FASTMath Unstructured Mesh Technologies

E. Boman[1], V. Dobrev[2], D.A. Ibanez[1], K.E. Jansen[3], T. Kolev[2], J.Merson[4], O. Sahni[4], M.S. Shephard[4], G.M. Slota[4], C.W. Smith[4] , V. Tomov[2]

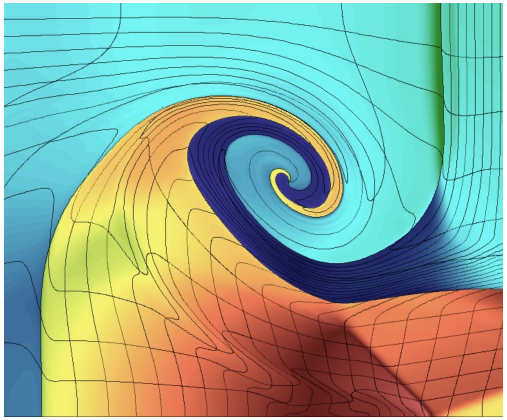
[1]Sandia National Laboratories
[2]Lawrence Livermore National Laboratory
[3]University of Colorado
[4]Rensselaer Polytechnic Institute

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***

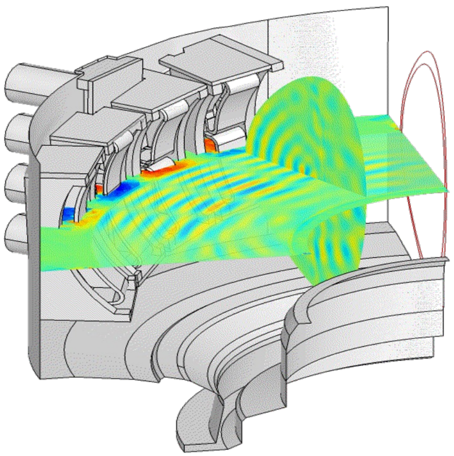
$$H(grad) \xrightarrow{\nabla} H(curl) \xrightarrow{\nabla\times} H(div) \xrightarrow{\nabla\cdot} L_2$$

"nodes"	"edges"	"faces"	"elems"
High-order kinematics	High-order MHD	High-order rad. diffusion	High-order thermodynamics



8th order Lagrangian simulation of shock triple-point interaction

- ***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



Core-Edge tokamak EM wave propagation

- ***Need new (interesting!) R&D for full benefits***
  - meshing, discretizations, solvers, AMR, UQ, visualization, …

Modular Finite Element Methods (MFEM)

Flexible discretizations on unstructured grids

- Triangular, quadrilateral, tetrahedral, hexahedral, prism; 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

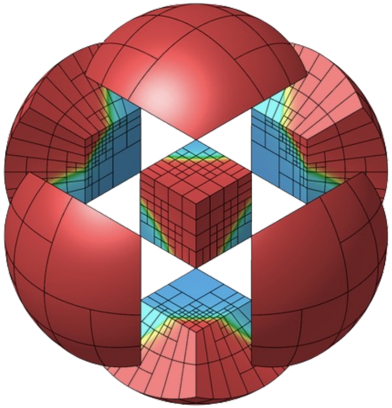
- Arbitrary-order H1, H(curl), H(div)- and L2 elements
- Arbitrary order curvilinear meshes
- MPI scalable to millions of cores + GPU accelerated
- Enables development from laptops to exascale machines.

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
(v4.7, May 2024)

Example 1 – Laplace equation

- Mesh

```
// 2. Read the mesh from the given mesh file. We can handle triangular,
//    quadrilateral, tetrahedral, hexahedral, surface and volume meshes with
//    the same code.
Mesh *mesh;
ifstream imesh(mesh_file);
if (!imesh)
{
   cerr << "\nCan not open mesh file: " << mesh_file << '\n' << endl;
   return 2;
}
mesh = new Mesh(imesh, 1, 1);
imesh.close();
int dim = mesh->Dimension();

// 3. Refine the mesh to increase the resolution. In this example we do
//    'ref_levels' of uniform refinement. We choose 'ref_levels' to be the
//    largest number that gives a final mesh with no more than 50,000
//    elements.
{
   int ref_levels =
      (int)floor(log(50000./mesh->GetNE())/log(2.)/dim);
   for (int l = 0; l < ref_levels; l++)
      mesh->UniformRefinement();
}
```

- Finite element space

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

- Initial guess, linear/bilinear forms

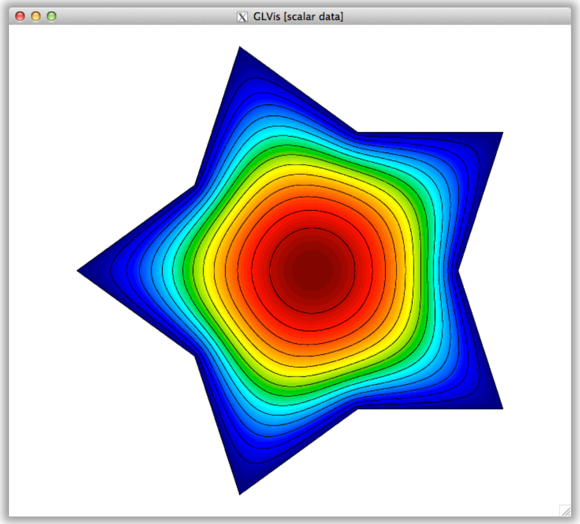
```
// 5. Set up the linear form b(.) which corresponds to the right-hand side of
//    the FEM linear system, which in this case is (1,phi_i) where phi_i are
//    the basis functions in the finite element fespace.
LinearForm *b = new LinearForm(fespace);
ConstantCoefficient one(1.0);
b->AddDomainIntegrator(new DomainLFIntegrator(one));
b->Assemble();

// 6. Define the solution vector x as a finite element grid function
//    corresponding to fespace. Initialize x with initial guess of zero,
//    which satisfies the boundary conditions.
GridFunction x(fespace);
x = 0.0;

// 7. Set up the bilinear form a(.,.) on the finite element space
//    corresponding to the Laplacian operator -Delta, by adding the Diffusion
//    domain integrator and imposing homogeneous Dirichlet boundary
//    conditions. The boundary conditions are implemented by marking all the
//    boundary attributes from the mesh as essential (Dirichlet). After
//    assembly and finalizing we extract the corresponding sparse matrix A.
BilinearForm *a = new BilinearForm(fespace);
a->AddDomainIntegrator(new DiffusionIntegrator(one));
a->Assemble();
Array<int> ess_bdr(mesh->bdr_attributes.Max());
ess_bdr = 1;
a->EliminateEssentialBC(ess_bdr, x, *b);
a->Finalize();
const SparseMatrix &A = a->SpMat();
```

- Linear solve

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

- Visualization

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



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

Example 1 – Laplace equation

- Mesh

```
   // 2. Read the mesh from the given mesh file. We can handle triangular,
   //    quadrilateral, tetrahedral, hexahedral, surface and volume meshes with
   //    the same code.
   Mesh *mesh;
   ifstream imesh(mesh_file);
   if (!imesh)
   {
      cerr << "\nCan not open mesh file: " << mesh_file << '\n' << endl;
      return 2;
   }
   mesh = new Mesh(imesh, 1, 1);
   imesh.close();
   int dim = mesh->Dimension();

   // 3. Refine the mesh to increase the resolution. In this example we do
   //    'ref_levels' of uniform refinement. We choose 'ref_levels' to be the
   //    largest number that gives a final mesh with no more than 50,000
   //    elements.
   {
      int ref_levels =
         (int)floor(log(50000./mesh->GetNE())/log(2.)/dim);
      for (int l = 0; l < ref_levels; l++)
         mesh->UniformRefinement();
   }
```

Example 1 – Laplace equation

- Finite element space

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

Example 1 – Laplace equation

- Initial guess, linear/bilinear forms

```
// 5. Set up the linear form b(.) which corresponds to the right-hand side of
//    the FEM linear system, which in this case is (1,phi_i) where phi_i are
//    the basis functions in the finite element fespace.
LinearForm *b = new LinearForm(fespace);
ConstantCoefficient one(1.0);
b->AddDomainIntegrator(new DomainLFIntegrator(one));
b->Assemble();

// 6. Define the solution vector x as a finite element grid function
//    corresponding to fespace. Initialize x with initial guess of zero,
//    which satisfies the boundary conditions.
GridFunction x(fespace);
x = 0.0;

// 7. Set up the bilinear form a(.,.) on the finite element space
//    corresponding to the Laplacian operator -Delta, by adding the Diffusion
//    domain integrator and imposing homogeneous Dirichlet boundary
//    conditions. The boundary conditions are implemented by marking all the
//    boundary attributes from the mesh as essential (Dirichlet). After
//    assembly and finalizing we extract the corresponding sparse matrix A.
BilinearForm *a = new BilinearForm(fespace);
a->AddDomainIntegrator(new DiffusionIntegrator(one));
a->Assemble();
Array<int> ess_bdr(mesh->bdr_attributes.Max());
ess_bdr = 1;
a->EliminateEssentialBC(ess_bdr, x, *b);
a->Finalize();
const SparseMatrix &A = a->SpMat();
```

Example 1 – Laplace equation

- Linear solve

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

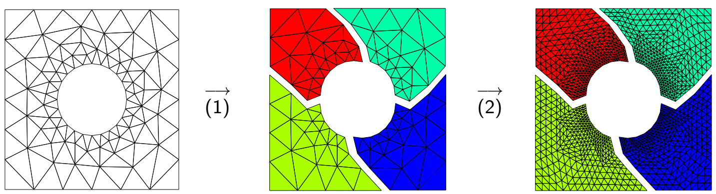
- Visualization

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

Example 1 – parallel Laplace equation

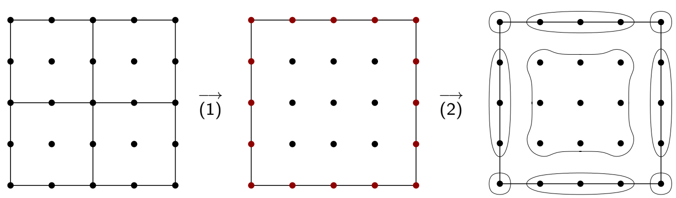
- Parallel mesh

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



- Parallel finite element space

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



$$P : true\_dof \mapsto dof$$

- Parallel initial guess, linear/bilinear forms

```
ParLinearForm *b = new ParLinearForm(fespace);
ParGridFunction x(fespace);
ParBilinearForm *a = new ParBilinearForm(fespace);
```

- Parallel assembly

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

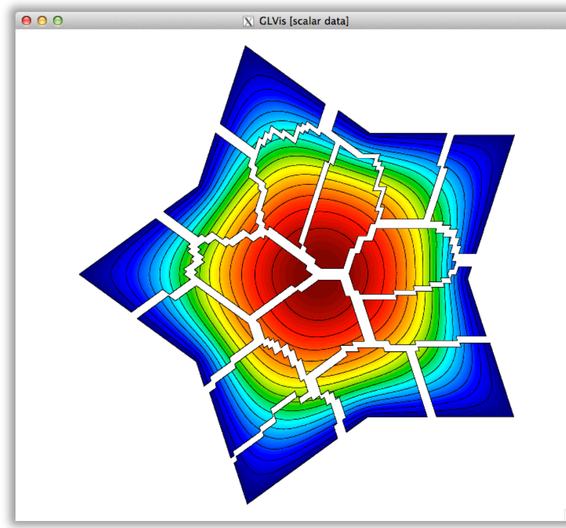
$$A = P^T a P \qquad B = P^T b \qquad x = PX$$

- Parallel linear solve with AMG

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

- Visualization

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



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

Example 1 – parallel Laplace equation

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

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

```
ParLinearForm *b = new ParLinearForm(fespace);
```

```
ParGridFunction x(fespace);
```

```
ParBilinearForm *a = new ParBilinearForm(fespace);
```

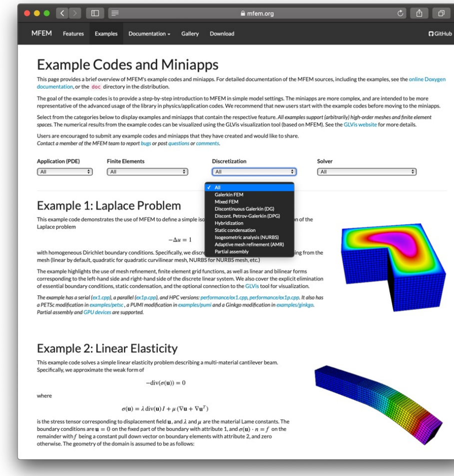
```
// 10. Define the parallel (hypre) matrix and vectors representing a(.,.),
//     b(.) and the finite element approximation.
HypreParMatrix *A = a->ParallelAssemble();
HypreParVector *B = b->ParallelAssemble();
HypreParVector *X = x.ParallelAverage();
```

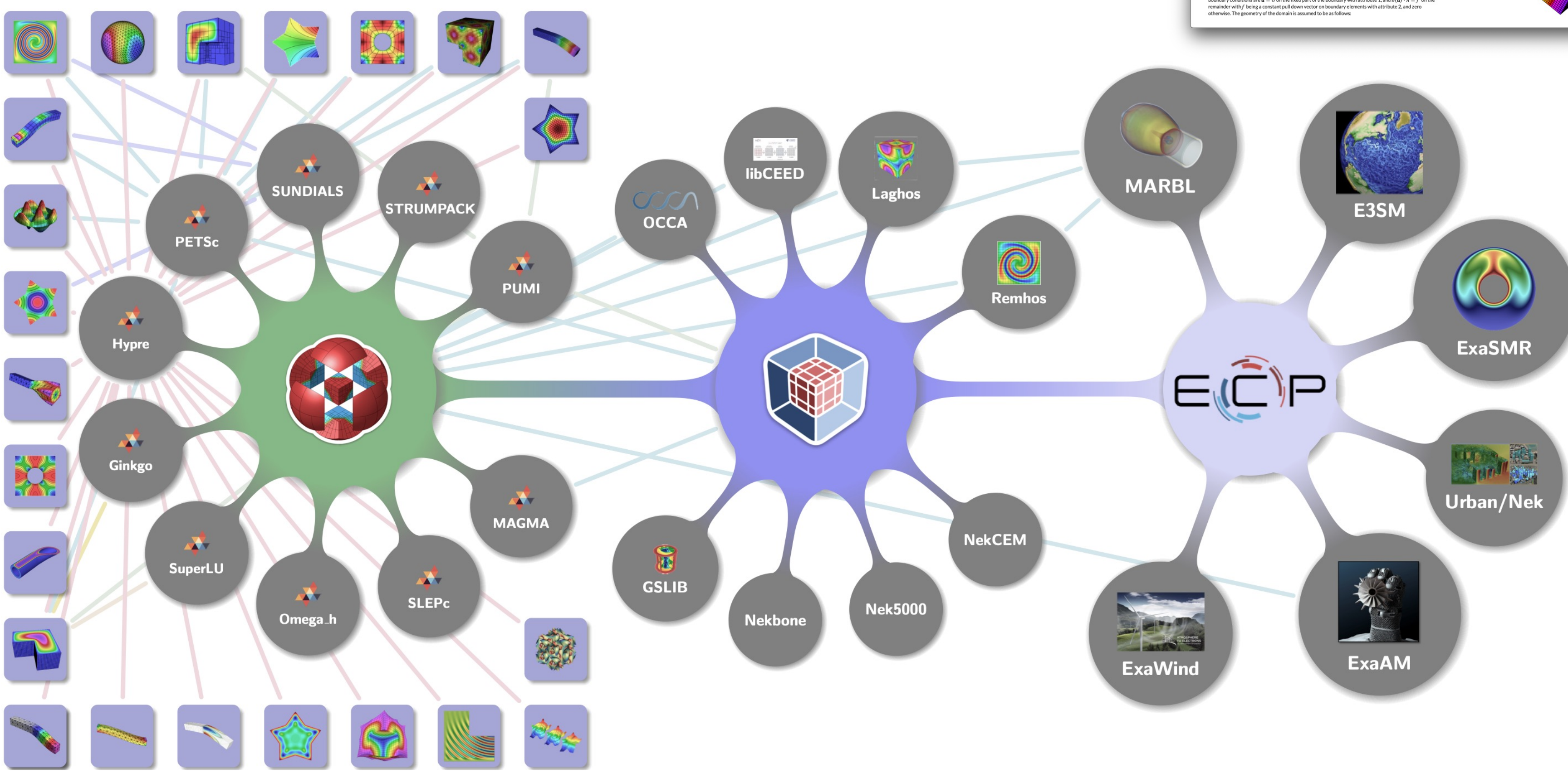
```
// 11. Define and apply a parallel PCG solver for AX=B with the BoomerAMG
//     preconditioner from hypre.
HypreSolver *amg = new HypreBoomerAMG(*A);
HyprePCG *pcg = new HyprePCG(*A);
pcg->SetTol(1e-12);
pcg->SetMaxIter(200);
pcg->SetPrintLevel(2);
pcg->SetPreconditioner(*amg);
pcg->Mult(*B, *X);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

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

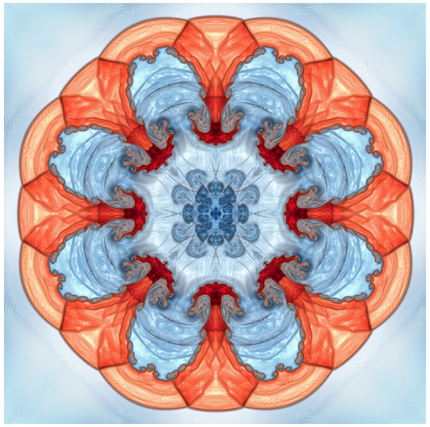
MFEM example codes: mfem.org/examples

- 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

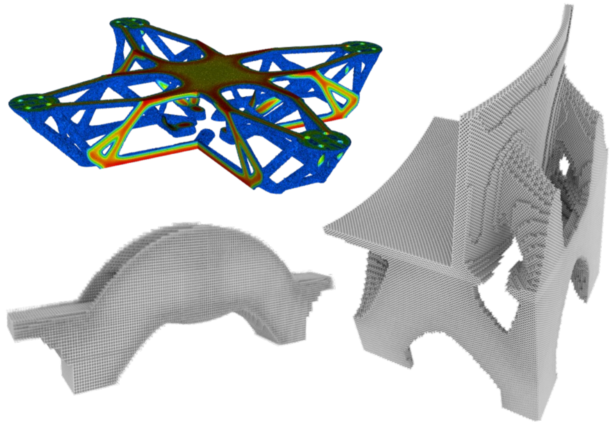




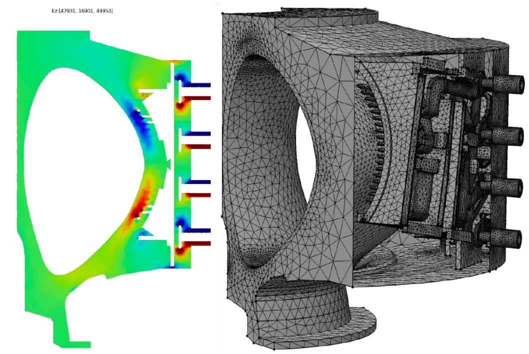
Some large-scale simulation codes powered by MFEM



Inertial confinement fusion (BLAST)

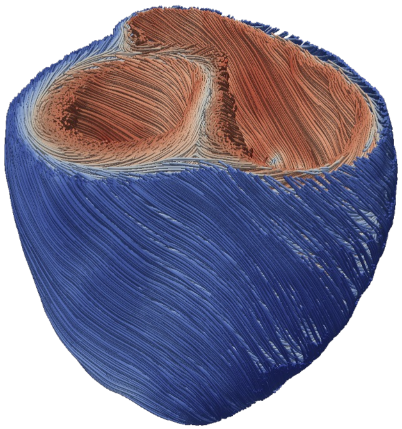


Topology optimization for additive manufacturing (LiDO)

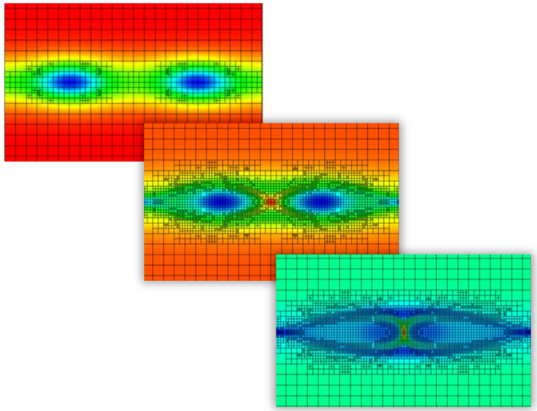


Core-edge tokamak EM wave propagation (SciDAC, RPI)

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

<u>Lagrange phase</u>

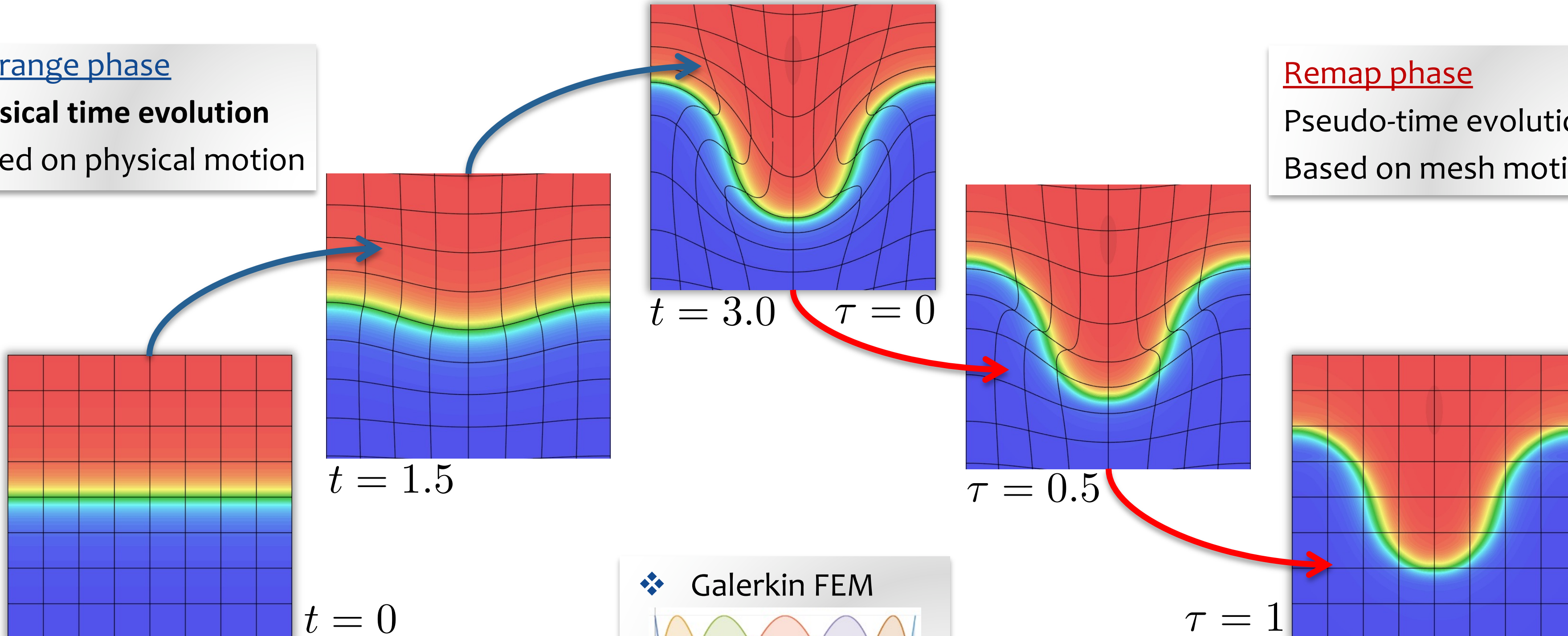
Physical time evolution

Based on physical motion

<u>Remap phase</u>

Pseudo-time evolution

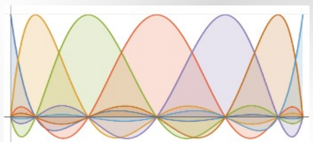
Based on mesh motion



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

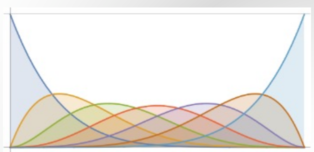
Momentum Conservation:	$\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \nabla \cdot \sigma$
Mass Conservation:	$\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\rho \nabla \cdot \vec{v}$
Energy Conservation:	$\rho \frac{\mathrm{d}e}{\mathrm{d}t} = \sigma : \nabla \vec{v}$
Equation of Motion:	$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{v}$

❖ Galerkin FEM



Gauss-Lobatto basis

❖ Discont. Galerkin

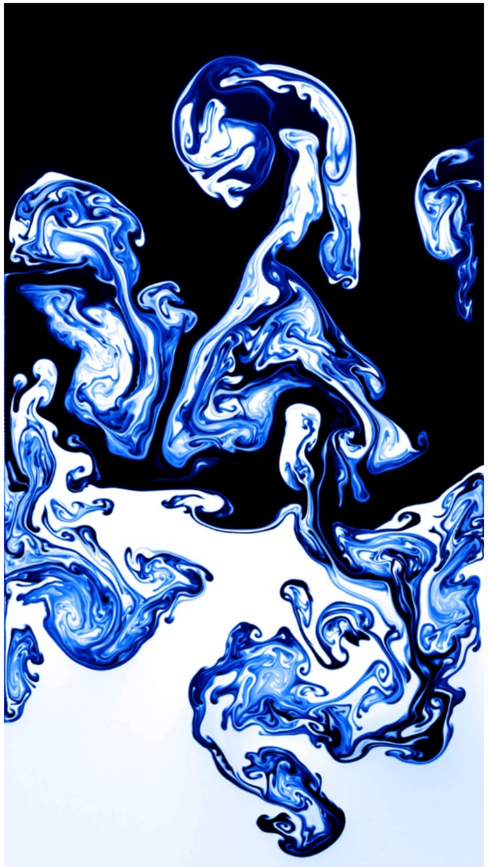


Bernstein basis

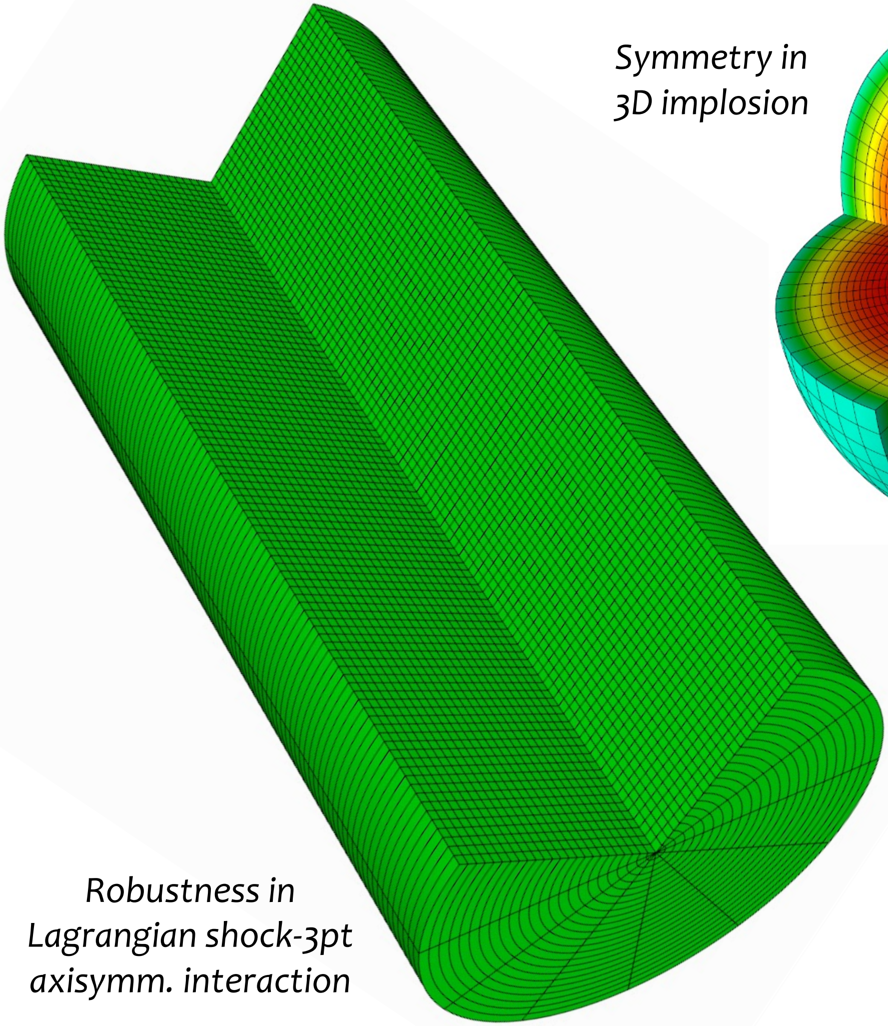
Advection phase ($\vec{c} = -\vec{v}_m$)

Momentum Conservation:	$\frac{\mathrm{d}(\rho\vec{v})}{\mathrm{d}\tau} = \vec{v}_m \cdot \nabla(\rho\vec{v})$
Mass Conservation:	$\frac{\mathrm{d}\rho}{\mathrm{d}\tau} = \vec{v}_m \cdot \nabla \rho$
Energy Conservation:	$\frac{\mathrm{d}(\rho e)}{\mathrm{d}\tau} = \vec{v}_m \cdot \nabla(\rho e)$
Mesh velocity:	$\vec{v}_m = \frac{\mathrm{d}\vec{x}}{\mathrm{d}\tau}$

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

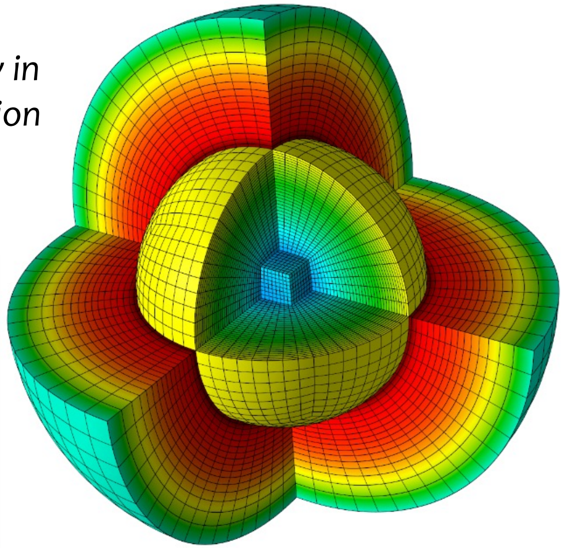


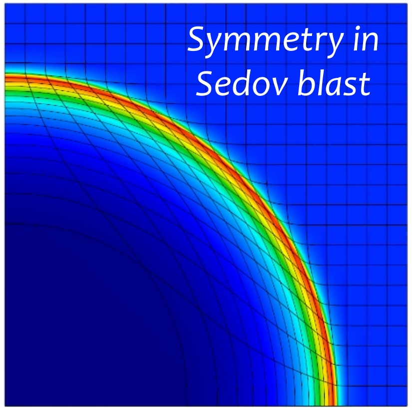
Parallel ALE for Q4 Rayleigh-Taylor instability (256 cores)



Robustness in Lagrangian shock-3pt axisymm. interaction

Symmetry in 3D implosion

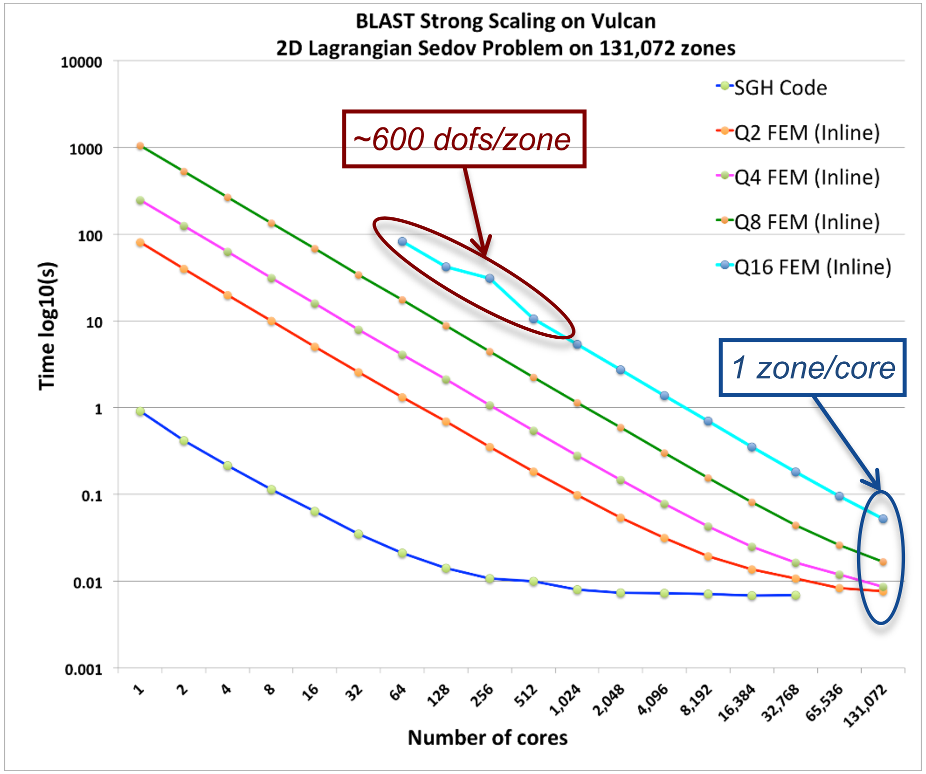




Symmetry in Sedov blast

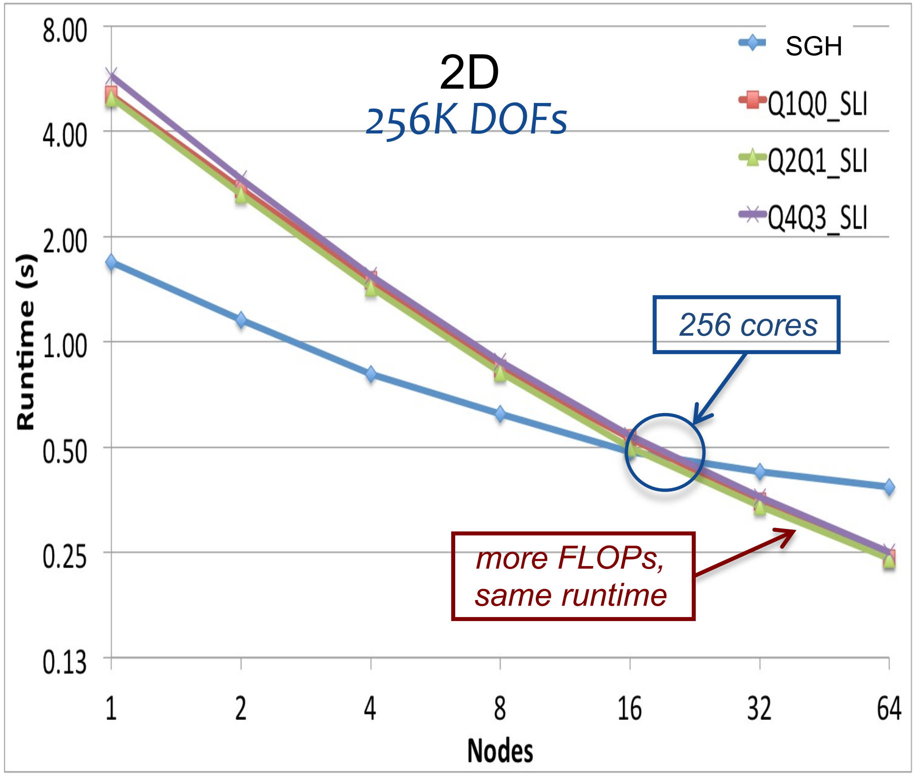
High-order finite elements have excellent strong scalability

Strong scaling, p-refinement



Finite element partial assembly

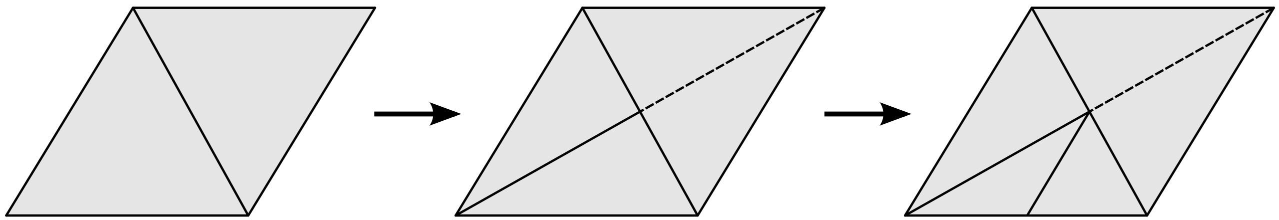
Strong scaling, fixed #dofs



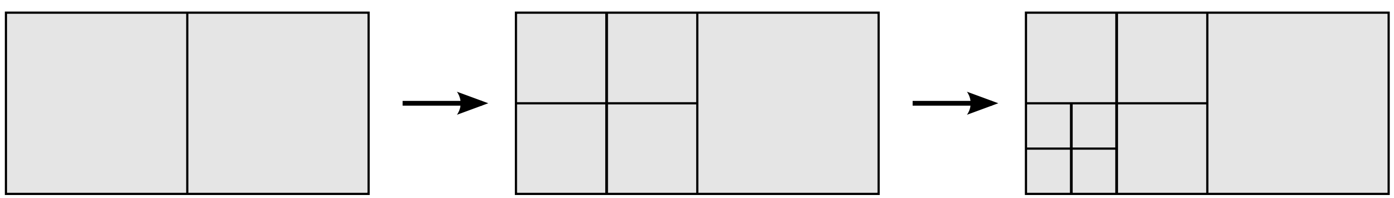
FLOPs increase faster than runtime

Conforming & Nonconforming Mesh Refinement

- Conforming refinement



- Nonconforming refinement



  - Natural for quadrilaterals and hexahedra

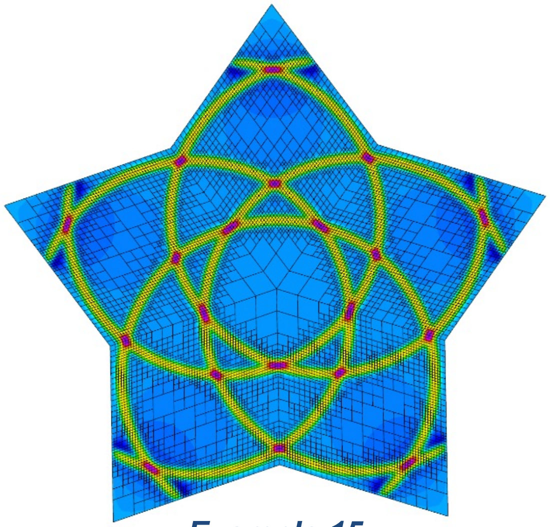
MFEM's unstructured AMR infrastructure

Adaptive mesh refinement on library level:

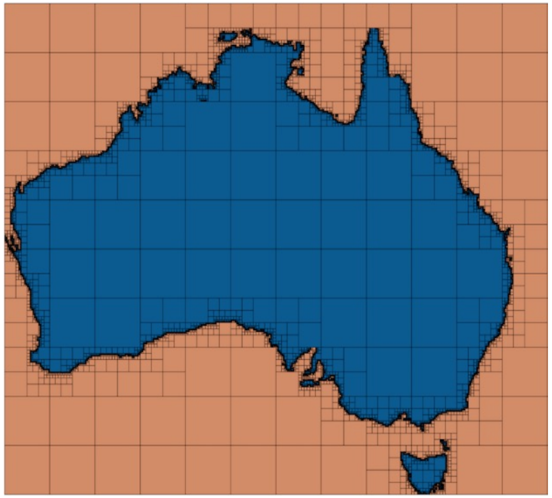
- Conforming local refinement on simplex meshes
- ***Non-conforming refinement for quad/hex meshes***
- h-refinement with fixed p

General approach:

- any high-order finite element space, H1, H(curl), H(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)



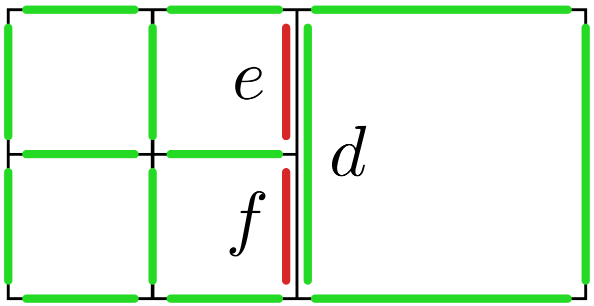
Example 15



Shaper miniapp

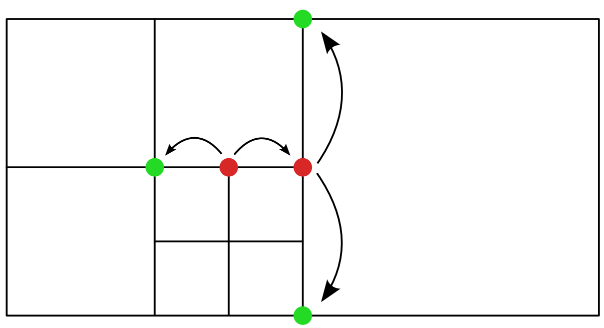
General nonconforming constraints

H(curl) elements



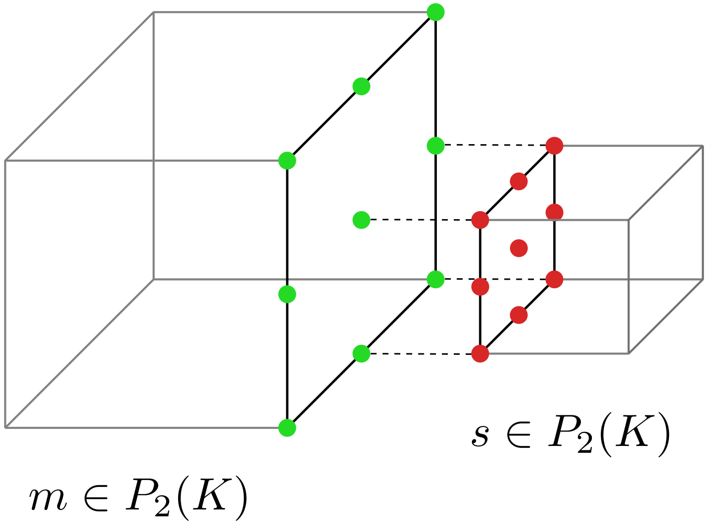
Constraint: $e = f = d/2$

Indirect constraints



More complicated in 3D…

High-order elements



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),

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

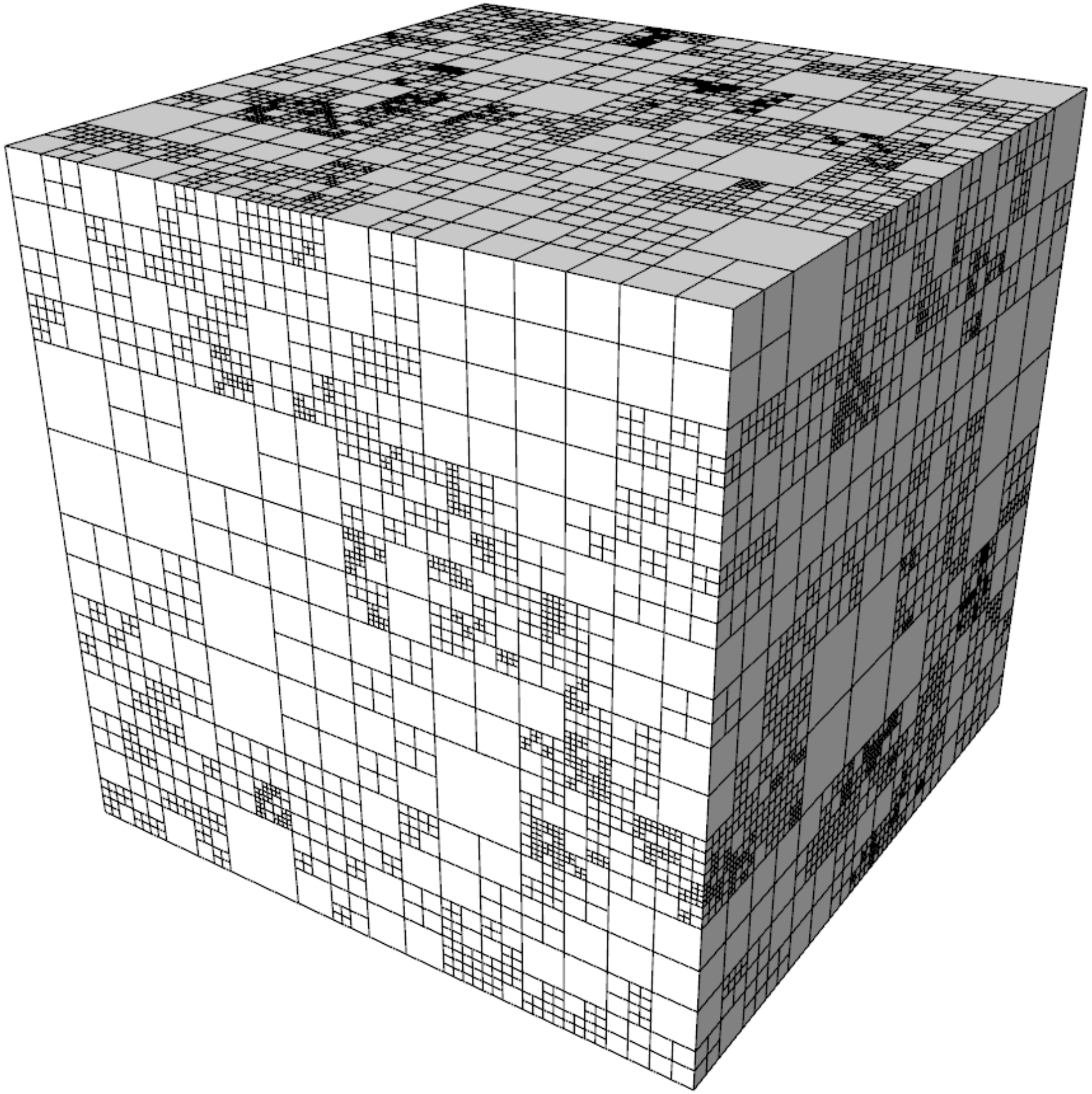
W – interpolation for slave DOFs

- Constrained problem:

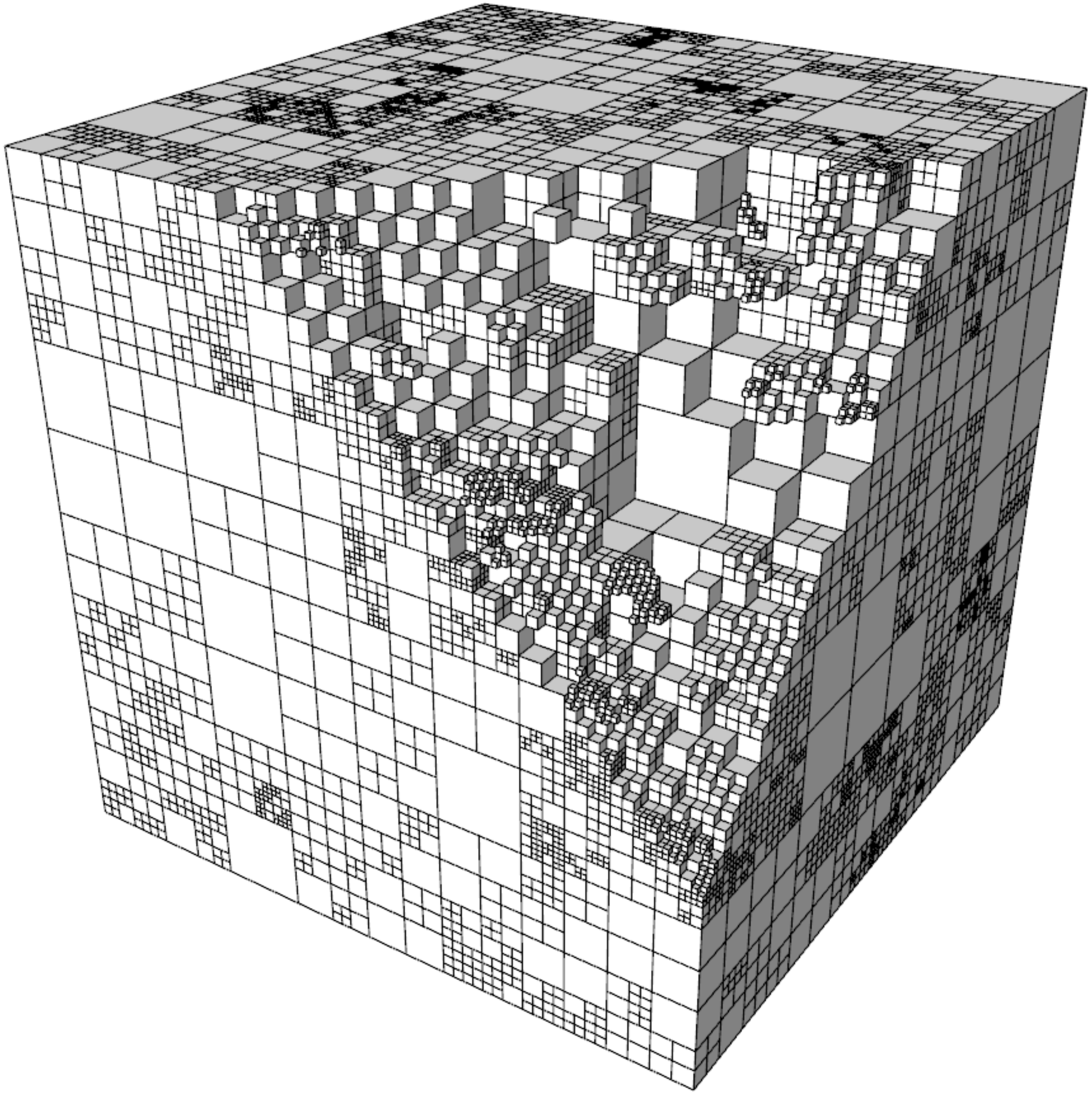
$$P^T A P x = P^T b,$$

$$y = Px.$$

Nonconforming variational restriction

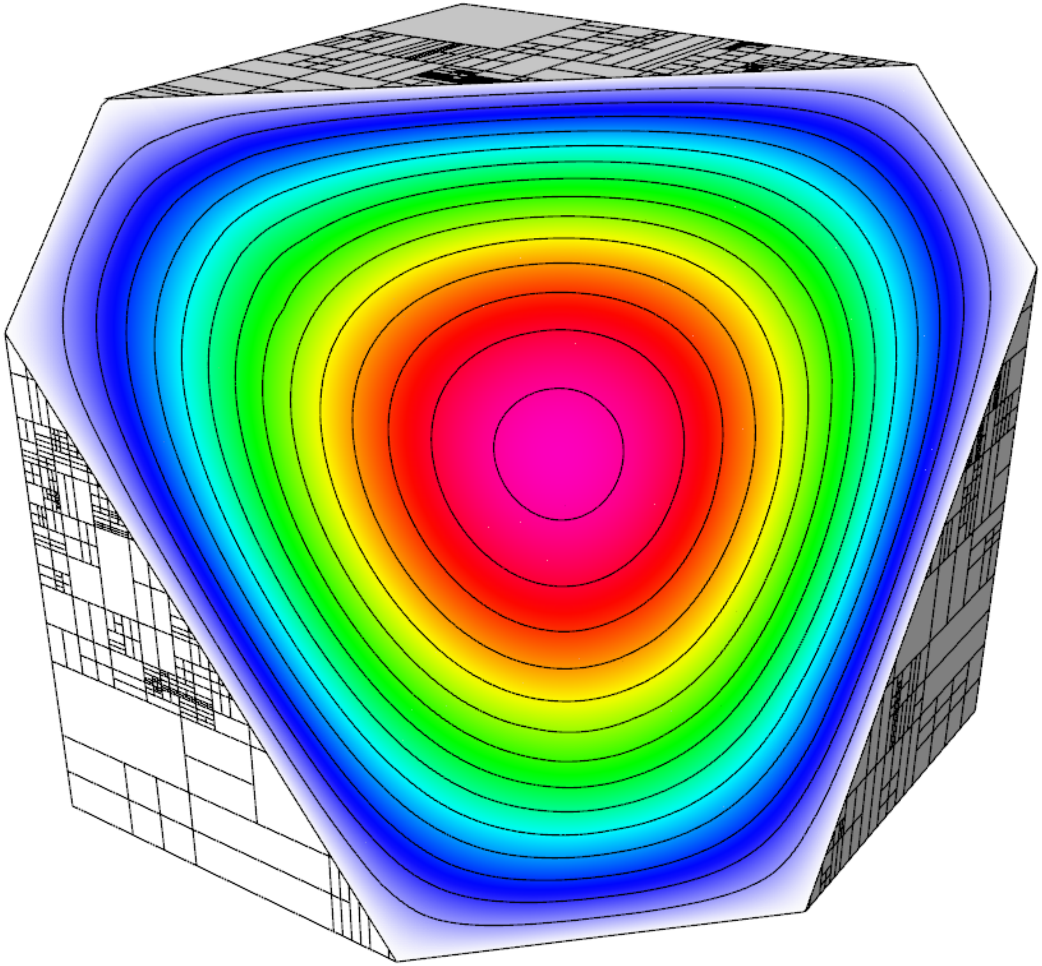


Nonconforming variational restriction



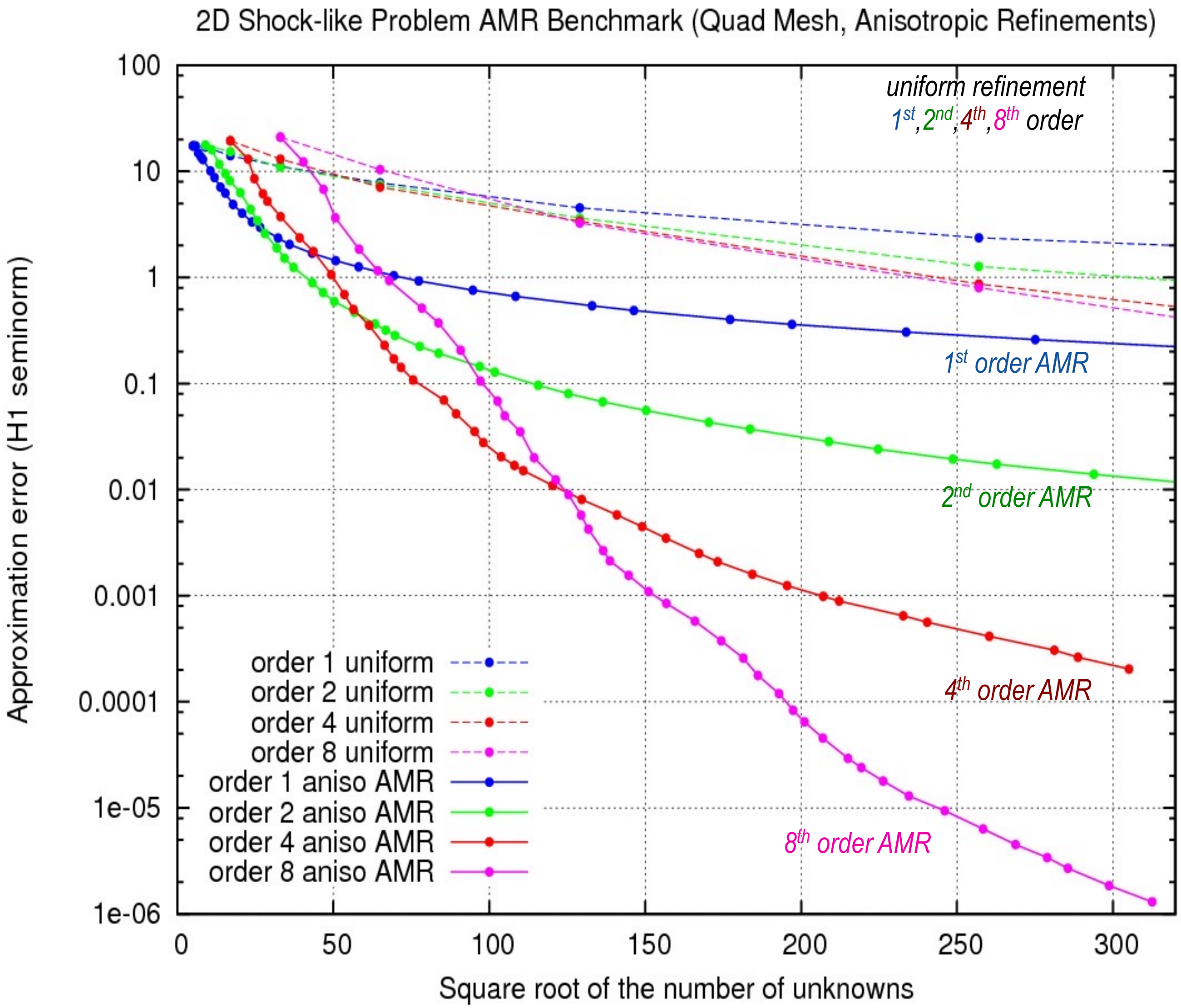
Regular assembly of A on the elements of the (cut) mesh

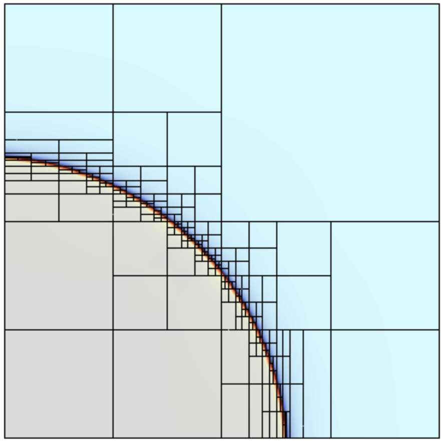
Nonconforming variational restriction

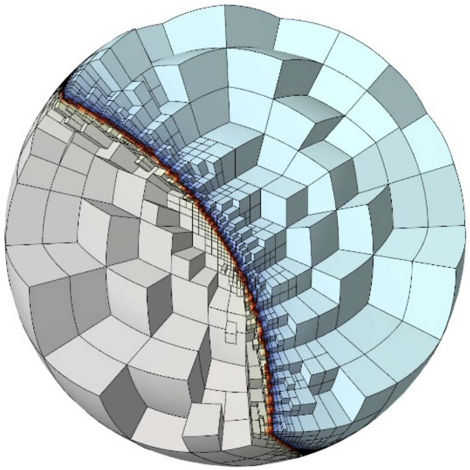


Conforming solution y = P x

AMR = smaller error for same number of unknowns

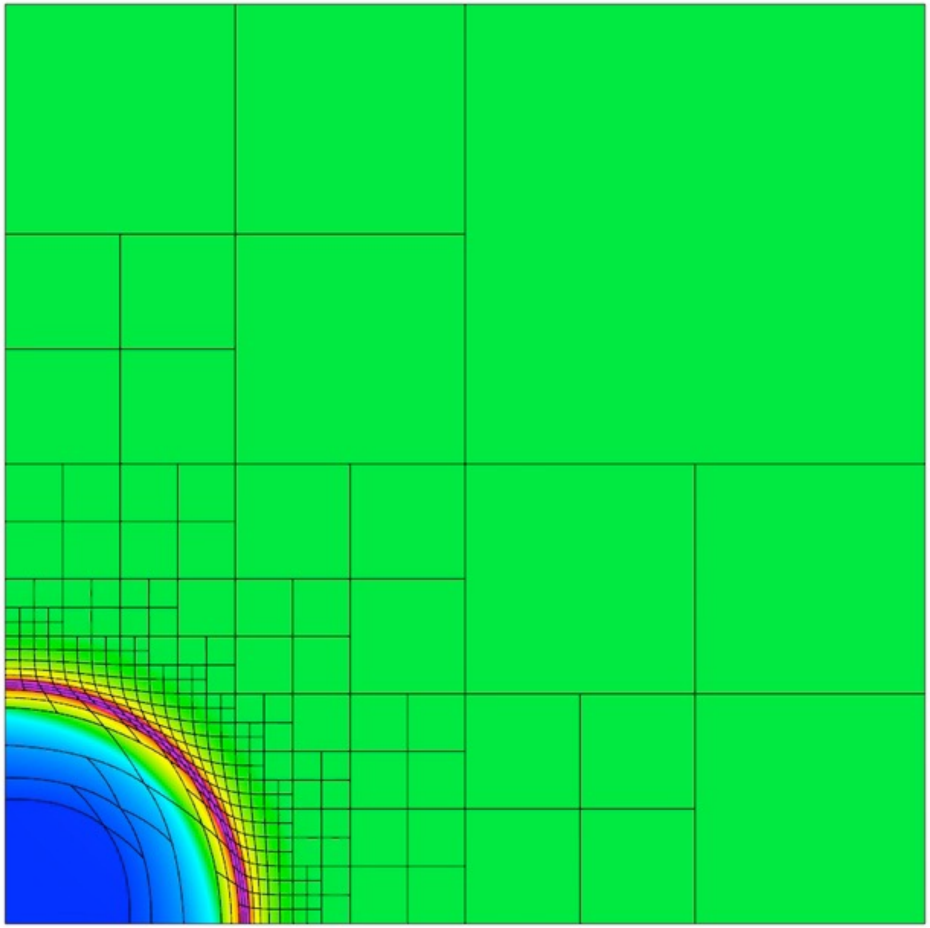




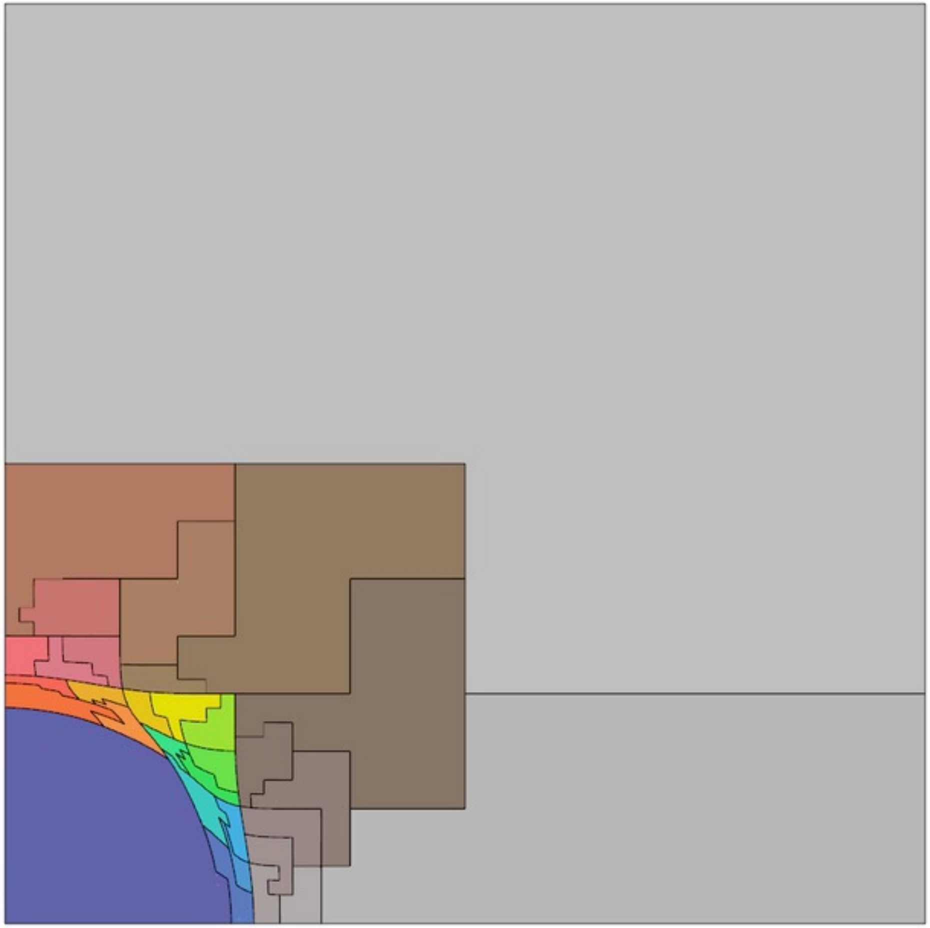


Anisotropic adaptation to shock-like fields in 2D & 3D

Parallel dynamic AMR, Lagrangian Sedov problem

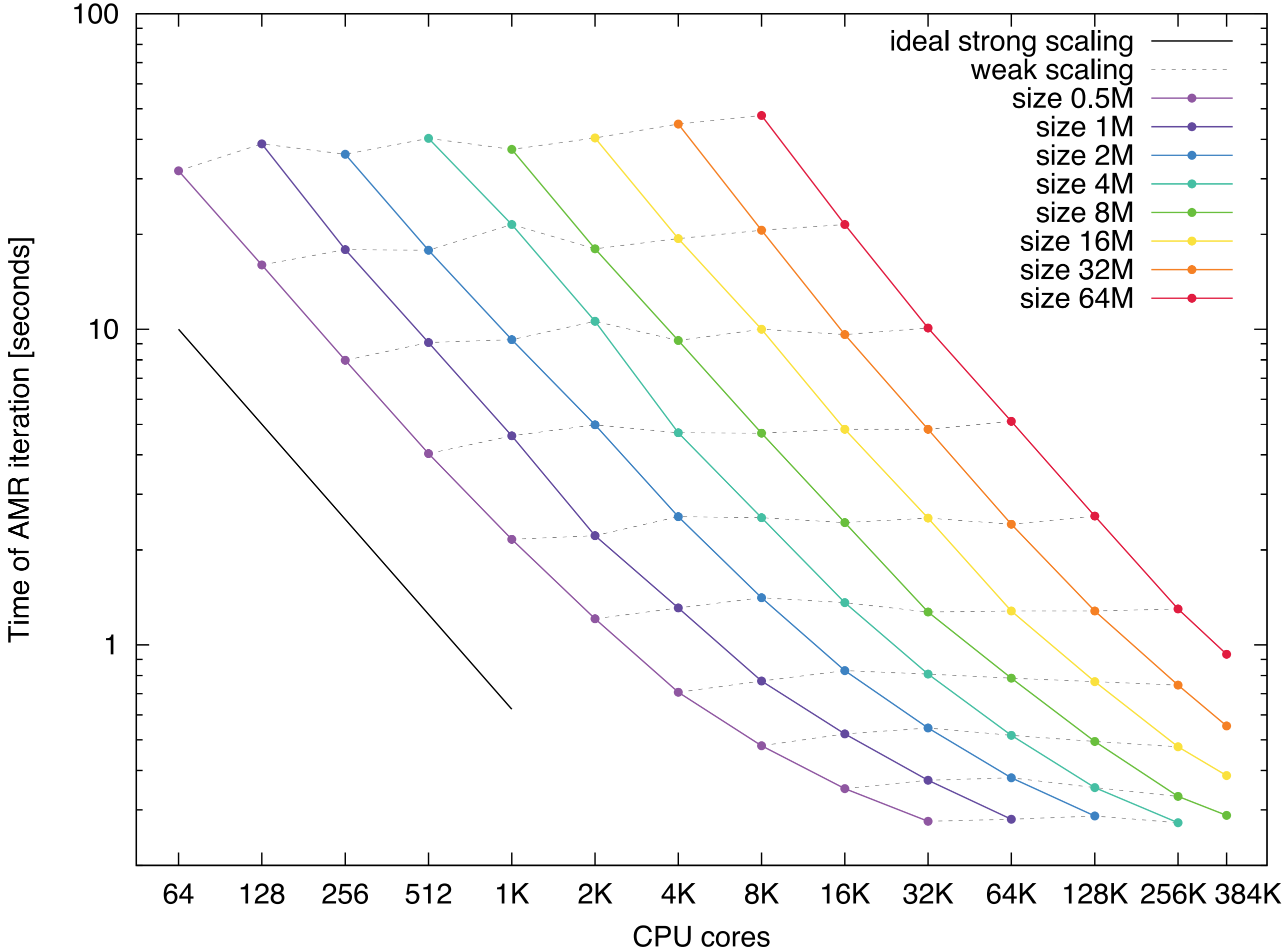


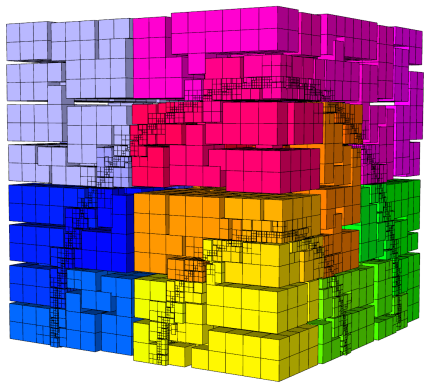
Adaptive, viscosity-based refinement and derefinement. 2nd order Lagrangian Sedov



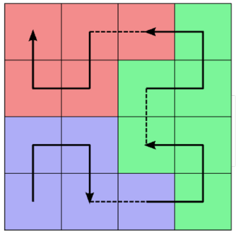
Parallel load balancing based on space-filling curve partitioning, 16 cores

Parallel AMR scaling to ~400K MPI tasks





Parallel decomposition (2048 domains shown)



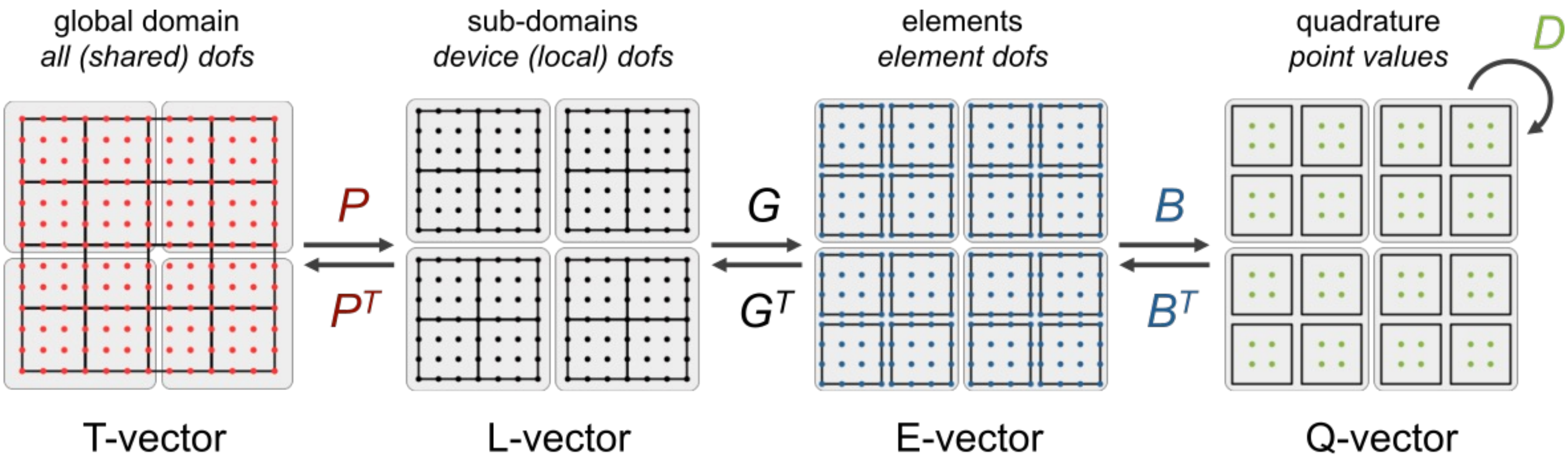
Parallel partitioning via Hilbert curve

- 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

* ***libCEED***, `github.com/ceed/libceed`

Example of a fast high-order operator

***Poisson problem** in variational form*

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

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

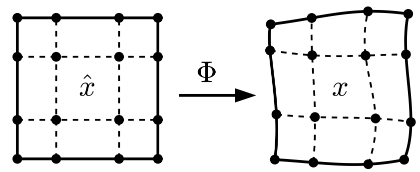
***Stiffness matrix** (unit coefficient)*

$$\underbrace{\int_\Omega \nabla\varphi_i \nabla\varphi_j}_{A_{ij}} = \sum_E \int_E \nabla\varphi_i \nabla\varphi_j$$

$$= \sum_E \sum_k \alpha_k\, J_E^{-1}(q_k)\hat{\nabla}\hat{\varphi}_i(q_k)\; J_E^{-1}(q_k)\hat{\nabla}\hat{\varphi}_j(q_k)\; |J_E(q_k)|$$

$$= \underbrace{\sum_E}_{G,G^T} \sum_k \underbrace{\hat{\nabla}\hat{\varphi}_i(q_k)}_{(B^T)_{ik}} \underbrace{\left(\alpha_k J_E^{-T}(q_k) J_E^{-1}(q_k) |J_E(q_k)|\right)}_{D_{kk}} \underbrace{\hat{\nabla}\hat{\varphi}_j(q_k)}_{B_{kj}}$$

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

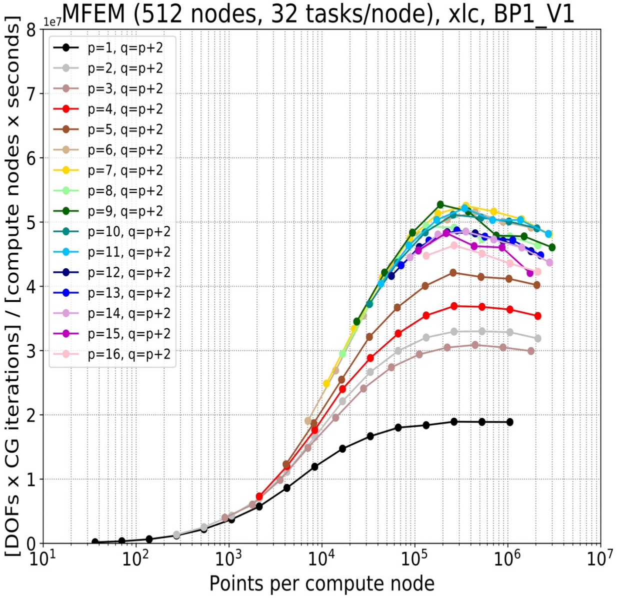


- *G* is usually Boolean (except AMR)
- Element matrices $A_E = B^T D B$, are full, account for bulk of the physics, can be applied in parallel

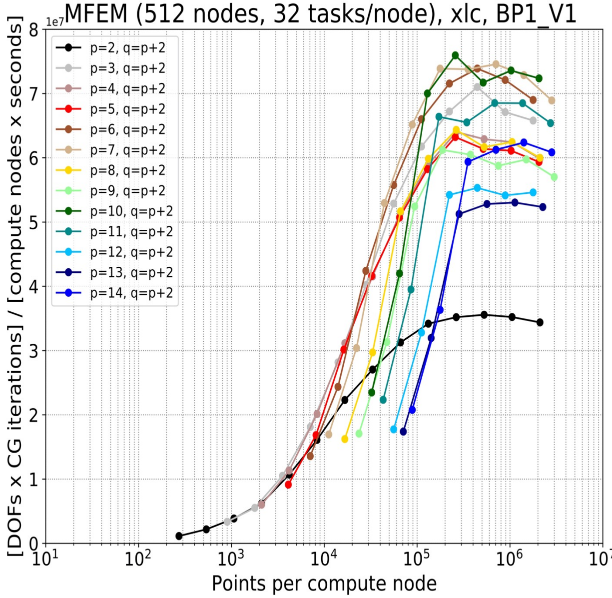
$$\begin{bmatrix} A^1 & & & \\ & A^2 & & \\ & & \ddots & \\ & & & A^4 \end{bmatrix}$$

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

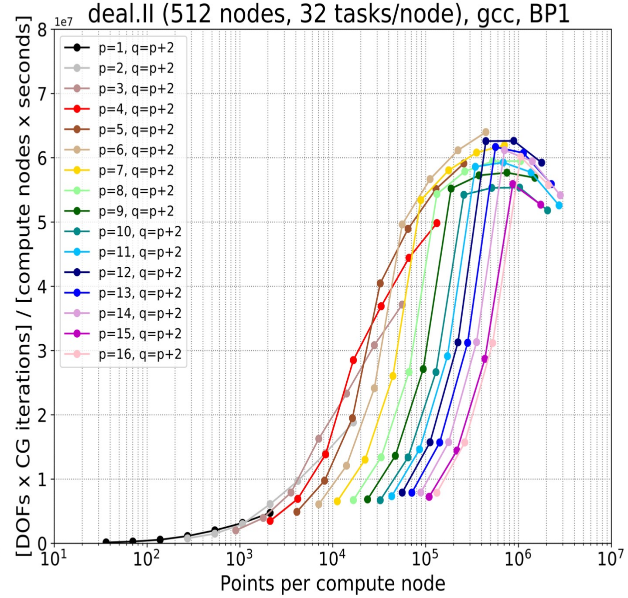
CEED BP1 bakeoff on BG/Q



Nek5000



MFEM-improved

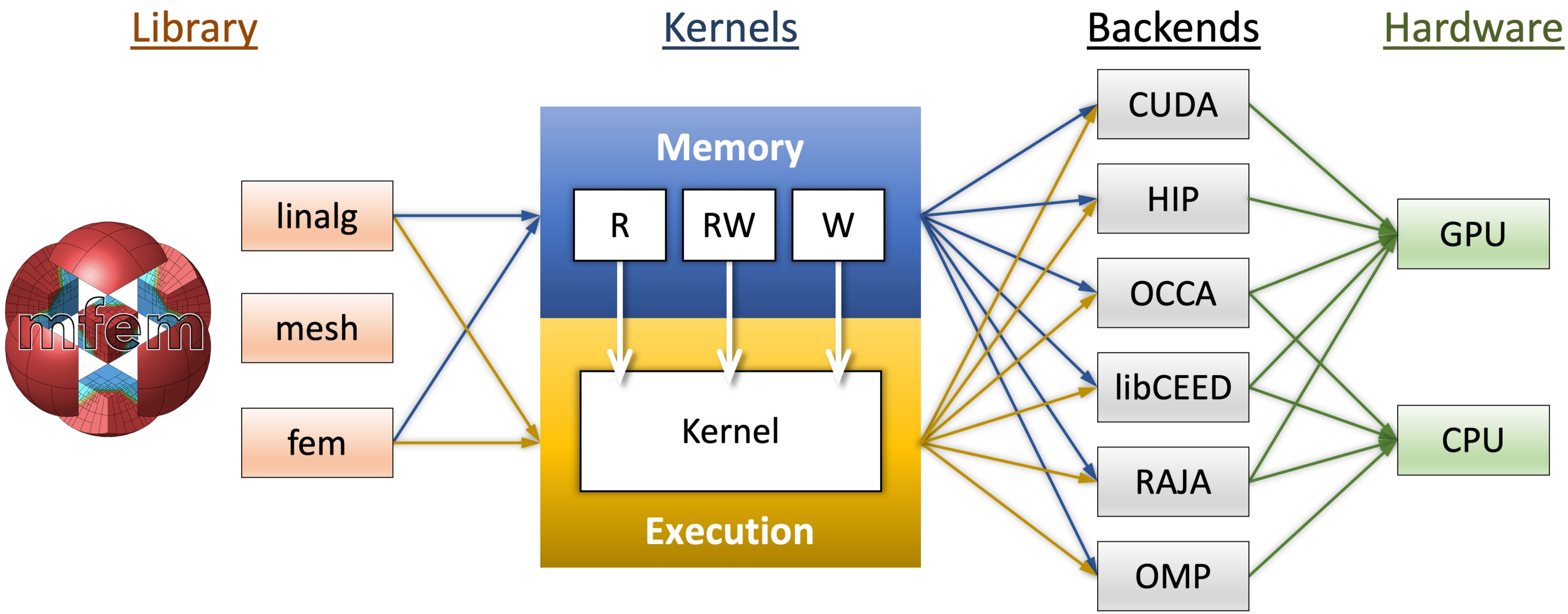


deal.ii

- ✔ All runs done on BG/Q (for repeatability), 16384 MPI ranks. Order p = 1, ...,16; quad. points q = p + 2.
- ✔ BP1 results of MFEM+xlc (left), MFEM+xlc+intrinsics (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

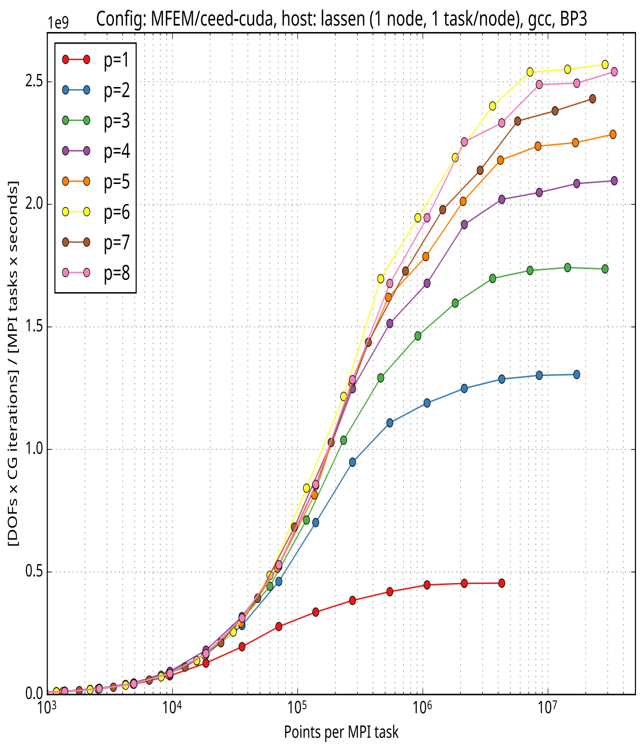
MFEM support GPU acceleration in many linear algebra and finite element operations



- 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, …

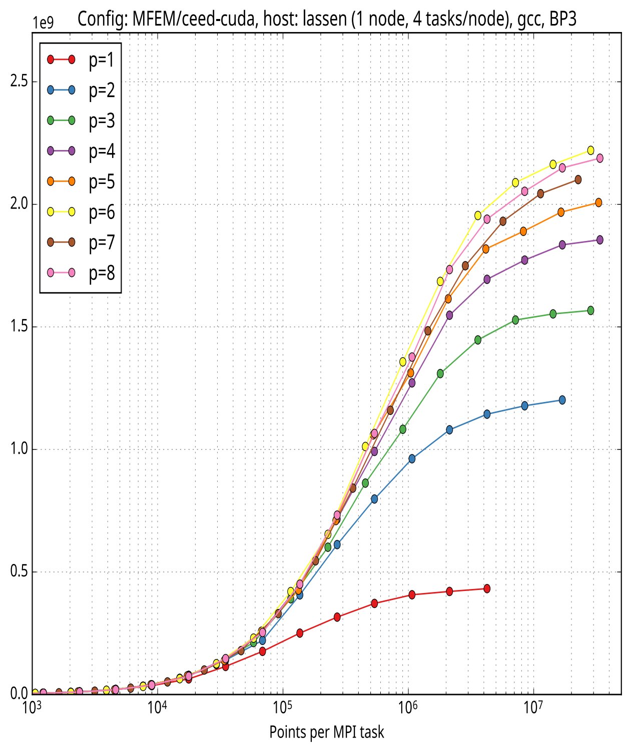
"MFEM: A modular finite element methods library", CAMWA 2020

MFEM performance on multiple GPUs

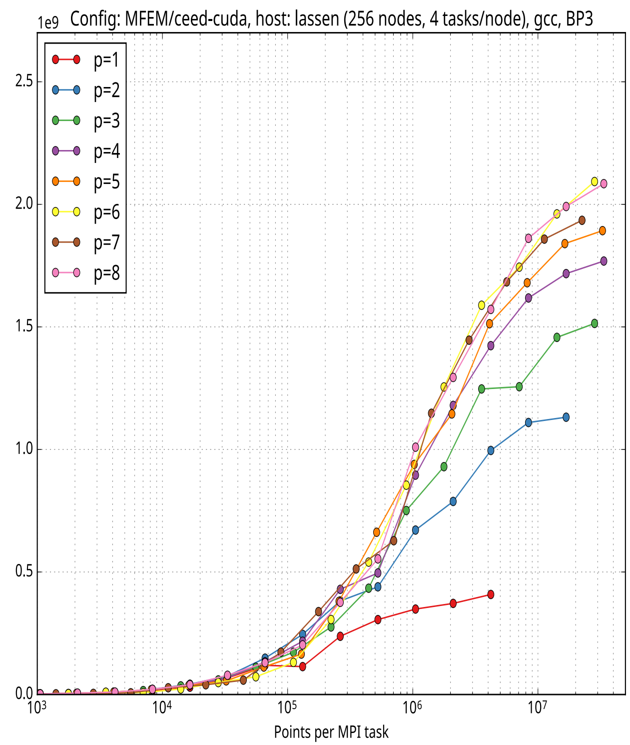


1 GPU

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



4 GPUs

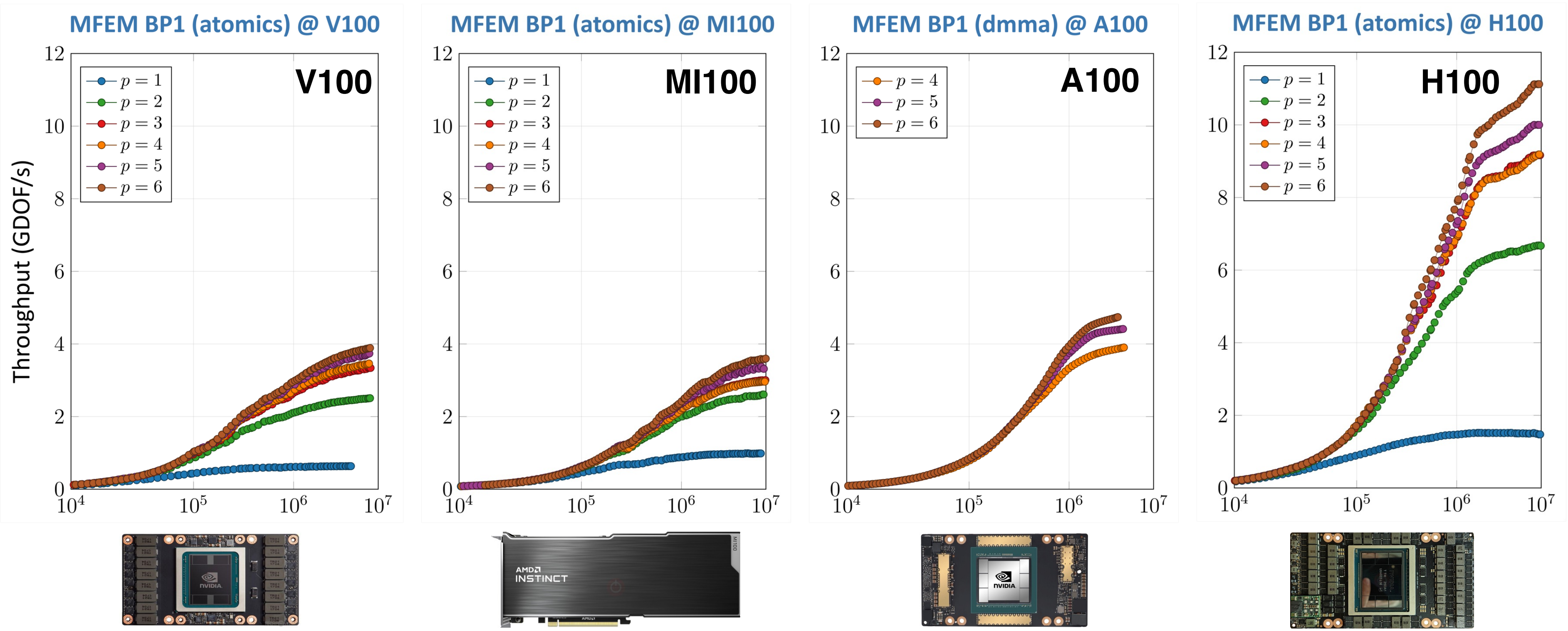


1024 GPUs

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,

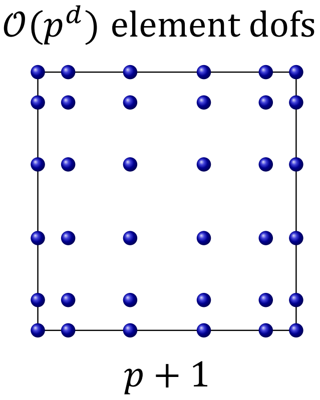
can utilize tensor cores on A100

achieve 10+ GDOFs on H100

MI250X results in the CEED-MS39 report: `ceed.exascaleproject.org/pubs`

Matrix-free preconditioning

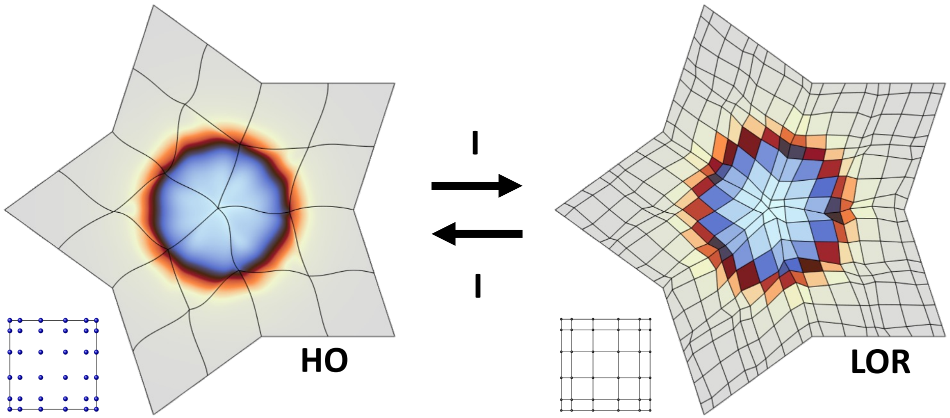
- ***Explicit matrix assembly impractical at high order:***
  - Polynomial degree p, spatial dimension d
  - Matrix assembly + sparse matvecs:
    - $\mathcal{O}(p^{2d})$ memory transfers
    - $\mathcal{O}(p^{3d})$ computations
    - can be reduced to $\mathcal{O}(p^{2d+1})$ computations by sum factorization
  - 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***

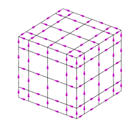


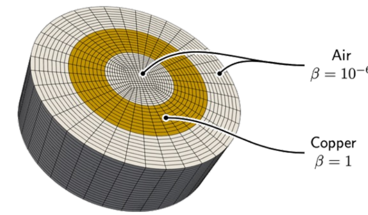
- ***Challenges:***
  - Traditional matrix-based preconditioners (e.g. AMG) not available
  - Condition number of diffusion systems grows like $\mathcal{O}(p^3/h^2)$

Low-Order-Refined (LOR) preconditioning

Efficient LOR-based preconditioning of H1, H(curl), H(div) and L2 high-order operators







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

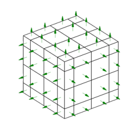
LOR–AMS						
p	Its.	Assembly (s)	AMG Setup (s)	Solve (s)	# DOFs	# NNZ
2	41	0.082	0.277	0.768	516,820	1.65×10^7
3	63	0.251	0.512	2.754	1,731,408	5.64×10^7
4	75	0.679	1.133	7.304	4,088,888	1.34×10^8
5	62	1.574	2.185	11.783	7,968,340	2.61×10^8
6	89	3.336	4.024	30.702	13,748,844	4.51×10^8
Matrix-Based AMS						
p	Its.	Assembly (s)	AMG Setup (s)	Solve (s)	# DOFs	# NNZ
2	39	0.140	0.385	1.423	516,820	5.24×10^7
3	44	1.368	1.572	9.723	1,731,408	4.01×10^8
4	49	9.668	5.824	45.277	4,088,888	1.80×10^9
5	53	61.726	15.695	148.757	7,968,340	5.92×10^9
6	56	502.607	40.128	424.100	13,748,844	1.59×10^{10}

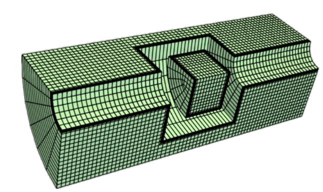
- Pick LOR space and HO basis so **P=R=I** (Gerritsma, Dohrmann)
- $\mathbf{A_{LOR}}$ is sparse and spectrally equivalent to $\mathbf{A_{HO}}$

Theorem 2. *Let $M_\star$ and $K_\star$ denote the mass and stiffness matrices, respectively, where $\star$ represents one of the above-defined finite element spaces with basis as in Section 4.3. Then we have the following spectral equivalences, independent of mesh size h and polynomial degree p.*

$$\begin{aligned} M_{V_h} &\sim M_{V_p}, & K_{V_h} &\sim K_{V_p}, \\ M_{\boldsymbol{W}_h} &\sim M_{\boldsymbol{W}_p}, & K_{\boldsymbol{W}_h} &\sim K_{\boldsymbol{W}_p}, \\ M_{\boldsymbol{X}_h} &\sim M_{\boldsymbol{X}_p}, & K_{\boldsymbol{X}_h} &\sim K_{\boldsymbol{X}_p}, \\ M_{Y_h} &\sim M_{Y_{p-1}}, & & \\ M_{Z_h} &\sim M_{Z_p}, & K_{Z_h} &\sim K_{Z_p}. \end{aligned}$$

- $\mathbf{(A_{HO})^{-1}} \approx \mathbf{(A_{LOR})^{-1}} \approx \mathbf{B_{LOR}}$ - can use BoomerAMG, AMS, ADS





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

p	LOR–ADS Runtime (s)	LOR–ADS Memory (GB)	Matrix-Based ADS Runtime (s)	Matrix-Based ADS Memory (GB)	Speedup
2	2.11	0.04	2.98	0.20	1.41×
3	6.64	0.15	22.58	1.84	3.40×
4	17.40	0.35	114.35	9.13	6.57×
5	43.70	0.68	422.74	32.21	9.67×
6	92.76	1.18	1324.94	91.09	14.28×

"Low-order preconditioning for the high-order de Rham complex", Pazner, Kolev, Dohrmann, 2022

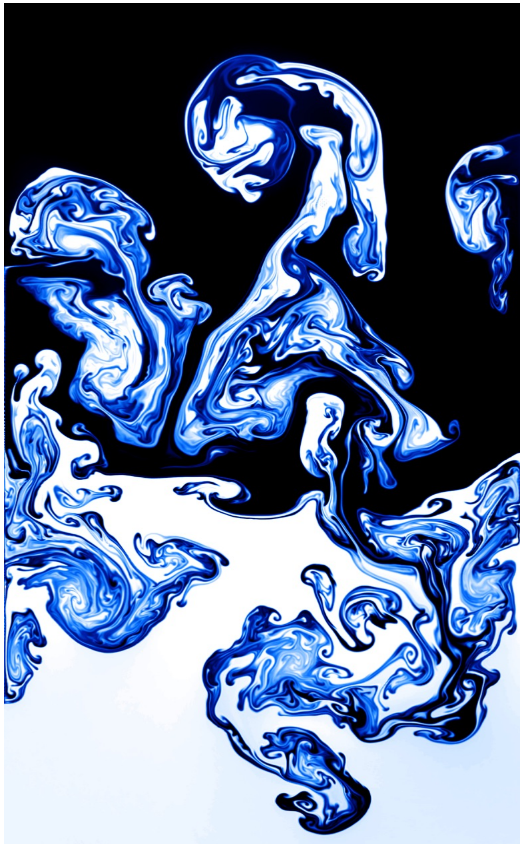
High-order methods show promise for high-quality & performant simulations on exascale platforms

- **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

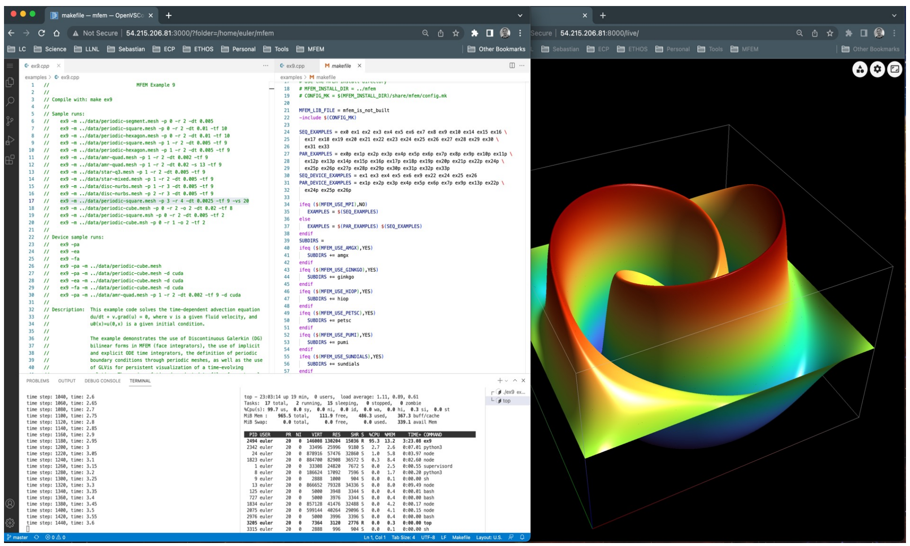


Q4 Rayleigh-Taylor single-material ALE on 256 processors

Upcoming MFEM Events

MFEM in the Cloud Tutorial

August 22, 2024



https://mfem.org/tutorial

MFEM Community Workshop

October 22-24, 2024

https://mfem.org/workshop

FEM@LLNL **Seminar series:** https://mfem.org/seminar

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
LLNL-PRES-755924

Lawrence Livermore National Laboratory

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 – makes matrix solves harder
- Non-tensor product elements not as computationally efficient

FASTMATH

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

FASTMath Unstructured Mesh Development Areas

- Unstructured Mesh Analysis Codes – Support application’s PDE solution needs – MFEM library is a 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 In Cell (PIC) Codes – Tools to support PIC on unstructured meshes
- Unstructured Mesh ML and UQ – ML for data reduction, adaptive mesh UQ
- Code Coupling Tools – Parallel geo./mesh/field coupling
- In Situ Vis and Data Analytics – Tools to gain insight as simulations execute

FASTMath Unstructured Mesh Tools and Components

- FE Analysis codes
  - MFEM (https://mfem.org/)
  - LGR (https://github.com/SNLComputation/lgrtk)
  - PHASTA (https://github.com/phasta/phasta)
- Unstructured Mesh Infrastructure
  - Omega_h (https://github.com/SNLComputation/omega_h)
  - PUMI/MeshAdapt (https://github.com/SCOREC/core)
  - PUMIpic (https://github.com/SCOREC/pumi-pic)
- Load balancing, task placement
  - Jet (https://github.com/sandialabs/Jet-Partitioner/)
  - Zoltan (https://github.com/sandialabs/Zoltan)
  - Zoltan2 (https://github.com/trilinos/Trilinos/tree/master/packages/zoltan2)
  - Xtra-PULP (https://github.com/HPCGraphAnalysis/PuLP)
  - EnGPar (http://scorec.github.io/EnGPar/)
- General code coupling
  - PCMS (https://github.com/SCOREC/pcms)

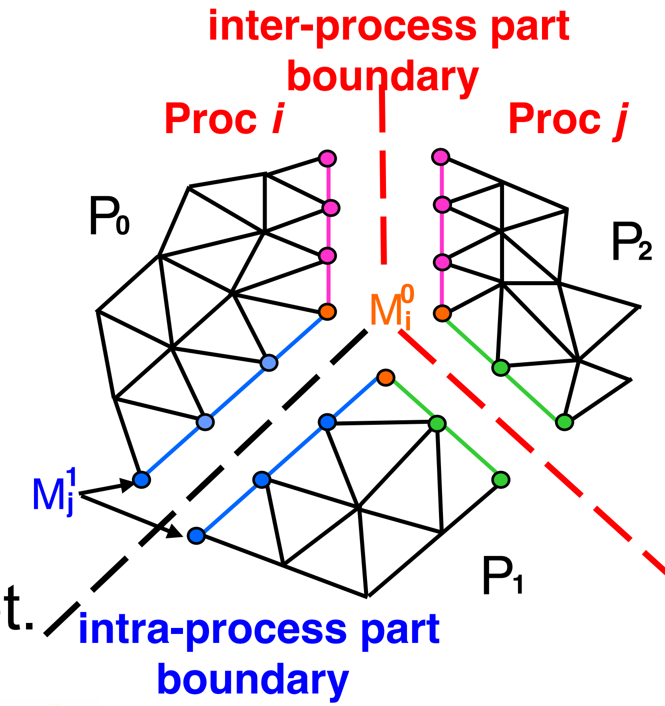
Parallel Unstructured Mesh Infrastructure

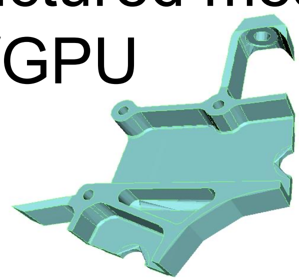
Support unstructured mesh interactions on exascale systems

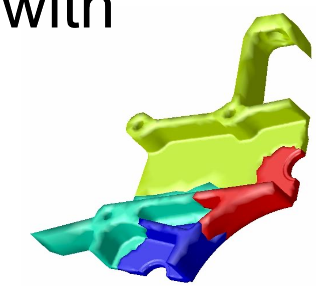
- Mesh hierarchy to support interrogation and modification
- Maintains linkage to original geometry
- Conforming mesh adaptation
- Inter-process communication
- Supports field operations

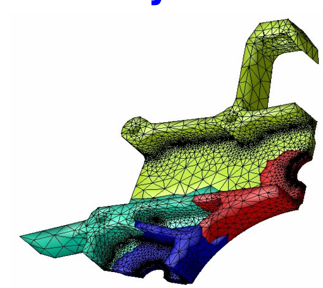
Tools

- Omega_h – CPU/GPU support
- PUMI – CPU based curved mesh adapt.
- PUMIPic – Unstructured mesh with particles for CPU/GPU




Geometric model

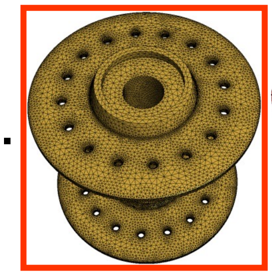

Partition model

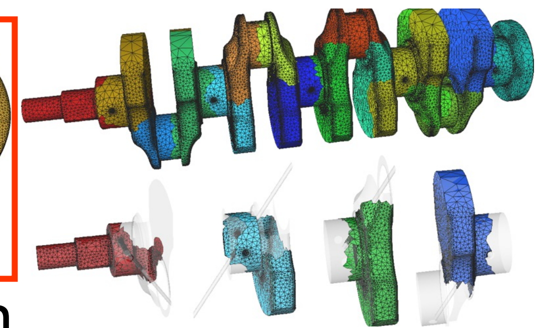

Distributed mesh

Mesh Generation and Control

Mesh Generation:

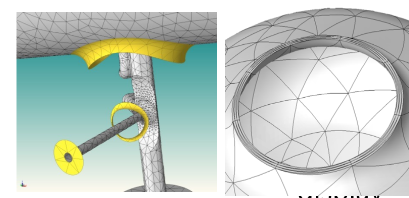
- Automatically mesh complex domains – should work directly from CAD, image data, etc.
- Use tools like Gmsh, Simmetrix, etc.

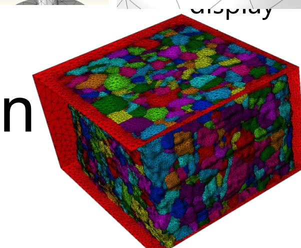




Mesh control:

- Use *a posteriori* information to improve mesh
- Curved geometry and curved mesh entities
- Support full range of mesh modifications – vertex motion, mesh entity curving, cavity based refinement and coarsening, etc. anisotropic adaptation
- Control element shapes as needed by the various discretization methods for maintaining accuracy and efficiency



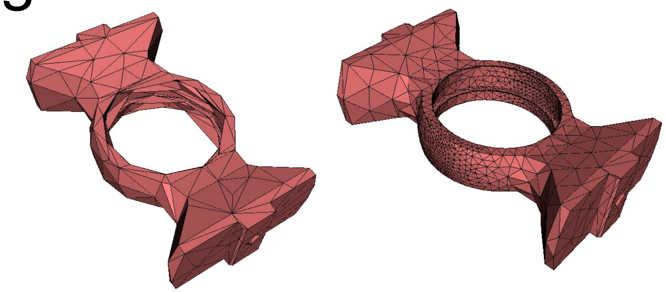


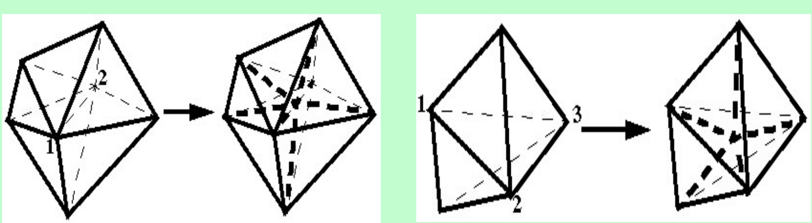
Parallel execution of all functions is critical on large meshes

FASTMATH

General Mesh Modification for Mesh Adaptation

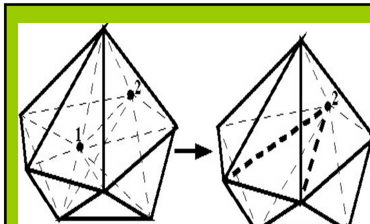
- Driven by an anisotropic mesh size field that can be set by any combination of criteria
- Employ a “complete set” of mesh modification operations to alter the mesh into one that matches the given mesh size field
- Advantages
  - Supports general anisotropic meshes
  - Can obtain level of accuracy desired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



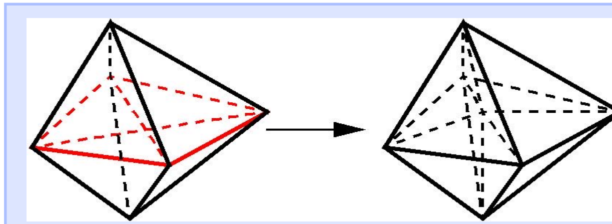


Edge split

face split



Edge collapse

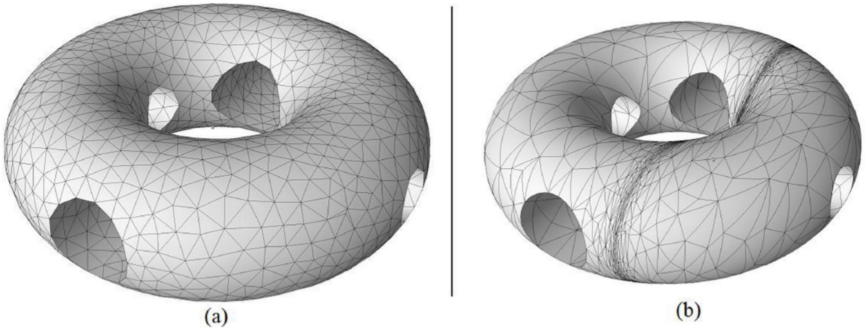


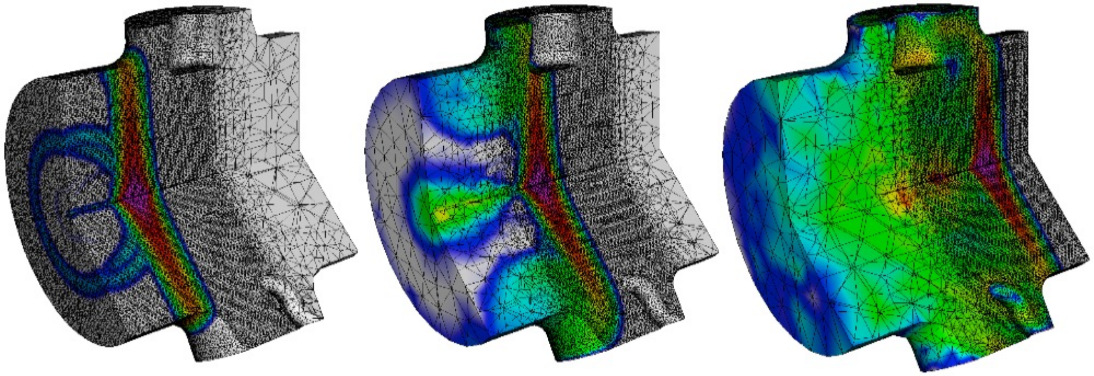
Double split collapse to remove sliver

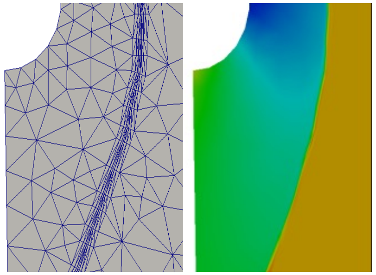
FASTMATH

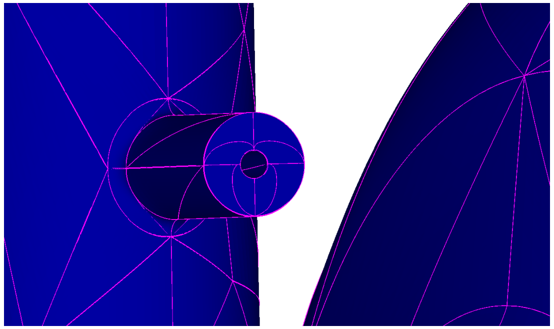
Mesh Adaptation

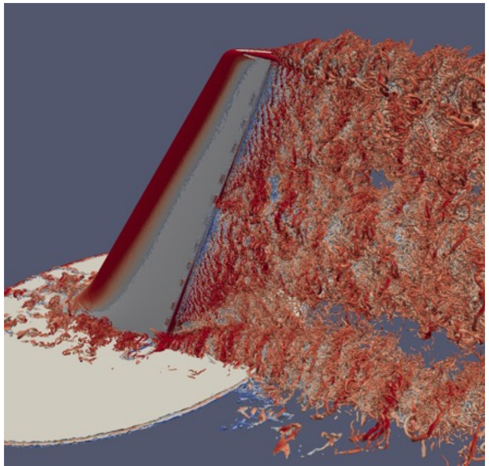
- Supports adaptation of curved elements
- Adaptation based on multiple criteria, examples
  - Level sets at interfaces
  - Tracking particles
  - Discretization errors
  - Controlling element shape in evolving geometry





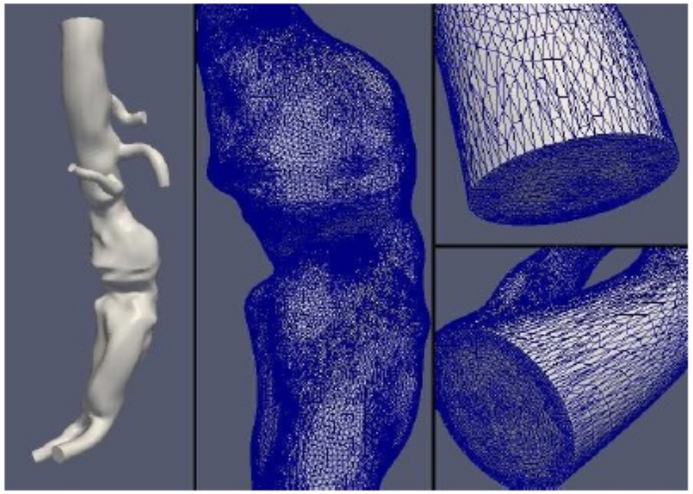


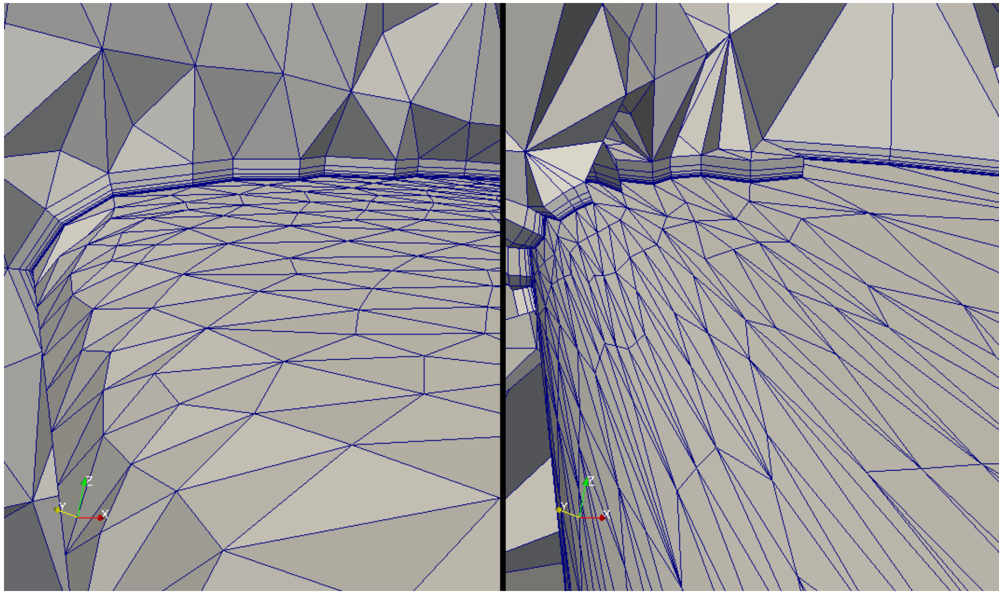


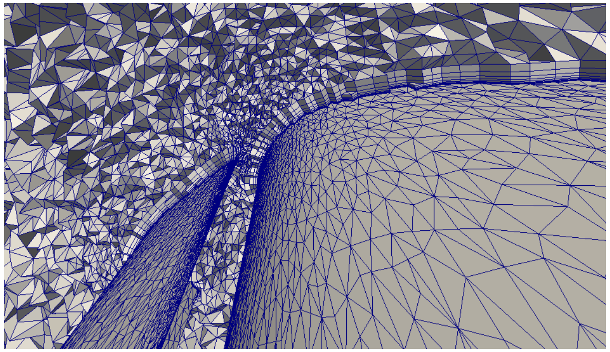


Mesh Adaptation

- Applied to very large scale models – 92B elements on 3.1M processes on ¾ million cores
- Local solution transfer supported through callback
- Effective storage of solution fields on meshes
- Supports adaptation with boundary layer meshes





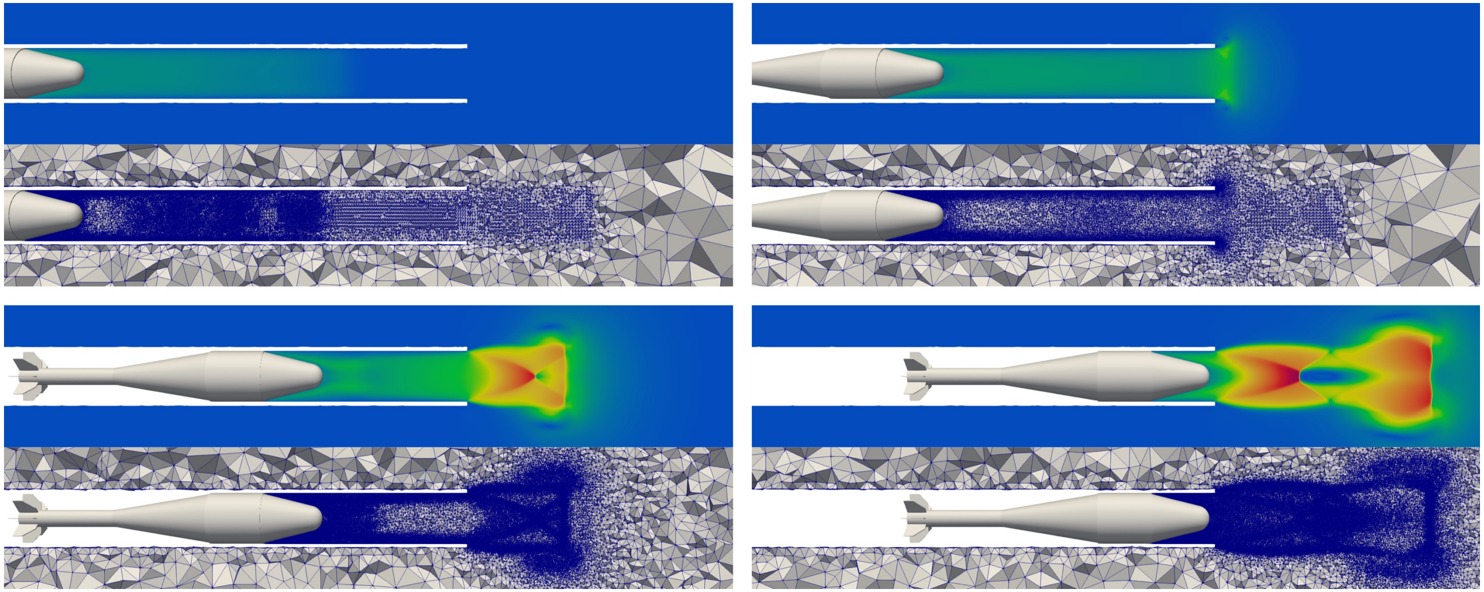


Mesh Adaptation of Evolving Geometry Problems

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

Adaptive loop:

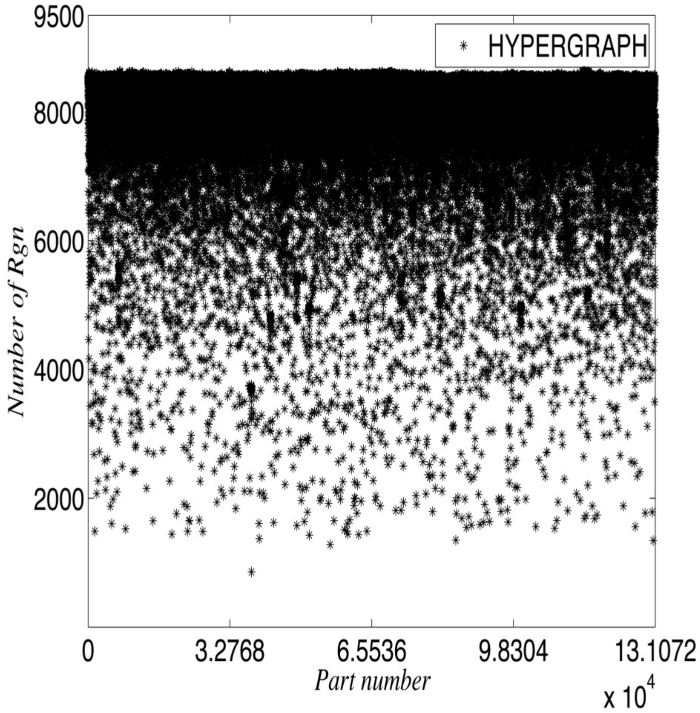
1. Initialize analysis case, generate initial mesh, start time stepping loop
2. Perform time steps employing mesh motion - monitor element quality and discretization errors
3. When element quality is not satisfactory or discretization errors too large – set mesh size field and perform mesh modification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



FASTMATH

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
  - EnGPar diffusive multi-criteria partition improvement
  - XtraPuLP multi-constraint multi-objective label propagation-based graph partitioner

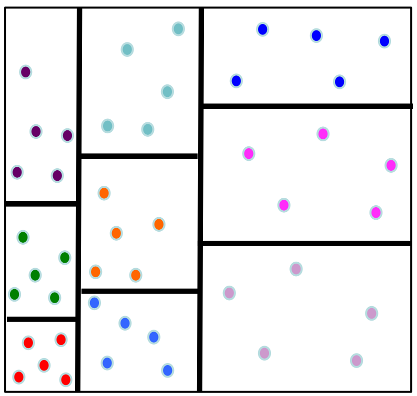


FASTMATH

Zoltan/Zoltan2 Toolkits: Partitioners

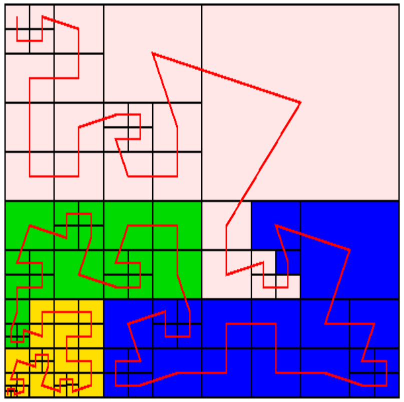
Suite of partitioners supports a wide range of applications; no single partitioner is best for all applications.

Geometric

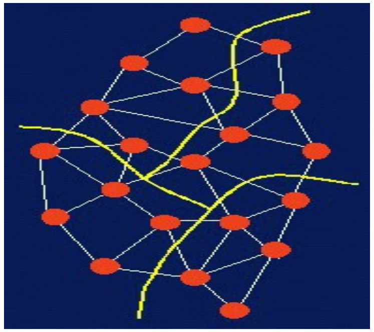


Recursive Coordinate Bisection
Recursive Inertial Bisection
Multi-Jagged Multi-section

Space Filling Curves

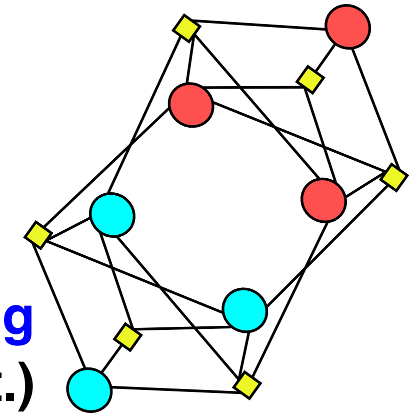


Topology-based



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

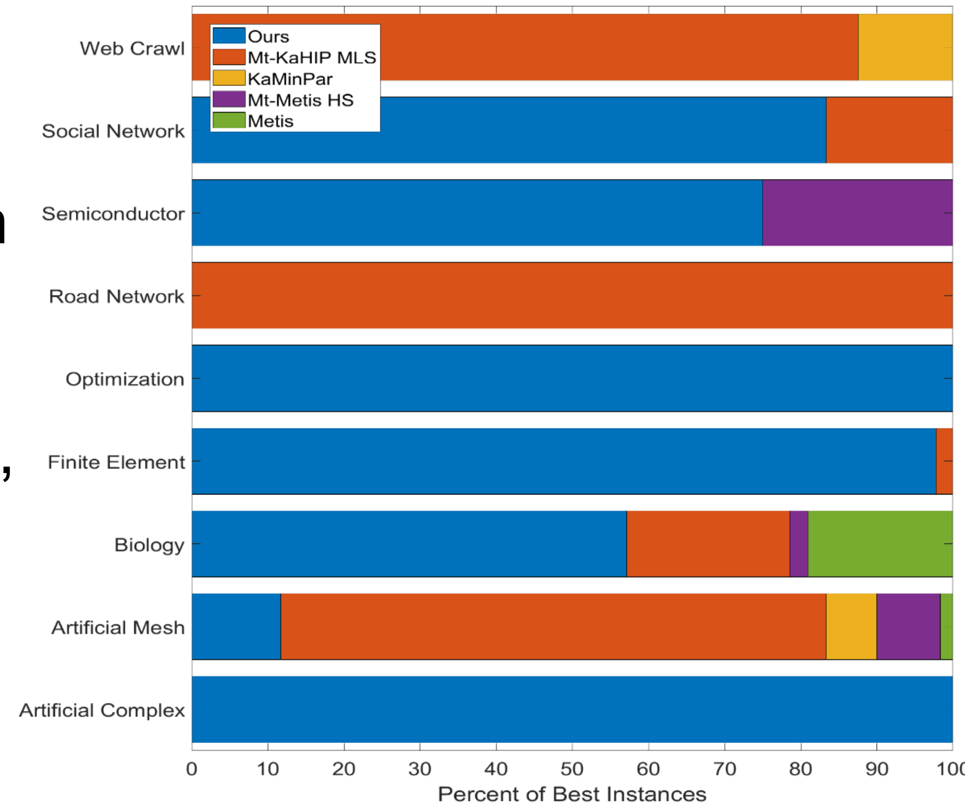
PHG Hypergraph Partitioning
Interface to PaToH (Ohio St.)



FASTMATH

A New Graph Partitioner for GPU: Jet

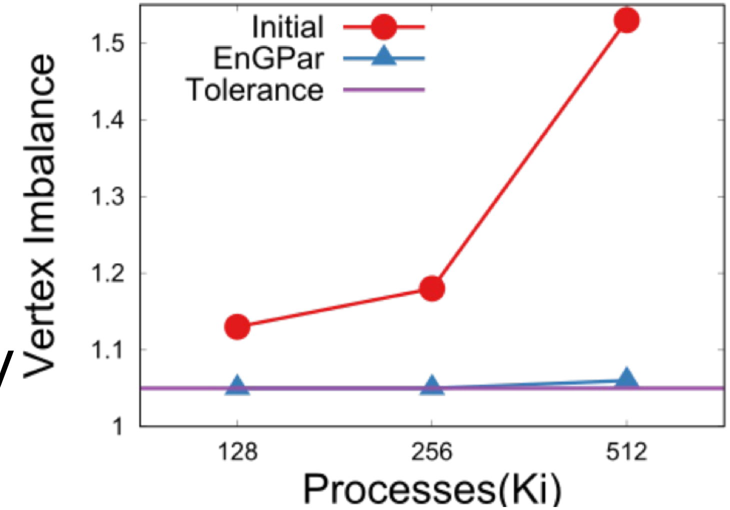
- Multilevel graph partitioner on GPU
- Uses new label propagation refinement algorithm
- Results (blue bars) slightly better than Metis/Parametis, but significant speedup due to GPU execution
- Best partitions for 98% of the test graphs from finite element meshes
- Currently single GPU (up to ~1B edges)
- Multi GPU - distributed memory version is under development

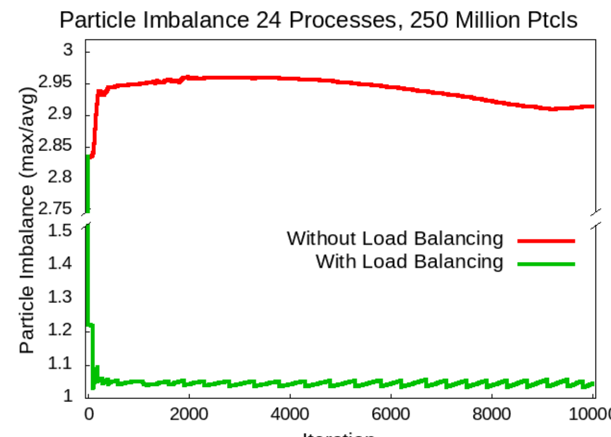


FASTMATH

EnGPar quickly reduces large imbalances on (hyper)graphs with billions of edges on up to 512K processes

- Multi-(hyper)graph supports multiple dependencies (edges) between application work/data items (vertices)
- Application defined vertex and edges
- Diffusion sending of work from heavily loaded parts to lightly loaded parts
- In 8 seconds, EnGPar 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 to support particle balance – 20% reduction in total run time





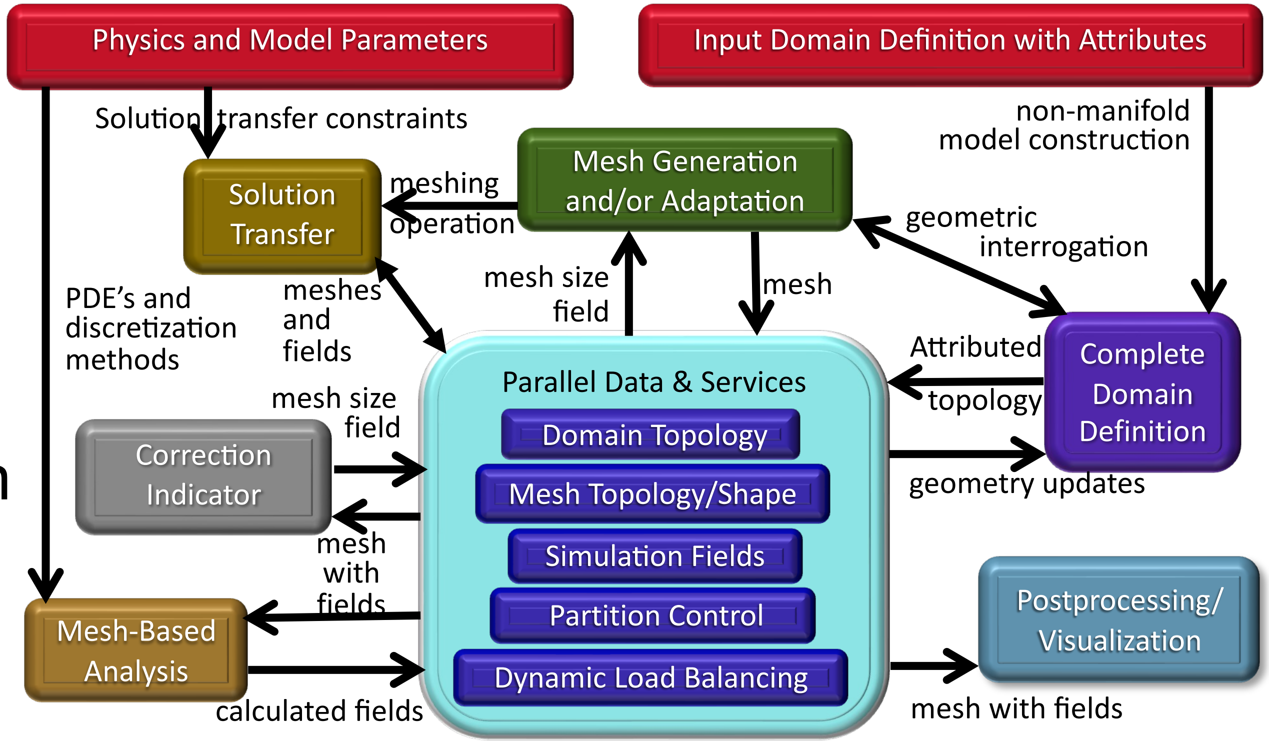
Application of EnGPar particle dynamic load balancing in a GITRm impurity transport simulation

FASTMATH

Creation of Parallel Adaptive Loops

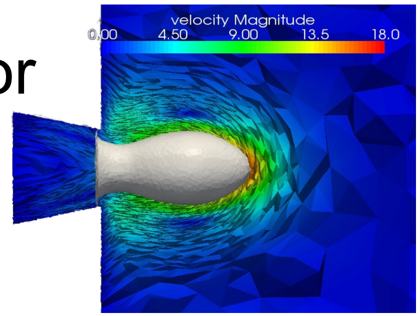
Parallel data and services used to develop adaptive loop

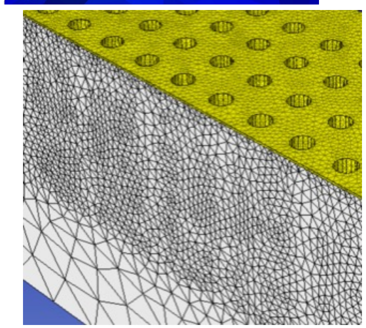
- Geometric model topology for domain linkage
- Mesh topology – it must be distributed
- Simulation fields distributed over geometric model and mesh
- Partition control
- Dynamic load balancing required at multiple steps
- API's to link to
  - CAD
  - Mesh generation and adaptation
  - Error estimation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

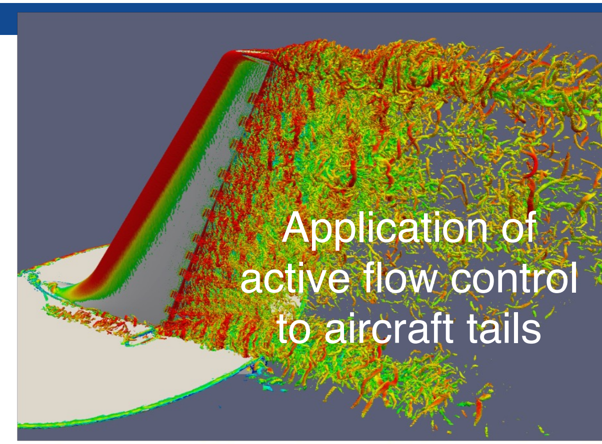


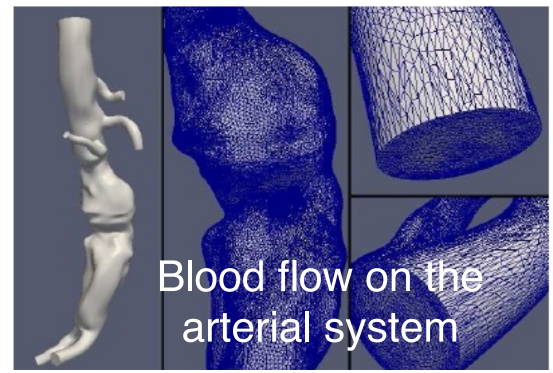
Parallel Adaptive Simulation Workflows

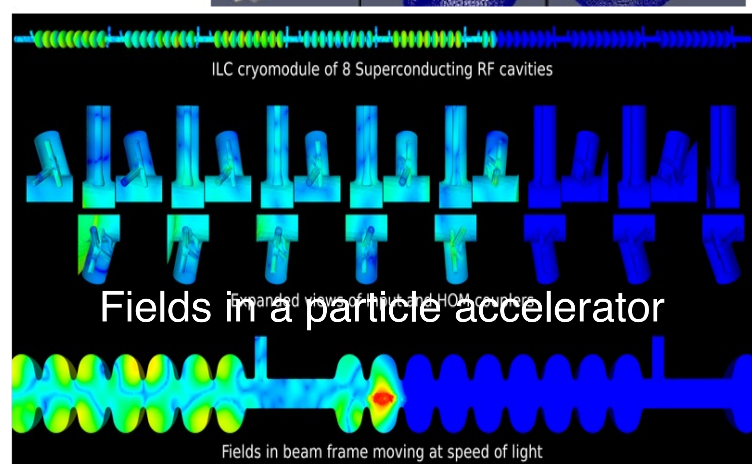
- In memory adaptive loops support effective data movement
- In-memory adaptive loops for
  - MFEM – High order FE framework
  - PHASTA – FE for NS
  - FUN3D – FV CFD
  - Proteus – multiphase FE
  - Albany – FE framework
  - ACE3P – High order FE electromagnetics
  - M3D-C1 – FE based MHD
  - Nektar++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





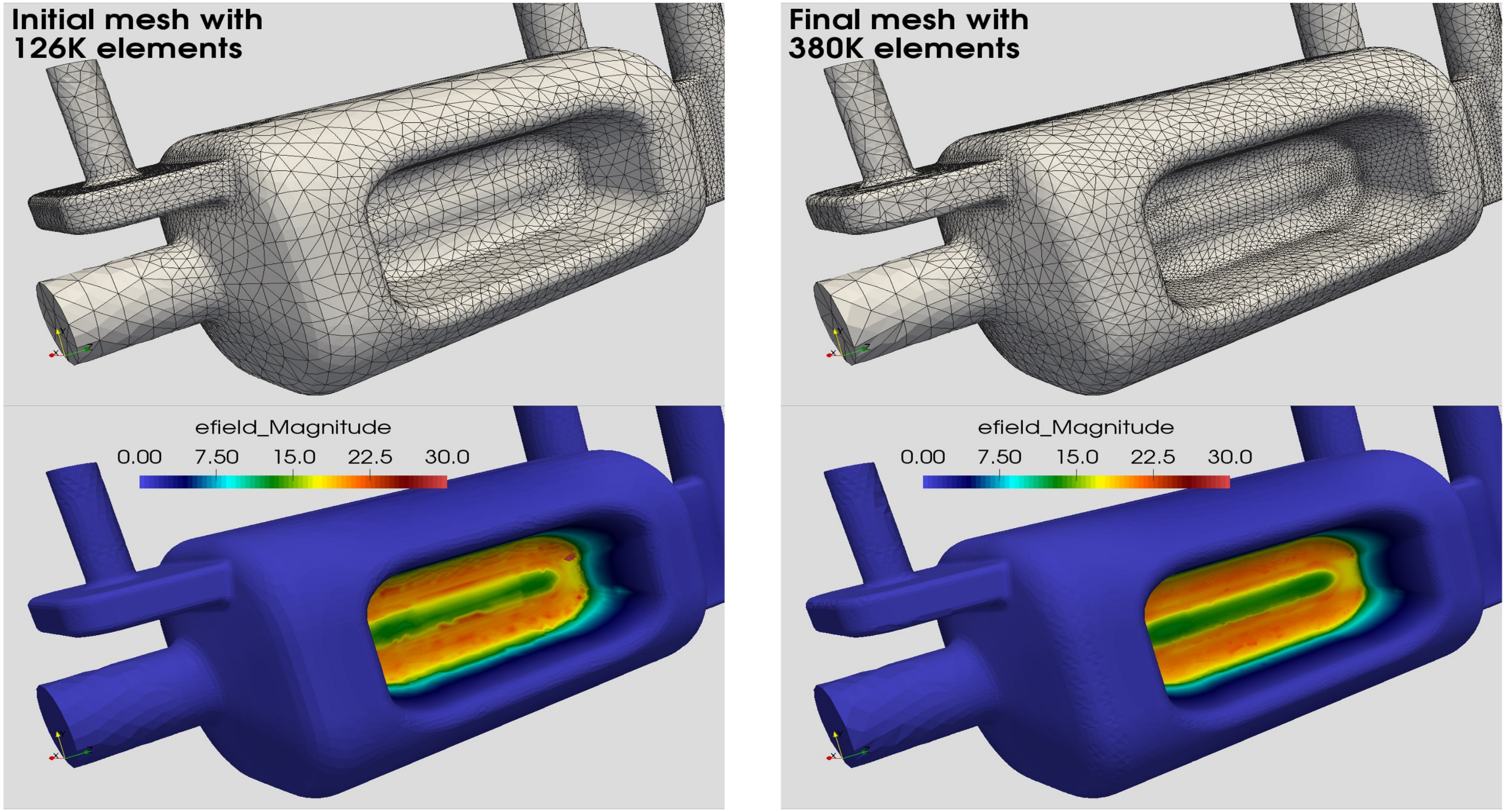






Application interactions – Accelerator EM

Omega3P Electro Magnetic Solver (second-order curved meshes)

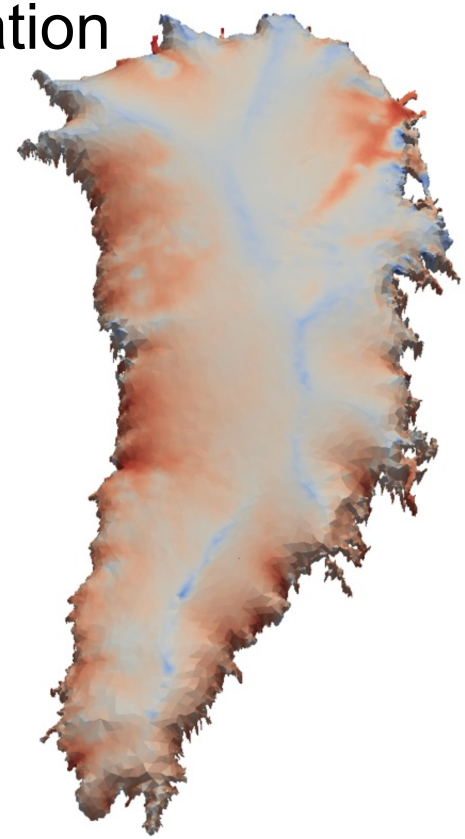


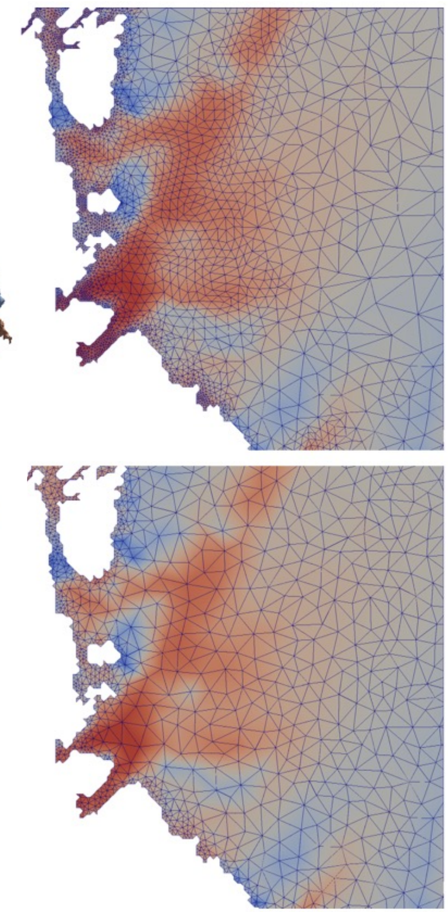
This figure shows the adaptation results for the CAV17 model. (top left) shows the initial mesh with ~126K elements, (top right) shows the final (after 3 adaptation levels) mesh with ~380K elements, (bottom left) shows the first eigenmode for the electric field on the initial mesh, and (bottom right)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

FASTMATH

Application interactions – Land Ice

- FELIX, a component of the Albany framework is the analysis code
- Omega_h 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

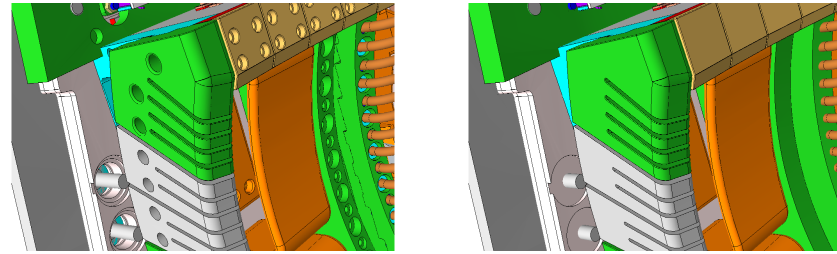




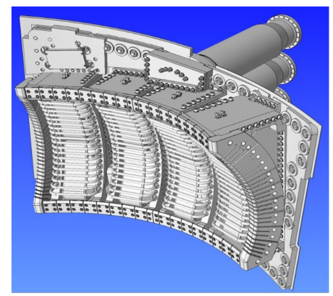
FASTMATH

Application interactions – RF Fusion

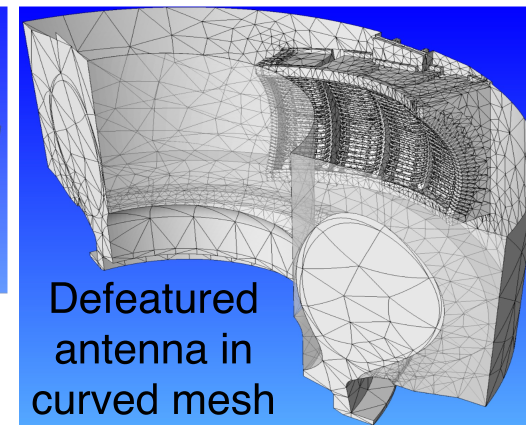
- Accurate RF simulations require
  - Detailed antenna CAD geometry
  - CAD geometry defeaturing
  - Extracted physics curves from GEQDSK equilibrium file
  - Analysis geometry combines CAD, and physics geometry
  - 3D meshes for accurate FE calculations in MFEM
  - Projection based error estimator
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



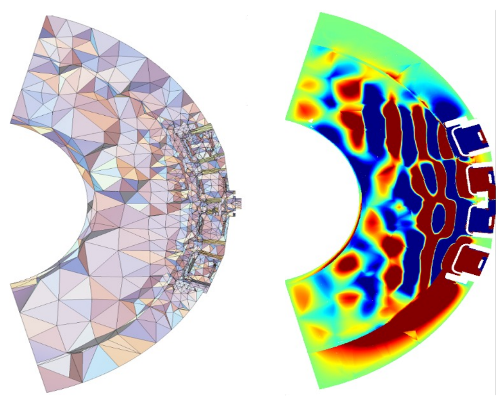
Fast elimination of unwanted features



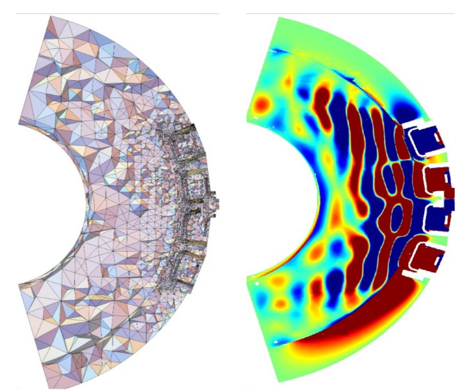
CAD of antenna array



Defeatured antenna in curved mesh



Initial Mesh

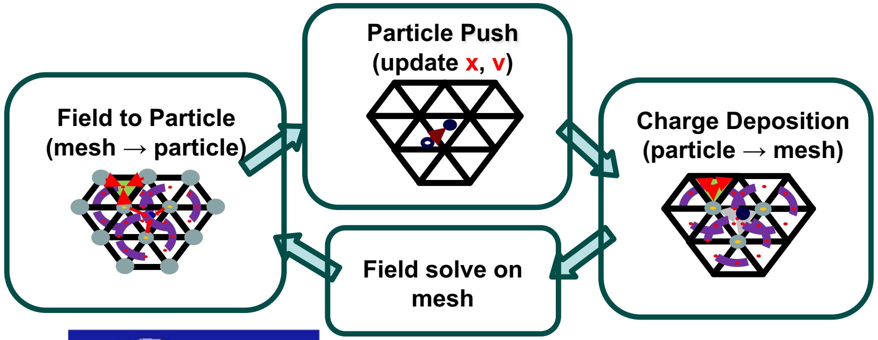


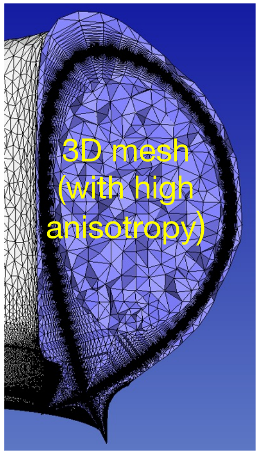
Final Adapted Mesh

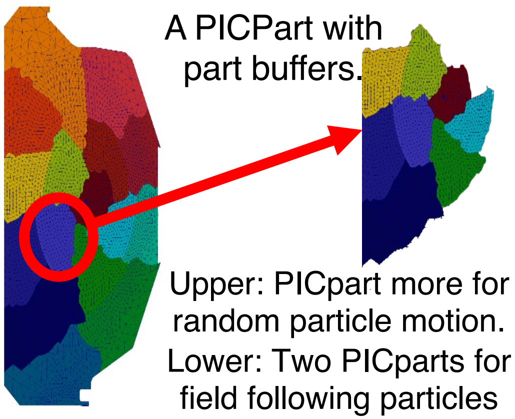
Supporting Unstructured Mesh for Particle-in-Cell Calculations

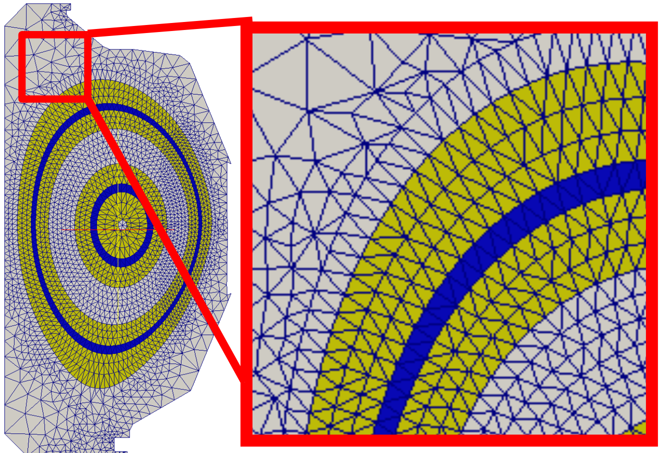
PUMIPic data structures are mesh centric

- 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



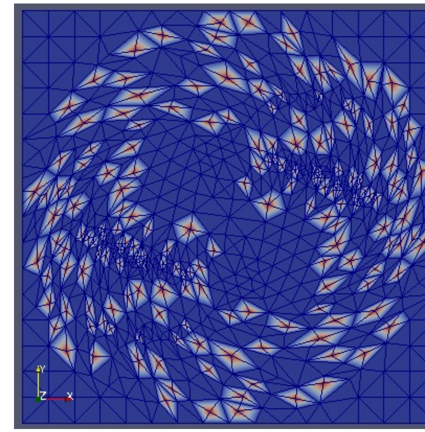




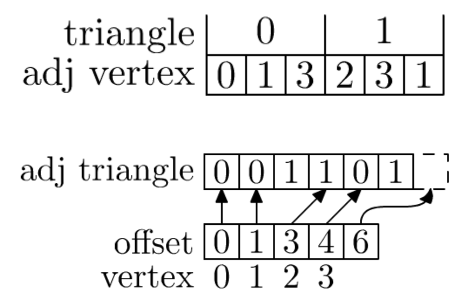


Mesh Data Structure for Heterogeneous Systems

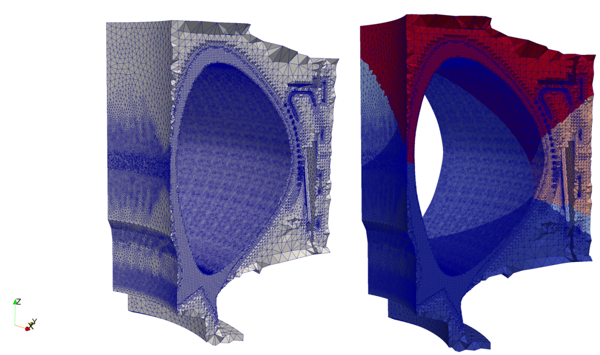
- Mesh topology/adaptation tool - Omega
  - Conforming mesh adaptation (coarsening past initial mesh, refinement, swap)
  - Manycore and GPU parallelism using Kokkos
  - Distributed mesh via mesh partitions with MPI communications
  - Support for mesh-based fields
- Recent developments:
  - Curved mesh adaptation
  - More efficient field storage
  - Kokkos implementation on latest NVIDIA, AMD and Intel GPUs



Adaptation following rotating flow field.



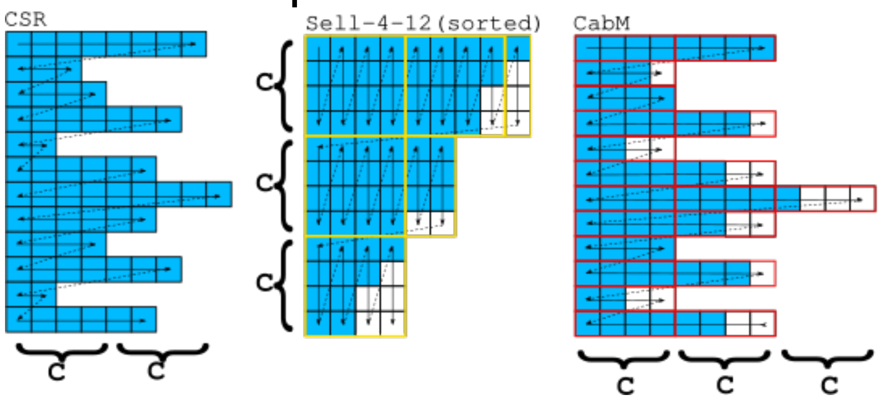
Mesh entity adjacency arrays.



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

PUMIPic Particle Data Structures

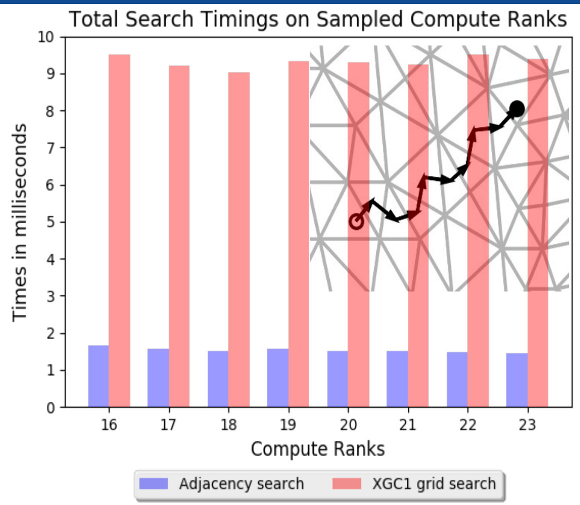
- Layout of particles in memory is critical to performance
  - Optimizes push (sort/rebuild), scatter, and gather operations
  - Associate particles with elements for large per element particle cases
  - Support changes in the number of particles per element
  - Evenly distributes work under a range 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
  - 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.

FASTMATH

PIC Operations Supported by PUMIPic



- 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

2020 PUMIPic Paper: https://www.scorec.rpi.edu/REPORTS/2020-2.pdf

FASTMATH

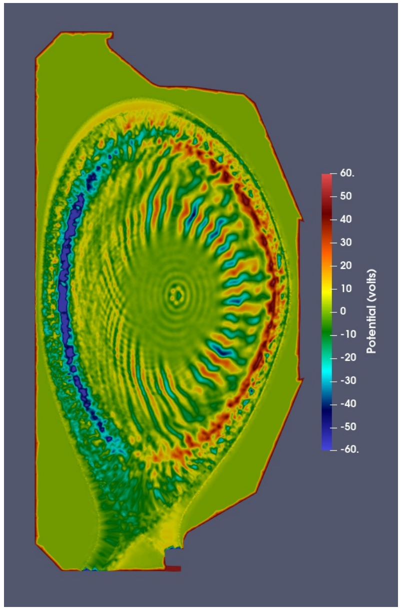
PUMIPic based XGCm Edge Plasma Code

XGCm is a version of XGC built on PUMIPic

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

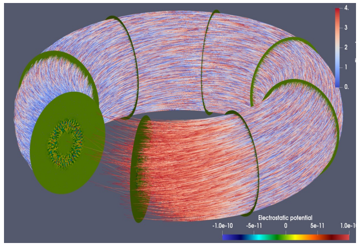
Testing of PUMIPic for use in XGC like push

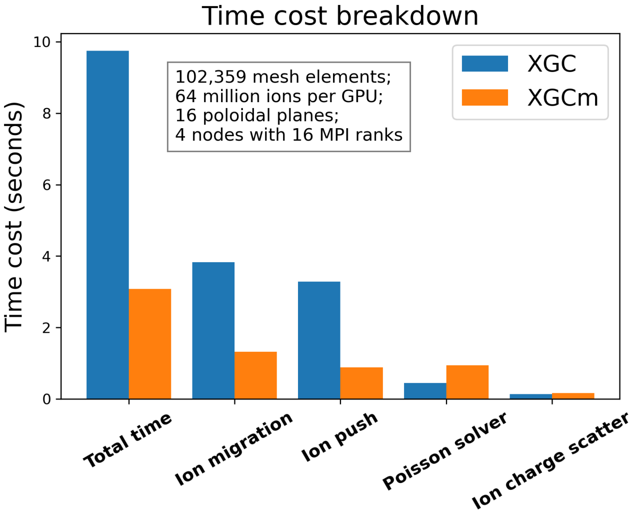
- 2M elements, 1M vertices, 2 to 128 poloidal planes
- Pseudo push and particle-to-mesh gyro scatter
- Tested on up to 24,576 GPUs of Summit with 1.1 trillion particles, for 100 iterations: push, adjacency
- PUMIPic weak scaling up to 24576 GPUs (4096 nodes) with 48 million particles per GPU



Total time comparison

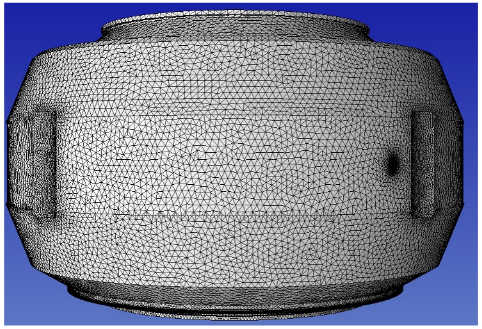
- Ran on NERSC's Perlmutter.
- XGCm 3 times faster than XGC for adiabatic electron case, 21% faster for kinetic electron case.

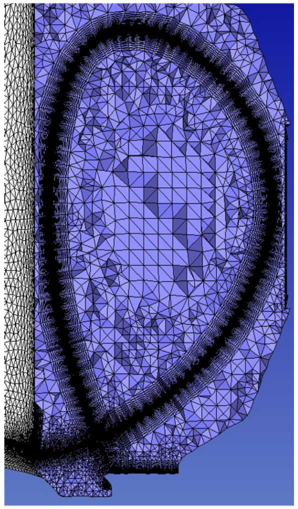


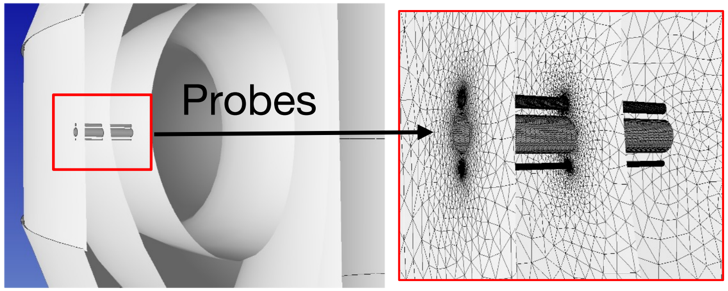


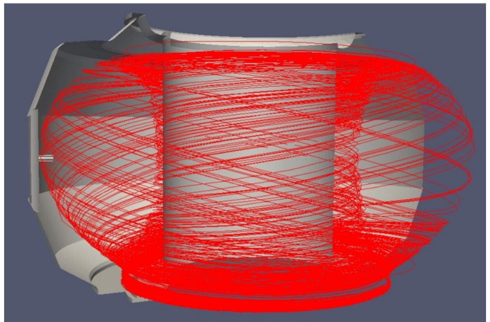
PUMIPic based GITRm Impurity Transport Code

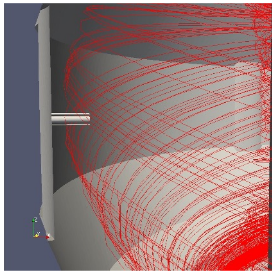
- Incorporates impurity transport capabilities of GITR
- 3D mesh for cases including divertors, tiles, limiters, specific diagnostics/probes etc.
- Status
  - Physics equivalent to GITR
  - Multi-species capabilities developed
  - Anisotropy mesh for accurate field transfer
  - Field transfer from SOLPS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







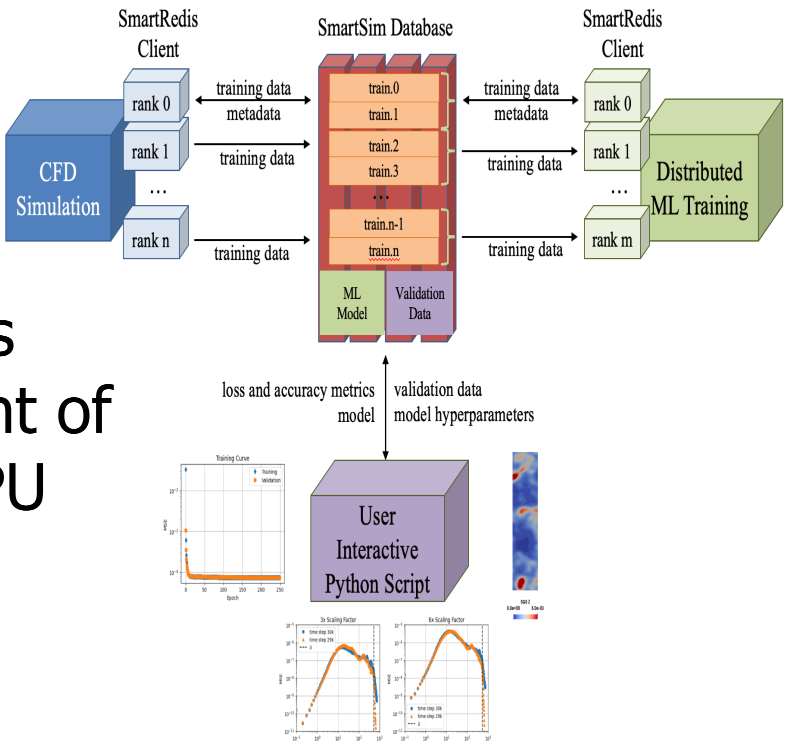




FASTMATH

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

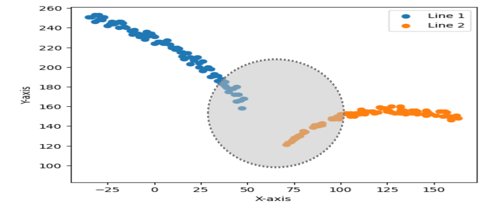
R.Balin et al., In Situ Framework for Coupling Machine Learning with Application to CFD, https://doi.org/10.48550/arXiv.2306.12900

FASTMATH

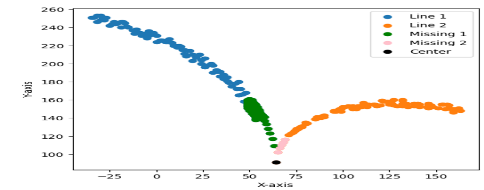
Mesh Related AI/ML Developments

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

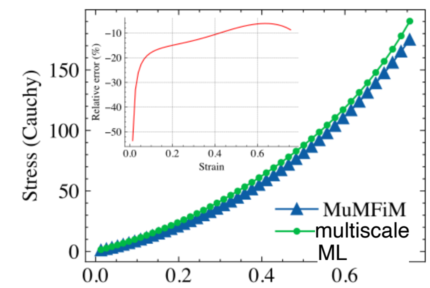


Improve sensor data



Physics informed material models

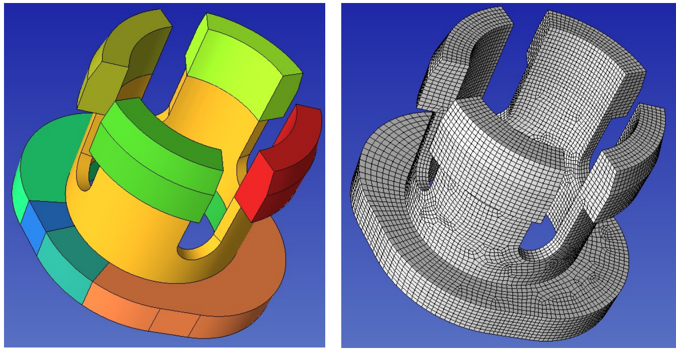
- Machine learned constitutive models can reduce runtime cost of upscaling FEM simulations by orders of magnitude while retaining 80% of accuracy.



Comparing ML to Multiscale

ML Agents for automated hex meshing

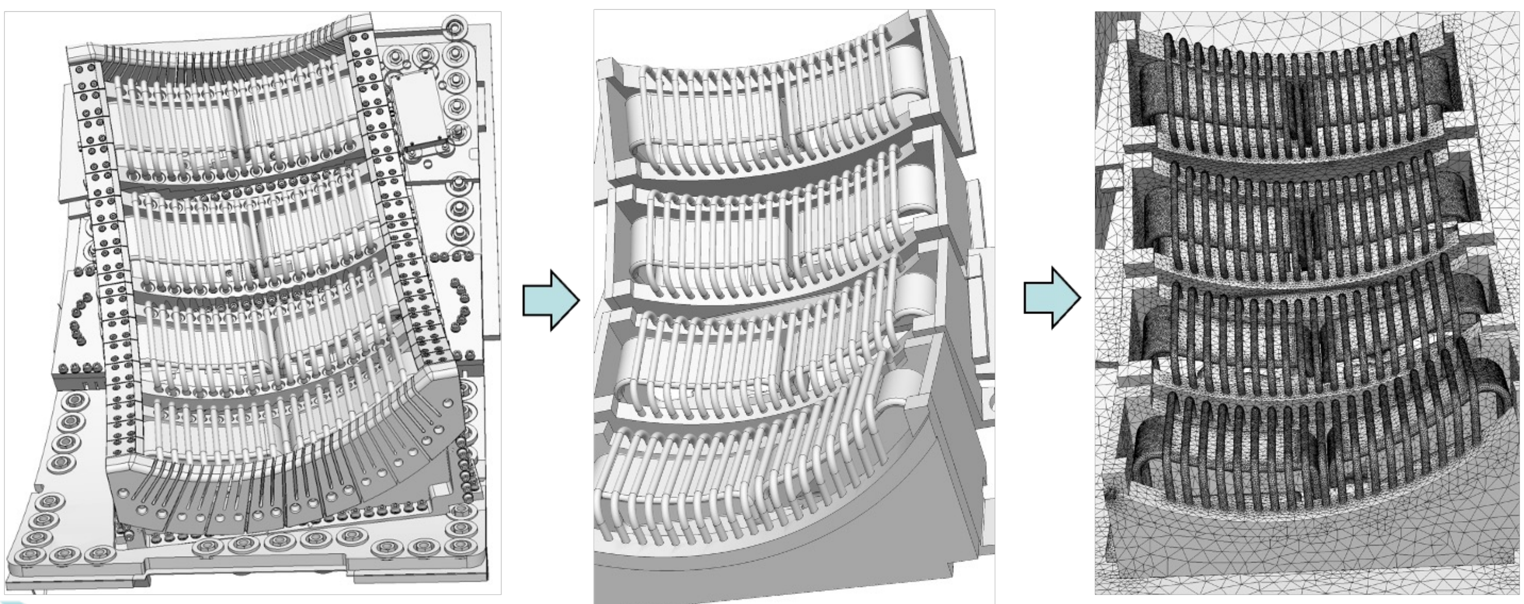
- Reinforced learning agent decomposes CAD modes into regions suitable for hexahedral mesh generation



Run the latest Simmetrix and PUMI software on RPI systems

We will help you run the latest Simmetrix and PUMI model preparation, mesh generation, and adaptation tools on **your problem** using HPC systems at RPI.

Contact Cameron Smith in Slack, during Speed-Dating, or via email at smithc11@rpi.edu for more information.



FASTMATH