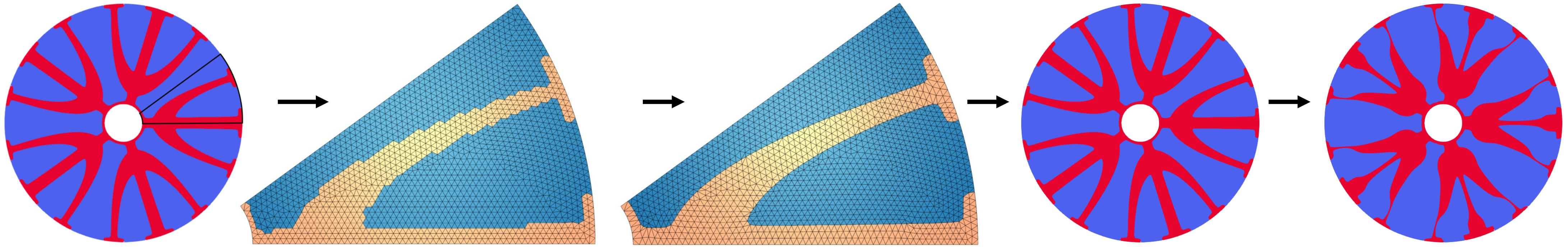
High-Order Mesh Adaptivity in MFEM

Boundary and Interface Alignment

- Mesh morphing for surface alignment:
  - Domain boundary and/or multimaterial interface is prescribed using a discrete level-set function.
  - Modified TMOP-based objective function is minimized to maintain good mesh quality while selected mesh nodes align with the prescribed interface/boundary.
  - Powerful to obtain curvilinear meshes starting from easy-to-generate meshes.
  - Robust in 2D and 3D for different element types.



Cartesian-aligned hex mesh morphed to align with the target boundary prescribed using a level-set function.



Fischer-Tropsch reactor domain.

Uniform triangular mesh morphed to align with the target interface prescribed using a level-set function.

Shape optimization to maximize energy production while keeping the volume of conducting fins (red) fixed.

Mittal et al., *High-Order Mesh Morphing for Boundary and Interface Fitting to Implicit Geometries.* Computer Aided Design Journal.

Progress of the Python Wrapper (PyMFEM)

Continue to closely follow the major MFEM releases:

- 4.4 Oct 2022, 4.5 Jan 2023, and 4.5.02 Mar 2023
- 4.6 preparation in progress

Major overhaul of Just-in-Time compiled coefficients:

- JIT function receives numpy-like array, not a pointer
- JIT function can receive other coefficients in addition to position
- Added supporting time-dependent coefficient and complex number coefficients

```
@mfem.jit.scalar
def scalar_ex(ptx):
    return s_func0(ptx[0], ptx[1],  ptx[2])

@mfem.jit.vector(vdim=3, dependency=(scalar_ex,))
def vector_ex(ptx, param):
    return np.array([0., param, param], dtype=np.float64)

@mfem.jit.matrix(shape=(2,2), dependency=(vector_ex,),
                 td=True, complex=True)
def matrix_ex(ptx, t, param):
    return np.array([[param[0], 1j*param[1]],
                     [param[2],  0j],], dtype=np.complex128)
```

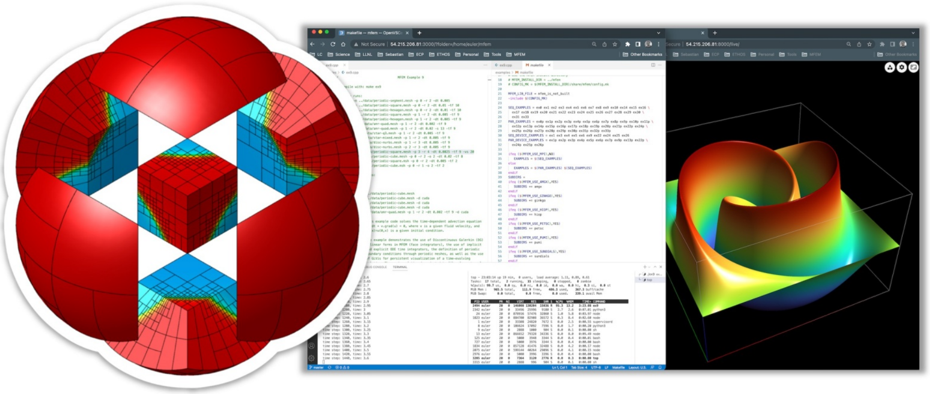
Use @mfem.jit to generate JIT-ed coefficient

LLNL staff joining PyMFEM improvement starting January 2024. Initial scope includes:

- Documentation improvement
- Python example update and refreshing
- Adding missing Pythonic method calls
- … any input for improvement is welcome!

MFEM events and resources

- FEM@LLNL seminar series, online, sign-up at mfem.org/seminar
- Online (cloud) tutorial: mfem.org/tutorial



CASC

Center for Applied
Scientific Computing

Lawrence Livermore
National Laboratory