



FASTMath Unstructured Mesh Technologies

E. Boman¹, V. Dobrev², T. Kolev², J. Merson³, O. Sahni³, J. Nikl²,
M.S. Shephard³, C.W. Smith³, M. Stowell², B. Sturdevant³

¹Sandia National Laboratories

²Lawrence Livermore National Laboratory

³Rensselaer Polytechnic Institute



BERKELEY LAB



Lawrence Livermore
National Laboratory



Sandia
National
Laboratories



Massachusetts
Institute of
Technology



UMBC



USC University of
Southern California

Unstructured Mesh Methods

Unstructured mesh – a spatial domain discretization composed of topological entities with general connectivity and shape

Advantages

- Automatic mesh generation for any level of geometric complexity.
- Can provide the highest accuracy on a per degree of freedom basis
- General mesh anisotropy possible.
- Meshes can easily be adaptively modified.
- Given a complete geometry, with analysis attributes defined on that model, the entire simulation workflow can be automated.

Disadvantages

- More complex data structures and increased program complexity, particularly in parallel.
- Requires careful mesh quality control (level of control required is a function of the unstructured mesh analysis code).
- Poorly shaped elements increase condition number of global system which makes iterative matrix solves slower and harder.
- Non-tensor product elements not as computationally efficient.

Unstructured Mesh Methods

Goal of FASTMath unstructured mesh developments include:

- Provide unstructured mesh components that are easily used by application code developers to extend their simulation capabilities.
- Ensure those components execute on exascale computing systems and support performant exascale application codes.
- Develop components to operate through multi-level APIs that increase interoperability and ease of integration.
- Address technical gaps by developing tools that address needs and/or eliminate/minimize disadvantages of unstructured meshes.
- Work with DOE application developers on integration of these components into their codes.
- Develop unstructured mesh version of applications.

FASTMath Unstructured Mesh Development Areas

- Unstructured Mesh Analysis Codes – Support application's PDE solution needs – MFEM library is the key example
- Performant Mesh Adaptation – Parallel mesh adaptation to integrate into analysis codes to ensure solution accuracy
- Dynamic Load Balancing and Task Management – Technologies to ensure load balance and effectively execute by optimal task placement
- Unstructured Mesh for Particle Codes – Tools to support particle operations on unstructured meshes
- Code Coupling Tools – Parallel geo./mesh/field coupling
- In Situ Vis and Data Analytics – Tools to gain insight as simulations execute
- Unstructured Mesh ML and UQ – ML for data reduction, adaptive mesh UQ

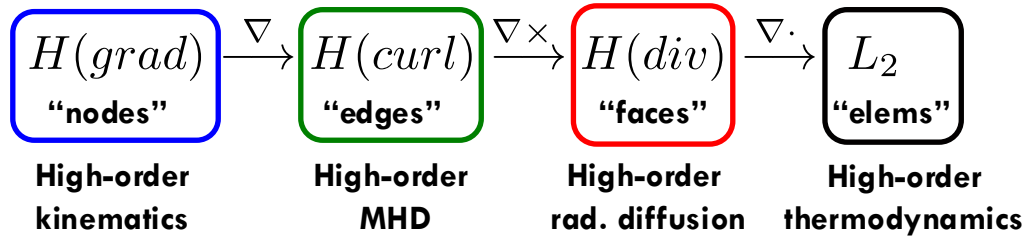
Modular Finite Element Method (MFEM)

MFEM is an open source, scalable C++ library for finite element methods

- MFEM webpage: <https://mfem.org/>
- MFEM gethub page: <https://github.com/mfem/mfem>
- Download available from MFEM main Webpage

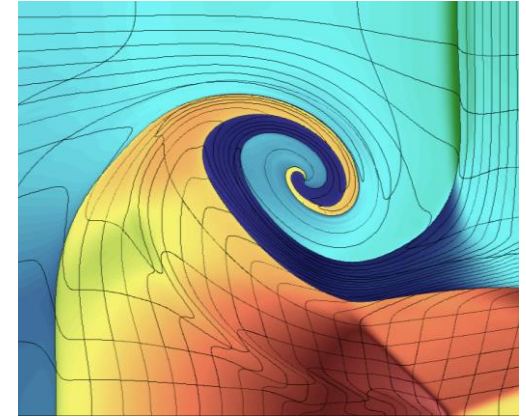
Finite elements are a good foundation for large-scale simulations on current and future architectures

- Backed by well-developed theory
- Naturally support unstructured and curvilinear grids.
- *Finite elements naturally connect different physics*

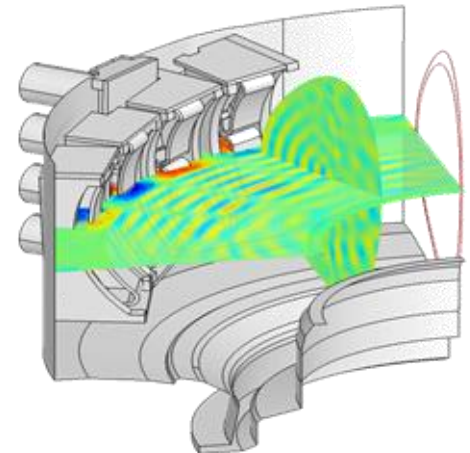


- *High-order finite elements on high-order meshes*
 - increased accuracy for smooth problems
 - sub-element modeling for problems with shocks
 - bridge unstructured/structured grids
 - bridge sparse/dense linear algebra
 - HPC utilization, FLOPs/bytes increase with the order
- *Need new (interesting!) R&D for full benefits*

- meshing, discretizations, solvers, AMR, UQ, visualization, ...



8th order Lagrangian simulation of shock triple-point interaction



Core-Edge tokamak EM wave propagation

Modular Finite Element Methods (MFEM)

Flexible discretizations on unstructured grids

- Triangular, quadrilateral, tetrahedral, hexahedral, prism, pyramid; volume, surface and topologically periodic meshes
- Bilinear/linear forms for: Galerkin methods, DG, HDG, DPG, IGA, ...
- Local conforming and non-conforming AMR, mesh optimization
- Hybridization and static condensation

High-order methods and scalability

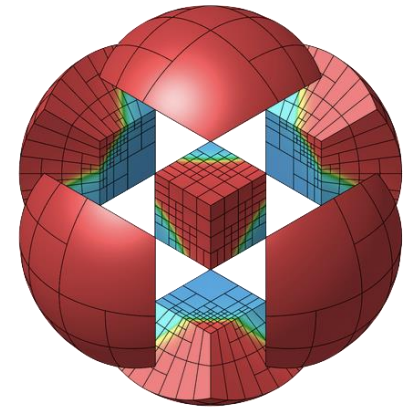
- Arbitrary-order H^1 , $H(\text{curl})$, $H(\text{div})$ - and L^2 elements
- Arbitrary order curvilinear meshes
- MPI scalable to millions of cores + GPU accelerated
- Enables development from laptops to exascale machines.

Solvers and preconditioners

- Integrated with: HYPRE, SUNDIALS, PETSc, SLEPc, SUPERLU, VisIt, ...
- AMG solvers for full de Rham complex on CPU+GPU, geometric MG
- Time integrators: SUNDIALS, PETSc, built-in RK, SDIRK, ...

Open-source software

- Open-source (GitHub) with 114 contributors, 50 clones/day
- Part of FASTMath, ECP/CEED, xSDK, OpenHPC, E4S, ...
- 75+ example codes & miniapps: mfem.org/examples



mfem.org

(v4.9, December 2025)



Example 1 – Laplace equation

■ Mesh

```
63 // 2. Read the mesh from the given mesh file. We can handle triangular,
64 // quadrilateral, tetrahedral, hexahedral, surface and volume meshes with
65 // the same code.
66 Mesh *mesh;
67 ifstream imesh(mesh_file);
68 if (!imesh)
69 {
70     cerr << "Can not open mesh file: " << mesh_file << '\n' << endl;
71     return 2;
72 }
73 mesh = new Mesh(imesh, 1, 1);
74 imesh.close();
75 int dim = mesh->Dimension();
76
77 // 3. Refine the mesh to increase the resolution. In this example we do
78 // 'ref_levels' of uniform refinement. We choose 'ref_levels' to be the
79 // largest number that gives a final mesh with no more than 50,000
80 // elements.
81 {
82     int ref_levels =
83         (int)floor(log(50000./mesh->GetNE())/log(2.)/dim);
84     for (int l = 0; l < ref_levels; l++)
85         mesh->UniformRefinement();
86 }
```

■ Finite element space

```
88 // 4. Define a finite element space on the mesh. Here we use continuous
89 // Lagrange finite elements of the specified order. If order < 1, we
90 // instead use an isoparametric/isogeometric space.
91 FiniteElementCollection *fec;
92 if (order > 0)
93     fec = new H1_FECollection(order, dim);
94 else if (mesh->GetNodes())
95     fec = mesh->GetNodes()->OwnFEC();
96 else
97     fec = new H1_FECollection(order = 1, dim);
98 FiniteElementSpace *fespace = new FiniteElementSpace(mesh, fec);
99 cout << "Number of unknowns: " << fespace->GetVSize() << endl;
```

■ Initial guess, linear/bilinear forms

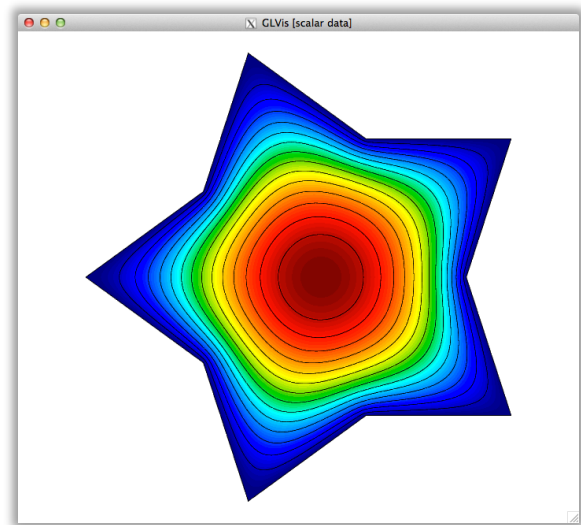
```
101 // 5. Set up the linear form b(.) which corresponds to the right-hand side of
102 // the FEM linear system, which in this case is (1,phi_i) where phi_i are
103 // the basis functions in the finite element fespace.
104 LinearForm *b = new LinearForm(fespace);
105 ConstantCoefficient one(1.0);
106 b->AddDomainIntegrator(new DomainLFIntegrator(one));
107 b->Assemble();
108
109 // 6. Define the solution vector x as a finite element grid function
110 // corresponding to fespace. Initialize x with initial guess of zero,
111 // which satisfies the boundary conditions.
112 GridFunction x(fespace);
113 x = 0.0;
114
115 // 7. Set up the bilinear form a(...) on the finite element space
116 // corresponding to the Laplacian operator -Delta, by adding the Diffusion
117 // domain integrator and imposing homogeneous Dirichlet boundary
118 // conditions. The boundary conditions are implemented by marking all the
119 // boundary attributes from the mesh as essential (Dirichlet). After
120 // assembly and finalizing we extract the corresponding sparse matrix A.
121 BilinearForm *a = new BilinearForm(fespace);
122 a->AddDomainIntegrator(new DiffusionIntegrator(one));
123 a->Assemble();
124 Array<int> ess_bdr(mesh->bdr_attributes.Max());
125 ess_bdr = 1;
126 a->EliminateEssentialBC(ess_bdr, x, *b);
127 a->Finalize();
128 const SparseMatrix &A = a->SpMat();
```

■ Linear solve

```
130 #ifndef MFEM_USE_SUITESPARSE
131 // 8. Define a simple symmetric Gauss-Seidel preconditioner and use it to
132 // solve the system Ax=b with PCG.
133 GSSm smoother M(A);
134 PCG(A, M, *b, x, 1, 200, 1e-12, 0.0);
135 #else
136 // 8. If MFEM was compiled with SuiteSparse, use UMFPACK to solve the system.
137 UMFPackSolver umf_solver;
138 umf_solver.Control[UMFPACK_ORDERING] = UMFPACK_ORDERING_METIS;
139 umf_solver.SetOperator(A);
140 umf_solver.Mult(*b, x);
141 #endif
```

■ Visualization

```
152 // 10. Send the solution by socket to a GLVis server.
153 if (visualization)
154 {
155     char vishost[] = "localhost";
156     int visport = 19916;
157     socketstream sol_sock(vishost, visport);
158     sol_sock.precision(8);
159     sol_sock << "solution\n" << *mesh << x << flush;
160 }
```



- works for any mesh & any H1 order
- builds without external dependencies

Example 1 – Laplace equation

- Mesh

```
63 // 2. Read the mesh from the given mesh file. We can handle triangular,
64 // quadrilateral, tetrahedral, hexahedral, surface and volume meshes with
65 // the same code.
66 Mesh *mesh;
67 ifstream imesh(mesh_file);
68 if (!imesh)
69 {
70     cerr << "\nCan not open mesh file: " << mesh_file << '\n' << endl;
71     return 2;
72 }
73 mesh = new Mesh(imesh, 1, 1);
74 imesh.close();
75 int dim = mesh->Dimension();
76
77 // 3. Refine the mesh to increase the resolution. In this example we do
78 // 'ref_levels' of uniform refinement. We choose 'ref_levels' to be the
79 // largest number that gives a final mesh with no more than 50,000
80 // elements.
81 {
82     int ref_levels =
83         (int)floor(log(50000./mesh->GetNE())/log(2.)/dim);
84     for (int l = 0; l < ref_levels; l++)
85         mesh->UniformRefinement();
86 }
```

Example 1 – Laplace equation

- Finite element space

```
88 // 4. Define a finite element space on the mesh. Here we use continuous
89 // Lagrange finite elements of the specified order. If order < 1, we
90 // instead use an isoparametric/isogeometric space.
91 FiniteElementCollection *fec;
92 if (order > 0)
93     fec = new H1_FECollection(order, dim);
94 else if (mesh->GetNodes())
95     fec = mesh->GetNodes()->OwnFEC();
96 else
97     fec = new H1_FECollection(order = 1, dim);
98 FiniteElementSpace *fespace = new FiniteElementSpace(mesh, fec);
99 cout << "Number of unknowns: " << fespace->GetVSize() << endl;
```

Example 1 – Laplace equation

- Initial guess, linear/bilinear forms

```
101 // 5. Set up the linear form b(.) which corresponds to the right-hand side of
102 // the FEM linear system, which in this case is (1,phi_i) where phi_i are
103 // the basis functions in the finite element fespace.
104 LinearForm *b = new LinearForm(fespace);
105 ConstantCoefficient one(1.0);
106 b->AddDomainIntegrator(new DomainLFIntegrator(one));
107 b->Assemble();
108
109 // 6. Define the solution vector x as a finite element grid function
110 // corresponding to fespace. Initialize x with initial guess of zero,
111 // which satisfies the boundary conditions.
112 GridFunction x(fespace);
113 x = 0.0;
114
115 // 7. Set up the bilinear form a(.,.) on the finite element space
116 // corresponding to the Laplacian operator -Delta, by adding the Diffusion
117 // domain integrator and imposing homogeneous Dirichlet boundary
118 // conditions. The boundary conditions are implemented by marking all the
119 // boundary attributes from the mesh as essential (Dirichlet). After
120 // assembly and finalizing we extract the corresponding sparse matrix A.
121 BilinearForm *a = new BilinearForm(fespace);
122 a->AddDomainIntegrator(new DiffusionIntegrator(one));
123 a->Assemble();
124 Array<int> ess_bdr(mesh->bdr_attributes.Max());
125 ess_bdr = 1;
126 a->EliminateEssentialBC(ess_bdr, x, *b);
127 a->Finalize();
128 const SparseMatrix &A = a->SpMat();
```

Example 1 – Laplace equation

- Linear solve

```
130 #ifndef MFEM_USE_SUITESPARSE
131 // 8. Define a simple symmetric Gauss-Seidel preconditioner and use it to
132 // solve the system Ax=b with PCG.
133 GSSmoothen M(A);
134 PCG(A, M, *b, x, 1, 200, 1e-12, 0.0);
135 #else
136 // 8. If MFEM was compiled with SuiteSparse, use UMFPACK to solve the system.
137 UMFPackSolver umf_solver;
138 umf_solver.Control[UMFPACK_ORDERING] = UMFPACK_ORDERING_METIS;
139 umf_solver.SetOperator(A);
140 umf_solver.Mult(*b, x);
141 #endif
```

- Visualization

```
152 // 10. Send the solution by socket to a GLVis server.
153 if (visualization)
154 {
155     char vishost[] = "localhost";
156     int visport = 19916;
157     socketstream sol_sock(vishost, visport);
158     sol_sock.precision(8);
159     sol_sock << "solution\n" << *mesh << x << flush;
160 }
```

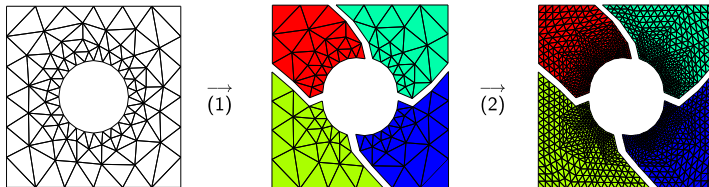
Example 1 – parallel Laplace equation

Parallel mesh

```

101 // 5. Define a parallel mesh by a partitioning of the serial mesh. Refine
102 // this mesh further in parallel to increase the resolution. Once the
103 // parallel mesh is defined, the serial mesh can be deleted.
104 ParMesh *pmesh = new ParMesh(MPI_COMM_WORLD, *mesh);
105 delete mesh;
106 {
107     int par_ref_levels = 2;
108     for (int l = 0; l < par_ref_levels; l++)
109         pmesh->UniformRefinement();
110 }

```

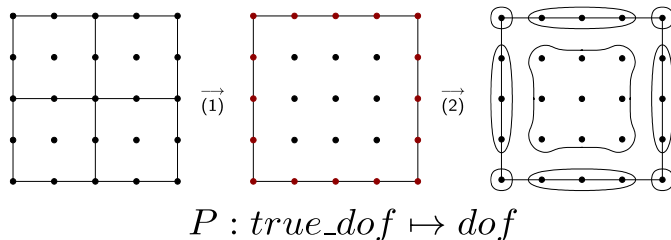


Parallel finite element space

```

122 ParFiniteElementSpace *fespace = new ParFiniteElementSpace(pmesh, fec);

```



Parallel initial guess, linear/bilinear forms

```

130 ParLinearForm *b = new ParLinearForm(fespace);
138 ParGridFunction x(fespace);
147 ParBilinearForm *a = new ParBilinearForm(fespace);

```

Parallel assembly

```

155 // 10. Define the parallel (hypr) matrix and vectors representing a(..),
156 // b(..) and the finite element approximation.
157 HyprParMatrix *A = a->ParallelAssemble();
158 HyprParVector *B = b->ParallelAssemble();
159 HyprParVector *X = x.ParallelAverage();

```

$$A = P^T a P \quad B = P^T b \quad x = P X$$

Parallel linear solve with AMG

```

164 // 11. Define and apply a parallel PCG solver for AX=B with the BoomerAMG
165 // preconditioner from hypre.
166 HyprSolver *amg = new HyprBoomerAMG(*A);
167 HyprPCG *pcg = new HyprPCG(*A);
168 pcg->SetTol(1e-12);
169 pcg->SetMaxIter(200);
170 pcg->SetPrintLevel(2);
171 pcg->SetPreconditioner(*amg);
172 pcg->Mult(*B, *X);

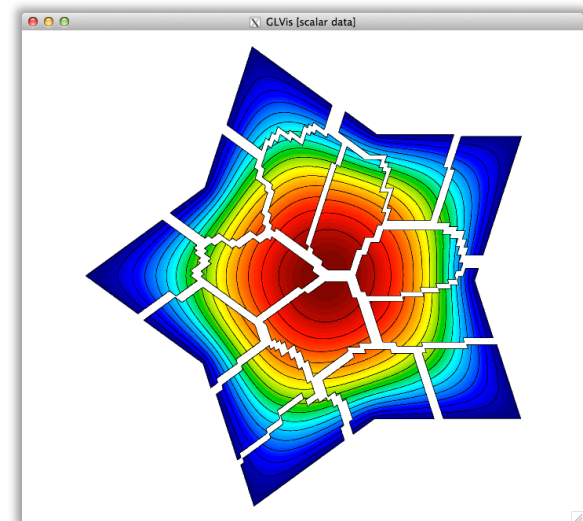
```

Visualization

```

194 // 14. Send the solution by socket to a GLVis server.
195 if (visualization)
196 {
197     char vishost[] = "localhost";
198     int visport = 19916;
199     socketstream sol_sock(vishost, visport);
200     sol_sock << "parallel " << num_procs << " " << myid << "\n";
201     sol_sock.precision(8);
202     sol_sock << "solution\n" << *pmesh << x << flush;
203 }

```



- highly scalable with minimal changes
- build depends on *hypr* and METIS

Example 1 – parallel Laplace equation

```
101 // 5. Define a parallel mesh by a partitioning of the serial mesh. Refine
102 // this mesh further in parallel to increase the resolution. Once the
103 // parallel mesh is defined, the serial mesh can be deleted.
104 ParMesh *pmesh = new ParMesh(MPI_COMM_WORLD, *mesh);
105 delete mesh;
106 {
107     int par_ref_levels = 2;
108     for (int l = 0; l < par_ref_levels; l++)
109         pmesh->UniformRefinement();
110 }

122 ParFiniteElementSpace *fespace = new ParFiniteElementSpace(pmesh, fec);
130 ParLinearForm *b = new ParLinearForm(fespace);
138 ParGridFunction x(fespace);
147 ParBilinearForm *a = new ParBilinearForm(fespace);

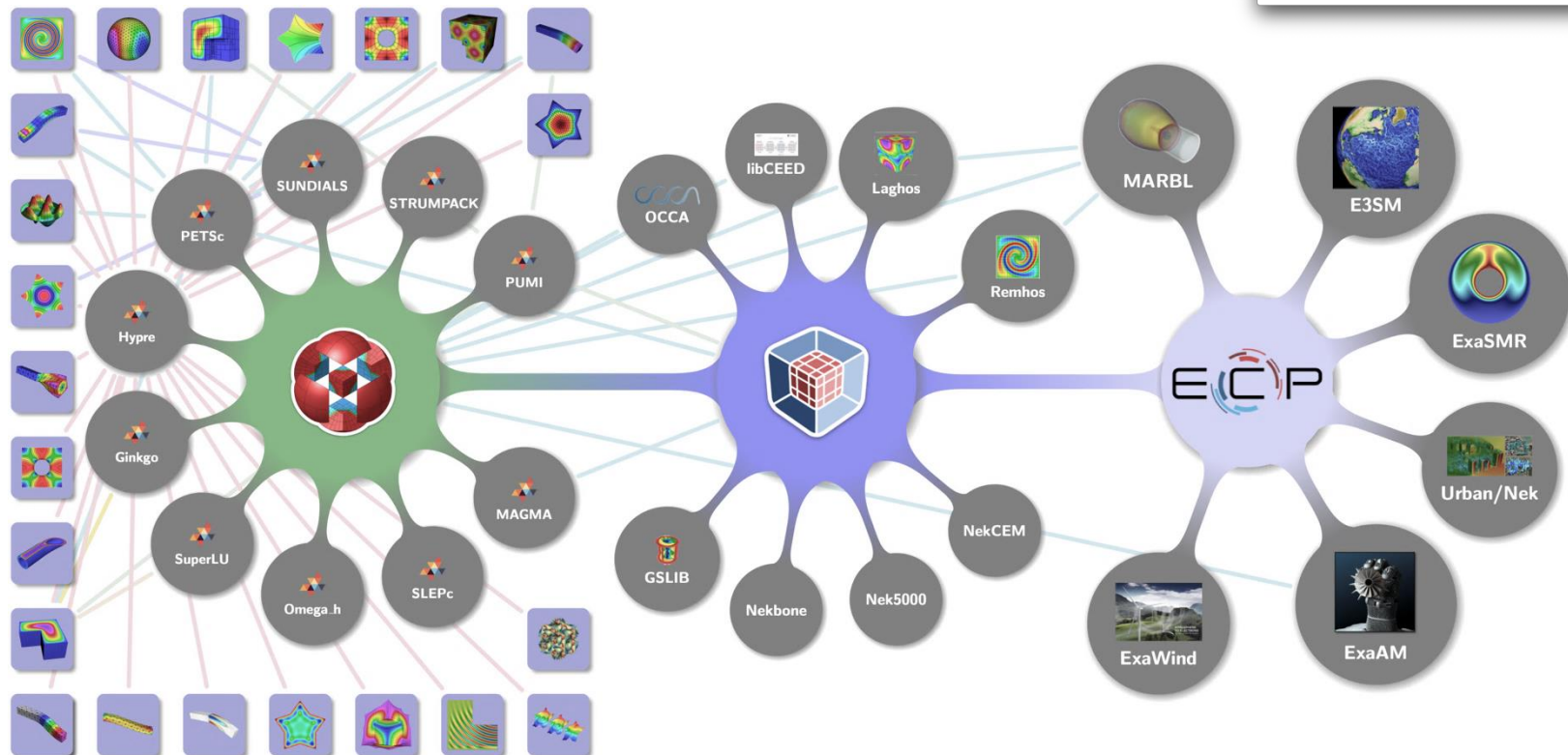
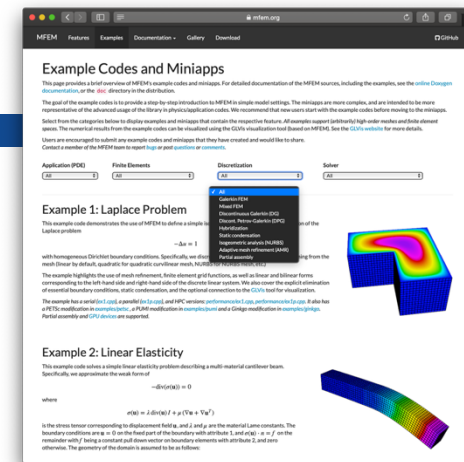
155 // 10. Define the parallel (hypre) matrix and vectors representing a(...),
156 // b(.) and the finite element approximation.
157 HypreParMatrix *A = a->ParallelAssemble();
158 HypreParVector *B = b->ParallelAssemble();
159 HypreParVector *X = x.ParallelAverage();

164 // 11. Define and apply a parallel PCG solver for AX=B with the BoomerAMG
165 // preconditioner from hypre.
166 HypreSolver *amg = new HypreBoomerAMG(*A);
167 HyprePCG *pcg = new HyprePCG(*A);
168 pcg->SetTol(1e-12);
169 pcg->SetMaxIter(200);
170 pcg->SetPrintLevel(2);
171 pcg->SetPreconditioner(*amg);
172 pcg->Mult(*B, *X);

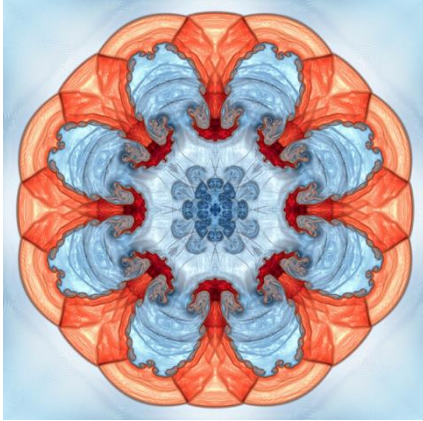
200 sol_sock << "parallel " << num_procs << " " << myid << "\n";
201 sol_sock.precision(8);
202 sol_sock << "solution\n" << *pmesh << x << flush;
```

MFEM example codes: mfem.org/examples

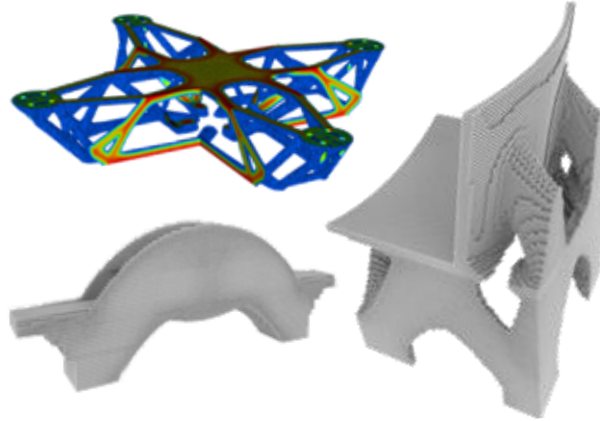
- 40+ example codes, most with both serial + parallel versions
- Tutorials to learn MFEM features
- Starting point for new applications
- Show integration with many external packages
- Miniapps: more advanced, ready-to-use physics solvers



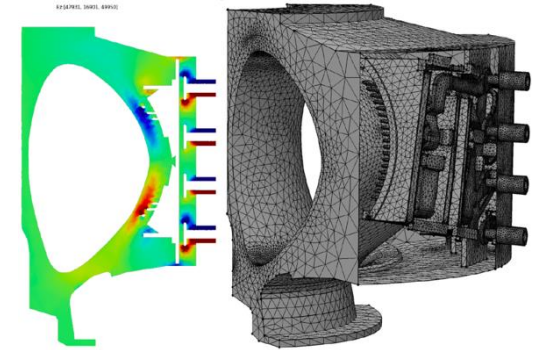
Some large-scale simulation codes powered by MFEM



Inertial confinement fusion (BLAST)



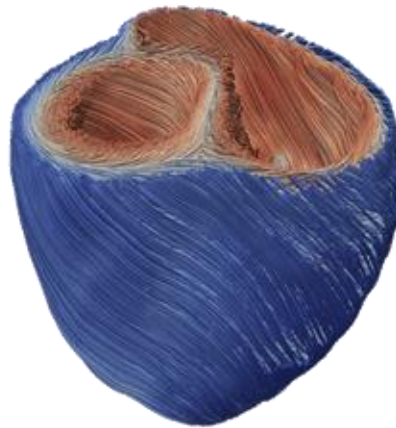
Topology optimization for additive manufacturing (LiDO)



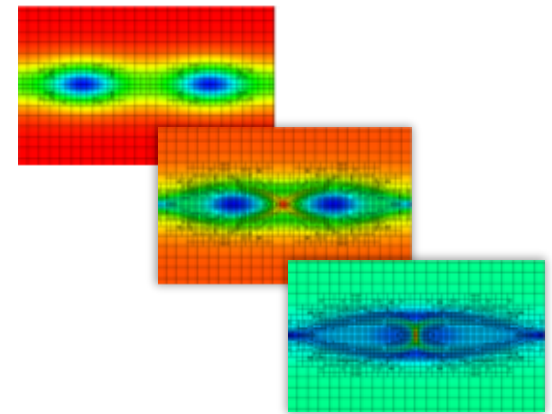
Core-edge tokamak EM wave propagation (SciDAC, RPI)



MRI modeling (Harvard Medical)



Heart modeling (Cardioid)



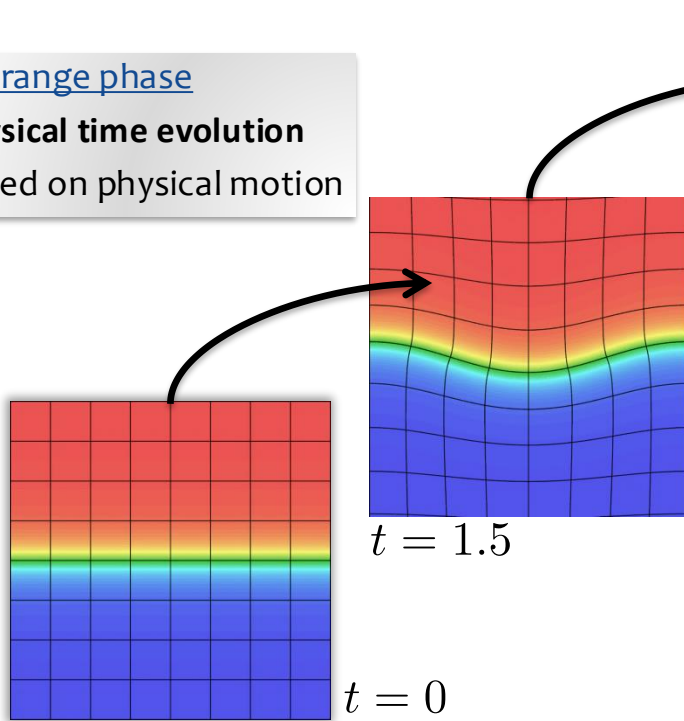
Adaptive MHD island coalescence (SciDAC, LANL)

BLAST models shock hydrodynamics using high-order FEM in both Lagrangian and Remap phases of ALE

Lagrange phase

Physical time evolution

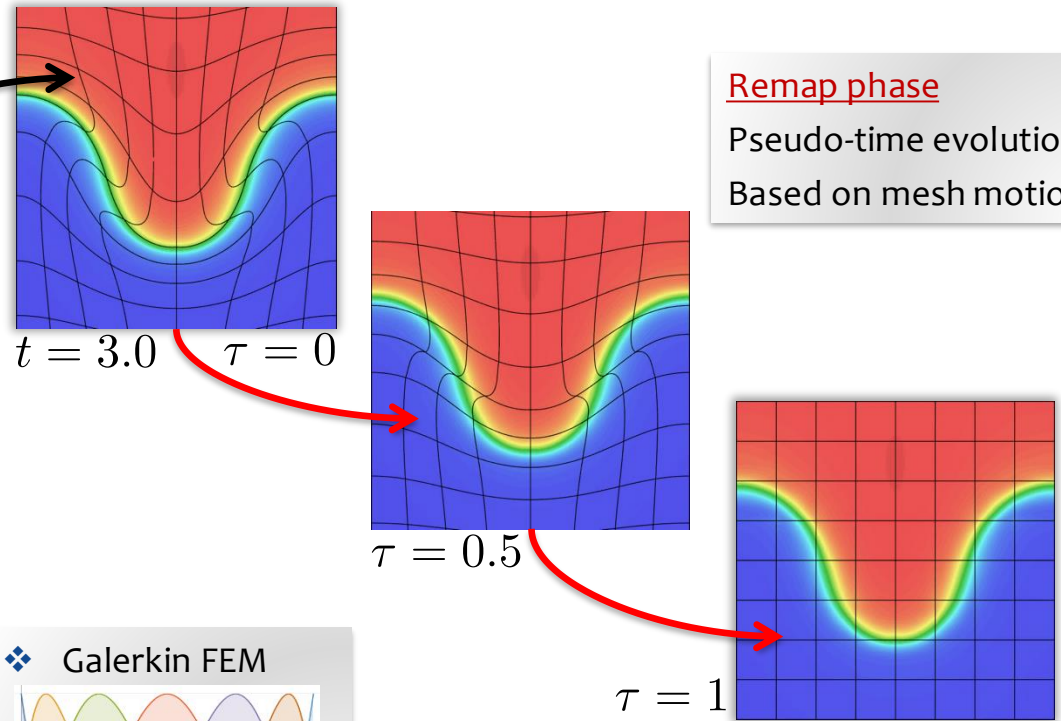
Based on physical motion



Remap phase

Pseudo-time evolution

Based on mesh motion



Lagrangian phase ($\vec{c} = \vec{0}$)

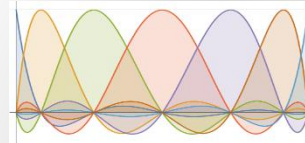
Momentum Conservation: $\rho \frac{d\vec{v}}{dt} = \nabla \cdot \sigma$

Mass Conservation: $\frac{d\rho}{dt} = -\rho \nabla \cdot \vec{v}$

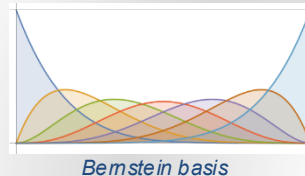
Energy Conservation: $\rho \frac{de}{dt} = \sigma : \nabla \vec{v}$

Equation of Motion: $\frac{d\vec{x}}{dt} = \vec{v}$

❖ Galerkin FEM



❖ Discont. Galerkin



Advection phase ($\vec{c} = -\vec{v}_m$)

Momentum Conservation: $\frac{d(\rho\vec{v})}{d\tau} = \vec{v}_m \cdot \nabla(\rho\vec{v})$

Mass Conservation: $\frac{d\rho}{d\tau} = \vec{v}_m \cdot \nabla \rho$

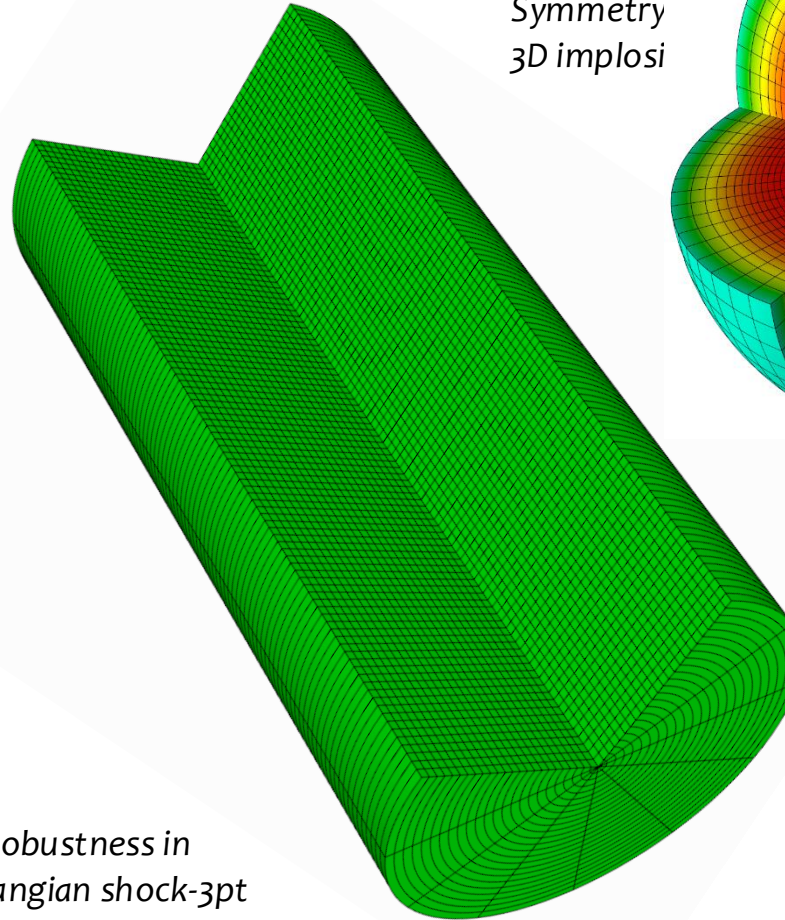
Energy Conservation: $\frac{d(\rho e)}{d\tau} = \vec{v}_m \cdot \nabla(\rho e)$

Mesh velocity: $\vec{v}_m = \frac{d\vec{x}}{d\tau}$

High-order finite elements lead to more accurate, robust and reliable hydrodynamic simulations

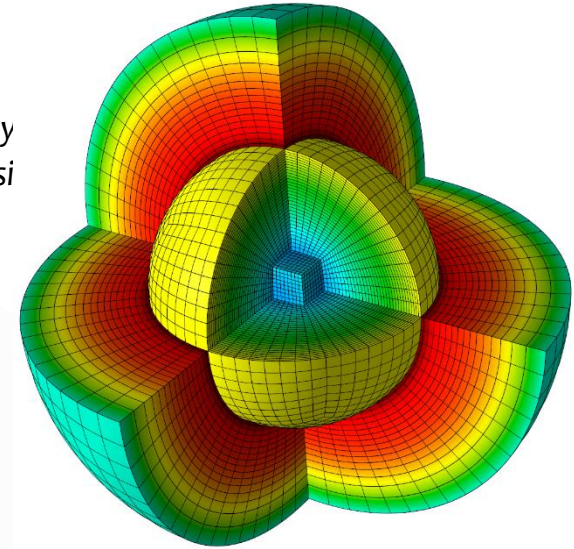


Parallel ALE for Q4 Rayleigh-Taylor instability (256 cores)

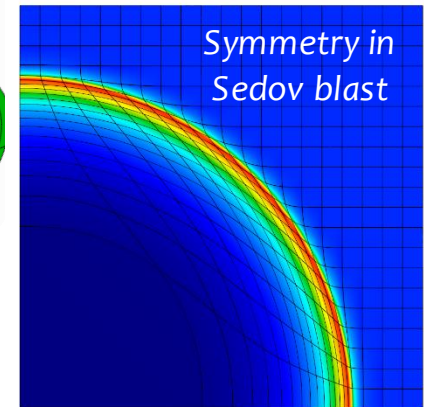


Robustness in
Lagrangian shock-3pt
axisymm. interaction

Symmetry
3D implosi

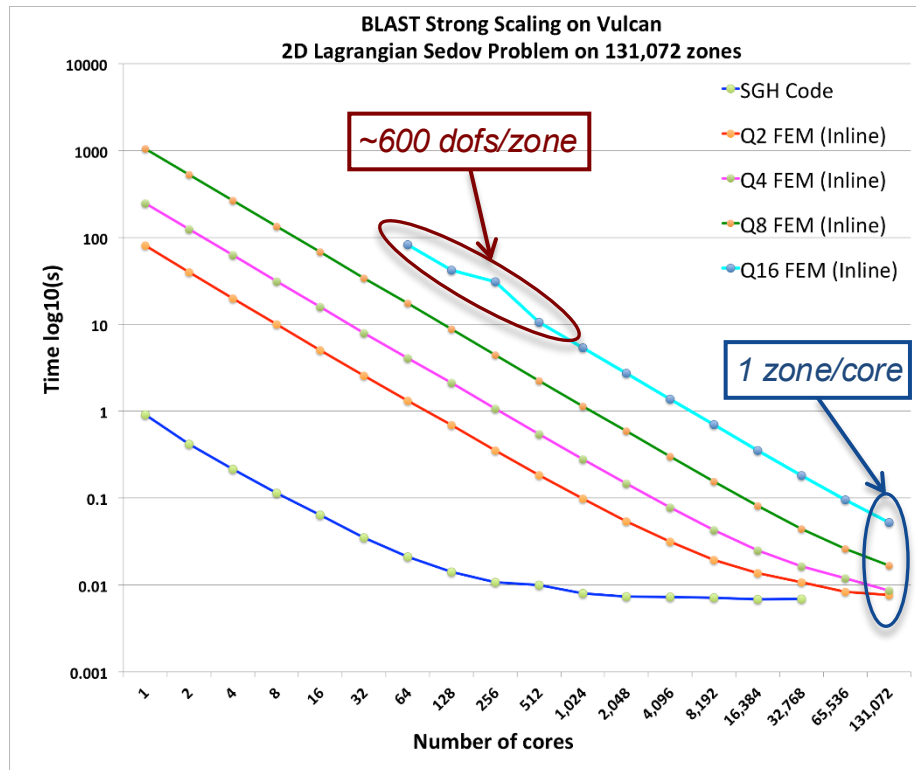


Symmetry in
Sedov blast



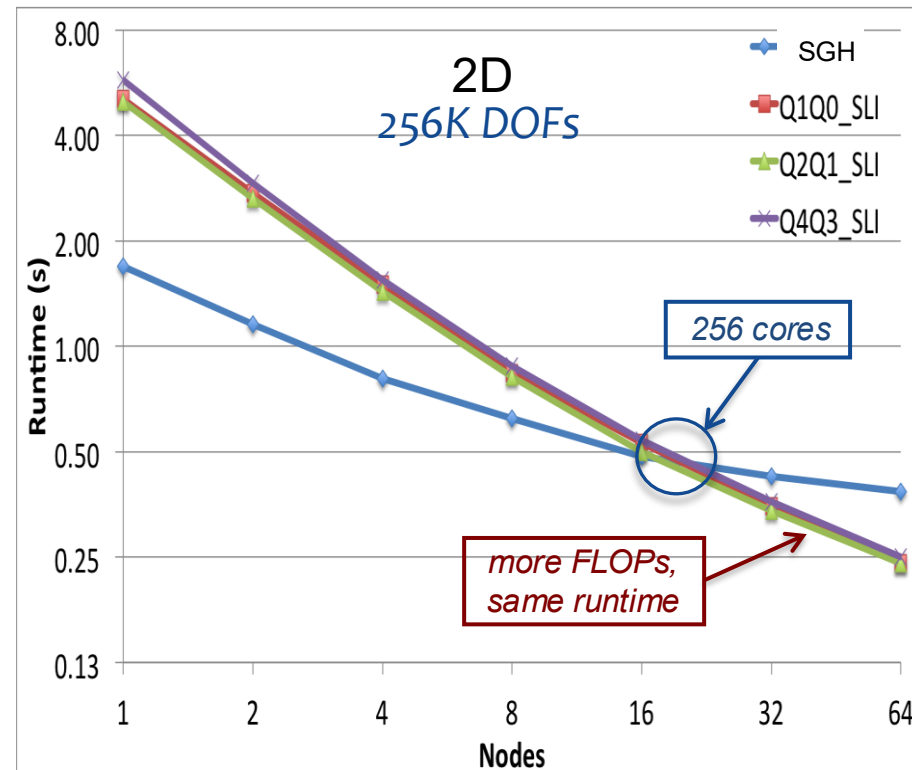
High-order finite elements have excellent strong scalability

Strong scaling, p -refinement



Finite element partial assembly

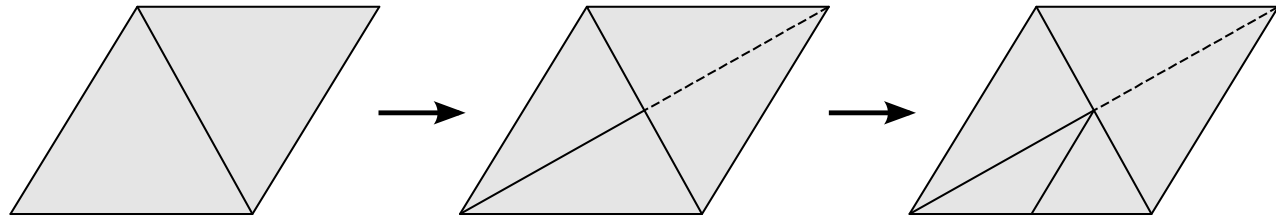
Strong scaling, fixed #dofs



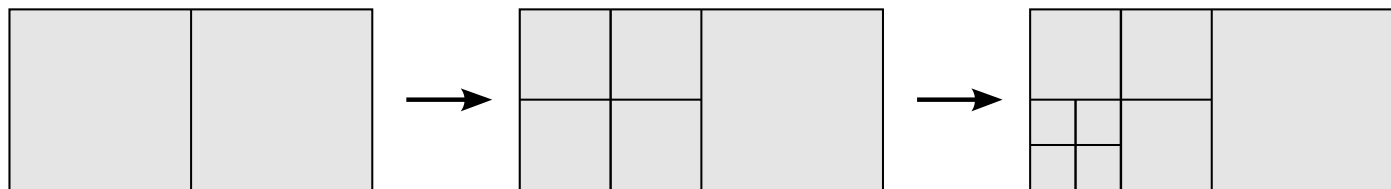
FLOPs increase faster than runtime

Conforming & Nonconforming Mesh Refinement

■ Conforming refinement



■ Nonconforming refinement



■ Natural for quadrilaterals and hexahedra

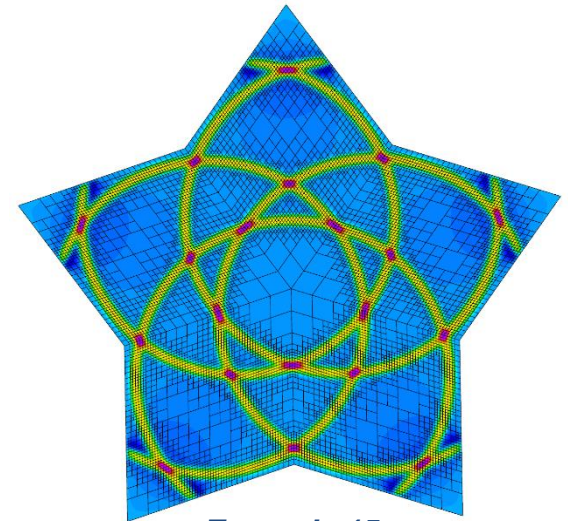
MFEM's unstructured AMR infrastructure

Adaptive mesh refinement on library level:

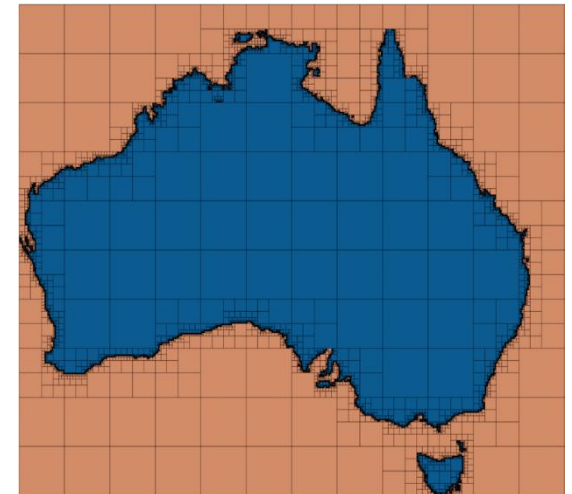
- Conforming local refinement on simplex meshes
- *Non-conforming refinement for quad/hex meshes*
- h-refinement with fixed p

General approach:

- any high-order finite element space, H_1 , $H(\text{curl})$, $H(\text{div})$, ..., on any high-order curved mesh
- 2D and 3D
- arbitrary order hanging nodes
- anisotropic refinement
- derefinement
- serial and parallel, including parallel load balancing
- independent of the physics (easy to incorporate in applications)



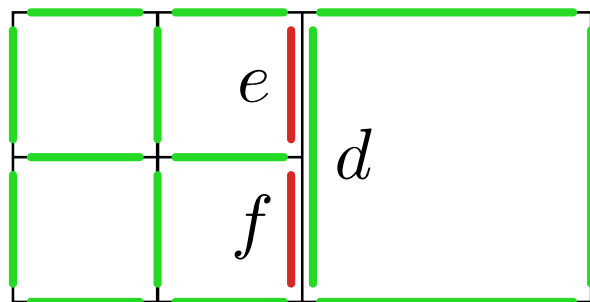
Example 15



Shaper miniapp

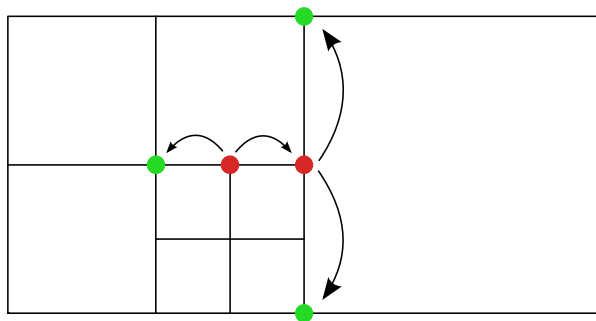
General nonconforming constraints

H(curl) elements



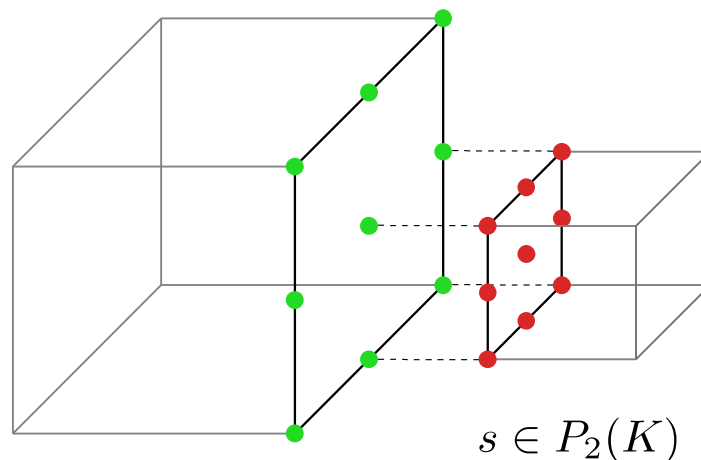
Constraint: $e = f = d/2$

Indirect constraints



More complicated in 3D...

High-order elements



$m \in P_2(K)$

$s \in P_2(K)$

Constraint: local interpolation matrix

$$s = Q \cdot m, \quad Q \in \mathbb{R}^{9 \times 9}$$

Nonconforming variational restriction

- General constraint:

$$y = Px, \quad P = \begin{bmatrix} I \\ W \end{bmatrix}.$$

x – conforming space DOFs,

y – nonconforming space DOFs (unconstrained + slave),

$$\dim(x) \leq \dim(y)$$

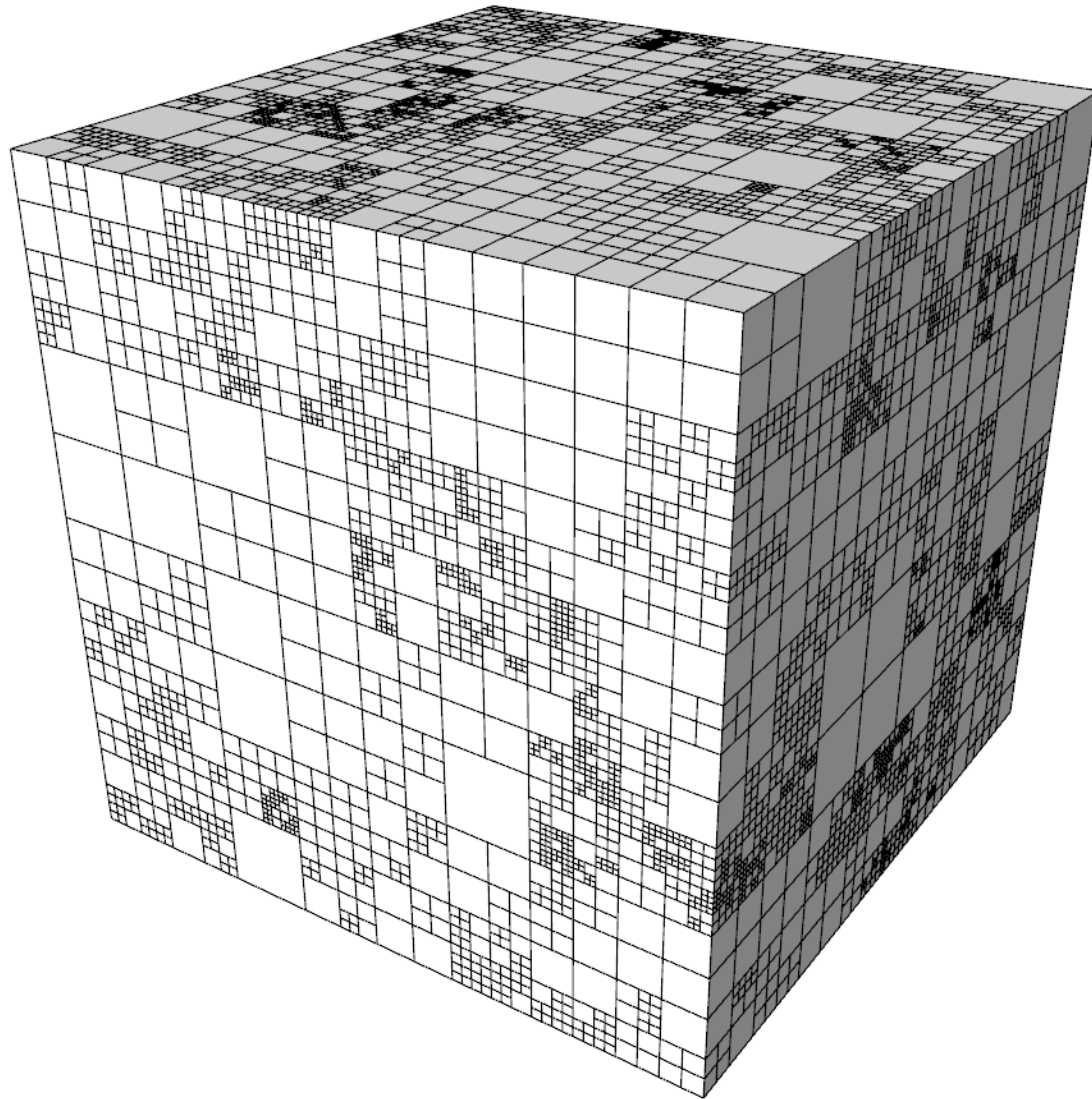
W – interpolation for slave DOFs

- Constrained problem:

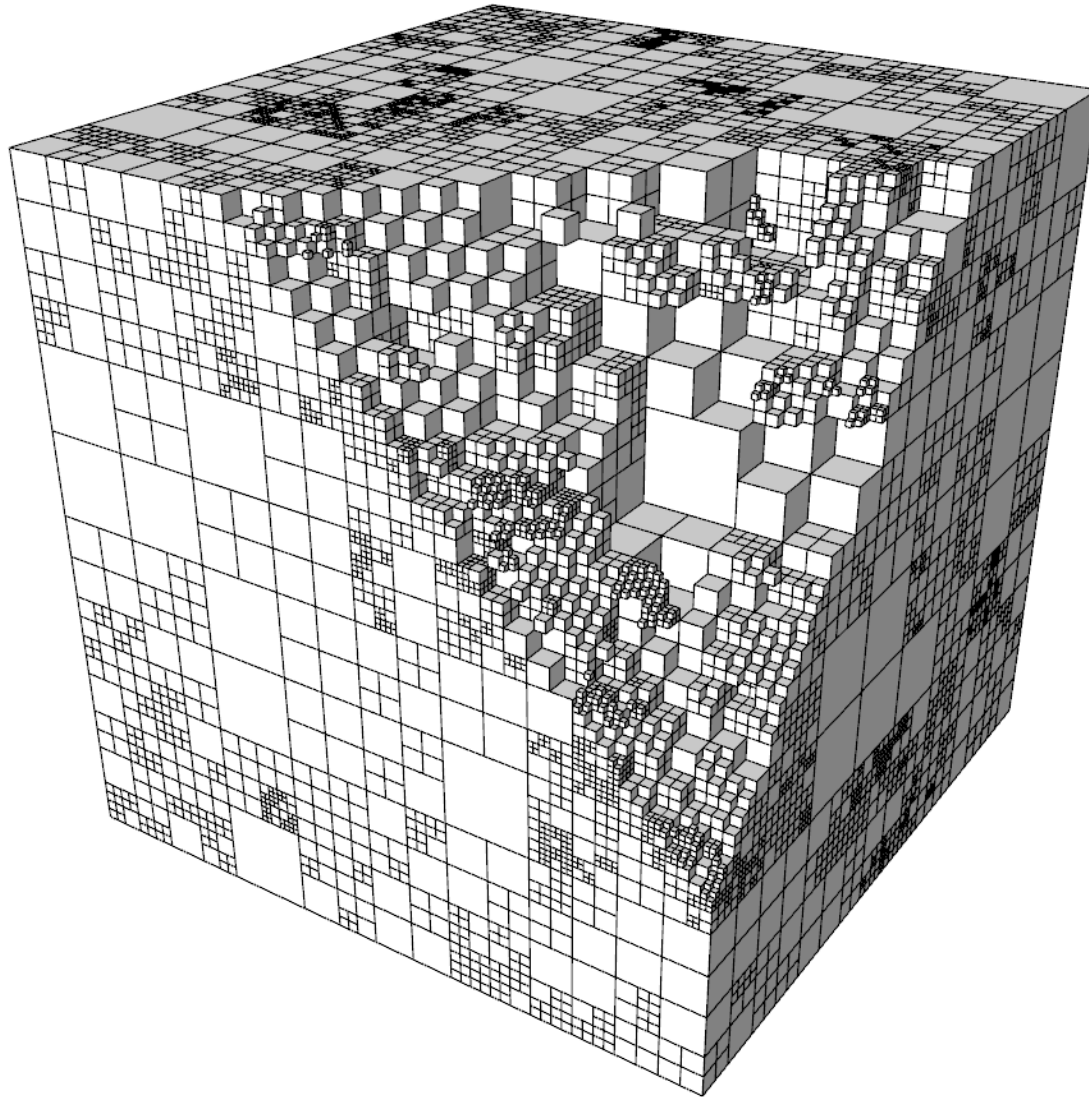
$$P^T A P x = P^T b,$$

$$y = Px.$$

Nonconforming variational restriction

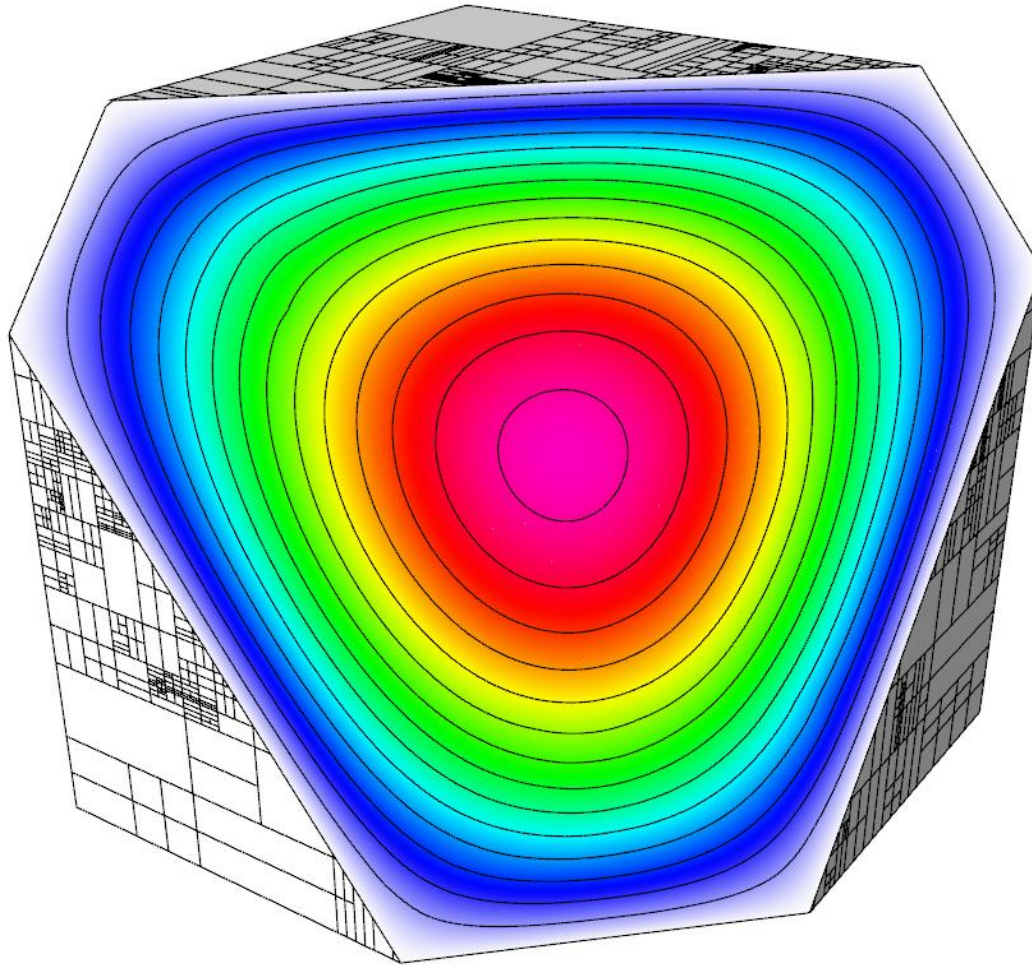


Nonconforming variational restriction



Regular assembly of A on the elements of the (cut) mesh

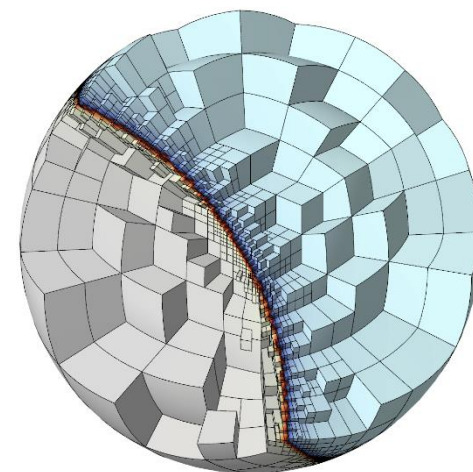
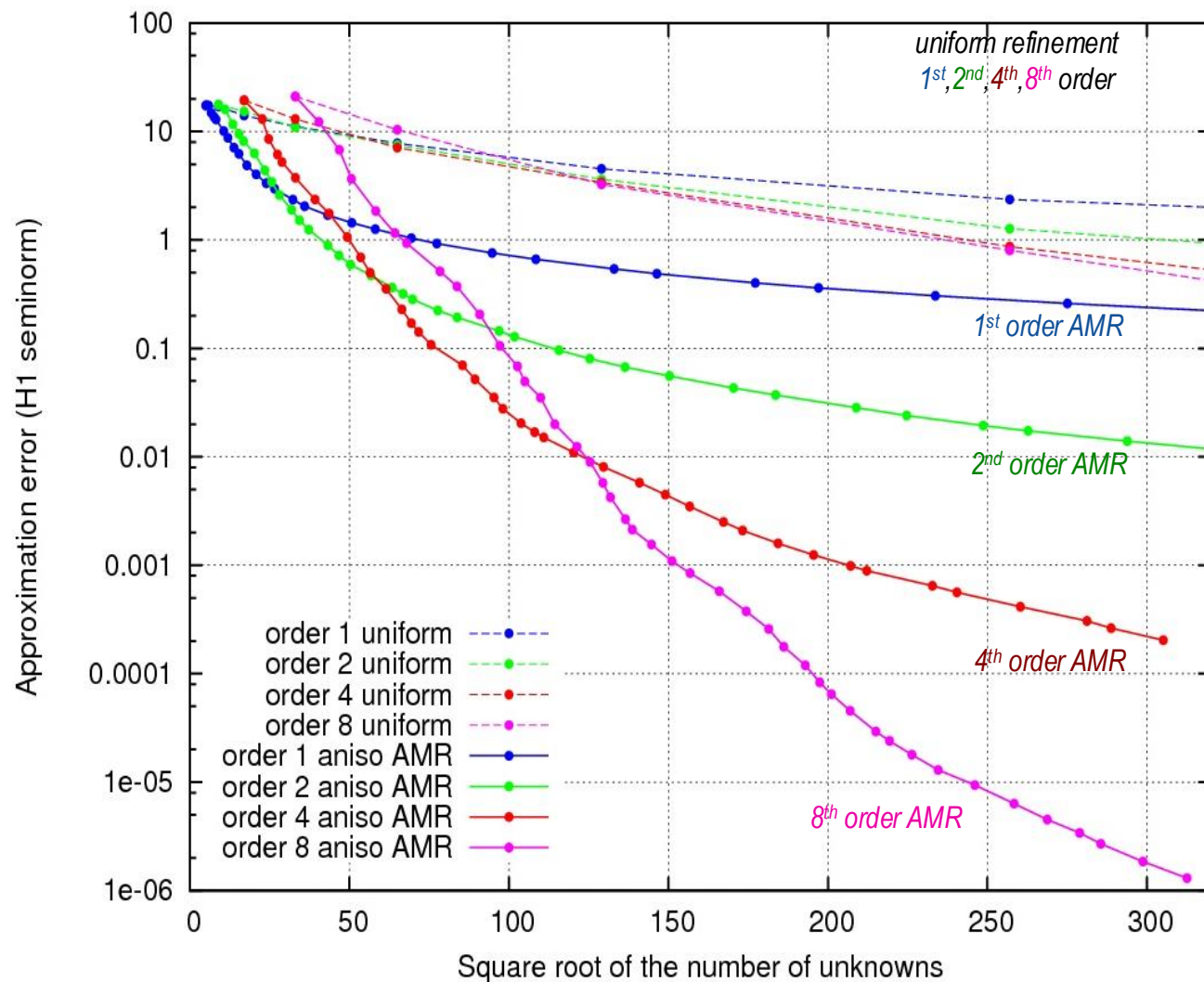
Nonconforming variational restriction



Conforming solution $y = P x$

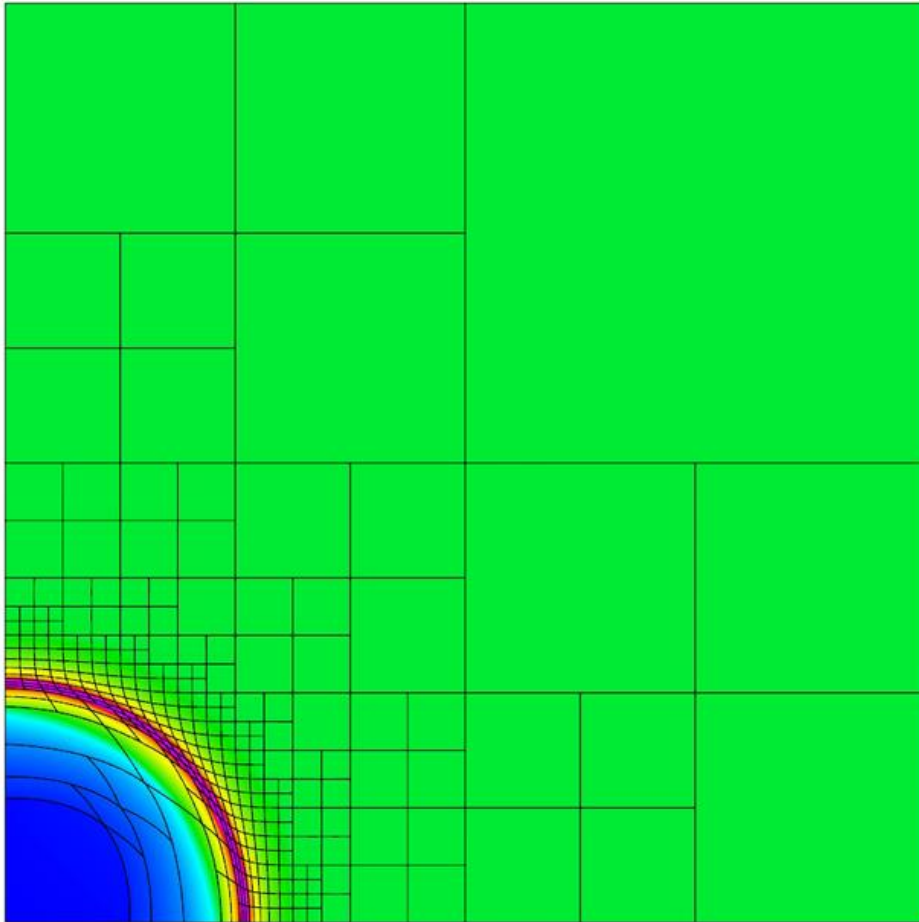
AMR = smaller error for same number of unknowns

2D Shock-like Problem AMR Benchmark (Quad Mesh, Anisotropic Refinements)

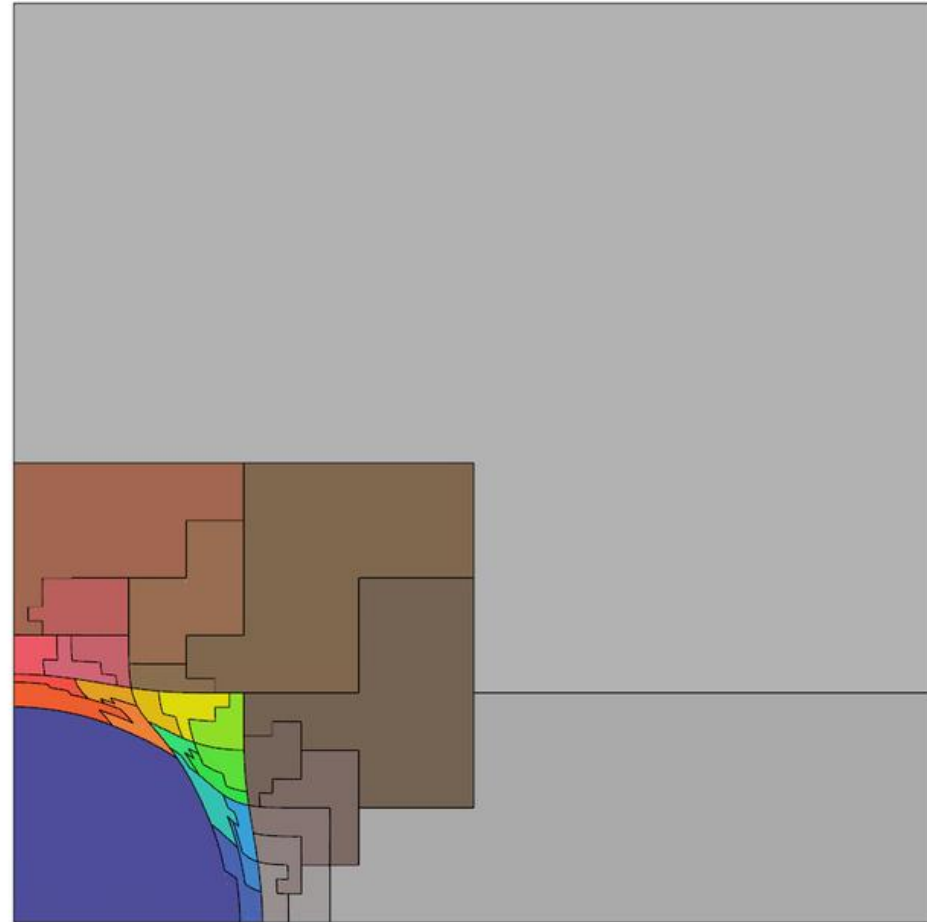


Anisotropic adaptation to shock-like fields in 2D & 3D

Parallel dynamic AMR, Lagrangian Sedov problem

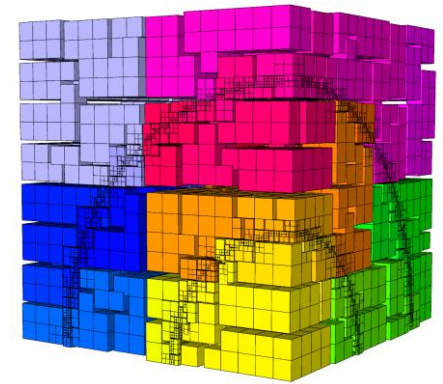
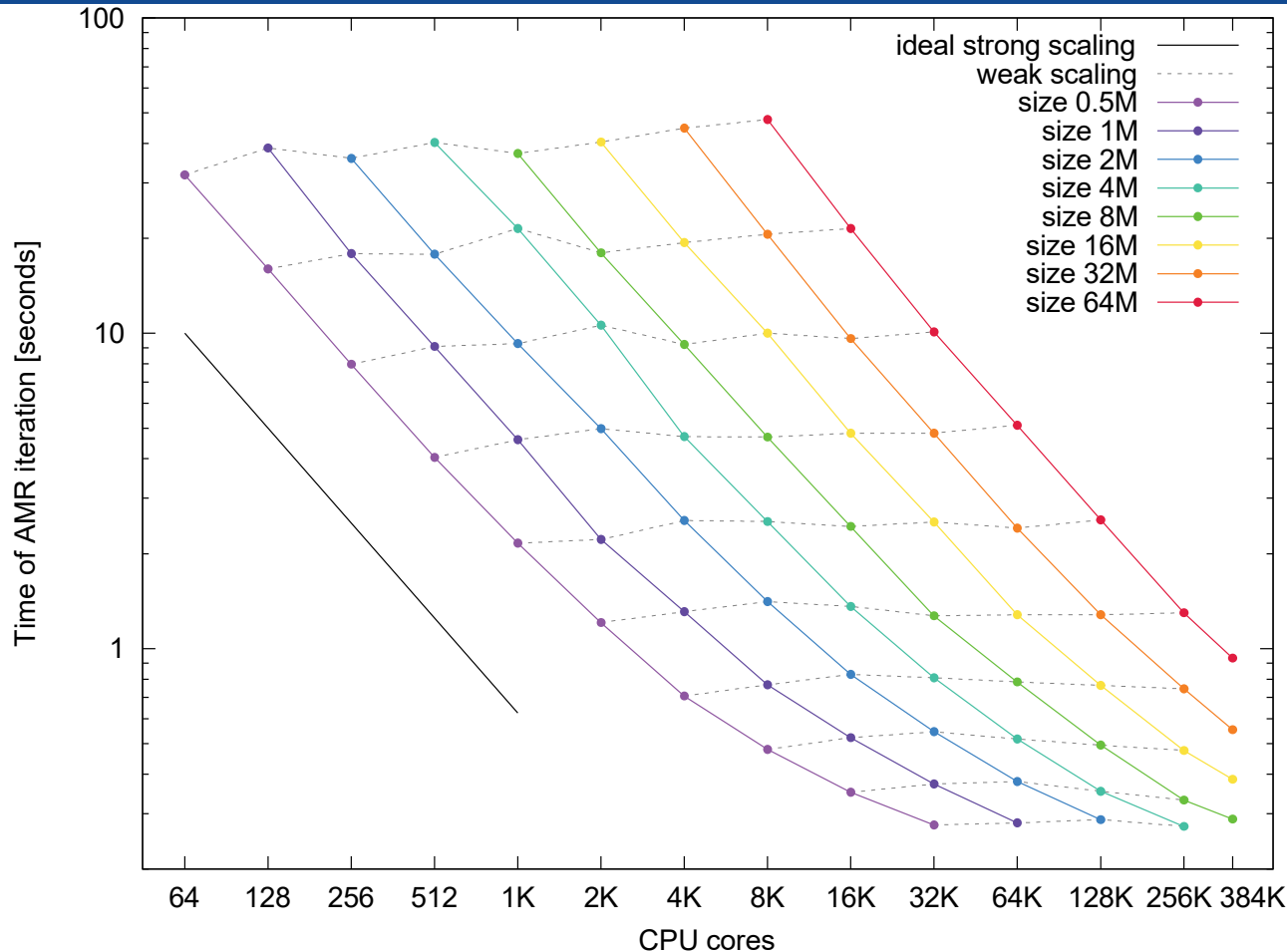


Adaptive, viscosity-based refinement and derefinement. 2nd order Lagrangian Sedov

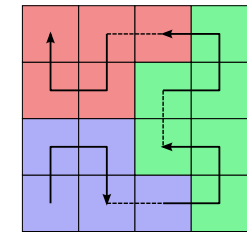


Parallel load balancing based on space-filling curve partitioning, 16 cores

Parallel AMR scaling to ~400K MPI tasks



*Parallel decomposition
(2048 domains shown)*



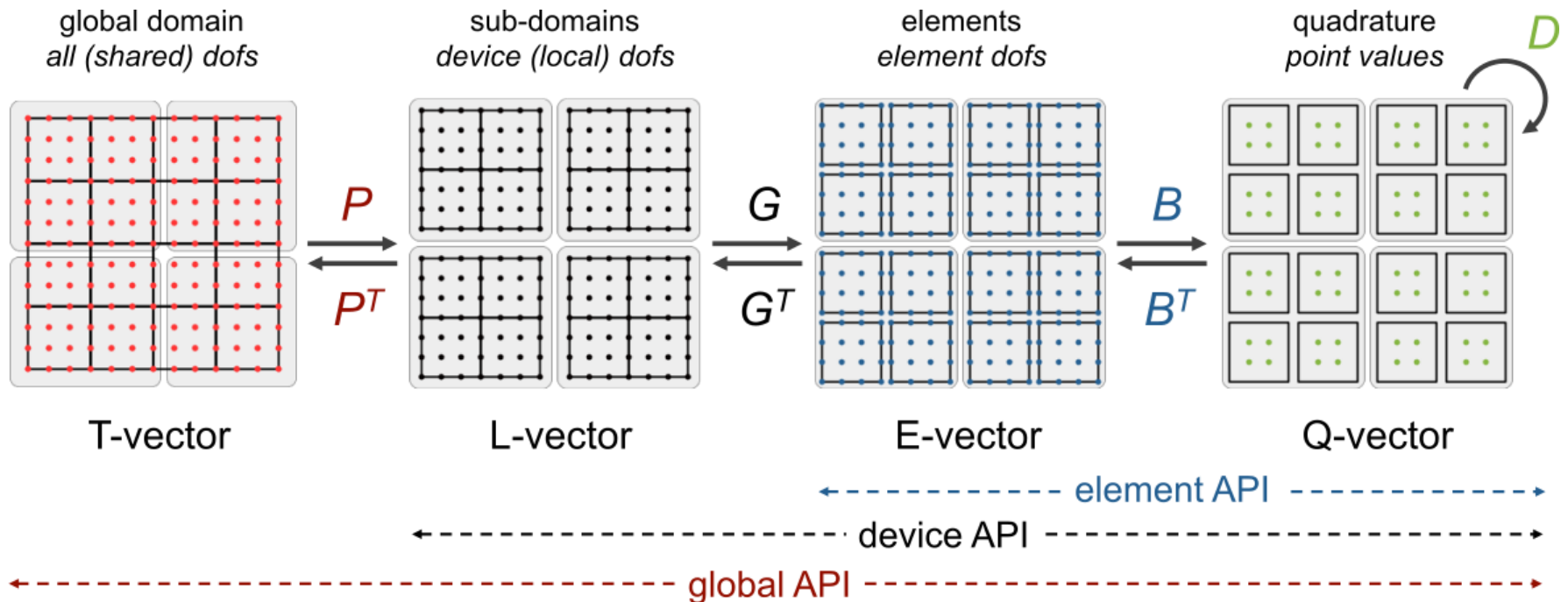
*Parallel partitioning via
Hilbert curve*

- weak+strong scaling up to ~400K MPI tasks on BG/Q
- **measure AMR only components:** interpolation matrix, assembly, marking, refinement & rebalancing (no linear solves, no “physics”)

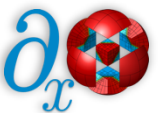
Fundamental finite element operator decomposition

The assembly/evaluation of FEM operators can be decomposed into **parallel**, **mesh topology**, **basis**, and **geometry/physics** components:

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



- ✓ **partial assembly** = store only D , evaluate B (tensor-product structure)
- ✓ better representation than A : *optimal memory, near-optimal FLOPs*
- ✓ purely algebraic
- ✓ high-order operator format
- ✓ AD-friendly



Example of a fast high-order operator

Poisson problem in variational form

Find $u \in Q_p \subset \mathcal{H}_0^1$ s.t. $\forall v \in Q_p$,

$$\int_{\Omega} \nabla u \cdot \nabla v = \int_{\Omega} f v$$

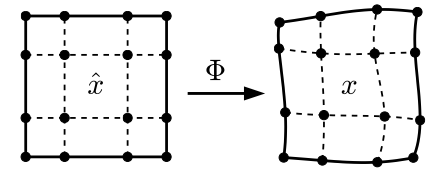
Stiffness matrix (unit coefficient)

$$\begin{aligned} \int_{\Omega} \nabla \varphi_i \nabla \varphi_j &= \sum_E \int_E \nabla \varphi_i \nabla \varphi_j \\ &= \sum_E \sum_k \alpha_k J_E^{-1}(q_k) \hat{\nabla} \hat{\varphi}_i(q_k) J_E^{-1}(q_k) \hat{\nabla} \hat{\varphi}_j(q_k) |J_E(q_k)| \\ &= \sum_E \sum_k \underbrace{\hat{\nabla} \hat{\varphi}_i(q_k)}_{(B^T)_{ik}} \underbrace{(\alpha_k J_E^{-T}(q_k) J_E^{-1}(q_k) |J_E(q_k)|)}_{D_{kk}} \underbrace{\hat{\nabla} \hat{\varphi}_j(q_k)}_{B_{kj}} \end{aligned}$$

$\uparrow$
 A_{ij}

G, G^T

- J is the Jacobian of the element mapping (geometric factors)

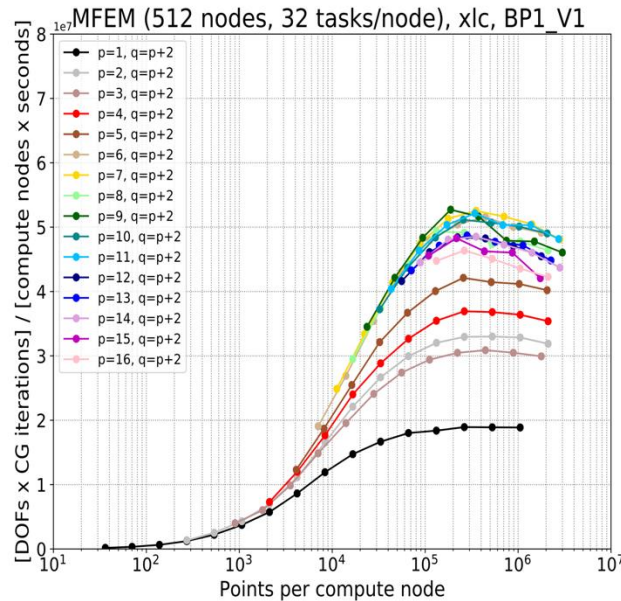


- G is usually Boolean (except AMR)
- Element matrices $A_E = B^T D B$, are full, account for bulk of the physics, can be applied in parallel

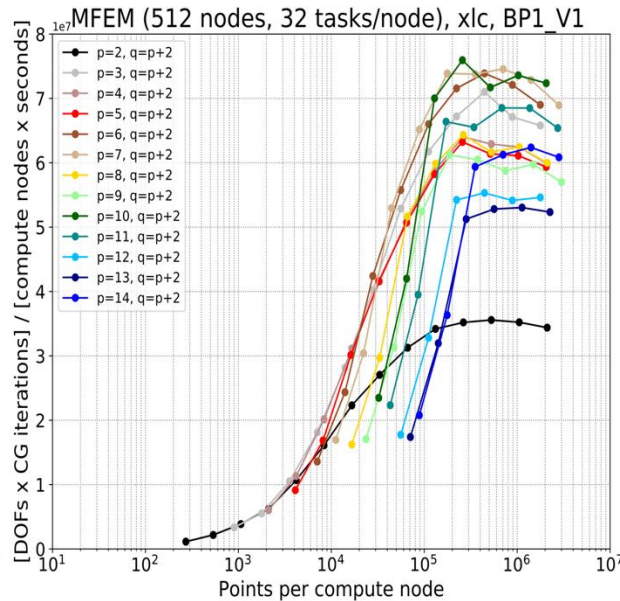
$$\begin{bmatrix} A^1 & & & \\ & A^2 & & \\ & & \ddots & \\ & & & A^4 \end{bmatrix}$$

- Never form A_E , just apply its action based on actions of B , B^T and D

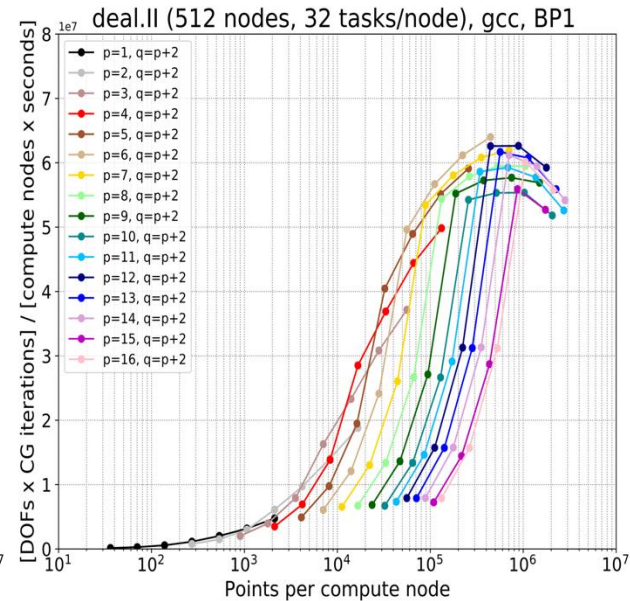
CEED BP1 bakeoff on BG/Q



Nek5000



MFEM-improved

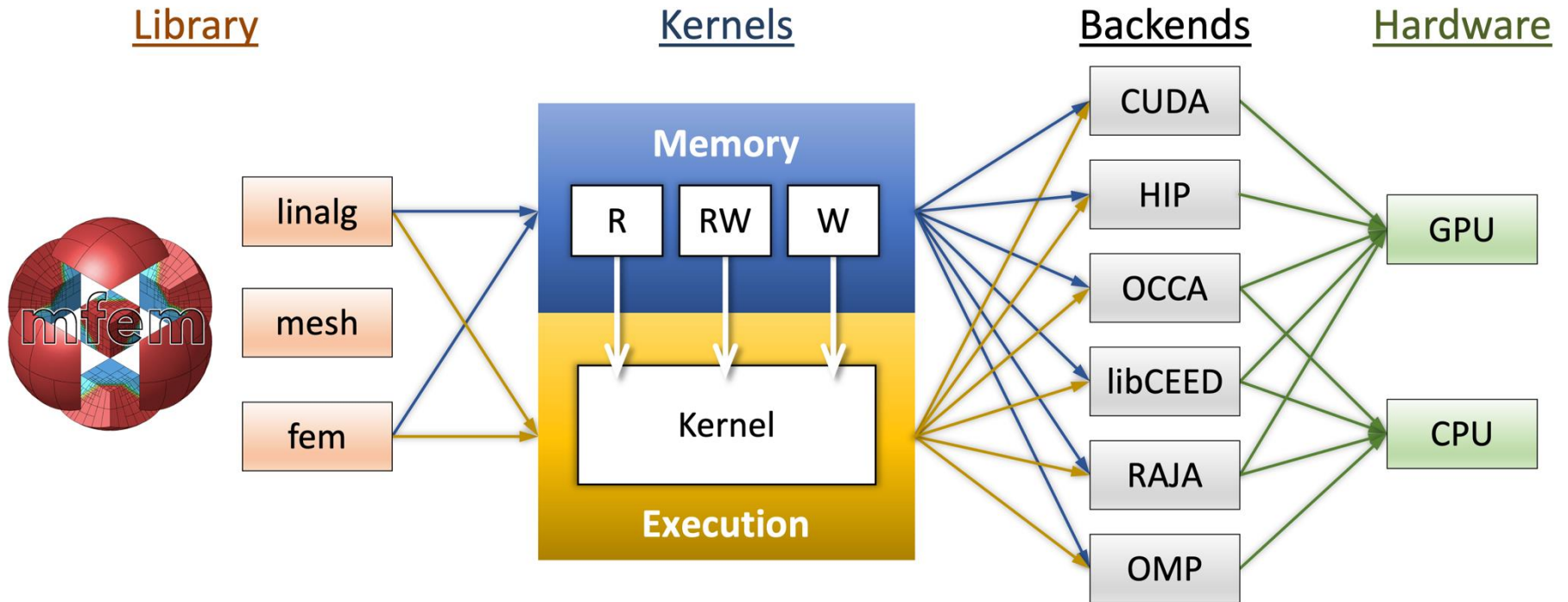


deal.II

- ✓ All runs done on BG/Q (for repeatability), 16384 MPI ranks. Order $p = 1, \dots, 16$; quad. points $q = p + 2$.
- ✓ BP1 results of MFEM+xlcr (left), MFEM+xlcr+intrinsic (center), and deal.II + gcc (right) on BG/Q.
- ✓ Paper: “Scalability of High-Performance PDE Solvers”, IJHPCA, 2020
- ✓ Cooperation/collaboration is what makes the bake-offs rewarding.

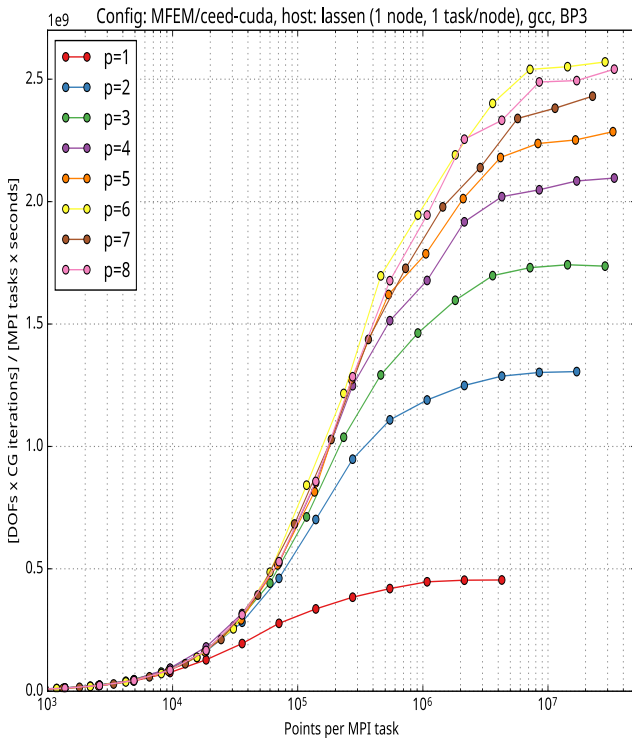
Device support in MFEM

MFEM support GPU acceleration in many linear algebra and finite element operations

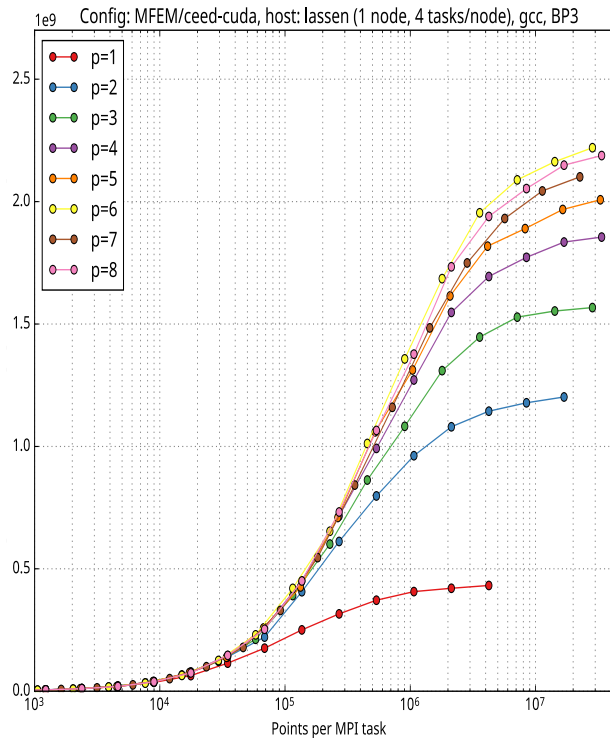


- Several MFEM examples + miniapps have been ported with small changes
- Many kernels have a single source for CUDA, RAJA and OpenMP backends
- **Backends are runtime selectable, can be mixed**
- **Recent improvements in CUDA, HIP, RAJA, SYCL, ...**

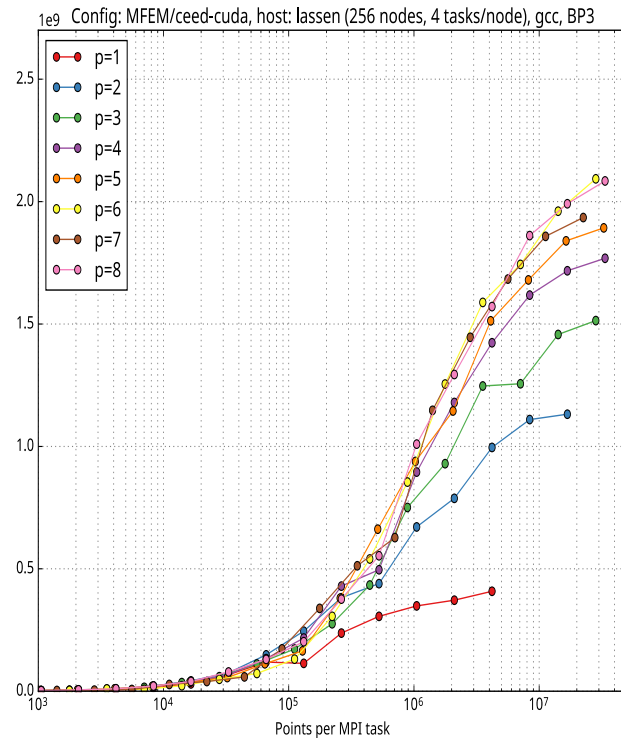
MFEM performance on multiple GPUs



1 GPU



4 GPUs



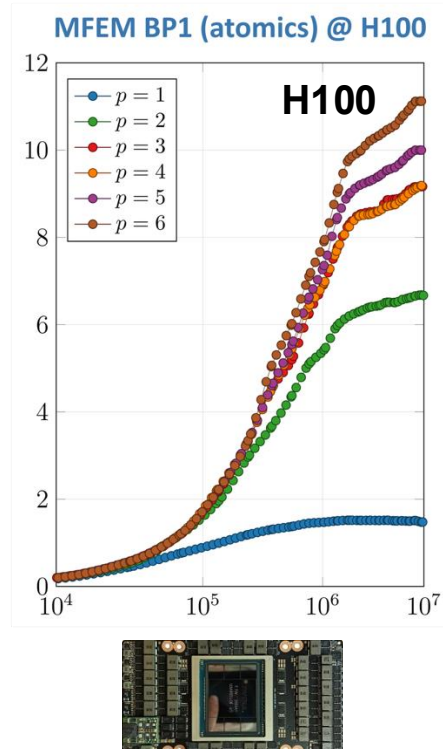
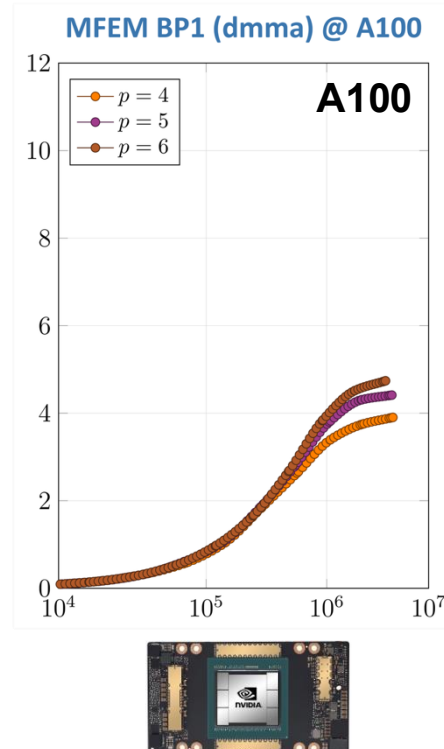
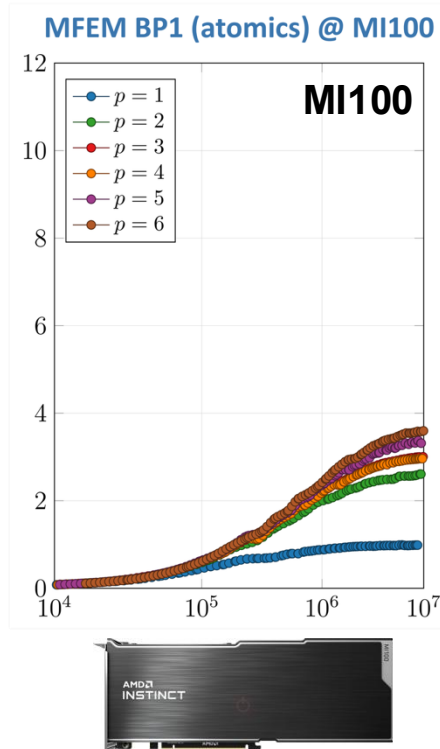
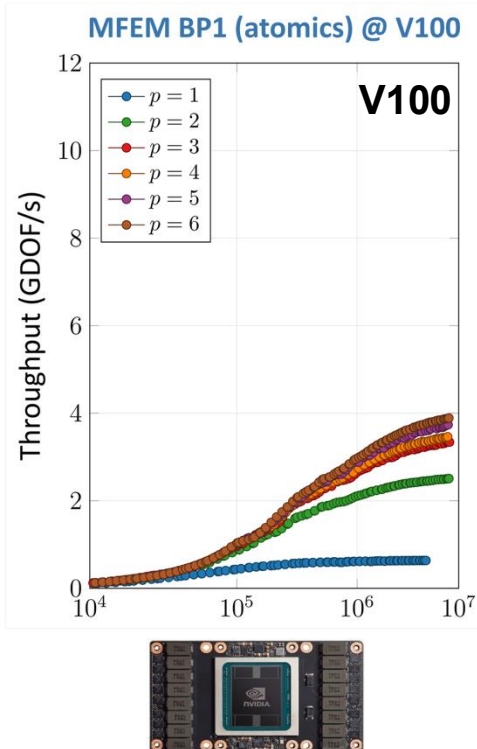
1024 GPUs

Single GPU performance: **2.6 GDOF/s**
Problem size: 10+ million

Best total performance: **2.1 TDOF/s**
Largest size: 34 billion

Optimized kernels for MPI buffer packing/unpacking on the GPU

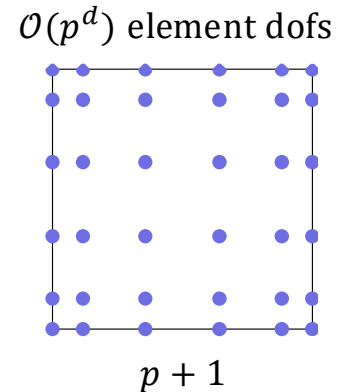
Recent improvements on NVIDIA and AMD GPUs



*New MFEM GPU kernels: perform on both V100 + MI100,
have better strong scaling,
can utilize tensor cores on A100
achieve 10+ GDOFs on H100*

Matrix-free preconditioning

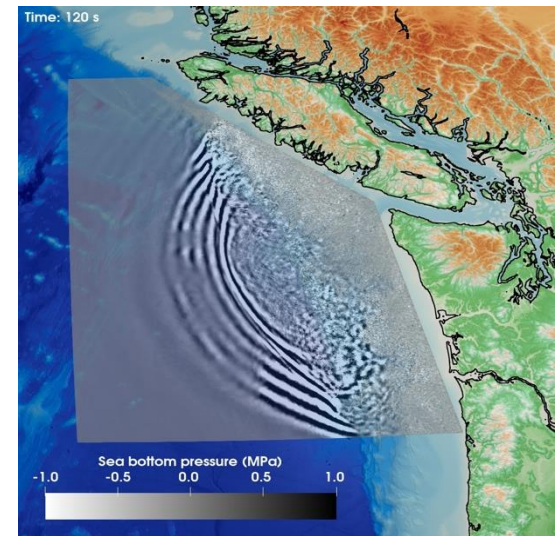
- **Explicit matrix assembly impractical at high order:**
 - Polynomial degree p , spatial dimension d
 - Matrix assembly + sparse matvecs:
 - $\mathcal{O}(p^{2d})$ memory transfers
 - $\mathcal{O}(p^{3d})$ computations
 - can be reduced to $\mathcal{O}(p^{2d+1})$ computations by sum factorization
 - Matrix-free action of the operator (partial assembly):
 - $\mathcal{O}(p^d)$ memory transfers – *optimal*
 - $\mathcal{O}(p^{d+1})$ computations – *nearly-optimal*
 - efficient iterative solvers ***if combined with effective preconditioners***
- **Challenges:**
 - Traditional matrix-based preconditioners (e.g. AMG) not available
 - Condition number of diffusion systems grows like $\mathcal{O}(p^3/h^2)$



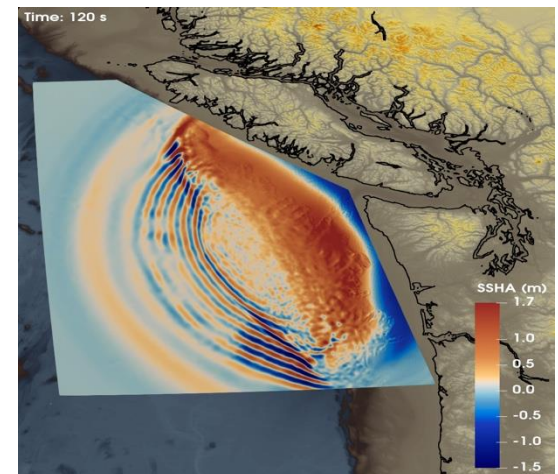
MFEM Executing Largest F.E. Calculation Ever

MFEM used for the execution of the high-fidelity simulations used in the development of a tsunami early warning System

- Winner of the 2025 ACM Gordon Bell Award
- Largest ever mesh finite element simulation with 55.5 trillion DOF on 43,520 GPUs
- Order of magnitude improvement compared to state-of-the-art
- Excellent weak scaling on Perlmutter, Alps and El Capitan
- GPU-accelerated high-order FE algorithms
- Adv. FE discretization techniques yielding well conditions systems for iterative solutions



Ocean bottom pressure and surface waves computed with acoustic-gravity model

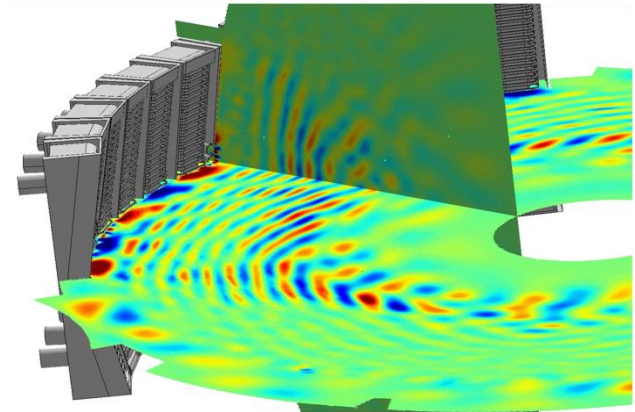


Design of More Robust Fusion System RF Antennas

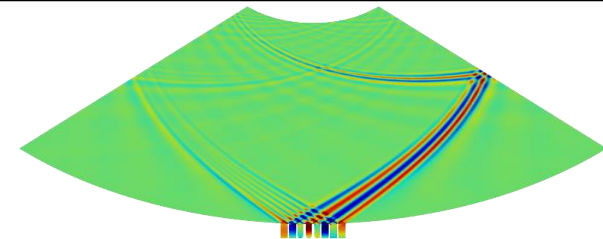
MFEM's high-order methods, efficient solvers and GPU support accurate modeling of fusion energy systems RF antennas.

MFEM developments to extend the simulation capabilities for the RF-PMI SciDAC Partnership:

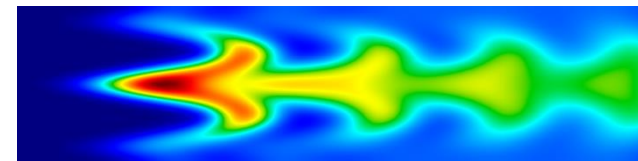
- The Hybridized Discontinuous Galerkin method offers a more efficient discretization for modeling non-linear anisotropic plasma transport on high-order meshes
- Discontinuous Petrov-Galerkin (DPG) preconditioners for the indefinite Maxwell equations enable higher fidelity solutions
- Integration with the MFEM adaptive mesh control methods and the Petra-M interface for execution of full geometric fidelity fusion energy systems and RF antenna



Fully resolved 3D HHFW field on NSTX-U (S. Shiraiwa, PPPL)



RF test case for new FE discretization methods

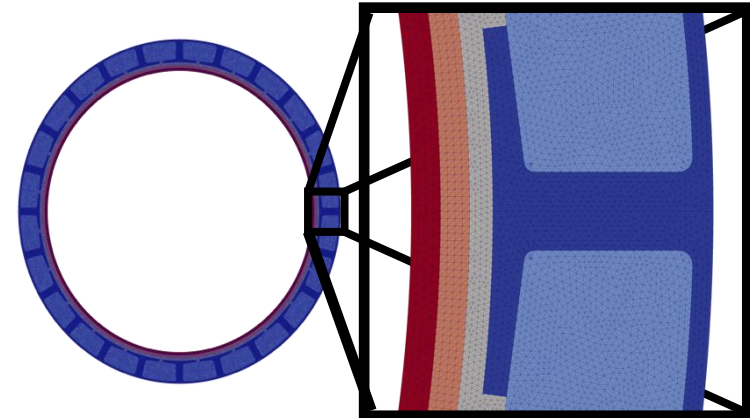
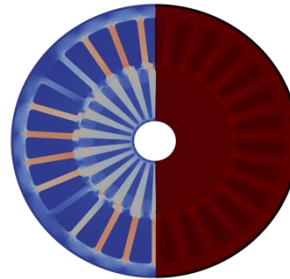
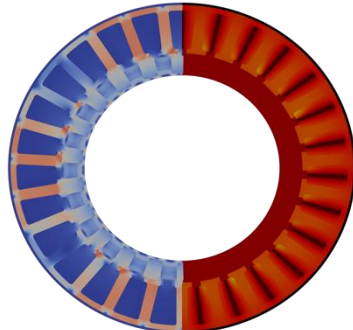
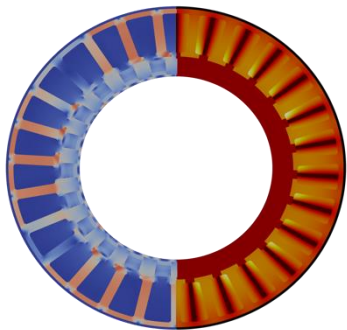
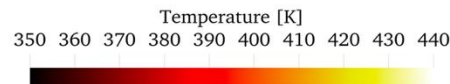
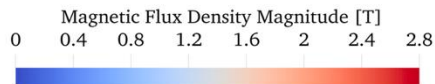
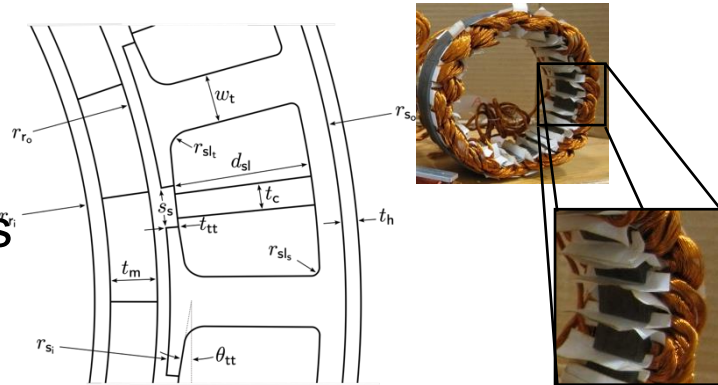


Contamination transport solved by high-order hybridized method

Modeling of Electric Motors

Modeling the motor

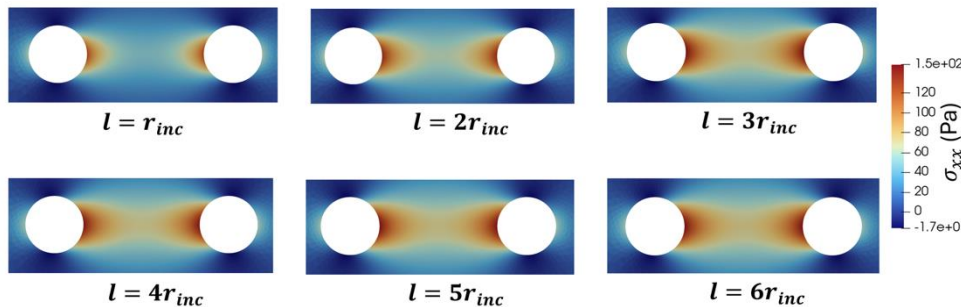
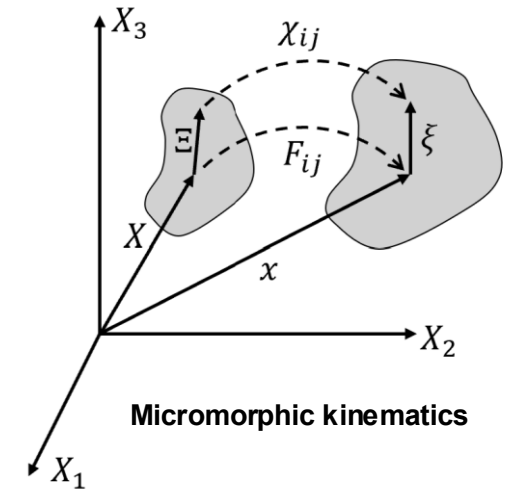
- Magnetostatic equation to model motor electromagnetics
- Heat equation to solve the thermal behavior
- Coupled models
- Multi-objective optimization that trades mass and efficiency



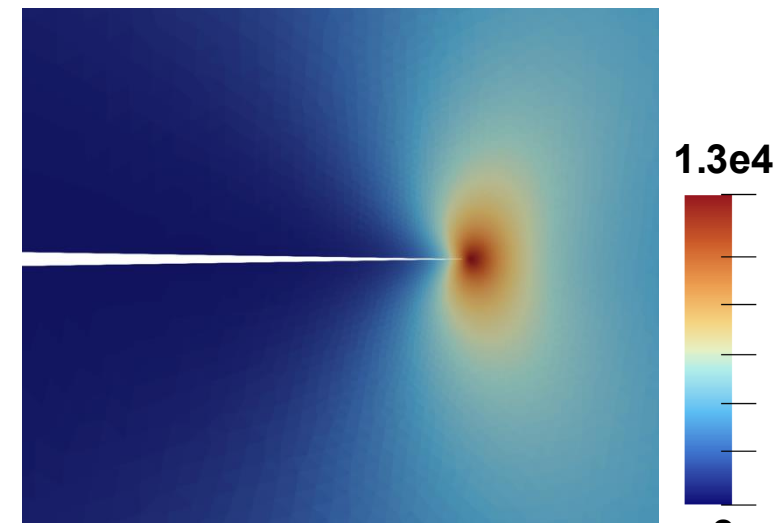
$$m_{\text{total}} = 9.27 \text{ kg}, \eta = 95.5\%$$

Modeling micromorphic continua in MFEM

- A micromorphic continuum is well-suited to model materials that exhibit nonlocal and nonaffine behavior
- Requires solving a mixed problem with displacements u_i and micro displacement gradients ϕ_{ij}
- We implemented a finite-strain formulation into MFEM using the BlockNonlinearForm framework



Length-scale effects capture long-range stress tethers.



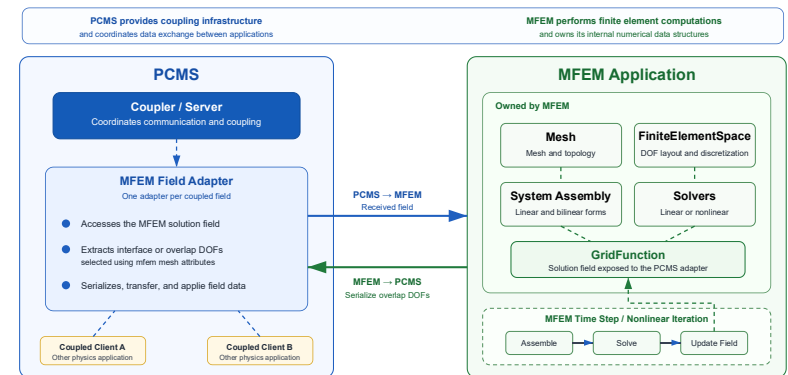
“Blunted” Mode 1 crack-tip σ_{22} stress field (Pa)

PCMS Integration with MFEM for Coupled Simulations

- PCMS enables partitioned multi-physics simulations by coupling multiple independent MFEM applications

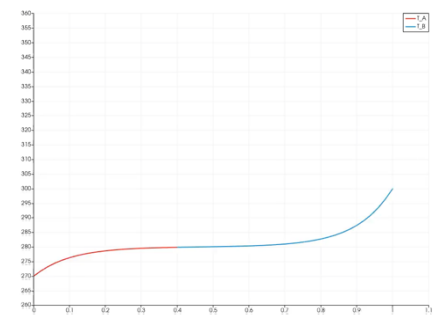
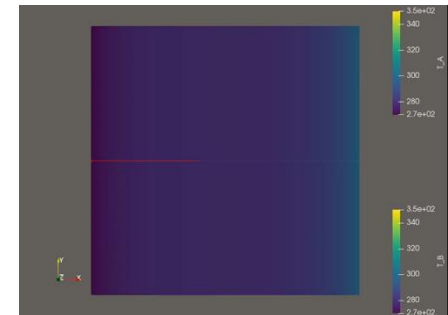
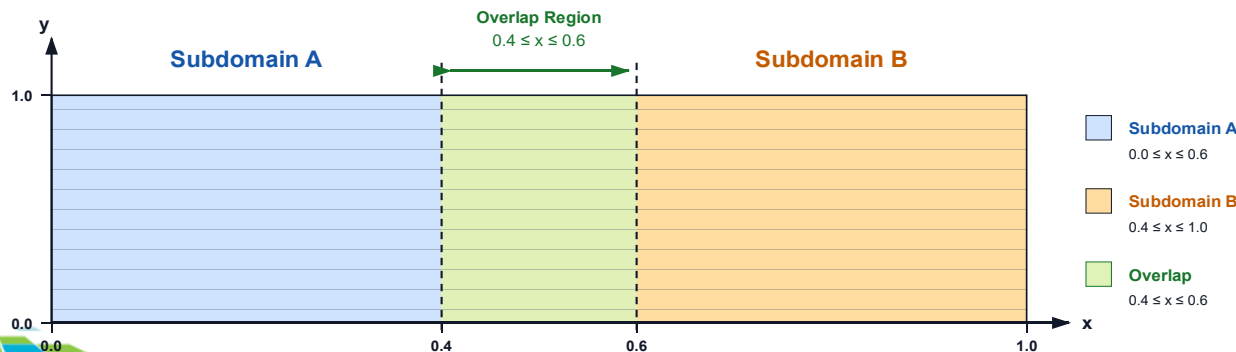
- PCMS supports:
 - Parallel field communication
 - Physics preserving field transfer
 - Interface and overlapping domain coupling
 - Steady and transient coupling workflows
- MFEM interfaces have been developed to support communication and field transfer needs.

PCMS ↔ MFEM Interaction



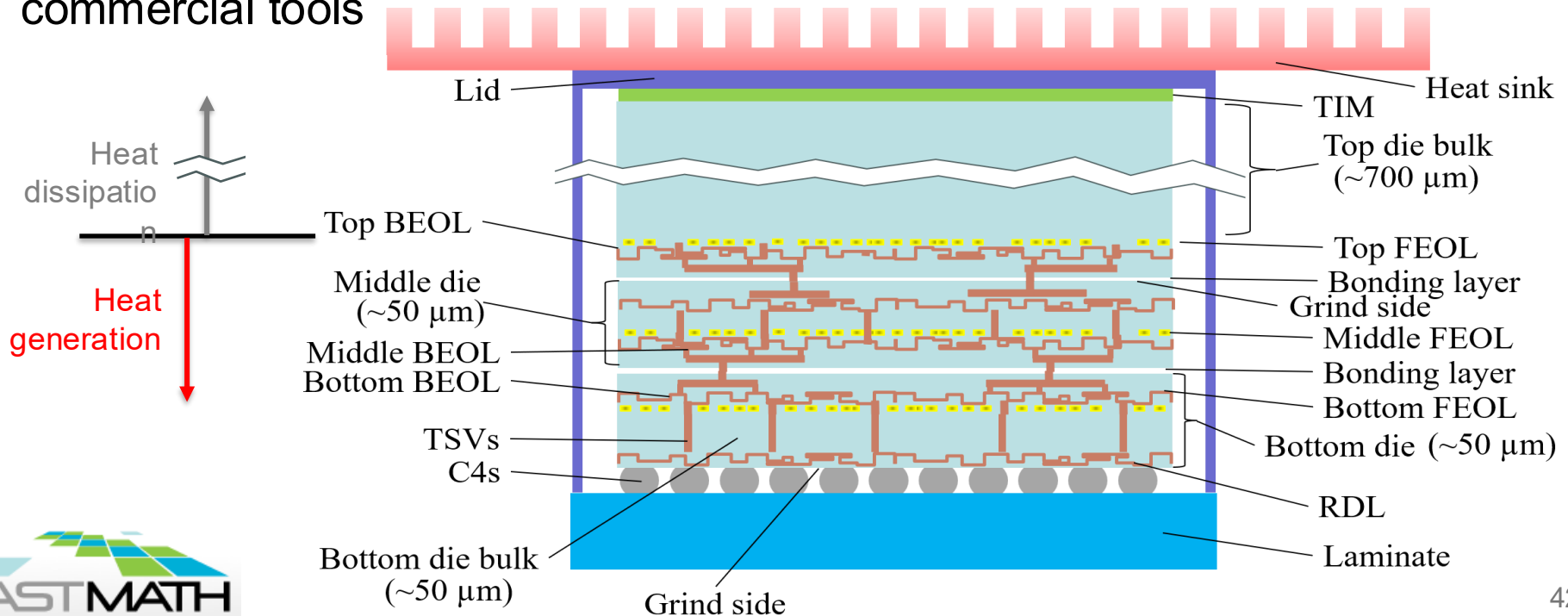
- Example: partitioned domain transient heat conduction simulation using MFEM-based PDE solver.

- Initial temperature; 280 K; BCs; Left: 270, Right: 300, Transient gaussian heat generation at $t=2s$

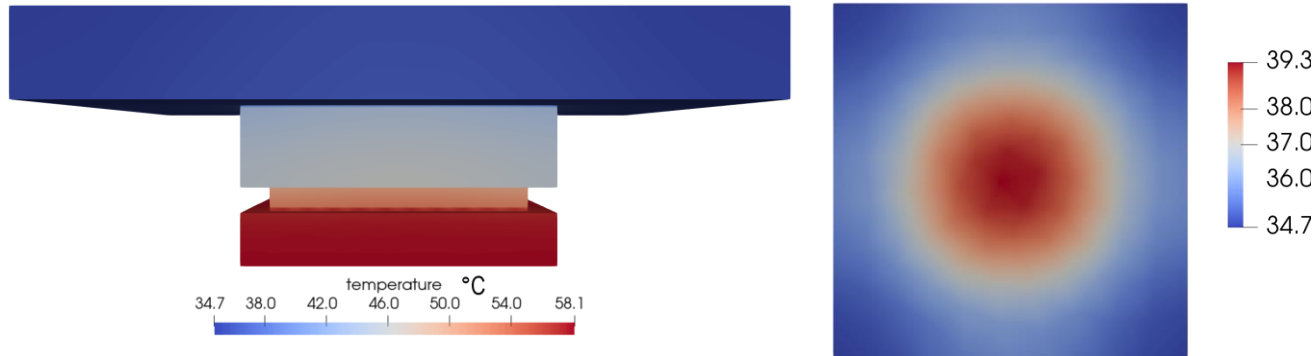


Multiscale Simulation of 3D ICs

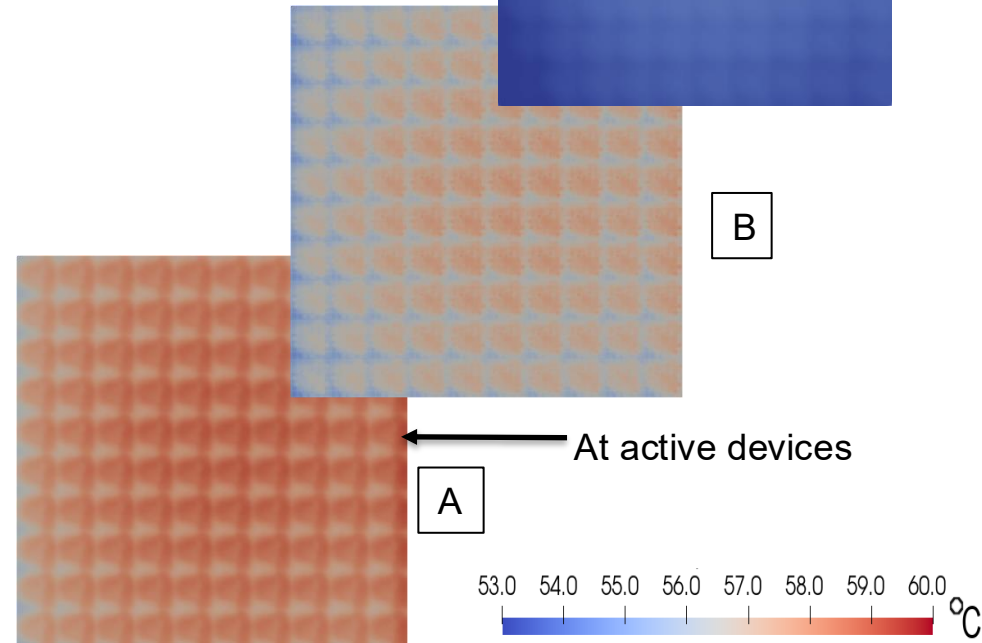
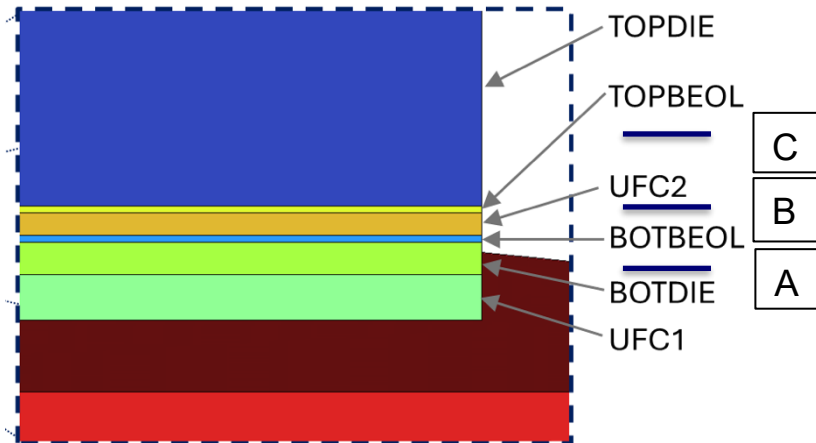
- 3D IC stacks live and die on their heat transfer performance
- Design requires feedback, preferably quickly!
- Micro/nanoscale heterogeneity impacts transistor-, die-, package-scale hot spots
 - Nanometer details, centimeter analysis - too big to mesh/solve.
- Software stack based on MFEM and Simmetrix components
- Described multiscale methods are optimized for integration with existing commercial tools

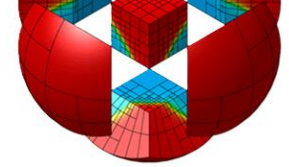


Multiscale Simulation of 3D ICS - Results



(Above) Package scale temperature solution including heat-sink, (Bottom left) Advanced package model including locations of temperature slices on (right)

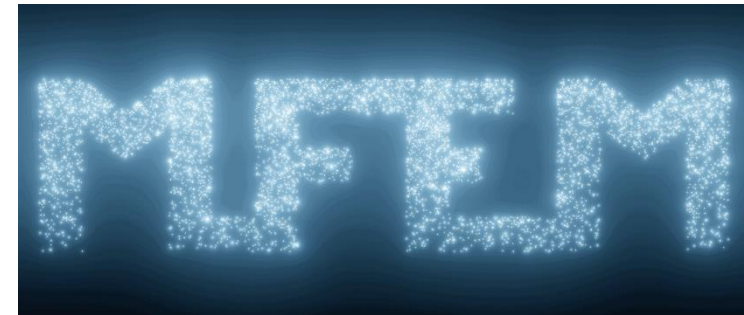




MFEM v4.9 (released 12/11/2025)

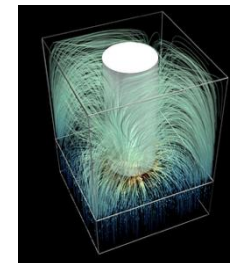
Scientific Achievement

- dFEM interface for differentiable simulations
- improved meshing algorithms TMOP, NURBS, AMR – *relevant to the SciDAC projects*
- optimized GPU kernels – *acceleration of existing applications*
- initial support for particle methods – *promising for future proposals*
- AMGF solver and miniapp for contact problems
- 10 new examples and miniapps to help DOE applications – *including the SciDAC projects*
- and much more, see <http://mfem.org>



Technical Approach

- 8 months in development (about 166 work days)
- 491K total lines of code (+33K from mfem-4.8, about 200/day)
- 625 changed files: 75,520 additions and 20,097 deletions
- 38 contributors
- 2456 commits (about 15/day)
- 131 issues closed (about 0.8/day)
- 247 PRs merged (about 1.5/day)



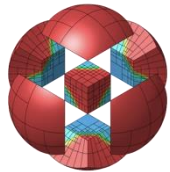
Illustrations of the new MFEM particle support (top), RF ablation application (bottom left), and the MFEM v4.9 file changes graph (bottom right)

High-order methods provide high-quality & performant simulations on exascale platforms

■ More information and publications

- MFEM – mfem.org
- BLAST – computation.llnl.gov/projects/blast
- CEED – ceed.exascaleproject.org

■ Open-source software



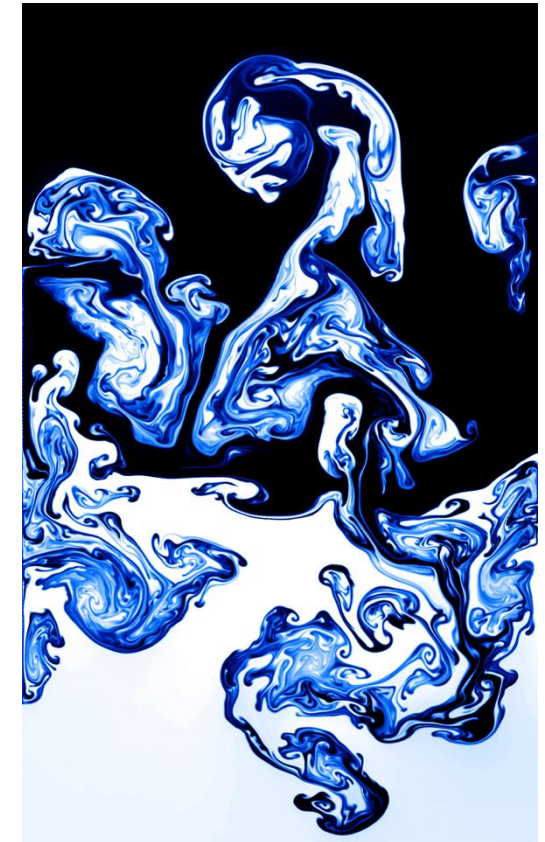
mfem



CEED
EXASCALE DISCRETIZATIONS

■ Ongoing R&D

- GPU-oriented algorithms for Frontier, Aurora, El Capitan
- Matrix-free scalable preconditioners
- Automatic differentiation, design optimization
- Deterministic transport, multi-physics coupling

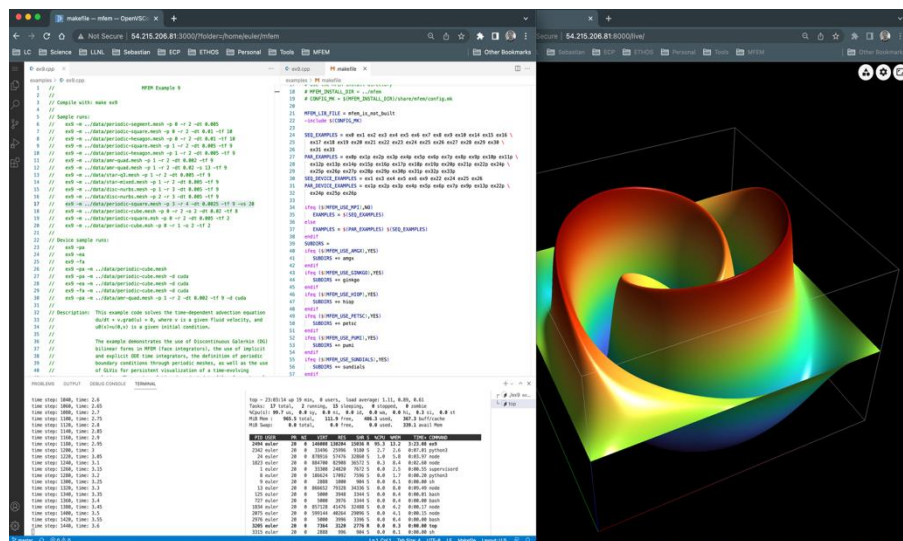


Q4 Rayleigh-Taylor single-material ALE on 256 processors

Upcoming MFEM Events

MFEM at LLNL HPC Tutorials

September 22, 2026



<https://hpcic.llnl.gov/>

MFEM Community Workshop

September 22-25, 2026

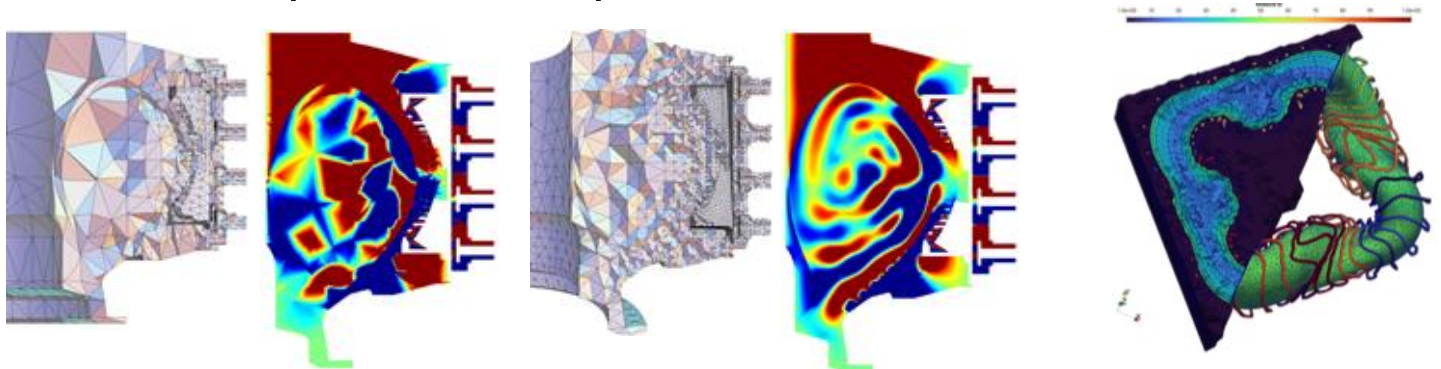
Georgia Institute of Technology + Virtual



<https://mfem.org/workshop>

FASTMath Unstructured Mesh Tools and Components

- Unstructured Mesh Infrastructure:
 - PUMI-General-Adapt (<https://github.com/SCOREC/core>): CPU based scalable conforming mesh adaptation based on local mesh modification
 - PUMI-Portable-Adapt (https://github.com/SCOREC/omega_h): Performant (CPUs and GPUs currently) infrastructure supporting scalable conforming mesh adaptation based on local mesh modification
 - PUMI-Particle (<https://github.com/SCOREC/pumi-pic>): Performance portable (CPUs and GPUs) with scalable mesh and particle data structures and operators to support particle-in-cell simulation codes
 - PUMI-Tally: Performance portable (CPUs and GPUs) infrastructure with scalable mesh and particle data structures and operators to support Monte Carlo neutral particle transport



FASTMath Unstructured Mesh Tools and Components

- General code coupling:
 - PCMS (<https://github.com/SCOREC/pcms>): Code coupling library for exascale applications (from file based to parallel in memory)
- Load balancing, task placement:
 - Jet (<https://github.com/sandialabs/Jet-Partitioner/>): Parallel graph partitioner that runs on most CPU and GPU systems
 - Zoltan (<https://github.com/sandialabs/Zoltan>): Dynamic load balancing library
 - Zoltan2 (<https://github.com/trilinos/Trilinos/tree/master/packages/zoltan2>): A package of combinatorial algorithms for scientific computing
 - PUMI-Balance (<http://scorec.github.io/EnGPar/>): A hyper-graph based parallel unstructured mesh dynamic partition improvement component

Geometry and Meshing

Critical to ensuring that fusion energy systems will succeed

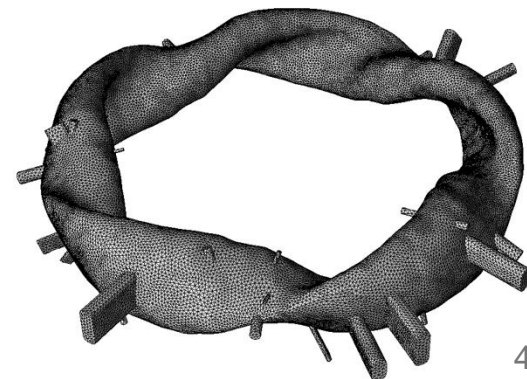
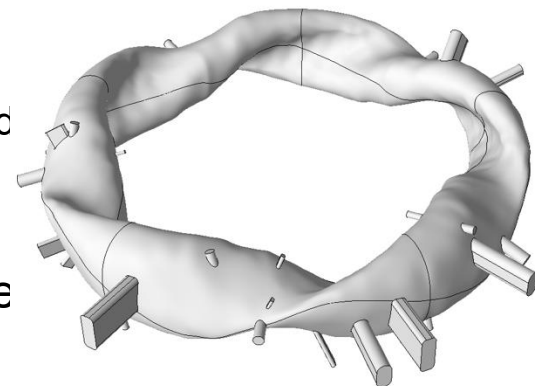
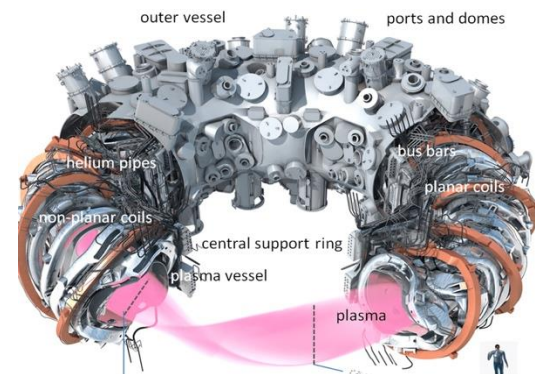
- Integration of simulations that account for the levels of geometric fidelity needed, particularly for plasma facing components
- Employ CAD technologies to define the analysis geometries – effort needed to coordinate with the CAE to provide the needed tools
- Unstructured mesh methods only reasonable means to address simulation workflows with complex geometries

When using the full set of available simulation automation technologies

- The generation and control of good meshes given a properly defined analysis geometry is possible using available tools.
- The generation of the proper analysis geometry is a challenge.

There is a Simmetrix/RPI/LLNL ASCR SBIR addressing some of these geometry construction needs including

- Automatic clean-up of CAD models to ensure meshes can be automatically generated
- Geometry and meshing for stellarators
- Working with SciDACs and fusion energy companies on supporting simulation workflows needing complex analysis geometries



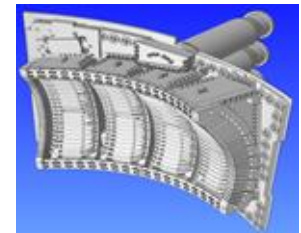
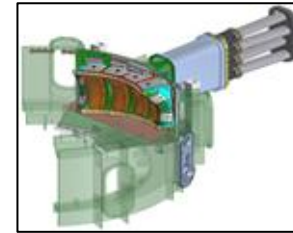
Example Analysis Geometry Construction

Desired properties of the analysis geometry for RF and impurity transport simulations

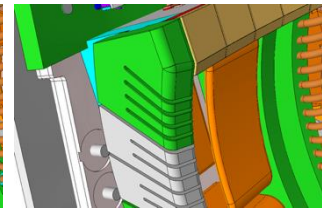
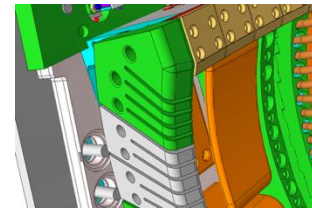
- Represents the physical domain to the level of geometric fidelity required
- Can include “physics” geometry of importance to the simulation process (example flux curves)
- Can be constructed from multiple input sources including CAD models, functions on grids, image data, mesh data, etc.
- Is a complete geometry in a form that can support the application of automatic mesh generation

Approach

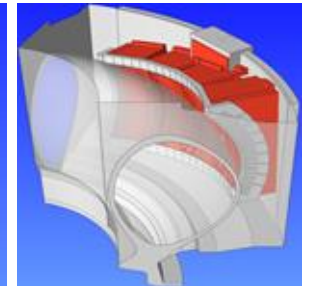
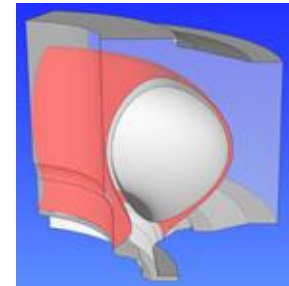
- Employ tools that support the combination of the various forms of input geometry into produce complete boundary representations as supported by CAD systems
- The Simmetrix GeomSim Core and SimModeler support these operations



Extracting antenna assembly geometry



Efficient elimination of unwanted components



Add Physics geometry Assembled Geometry

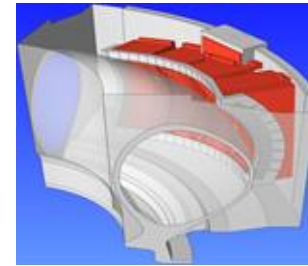
Automatic Mesh Generation

Desired properties of the mesh generators

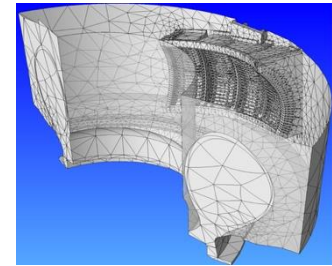
- Produce optimal meshes
- Automatically generated given geometry and mesh control information defined on model entities
- Designed to provide the desired level of accuracy

Approach

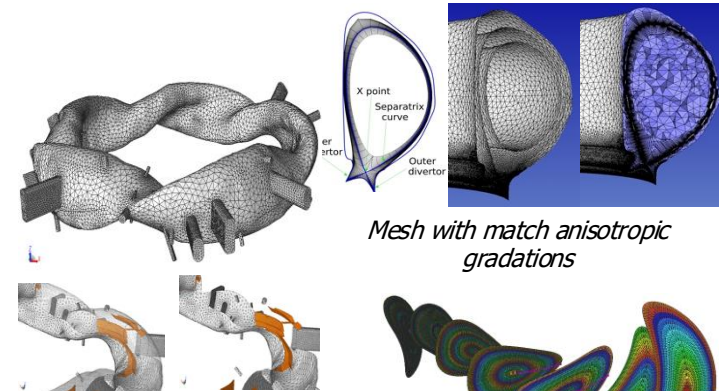
- Unstructured mesh technologies and support coupling to structured meshes
- Employ automatic mesh generation procedures such as Simmetrix MeshSim or opensource tools such as GMSH
- Provide geometry construction and meshing tools building on CAD and automatic mesh generators for specific applications
- Employ adaptive mesh control to ensure mesh effectively solves the problem



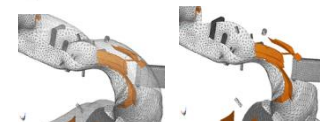
Analysis geometry



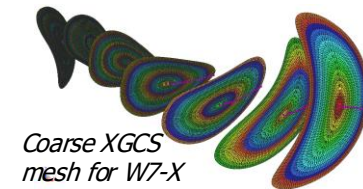
High-order curved mesh



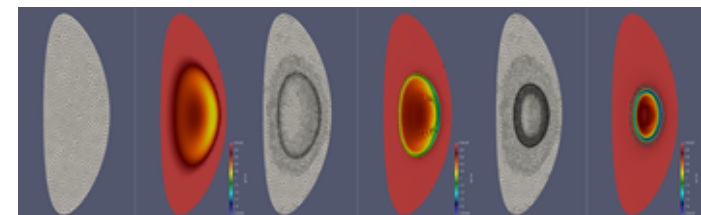
Mesh with match anisotropic gradations



Coarse 3D mesh for W7-X



Coarse XGC5 mesh for W7-X



Adapted meshes for M3D-C1 pellet injection simulation

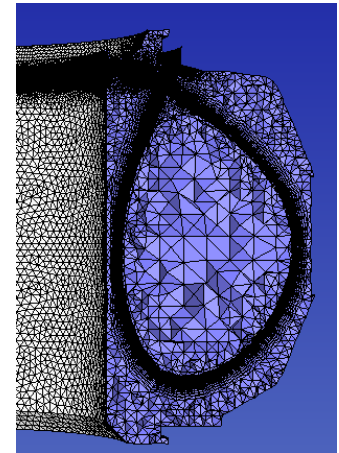
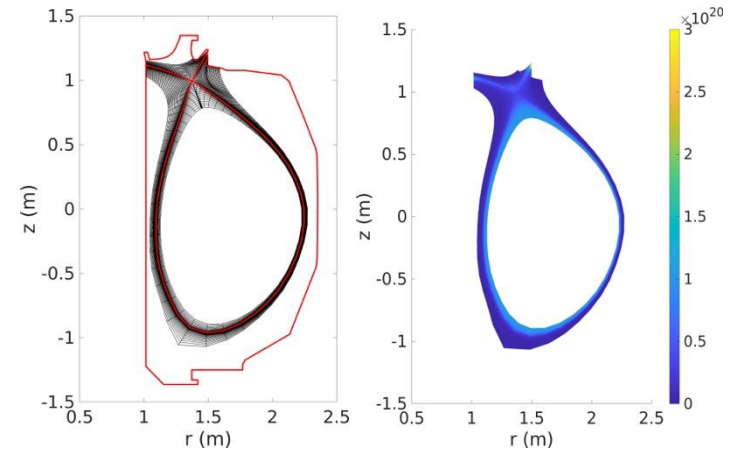
Matching Mesh with Coupled Meshes

Many simulation require field coupling:

- Maintaining solution accuracy requires accurate field transfer
- Accurate field transfer enhanced by matching mesh size and anisotropy

Example: GITRm requires background field from an edge plasma code:

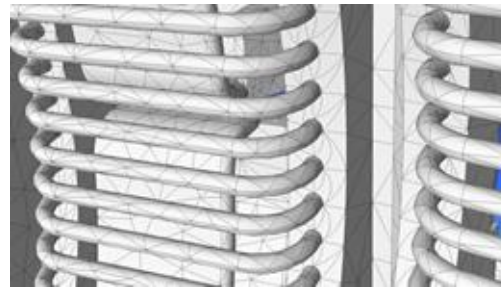
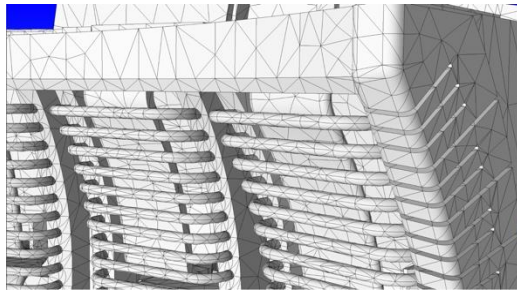
- Edge plasma code mesh uses is block structured, field following, anisotropic and attached to separatrix
- Include separatrix and selected flux surfaces in GITRm analysis geometry
- Set mesh type in that area to boundary layer and match mesh size and anisotropy
- Able to deal with full range of 3D geometry



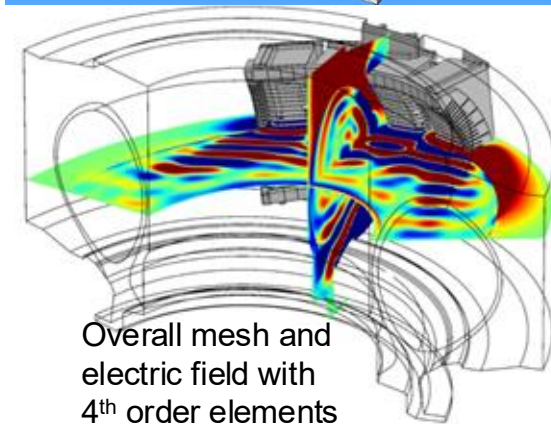
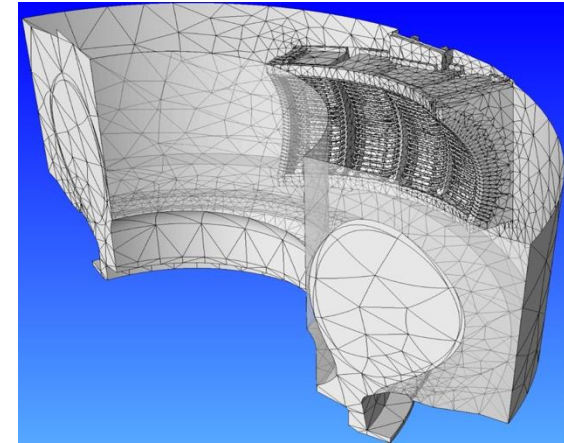
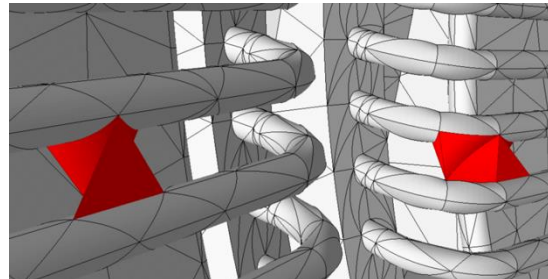
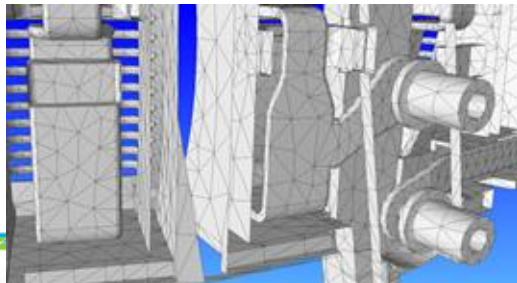
Mesh Generation to Support RF Simulation

Time-harmonic Maxwell's equations solved using high-order Nédélec finite elements:

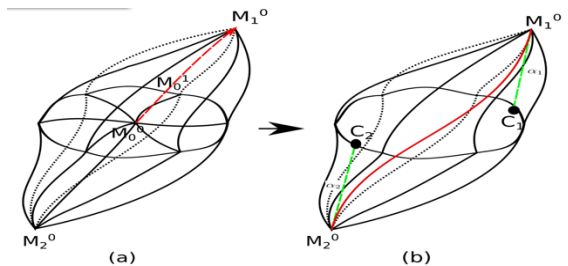
- High-order elements required generation of coarse curved meshes
- Simulation done using MFEM using up to 5th order elements
- Curved meshes automatically generated and adapted



Close-up views of coarse curved mesh



Overall mesh and electric field with 4th order elements

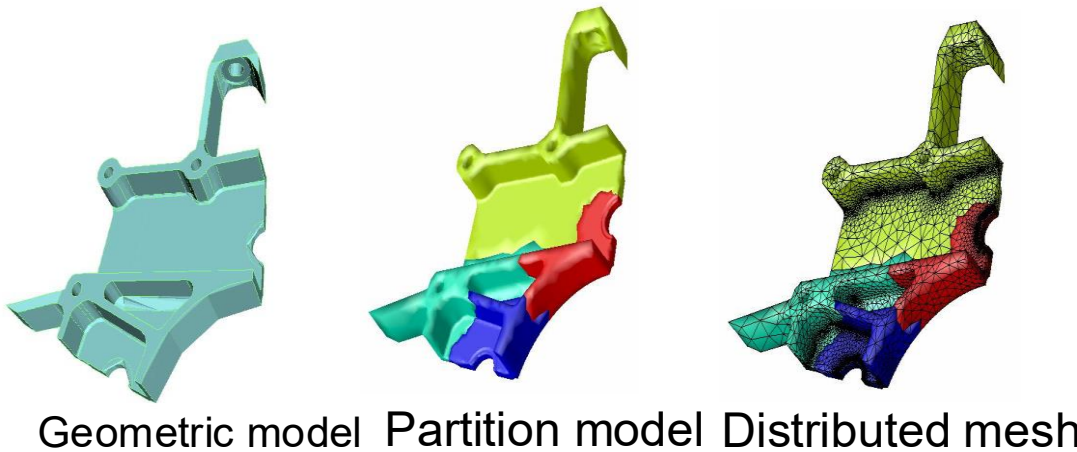


Curved Mesh Adaptation

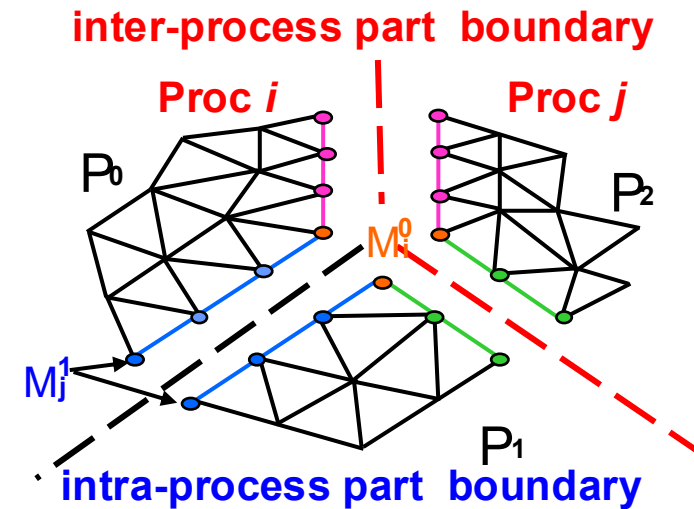
Parallel Unstructured Mesh Infrastructure

Support unstructured mesh interactions on exascale systems

- Mesh hierarchy to support interrogation and modification of meshes
- Maintains linkage to original geometry
- Conforming mesh adaptation
- Inter-process communication
- Supports field operations



Geometric model Partition model Distributed mesh

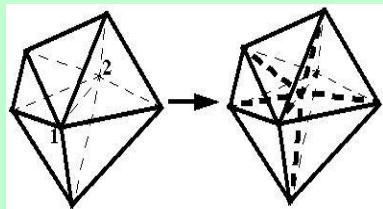
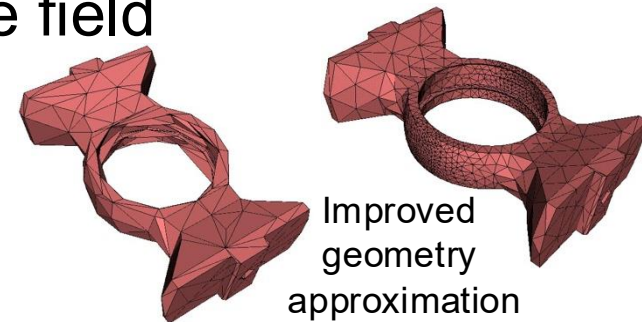


Mesh Entities	Geometry Entities
mesh region, M_i^3	geometry region, G_j^3
mesh face, M_i^2	geometry face, G_j^2
mesh edge, M_i^1	geometry edge, G_j^1
mesh vertex, M_i^0	geometry vertex, G_j^0

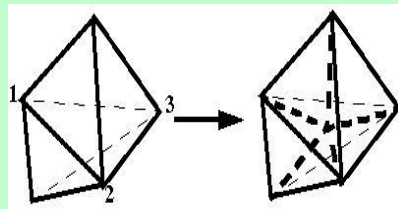
$\updownarrow$ indicates upward and downward adjacency
 $\overleftrightarrow{\text{or}}$ indicates classification on equal or higher dimension model entity
 $\leftarrow$ indicates reverse classification

General Mesh Modification for Mesh Adaptation

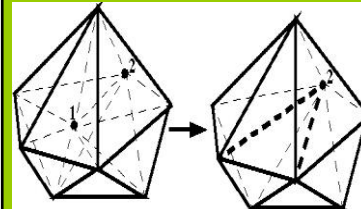
- Driven by an anisotropic mesh size field that can be set by any combination of criteria
- Employ a set of mesh modification operations to alter the mesh into one that matches the given mesh size field
- Advantages:
 - Supports general anisotropic meshes
 - Can obtain level of accuracy desired
 - Can deal with any level of geometric domain complexity
 - Solution transfer can be applied incrementally - provides more control to satisfy conservation constraints



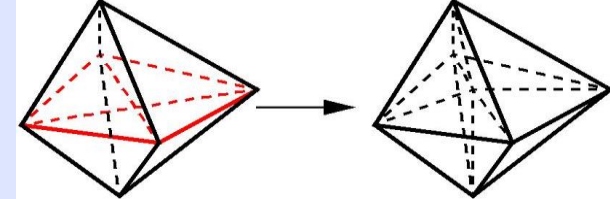
Edge split



face split

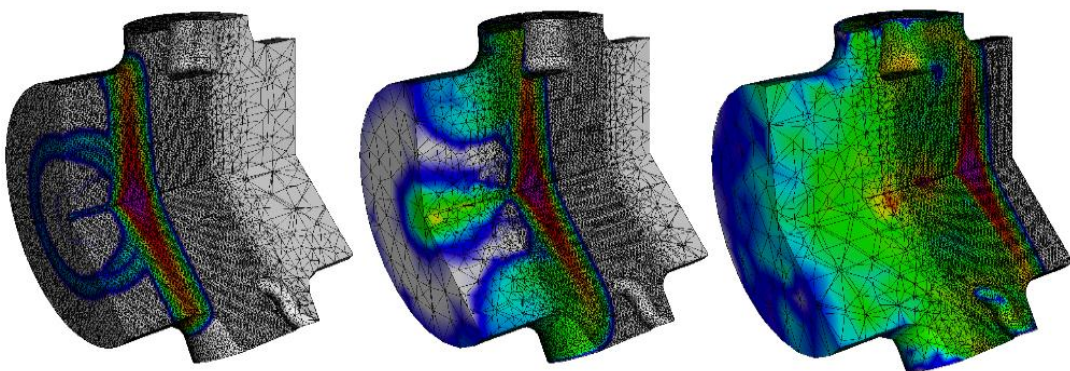


Edge collapse

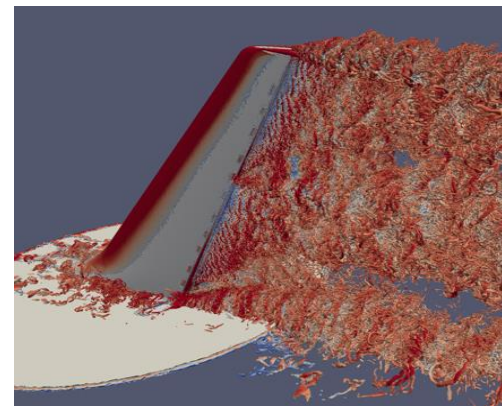


Double split collapse to remove sliver

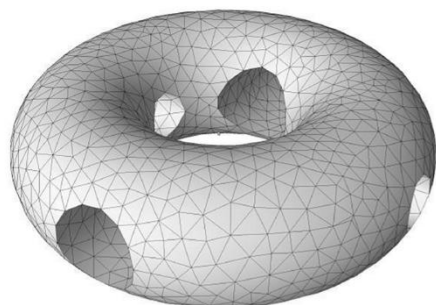
Mesh Adaptation Capabilities



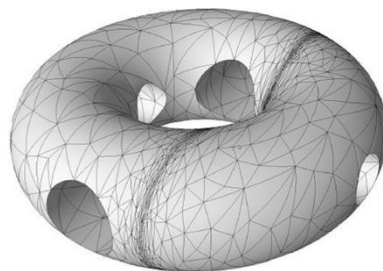
Tracking evolving solution features



Parallel adaptation of mesh with > 90 billion elements

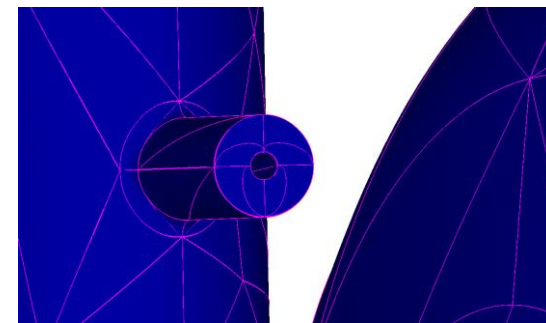


(a)

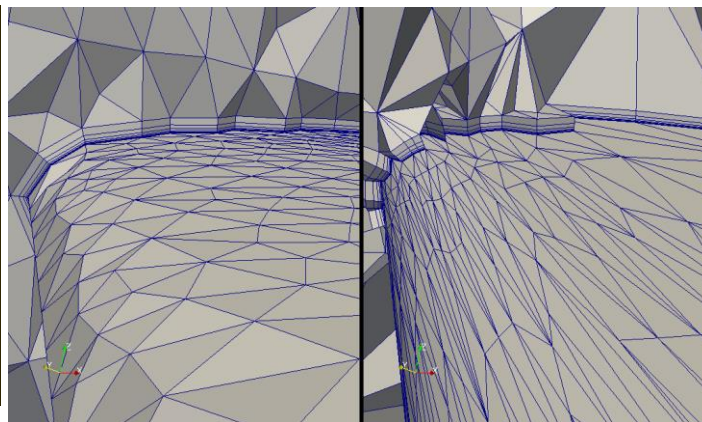
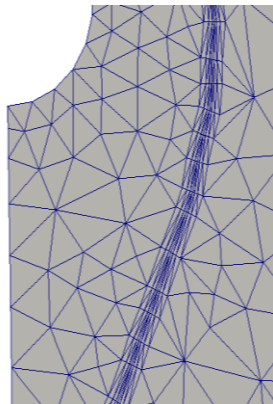


(b)

Adaptation of curved high-order elements



Automatic detection and isolation of solution features



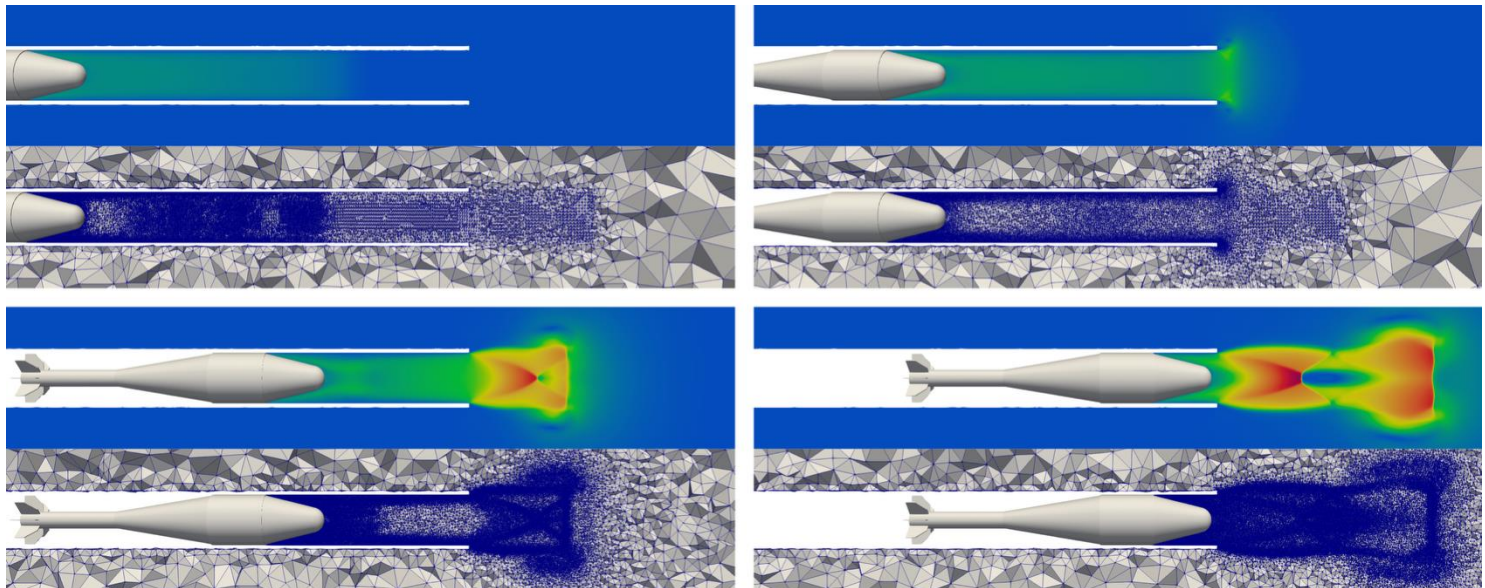
Mixed element topology boundary layer mesh adaptation

Mesh Adaptation of Evolving Geometry Problems

Many applications have geometry that evolves as a function of the results: Effective adaptive loops combine mesh motion and mesh modification.

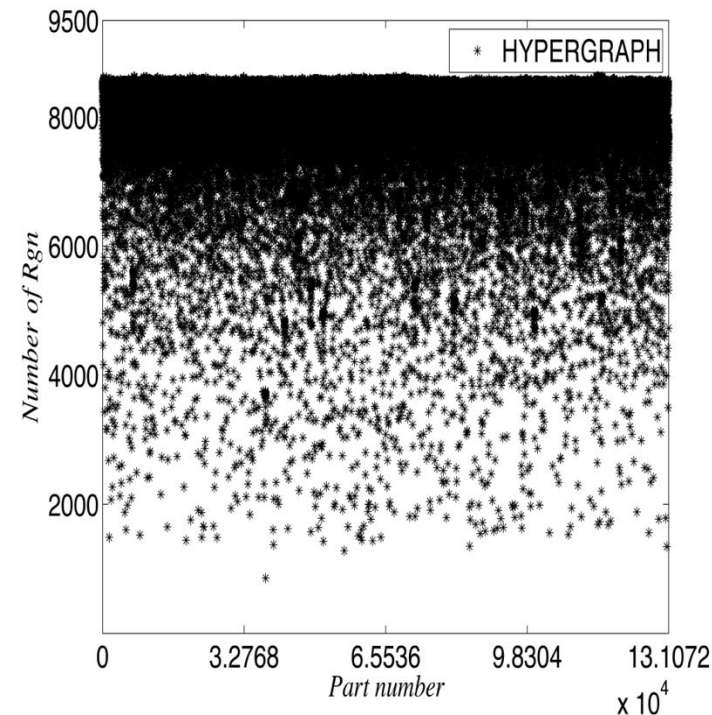
Adaptive loop:

1. Initialize analysis case, generate initial mesh, start time stepping loop
2. Perform time steps employing mesh motion - monitor element quality and discretization errors
3. When element quality is not satisfactory or discretization errors too large – set mesh size field and perform mesh modification
4. Return to step 2



Load Balancing, Dynamic Load balancing

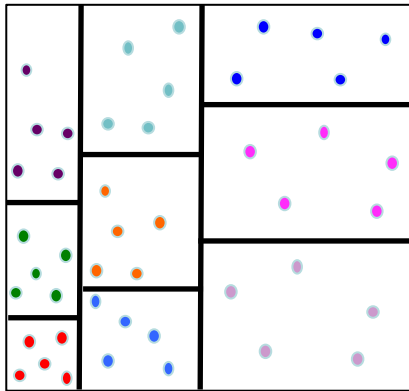
- Purpose: Balance or rebalance computational load while controlling communications
 - Equal “work load” with minimum inter-process communications
- FASTMath load balancing tools:
 - Jet library is a multilevel graph partitioner that runs on a GPU (distributed mesh version under development)
 - Zoltan/Zoltan2 libraries provide multiple dynamic partitioners with general control of partition objects and weights
 - PUMI-Balance diffusive multi-criteria partition improvement



Zoltan/Zoltan2 Toolkits: Partitioners

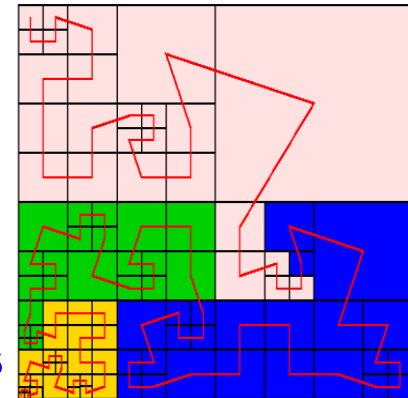
*Suite of partitioners supports a wide range of applications;
no single partitioner is best for all applications.*

Geometric

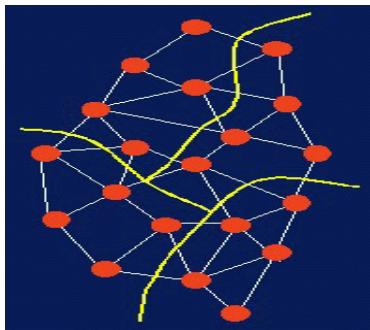


Recursive Coordinate Bisection
Recursive Inertial Bisection
Multi-Jagged Multi-section

Space Filling Curves

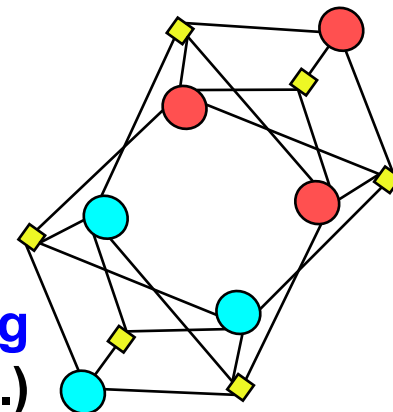


Topology-based



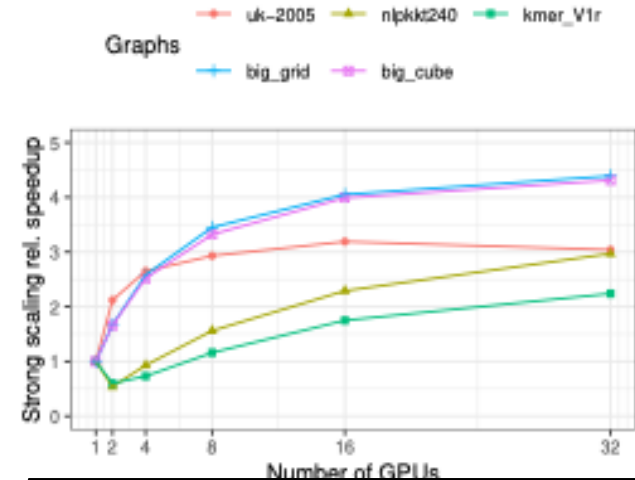
PHG Graph Partitioning
Interface to ParMETIS (U. Minnesota)
Interface to PT-Scotch (U. Bordeaux)

PHG Hypergraph Partitioning
Interface to PaToH (Ohio St.)

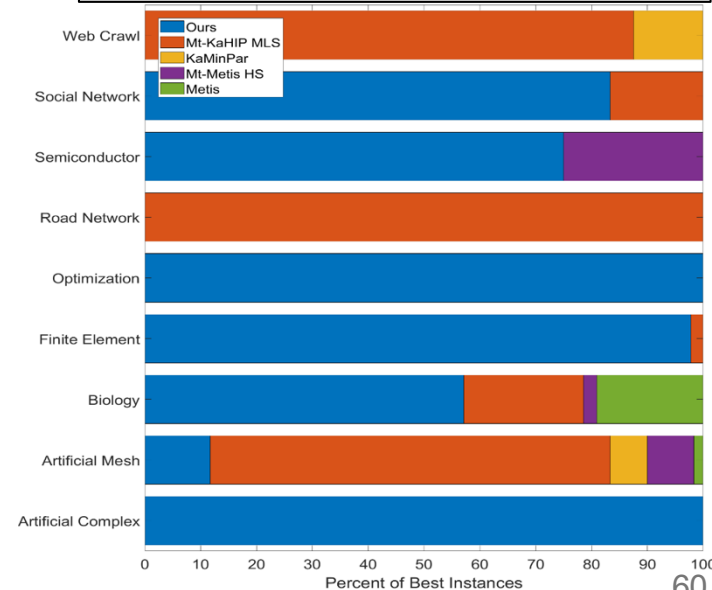


GPU Enabled Multilevel Graph Partitioner - Jet

- Current graph partitioners (e.g., Metis and ParMetis) are limited to CPU only
- Jet is a new Kokkos-based partitioner that is optimized for GPU
- Multi-GPU version is under development, will be available in Trilinos/Zoltan2
- Results (blue bars) slightly better than Metis/Parametis, but significant speedup due to GPU execution, ~20X faster than Metis
- Best partitions for 98% of the test graphs from finite element meshes
- Plan include it use in dynamic load balancing in PUMI adaptive applications

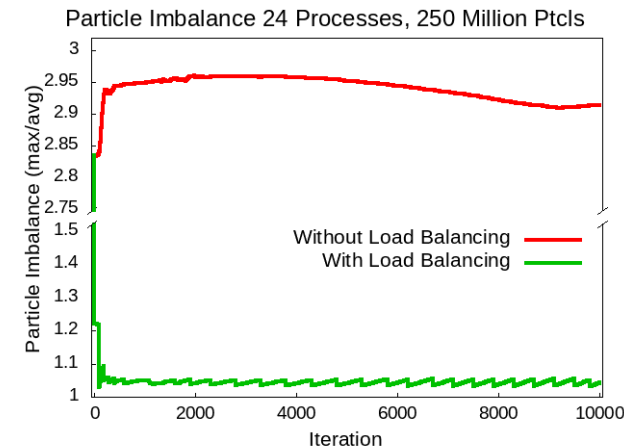
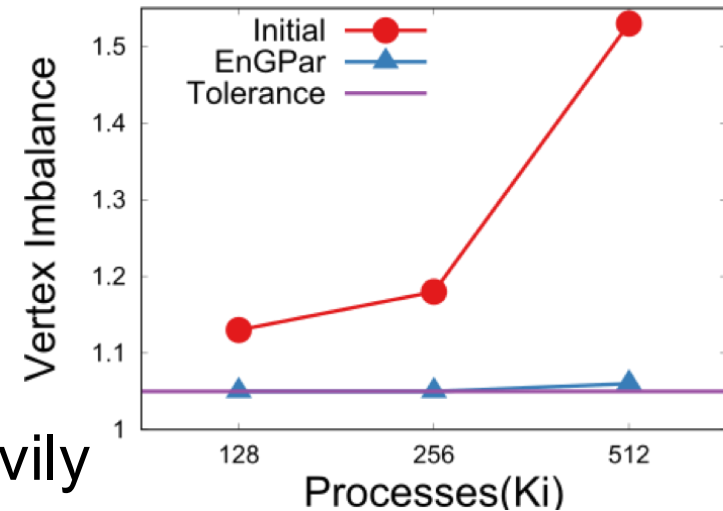


Strong scaling of the Jet partitioner on multiple GPUs on Perlmutter (NERSC).



PUMI-Balance Reduces Hyper-Graph Imbalance

- Hyper-graph supports multiple dependencies (edges) between application work/data items (vertices)
- Application defined graph vertices and edges
- Diffusion movement of work from heavily loaded parts to lightly loaded parts
- In 8 seconds, PUMI-Balance reduced a 53% vtx imbalance to 6%, at a cost of 5% elm imbalance, and edge cut increase by 1% on a 1.3B element mesh
- Applied to PIC calculations for particle balance – 20% reduction in run time

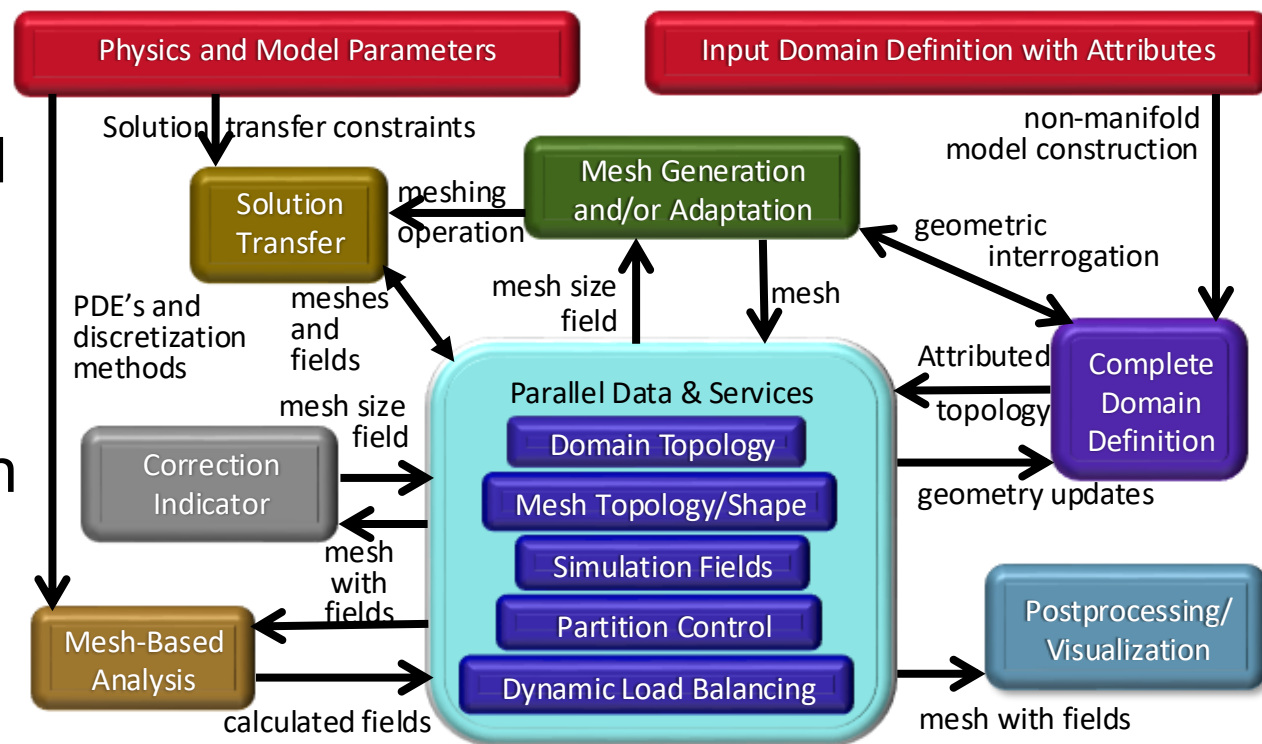


Application of EnGPar particle dynamic load balancing in a GITRm impurity transport simulation

Creation of Parallel Adaptive Loops

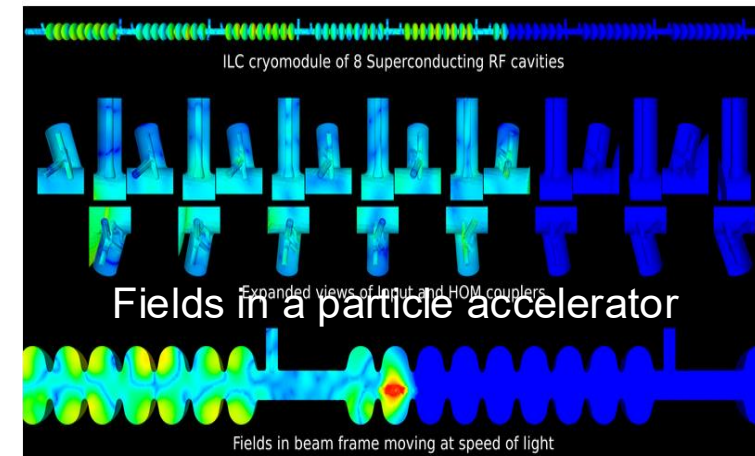
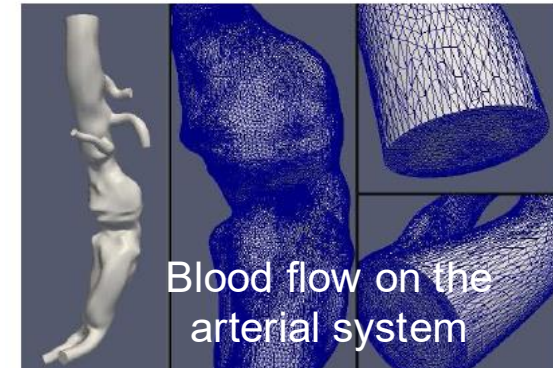
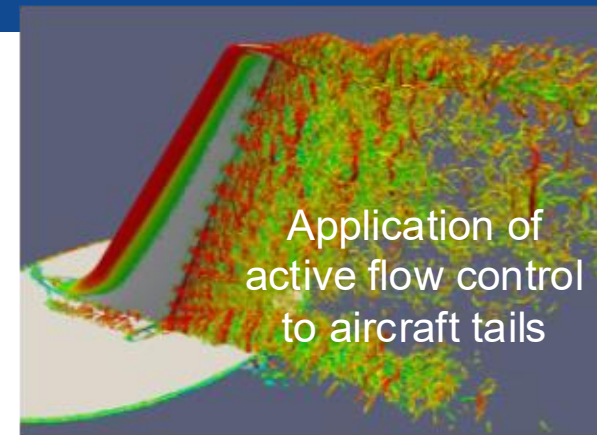
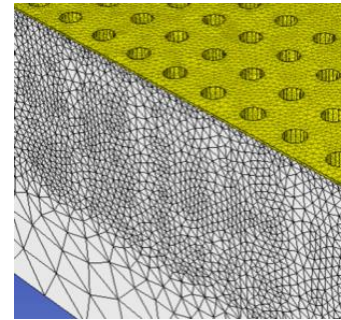
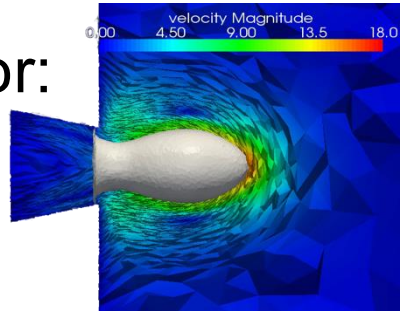
Parallel data and services used to develop adaptive simulations:

- Geometric model topology for domain linkage
- Mesh topology – it must be distributed
- Simulation fields distributed over geometric model and mesh.
- Partition control
- Dynamic load balancing required at multiple steps
- API's to link to:
 - CAD/Geometry
 - Mesh generation and adaptation
 - Error estimation
 - Field transfer



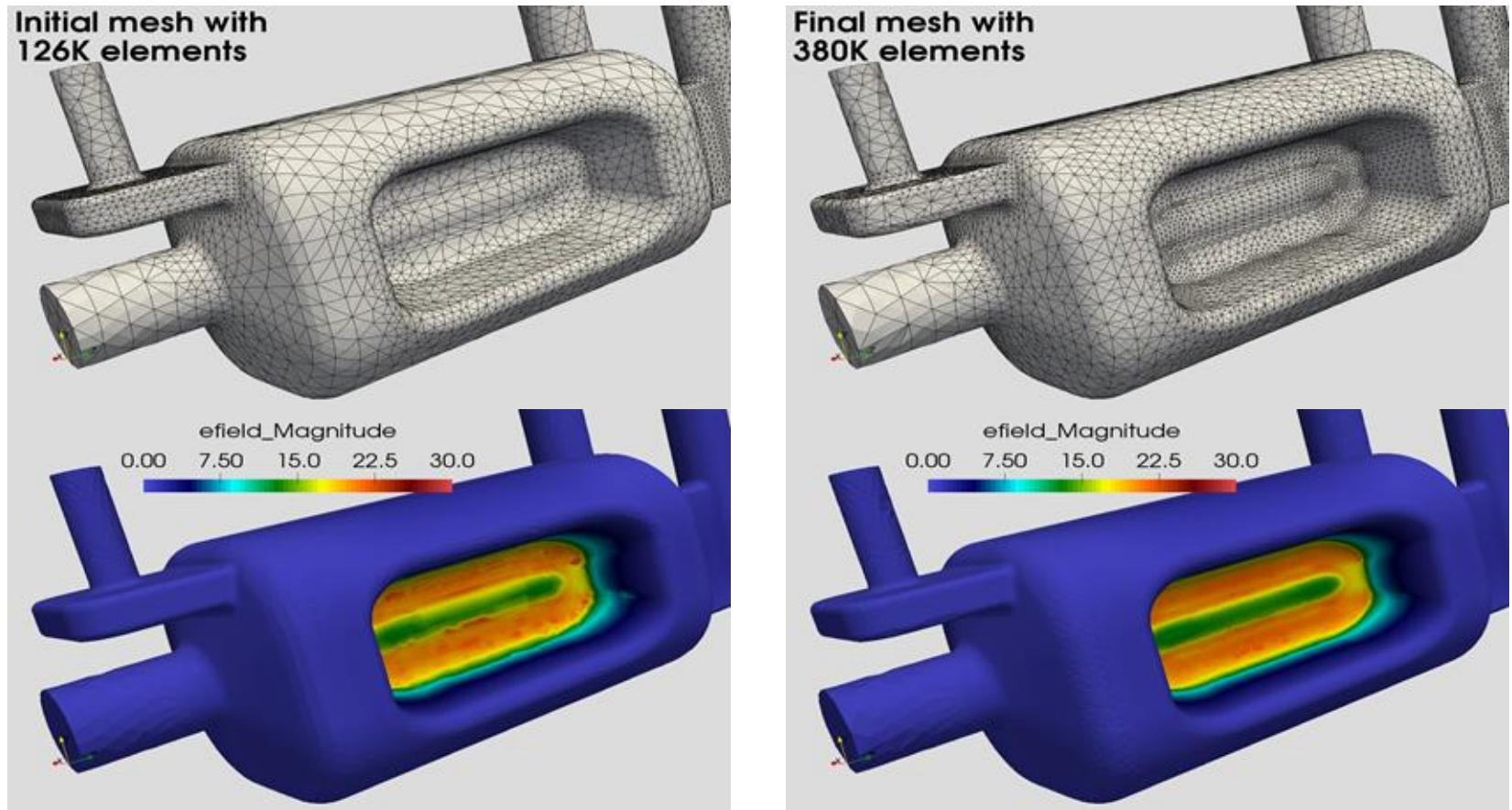
Parallel Adaptive Simulation Workflows

- In memory adaptive loops support effective data movement.
- In-memory adaptive loops for:
 - MFEM – High order FE framework
 - PHASTA – FE for NS
 - FUN3D – FV CFD
 - Proteus – multiphase FE
 - Albany – FE framework
 - ACE3P – High order FE electromagnetics
 - M3D-C1 – FE based MHD
 - Nektar++ – High order FE flow



Application interactions – Accelerator EM

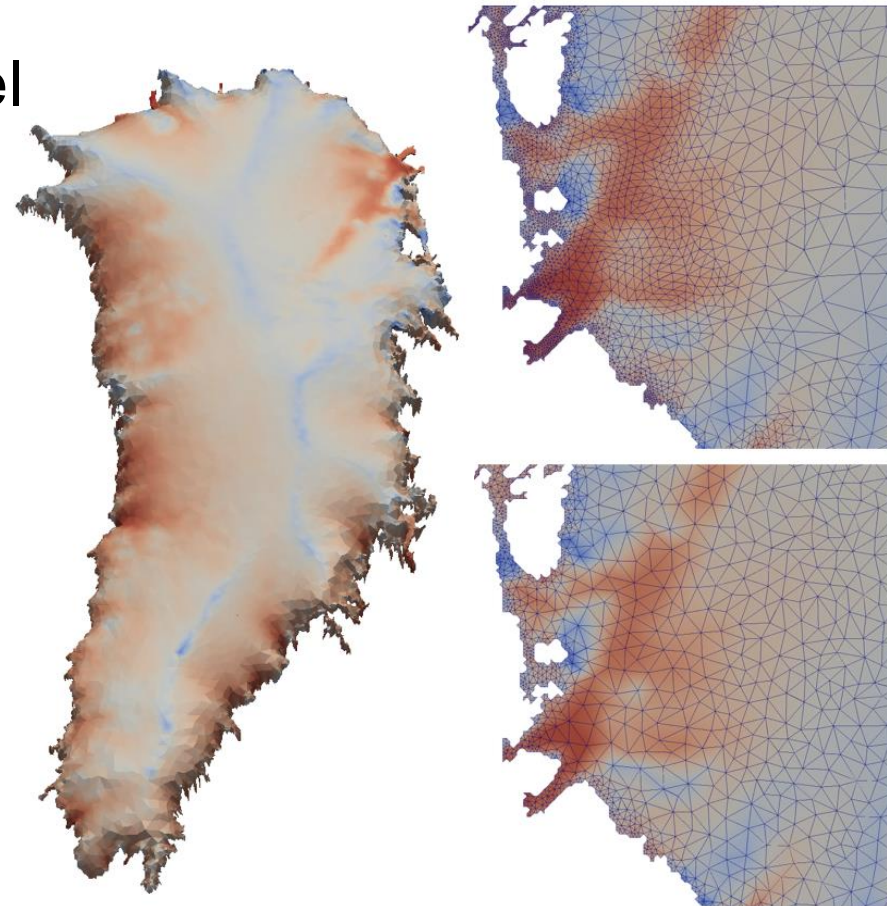
Omega3P Electro Magnetic Solver (second-order curved meshes)



This figure shows the adaptation results for the CAV17 model. (top left) shows the initial mesh with ~126K elements, (top right) shows the final (after 3 adaptation levels) mesh with ~380K elements, (bottom left) shows the first eigenmode for the electric field on the initial mesh, and (bottom right) shows the first eigenmode of the electric field on the final (adapted) mesh.

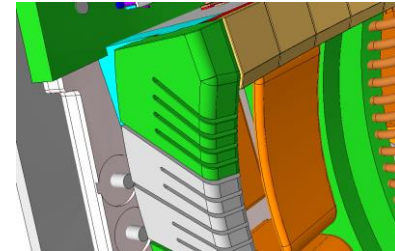
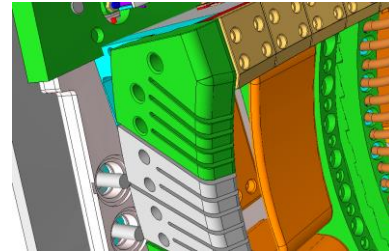
Application interactions – Land Ice

- FELIX, a component of the Albany framework is the analysis code
- PUMI-Perfromant-Adapt parallel mesh adaptation is integrated with Albany to do:
 - Estimate error
 - Adapt the mesh
- Ice sheet mesh is modified to minimize degrees of freedom
- Field of interest is the ice sheet velocity

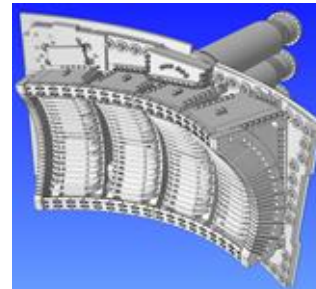


Application interactions – RF Fusion

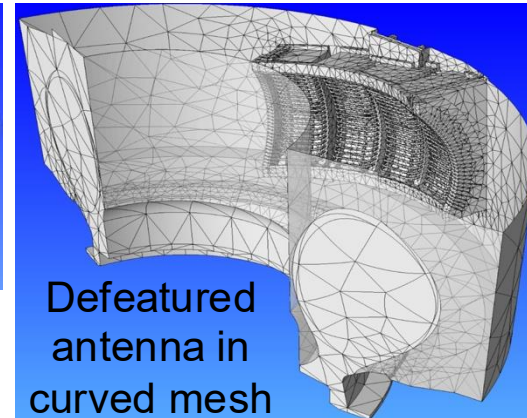
- Accurate RF simulations require:
 - Detailed antenna CAD geometry
 - CAD geometry defeaturing
 - Extracted physics curves from GEQDSK equilibrium file
 - Analysis geometry combines CAD, and physics geometry
 - 3D meshes for accurate FE calculations in MFEM
 - Projection based error estimator
 - Conforming mesh adaptation with PUMI



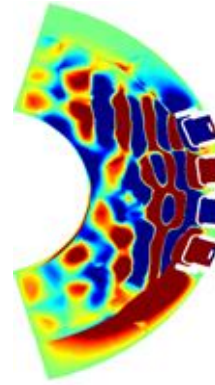
Fast elimination of unwanted features



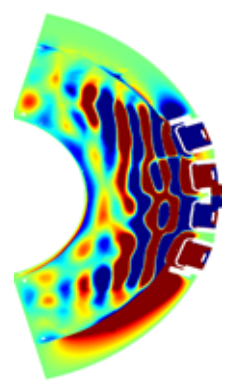
CAD of antenna array



Defeatured antenna in curved mesh



Initial Mesh

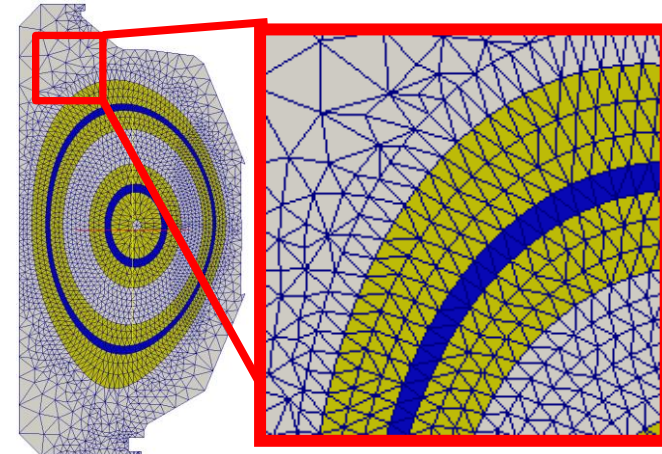
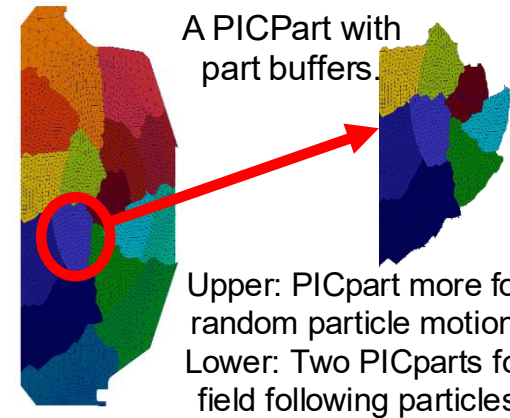
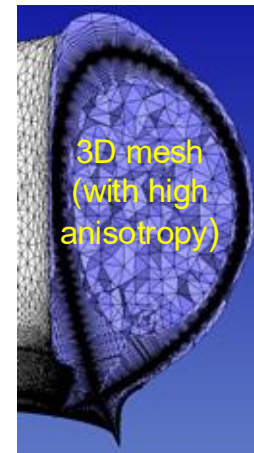
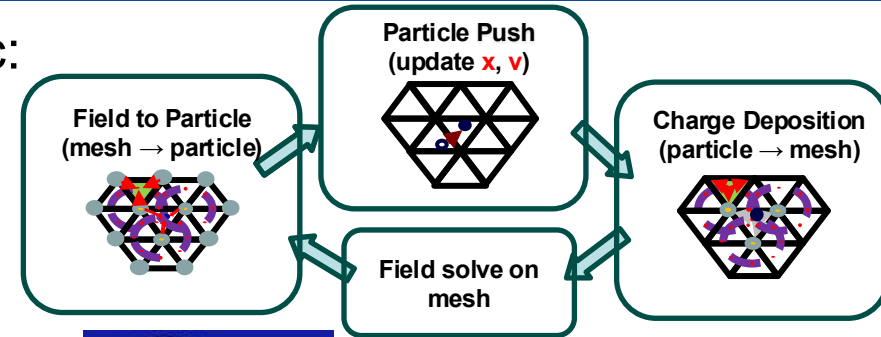


Final Adapted Mesh

Supporting Unstructured Mesh for Particle-in-Cell Calculations

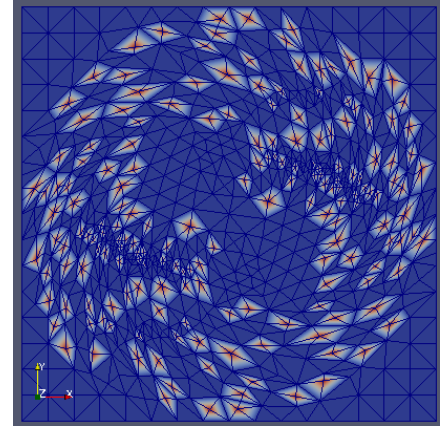
PUMI-Pic data structures are mesh centric:

- Mesh is distributed as needed by the application in terms of PICparts
- Mesh can be graded and anisotropic
- Particle data associated with elements
- Operations take advantage of distributed mesh topology
- Mesh relation to geometry used to speed calculation for near surface physics
- Mesh distributed in PICparts:
 - Start with a partition of mesh into a set of “core parts”
 - A PICpart is defined by a “core part” and sufficient buffer to keep particles on process for one or more pushes

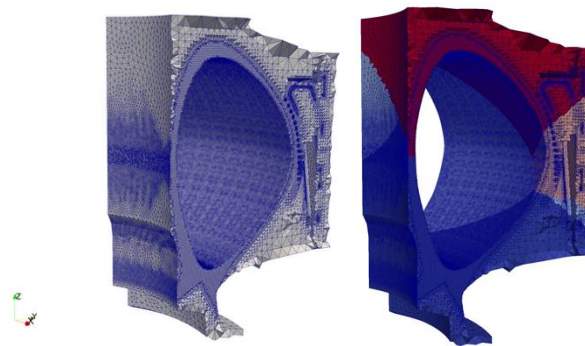


Mesh Data Structure for Heterogeneous Systems

- Mesh topology/adaptation tool – PUMI-Portable-Adapt:
 - Conforming mesh adaptation (coarsening past initial mesh, refinement, swap)
 - Manycore and GPU parallelism using Kokkos.
 - Distributed mesh via mesh partitions with MPI communications
 - Support for mesh-based fields
 - Curved mesh adaptation
 - Efficient field storage
 - Kokkos implementation on latest NVIDIA, AMD and Intel GPUs



Adaptation following rotating flow field.



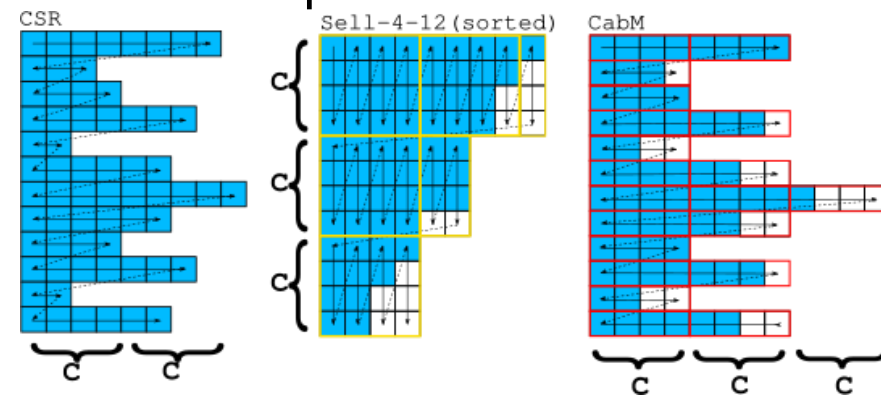
Serial and RIB partitioned mesh of RF antenna and vessel model.

triangle	0			1		
adj vertex	0	1	3	2	3	1
adj triangle	0	0	1	1	0	1
offset	0	1	3	4	6	
vertex	0	1	2	3		

Mesh entity adjacency arrays.

PUMI-Pic Particle Data Structures

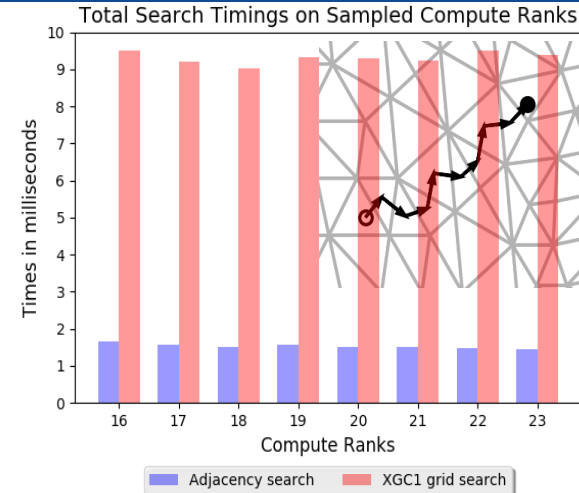
- Layout of particles in memory is critical to performance:
 - Optimizes push (sort/rebuild), scatter, and gather operations
 - Associate particles with elements for large per element particle cases
 - Support changes in the number of particles per element
 - Evenly distributes work under a range of particle distributions (e.g. uniform, Gaussian, exponential, etc.)
 - Stores a lot of particles per GPU – low overhead
- Particle data structure interface and implementation:
 - API abstracts implementation for PIC code developers
 - CSR, Sell-C- σ , CabanaM, DPS
 - Performance is a function of particle distribution
 - Cabana AoSoA w/a CSR index of elements-to-particles are promising
 - DPS particle structure for low particle density applications



Left to Right: CSR, SCS with vertical slicing (yellow boxes), CabanaM (red boxes are SOAs). C is a team of threads.

PIC Operations Supported by PUMI-Pic

- Particle push
- Adjacency based search
 - Faster than grid based search
- Element-to-particle association update
- Particle migration
- Particle path visualization
- Mesh partitioning w/buffer regions
- Mesh field association
- Fast construction of elements within given distance of mesh faces on model surface
- Poisson field solve using PETSc DMplex on GPUs



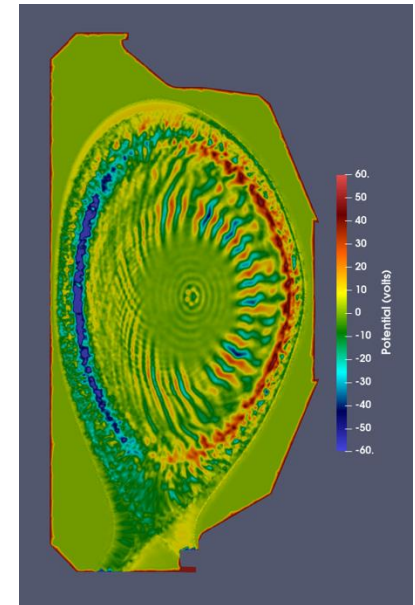
PUMI-Pic based XGCm Edge Plasma Code

XGCm is a version of XGC built on PUMI-Pic:

- All operations on GPUs – push, gather/scatter, etc.

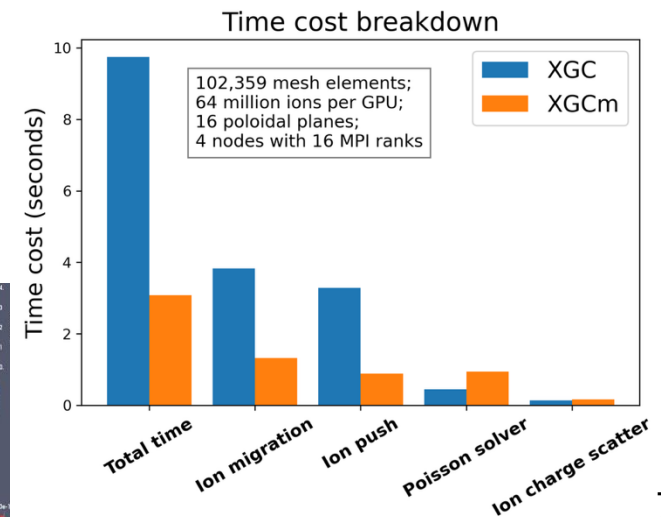
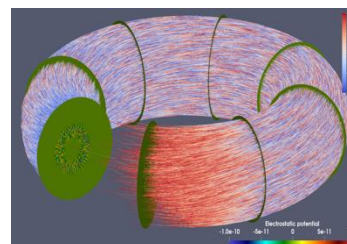
Testing of PUMI-Pic for use in XGC like push:

- 2M elements, 1M vertices, 2 to 128 poloidal planes.
- Pseudo push and particle-to-mesh gyro scatter.
- Tested on up to 24,576 GPUs with 1.1 trillion particles, for 100 iterations: push, adjacency.
- PUMI-Pic weak scaling up to 24,576 GPUs (4096 nodes) with 48 million particles per GPU.



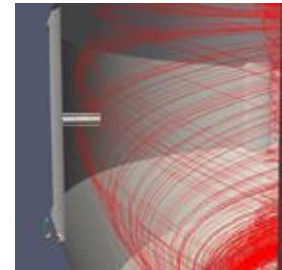
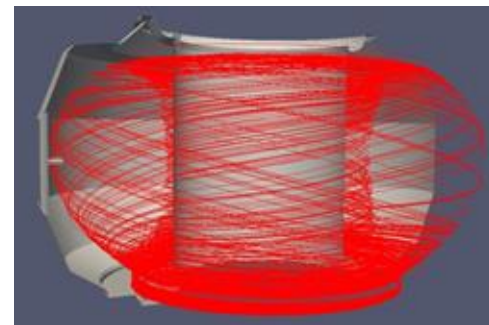
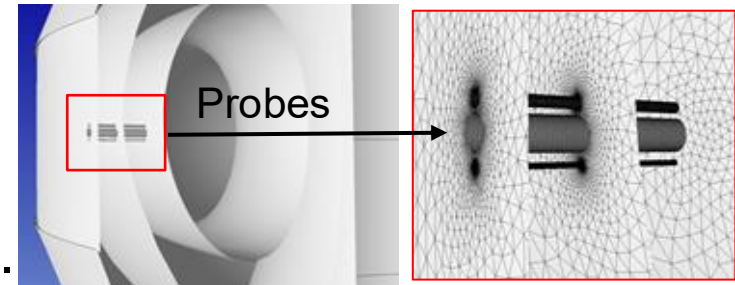
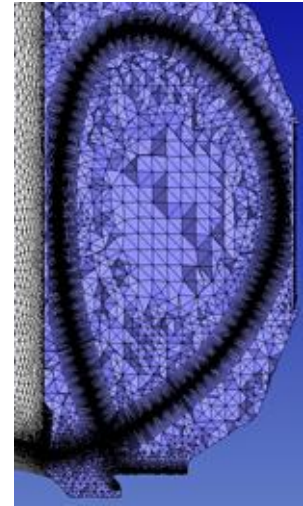
Total time comparison:

- Ran on NERSC's Perlmutter.
- XGCm 3 times faster than XGC for adiabatic electron case, 21% faster for kinetic electron case.



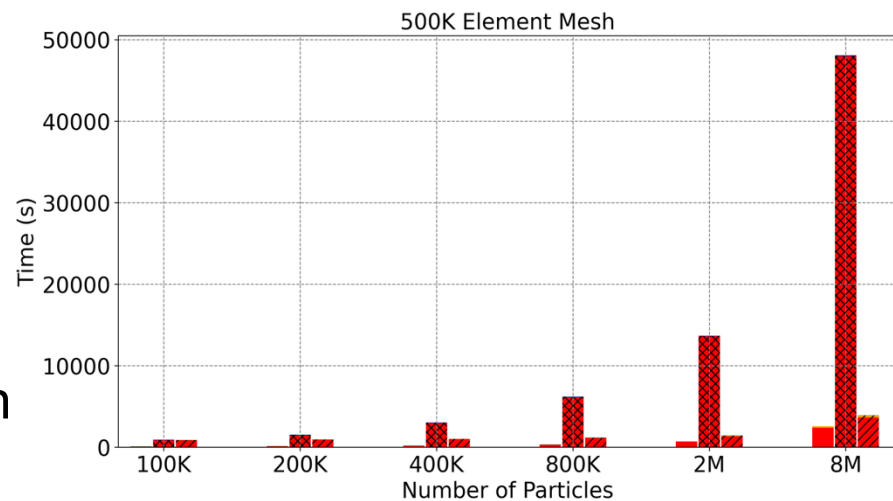
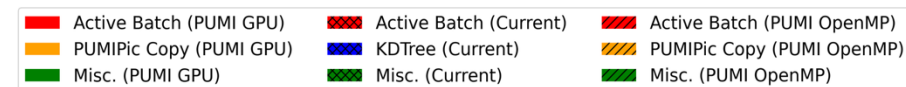
PUMIPic based GITRm Impurity Transport Code

- Incorporates impurity transport capabilities of GITR.
- 3D mesh for cases including divertors, tiles, limiters, specific diagnostics/probes etc.
- Status:
 - Physics equivalent to GITR.
 - Efficient Multi-species capabilities.
 - Anisotropy mesh for accurate field transfer.
 - Field transfer from SOLPS to 3D mesh.
 - Non-uniform particle distribution – evolves quickly in time.
 - Load balancing particles via PUMI-Balance.
 - Distance to boundary for sheath E field.
 - Post-processing on 3D unstructured mesh.

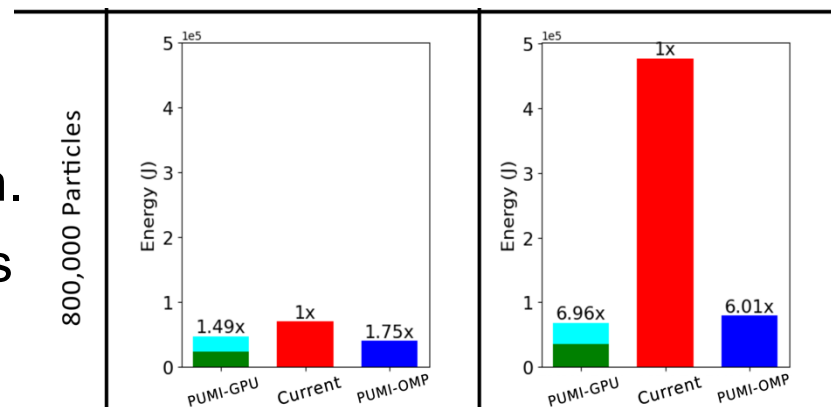


PUMI-Tally - GPU Acceleration of Monte Carlo Tallies

- Builds on PUMI-PIC core to support mesh-based tallies.
- Implements track length tallies.
 - Sum weighted length of moves.
- Batch the particles for data parallel.
- Tested on both GPU and CPU's.
- Compared to OpenMC implementation (does not batch particles):
 - 19.7 times faster on and NVIDIA A100.
 - 9.2 times faster using OpenMP on two AMD EPYC 7763 CPUs.
 - Field transfer from SOLPS to 3D mesh.
 - GPU version demonstrated a 6.7 times improvement in energy consumption.



Computation Time

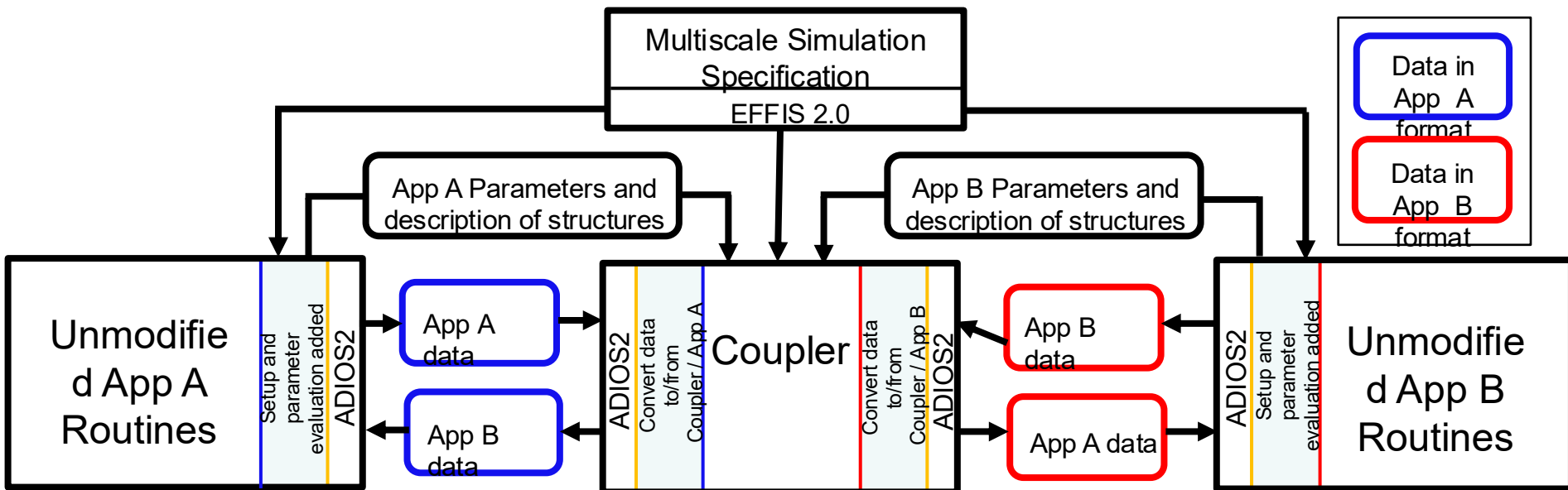


Energy consumption

Parallel Coupler for Multimodel Simulations (PCMS)

Goals of PCMS:

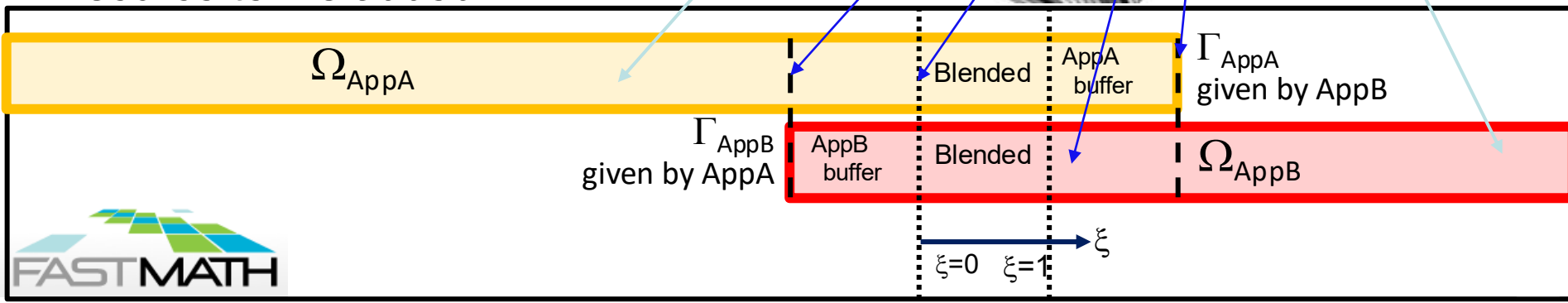
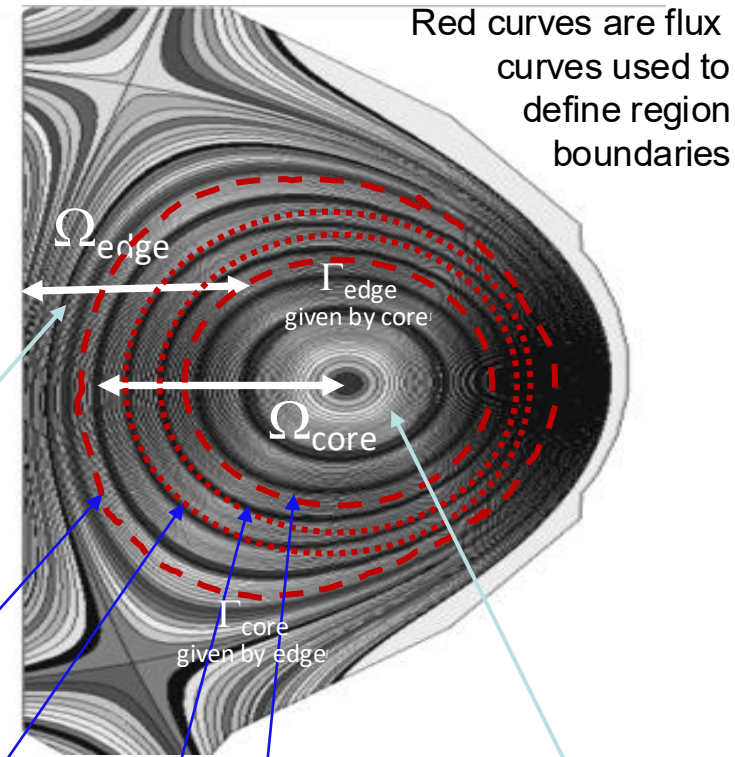
- Keep applications clean. Only modification to application codes is access its data structures.
- General structures and functions for coupling operations.
- Make effective use of the massively parallel, heterogeneous computing systems.



In-Memory Coupling of Fusion Codes

Each application solves its model(s) over a portion of the domain.

- The domains overlap: The overlap can include three subregions
- The blended region in which the fields are coupled based on a field blending strategy
- A buffer region for Application A (edge) in which the “right” end boundary conditions are determined by Application B (core) and/or source terms added
- A buffer region for Application B (core) in which the “left” end boundary conditions are determined by Application A (edge) and/or source terms added



“Rendezvous” Algorithm to Control Coupler Domain Partition

Challenge:

- The coupled applications need to control their domain partitioning to meet their specific needs

Approach:

- Coupling applications A and B, each of which has its own partitions
- Rendezvous algorithm uses a third partition to coordinate data transfers between the applications
- “Rendezvous” algorithm “enables scalable algorithms which are most useful when processors neither know which other processors to send data to, nor which other processors will be sending data to them”

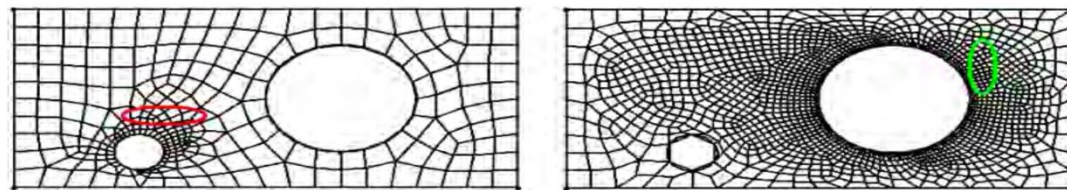


Fig. 4. Thermal (left) and stress (right) grids. The colored ovals are clumps of grid cells the same processor owns in the two grids.

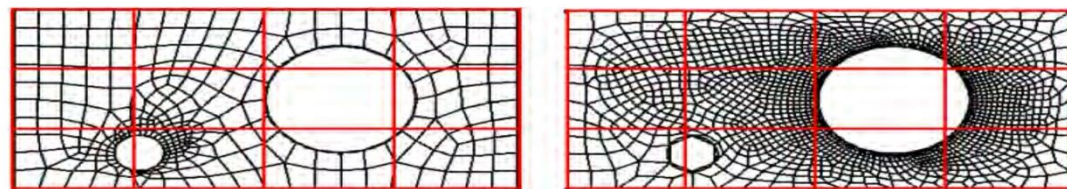
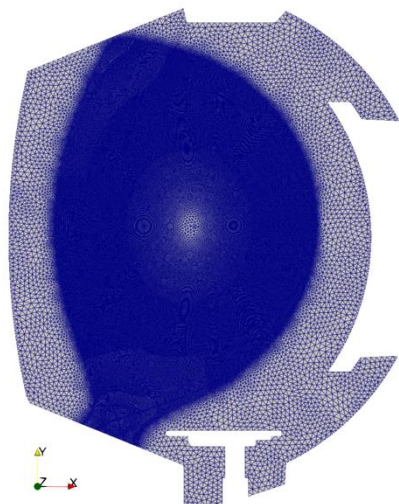
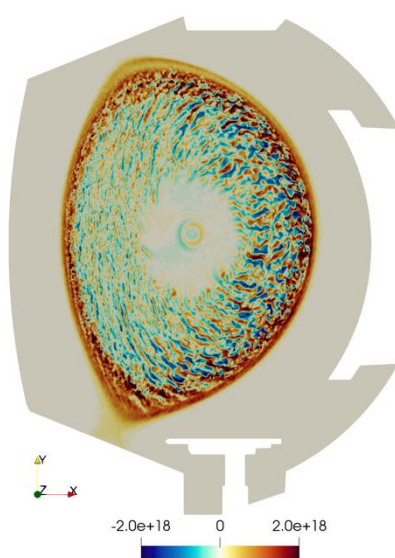


Fig. 5. Rendezvous decomposition overlaid on thermal and stress grids.

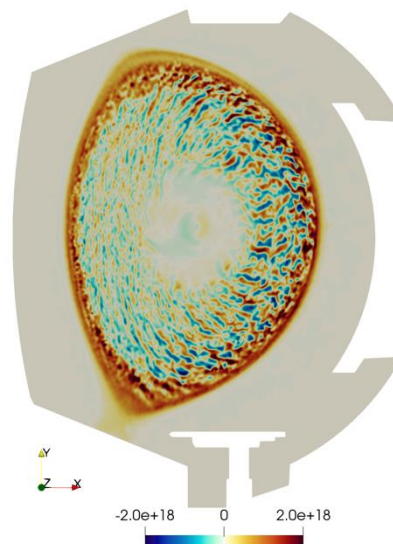
Example Field Transfer with RBF on LCPP



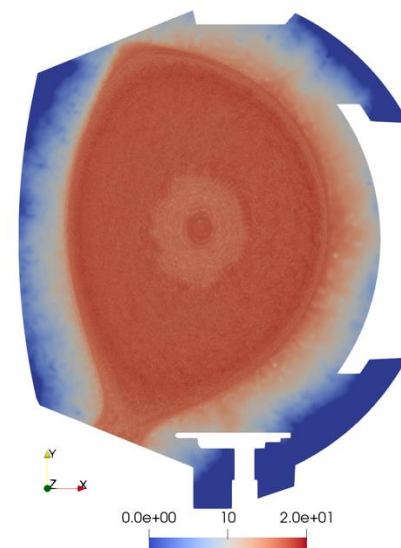
XGC mesh with 611,359 elements and 306,002 nodes



density loaded as the **exact/initial field** in the coupler app



density after 10 **MLS-RBF** interpolation operation: cell to node and node to cell in each iteration. $\rho = 0.007$

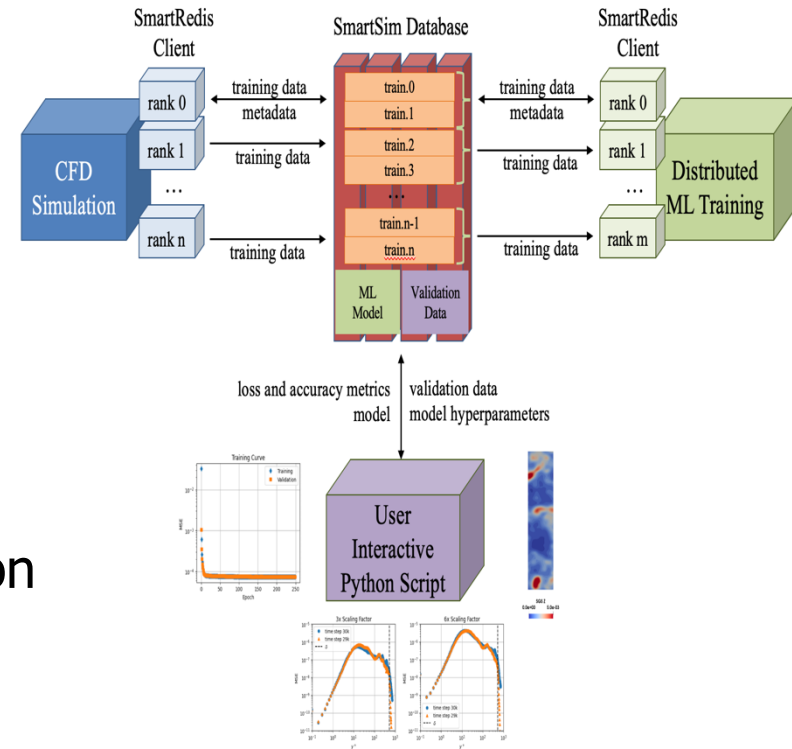


density error field after 10 **MLS-RBF** interpolation operation: cell to node and node to cell in each iteration. $\rho = 0.007$

$$\text{error} = \log_{10}|f_i - g_i|$$

In Situ Machine Learning During Simulations

- Simulation data too large to save
=> need Online Machine Learning
- Dynamic PDE data stream provides more and better training data
- Training using Smartsim/SmartRedis
- Clustered and co-located deployment of components utilizing CPU and/or GPU
- Scalable: negligible overhead on simulation
- Data parallel training with Horovod and PyTorch DDP
- No dependency on analysis code/ARCH (tested with PHASTA (legacy) and libCEED (ECP) on Aurora and Polaris)



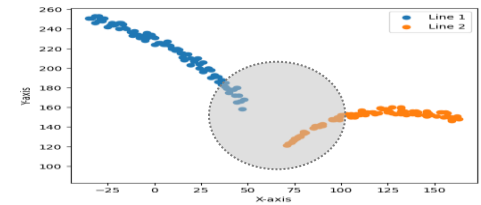
Online training with user interactive script

R.Balin et al., In Situ Framework for Coupling Machine Learning with Application to CFD, <https://doi.org/10.48550/arXiv.2306.12900>

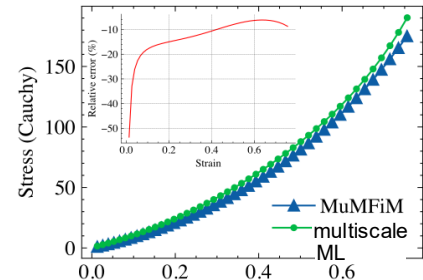
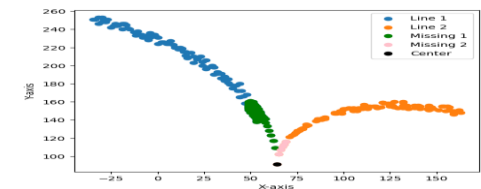
Mesh Related AI/ML Developments

Robust feature detection/processing:

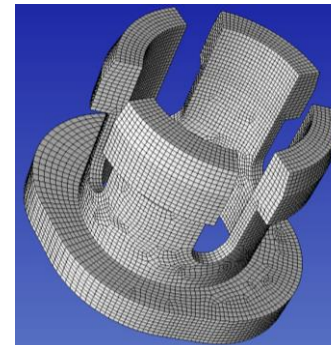
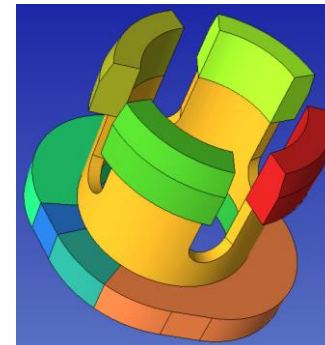
- Physics-based and AI/ML sensors: neural network processes fragmented/noisy data
- Application: anisotropic diffusion transport in fusion devices, ground line dynamics in ice sheets
- Physics informed material models
- Machine learned constitutive models can reduce runtime cost of upscaling FEM simulations by orders of magnitude while retaining 80% of accuracy
- ML Agents for automated hex meshing
- Reinforced learning agent decomposes CAD modes into regions suitable for hexahedral mesh generation



Improve sensor data



Comparing ML to Multiscale



Closing Remarks

The FASTMath unstructured mesh tools support the development of simulation workflows that:

- Can address complex physics problems
- Easily integrated with CAD models of general 3D domains
- Support the application of simulation automation and adaptive mesh control
- Make effective use of the massively parallel, heterogeneous computing systems

For more information contact Mark Shephard,
shephard@rpi.edu