Unstructured Meshing Technologies

Presented to
ATPESC 2020 Participants

Aaron Fisher (LLNL) & Mark Shephard (RPI)

Date 08/04/2020

U.S. DEPARTMENT OF ENERGY | Office of Science

ATPESC Numerical Software Track

EXASCALE COMPUTING PROJECT

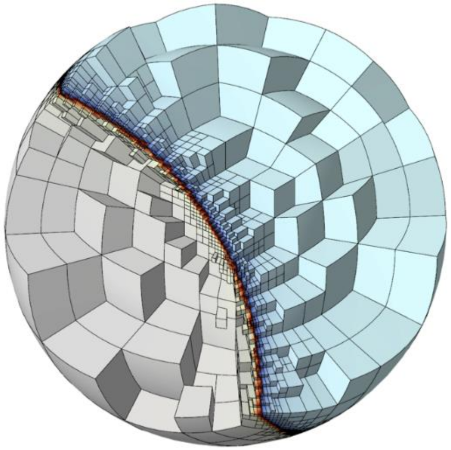
Argonne NATIONAL LABORATORY · BERKELEY LAB · Sandia National Laboratories · Rensselaer · SMU

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

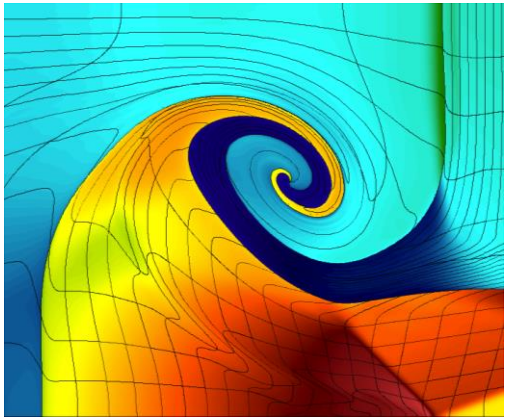
- Backed by well-developed theory.
- Naturally support unstructured and curvilinear grids.
- **High-order finite elements on high-order meshes**
  - Increased accuracy for smooth problems
  - Sub-element modeling for problems with shocks
  - Bridge unstructured/structured grids
  - Bridge sparse/dense linear algebra
  - FLOPs/bytes increase with the order
- Demonstrated match for compressible shock hydrodynamics (BLAST).
- Applicable to variety of physics (DeRham complex).

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

$H(grad)$	$H(curl)$	$H(div)$	L_2
"nodes"	"edges"	"faces"	"zones"
High-order kinematics	High-order MHD	High-order rad. diff.	High-order thermodynamics



Non-conforming mesh refinement on high-order curved meshes

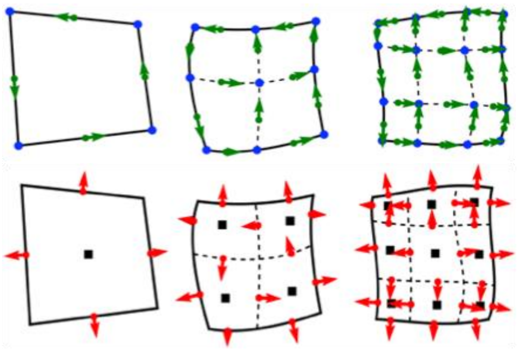


8th order Lagrangian hydro simulation of a shock triple-point interaction

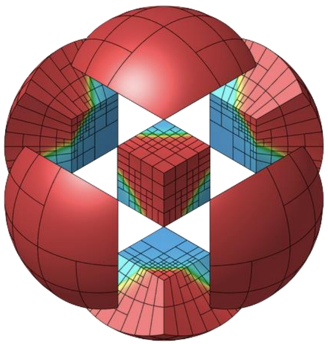
Modular Finite Element Methods (MFEM)

MFEM is an *open-source C++ library* for scalable FE research and fast application prototyping

- Triangular, quadrilateral, tetrahedral and hexahedral; volume and surface meshes
- Arbitrary order curvilinear mesh elements
- Arbitrary-order H1, H(curl), H(div)- and L2 elements
- Local conforming and non-conforming refinement
- NURBS geometries and discretizations
- Bilinear/linear forms for variety of methods (Galerkin, DG, DPG, Isogeometric, …)
- Integrated with: *HYPRE, SUNDIALS, PETSc, SUPERLU, PUMI, VisIt, Spack, xSDK, OpenHPC,* and more …
- Parallel and highly performant
- Main component of *ECP's co-design Center for Efficient Exascale Discretizations (CEED)*
- Native "in-situ" visualization: *GLVis*, glvis.org



Linear, quadratic and cubic finite element spaces on curved meshes



mfem.org
(v3.4, May/2018)

ECP CEED EXASCALE DISCRETIZATIONS FASTMATH Spack xSDK

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

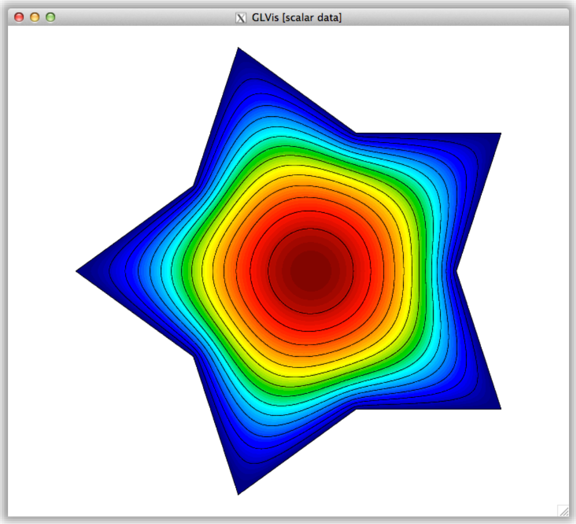
```
// 5. Set up the linear form b(.) which corresponds to the right-hand side of
//    the FEM linear system, which in this case is (1,phi_i) where phi_i are
//    the basis functions in the finite element fespace.
LinearForm *b = new LinearForm(fespace);
ConstantCoefficient one(1.0);
b->AddDomainIntegrator(new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

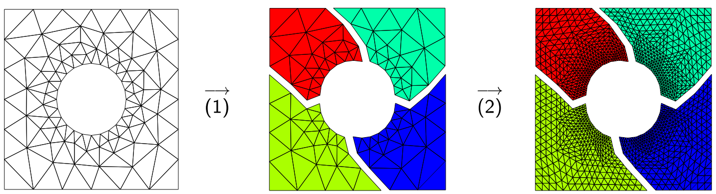
- Visualization

```
   // 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

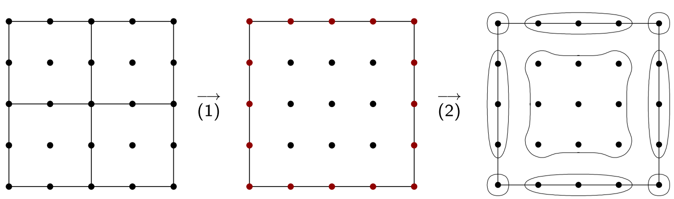
- Parallel mesh

```
// 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();
```

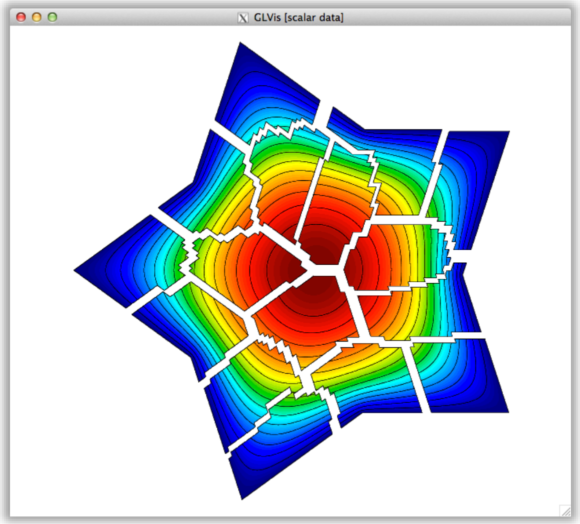
$$A = P^T a P \qquad B = P^T b \qquad x = PX$$

- 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);
```

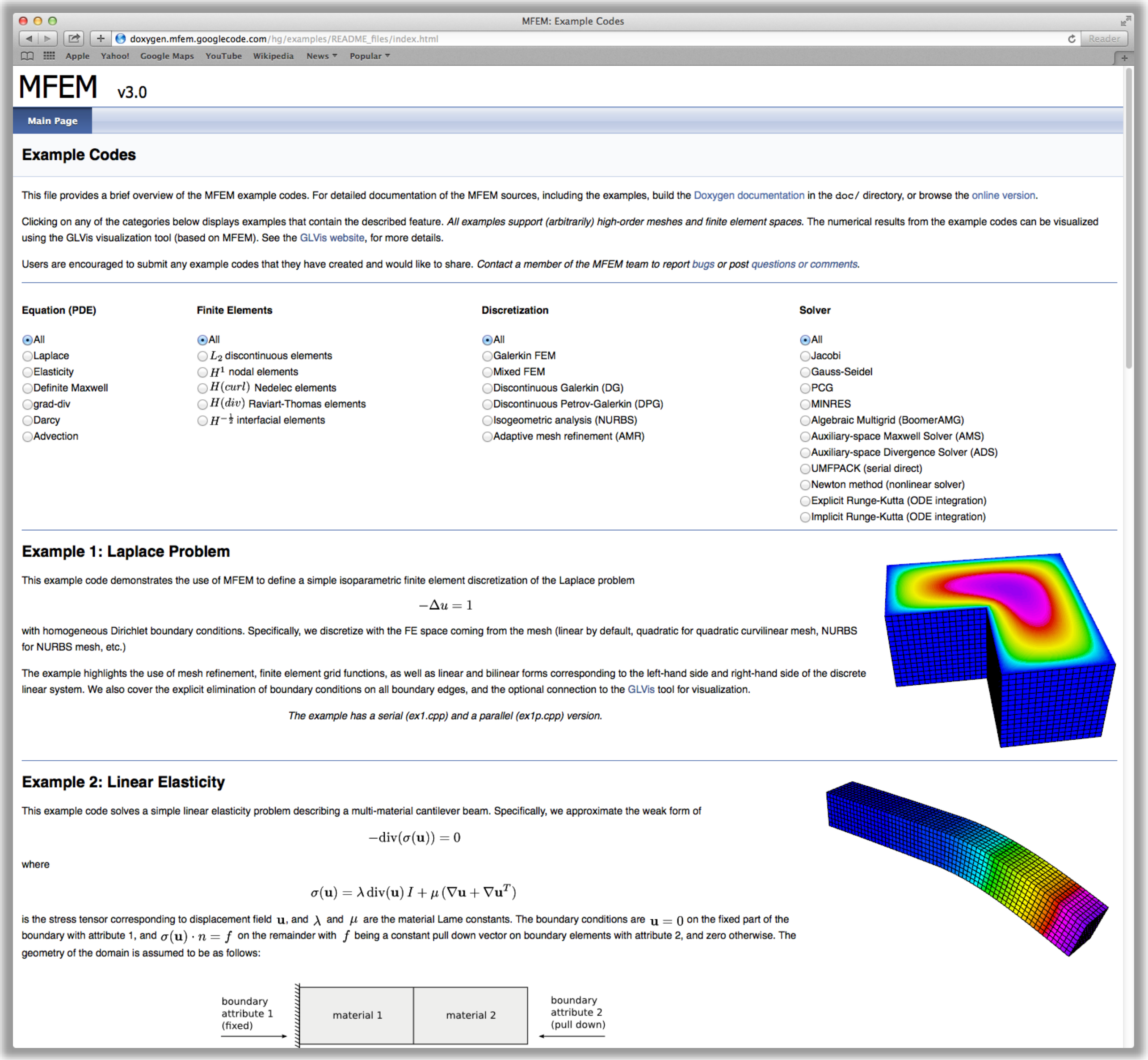
- Visualization

```
// 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

MFEM example codes – mfem.org/examples



MFEM: Example Codes

doxygen.mfem.googlecode.com/hg/examples/README_files/index.html

Apple Yahoo! Google Maps YouTube Wikipedia News Popular

MFEM v3.0

Main Page

Example Codes

This file provides a brief overview of the MFEM example codes. For detailed documentation of the MFEM sources, including the examples, build the Doxygen documentation in the doc/ directory, or browse the online version.

Clicking on any of the categories below displays examples that contain the described feature. *All examples support (arbitrarily) high-order meshes and finite element spaces.* The numerical results from the example codes can be visualized using the GLVis visualization tool (based on MFEM). See the GLVis website, for more details.

Users are encouraged to submit any example codes that they have created and would like to share. *Contact a member of the MFEM team to report bugs or post questions or comments.*

Equation (PDE)

- All
- Laplace
- Elasticity
- Definite Maxwell
- grad-div
- Darcy
- Advection

Finite Elements

- All
- L_2 discontinuous elements
- H^1 nodal elements
- $H(curl)$ Nedelec elements
- $H(div)$ Raviart-Thomas elements
- $H^{-\frac{1}{2}}$ interfacial elements

Discretization

- All
- Galerkin FEM
- Mixed FEM
- Discontinuous Galerkin (DG)
- Discontinuous Petrov-Galerkin (DPG)
- Isogeometric analysis (NURBS)
- Adaptive mesh refinement (AMR)

Solver

- All
- Jacobi
- Gauss-Seidel
- PCG
- MINRES
- Algebraic Multigrid (BoomerAMG)
- Auxiliary-space Maxwell Solver (AMS)
- Auxiliary-space Divergence Solver (ADS)
- UMFPACK (serial direct)
- Newton method (nonlinear solver)
- Explicit Runge-Kutta (ODE integration)
- Implicit Runge-Kutta (ODE integration)

Example 1: Laplace Problem

This example code demonstrates the use of MFEM to define a simple isoparametric finite element discretization of the Laplace problem

$$-\Delta u = 1$$

with homogeneous Dirichlet boundary conditions. Specifically, we discretize with the FE space coming from the mesh (linear by default, quadratic for quadratic curvilinear mesh, NURBS for NURBS mesh, etc.)

The example highlights the use of mesh refinement, finite element grid functions, as well as linear and bilinear forms corresponding to the left-hand side and right-hand side of the discrete linear system. We also cover the explicit elimination of boundary conditions on all boundary edges, and the optional connection to the GLVis tool for visualization.

The example has a serial (ex1.cpp) and a parallel (ex1p.cpp) version.

Example 2: Linear Elasticity

This example code solves a simple linear elasticity problem describing a multi-material cantilever beam. Specifically, we approximate the weak form of

$$-\mathrm{div}(\sigma(\mathbf{u})) = 0$$

where

$$\sigma(\mathbf{u}) = \lambda\,\mathrm{div}(\mathbf{u})\,I + \mu\,(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$$

is the stress tensor corresponding to displacement field $\mathbf{u}$, and λ and μ are the material Lame constants. The boundary conditions are $\mathbf{u} = 0$ on the fixed part of the boundary with attribute 1, and $\sigma(\mathbf{u})\cdot n = f$ on the remainder with f being a constant pull down vector on boundary elements with attribute 2, and zero otherwise. The geometry of the domain is assumed to be as follows:

boundary attribute 1 (fixed)
material 1
material 2
boundary attribute 2 (pull down)

Discretization Demo & Lesson

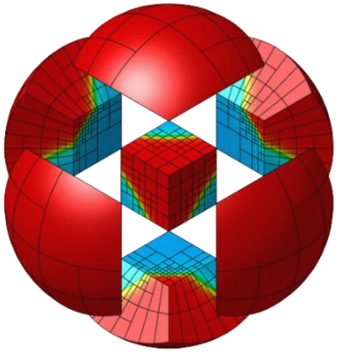
https://xsdk-project.github.io/MathPackagesTraining2020/lessons/mfem_convergence/

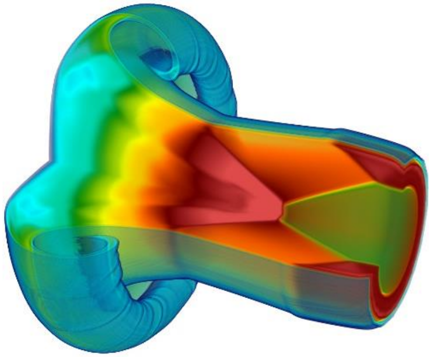
Application to high-order ALE shock hydrodynamics

hypre: *Scalable linear solvers library*

MFEM: *Modular finite element methods library*

BLAST: *High-order ALE shock hydrodynamics research code*





www.llnl.gov/casc/hypre

mfem.org

www.llnl.gov/casc/blast

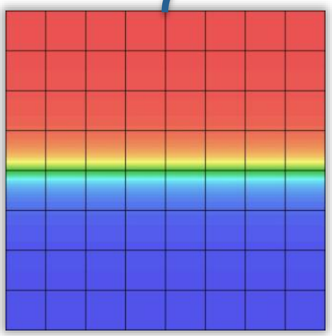
- *hypre* provides scalable algebraic multigrid solvers
- MFEM provides finite element discretization abstractions
  - uses *hypre*'s parallel data structures, provides finite element info to solvers
- BLAST solves the Euler equations using a high-order ALE framework
  - combines and extends MFEM's objects

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

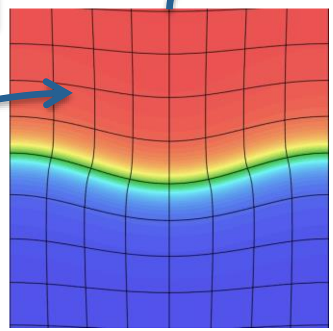
Lagrange phase

Physical time evolution

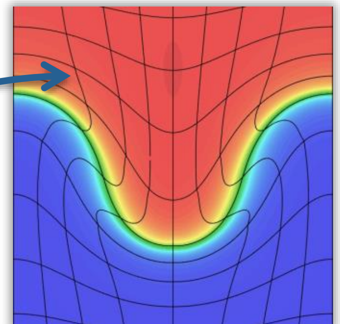
Based on physical motion



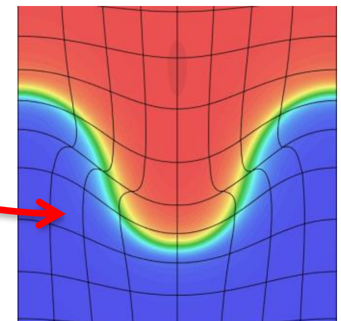
$t = 0$



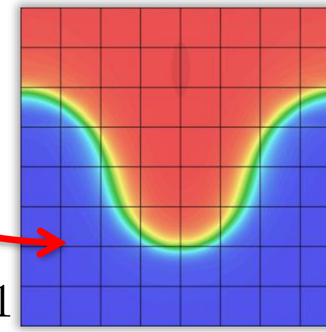
$t = 1.5$



$t = 3.0 \quad \tau = 0$



$\tau = 0.5$



$\tau = 1$

Remap phase

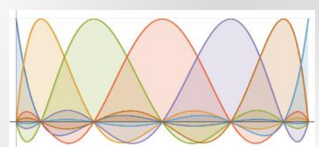
Pseudo-time evolution

Based on mesh motion

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

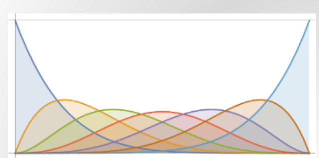
Momentum Conservation:	$\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \nabla \cdot \sigma$
Mass Conservation:	$\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\rho \nabla \cdot \vec{v}$
Energy Conservation:	$\rho \frac{\mathrm{d}e}{\mathrm{d}t} = \sigma : \nabla \vec{v}$
Equation of Motion:	$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t}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

- Galerkin FEM



Gauss-Lobatto basis

- Discont. Galerkin

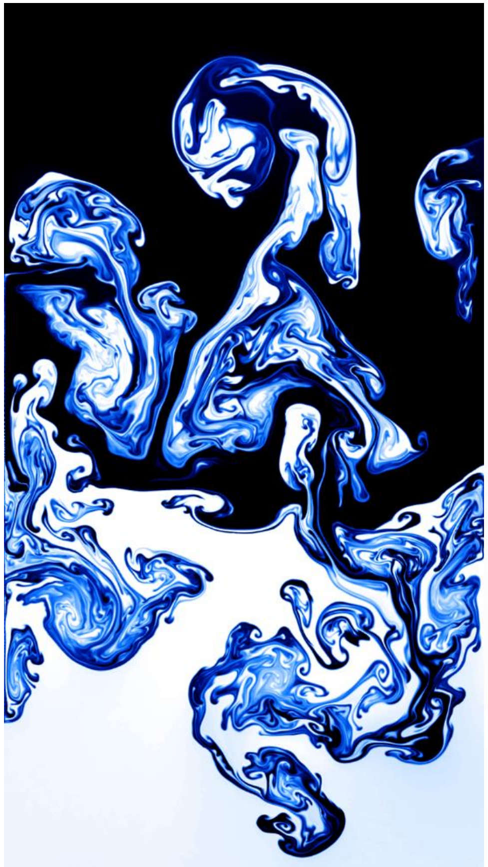


Bernstein basis

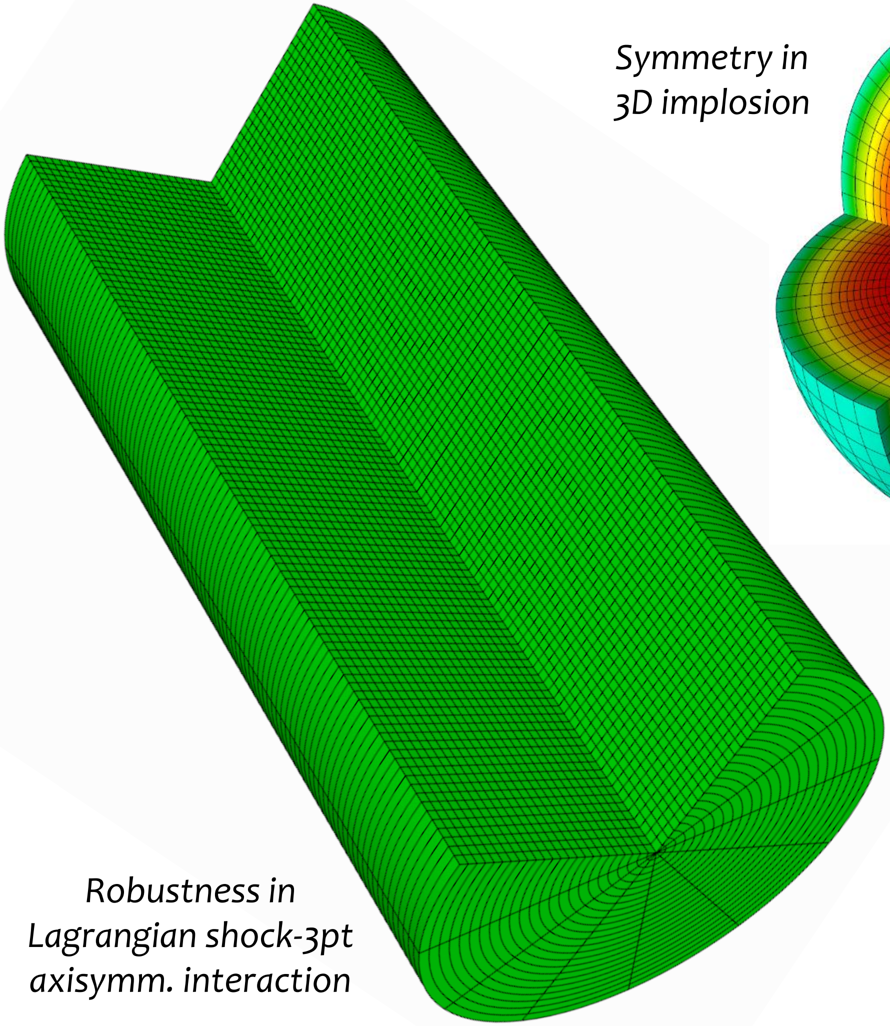
Advection phase ($\vec{c} = -\vec{v}_m$)

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

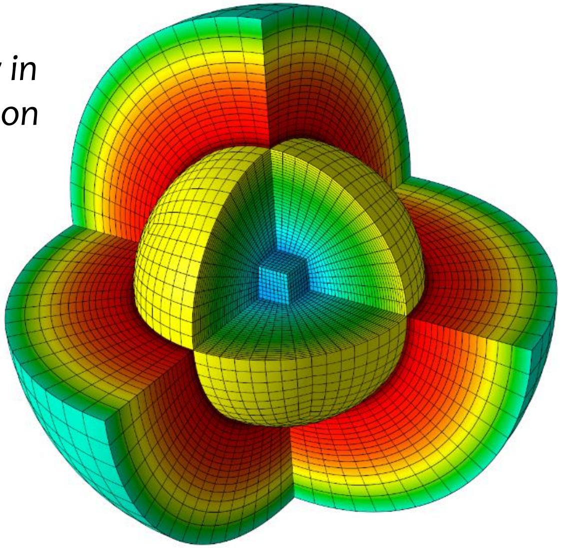


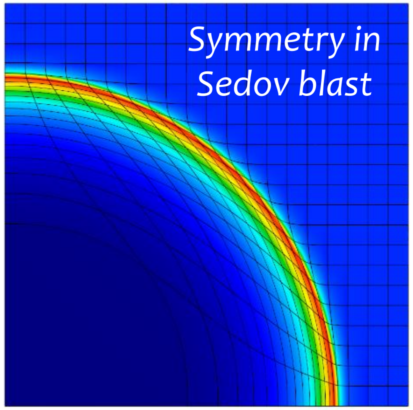
Parallel ALE for Q4 Rayleigh-Taylor instability (256 cores)



Robustness in Lagrangian shock-3pt axisymm. interaction

Symmetry in 3D implosion

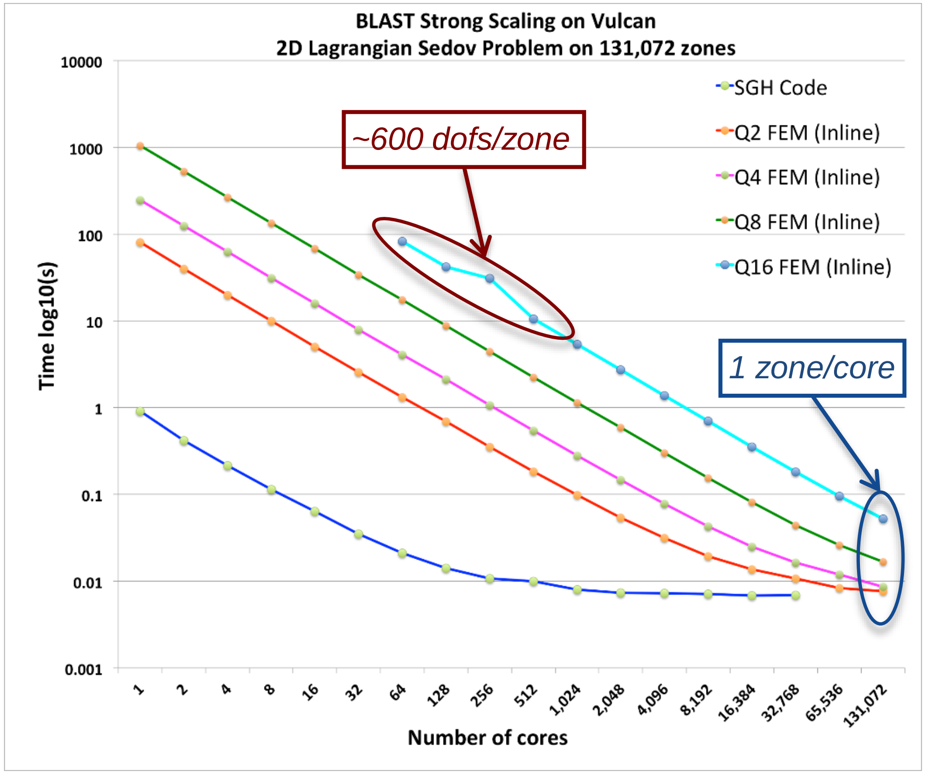




Symmetry in Sedov blast

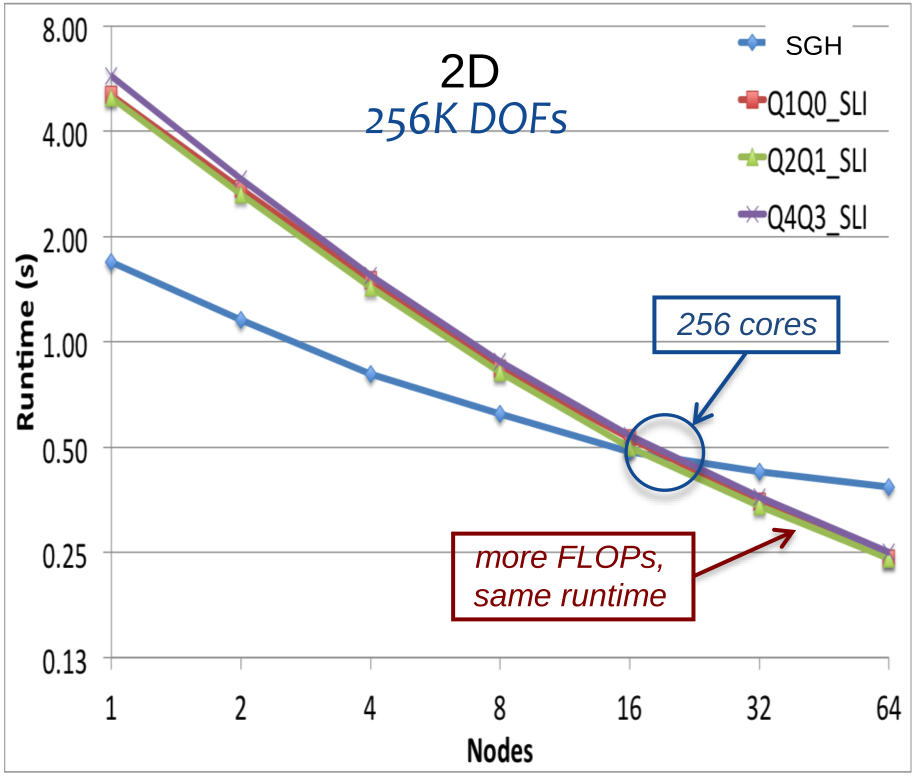
High-order finite elements have excellent strong scalability

Strong scaling, p-refinement



Finite element partial assembly

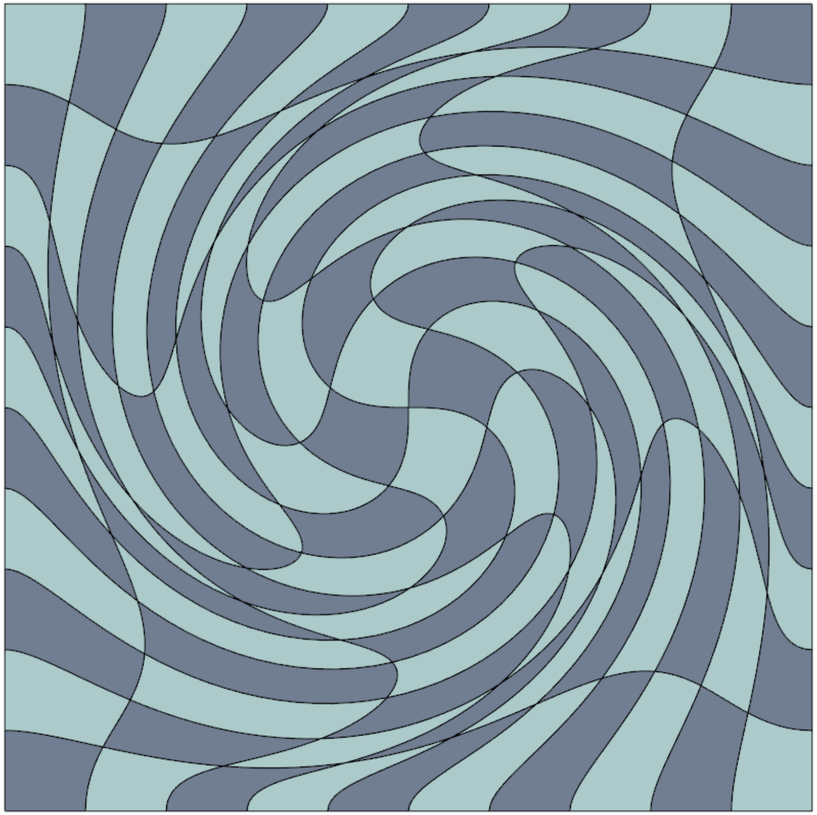
Strong scaling, fixed #dofs



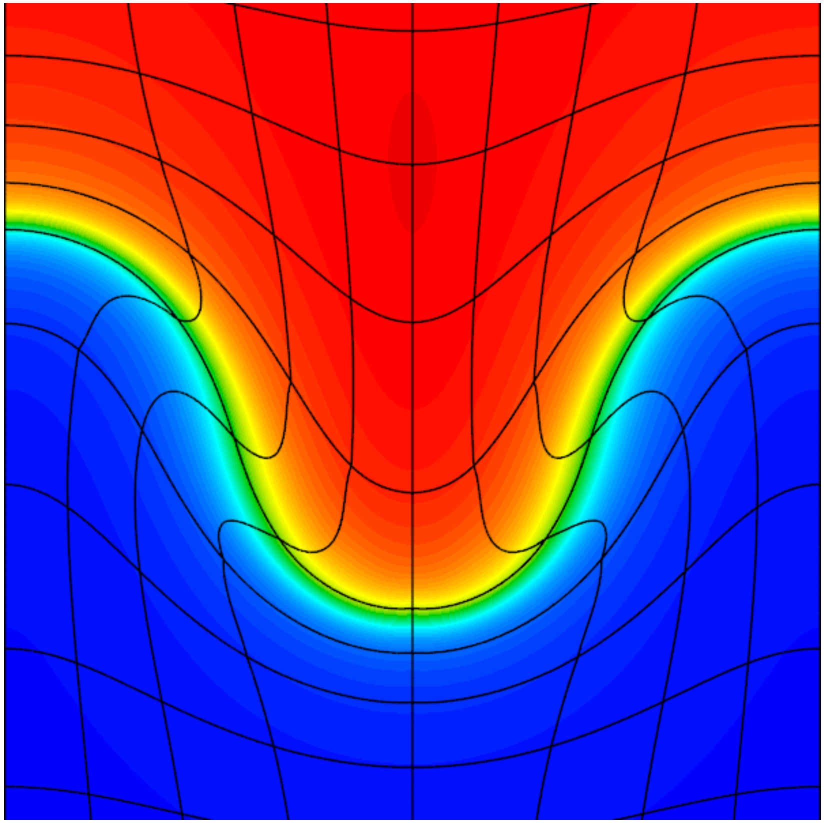
FLOPs increase faster than runtime

Unstructured Mesh R&D: Mesh optimization and high-quality interpolation between meshes

We target *high-order curved elements* + *unstructured meshes* + *moving meshes*



High-order mesh relaxation by neo-Hookean evolution (Example 10, ALE remesh)

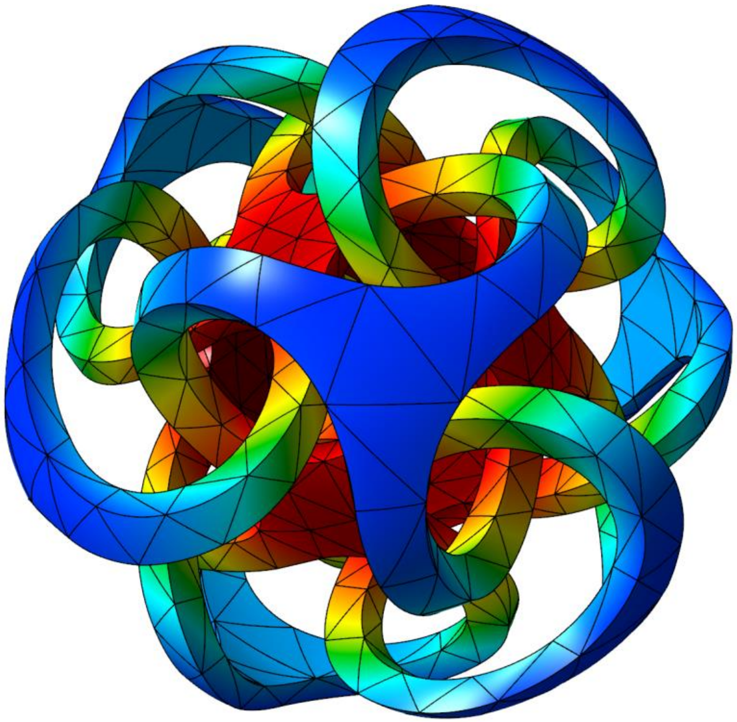


DG advection-based interpolation (ALE remap, Example 9, radiation transport)

Unstructured Mesh R&D: Accurate and flexible finite element visualization

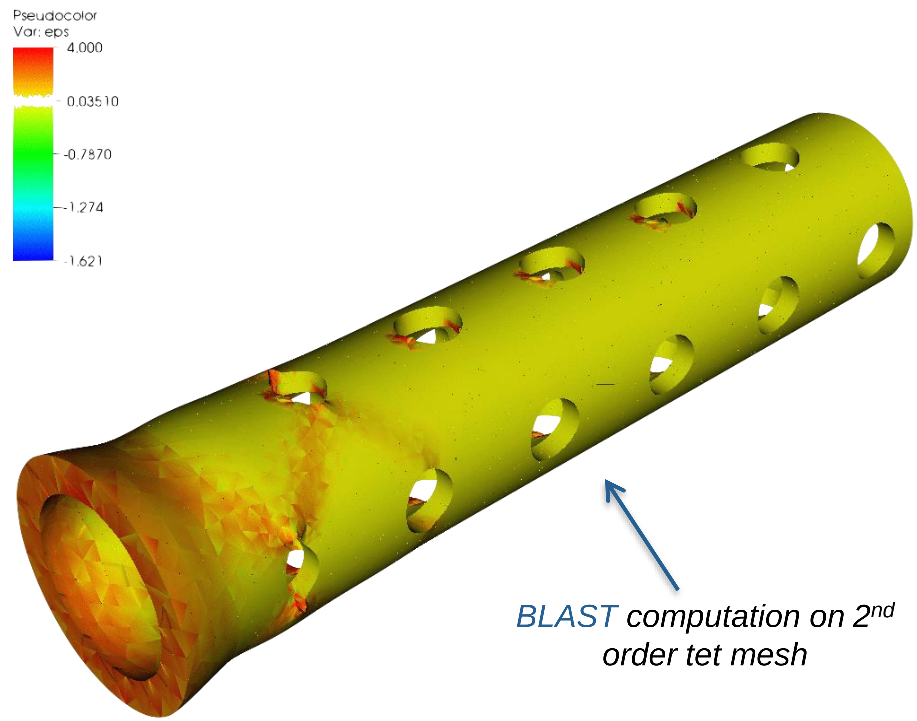
Two visualization options for high-order functions on high-order meshes

GLVis: *native MFEM lightweight OpenGL visualization tool*



glvis.org

VisIt: *general data analysis tool, MFEM support since version 2.9*



BLAST computation on 2nd order tet mesh

visit.llnl.gov

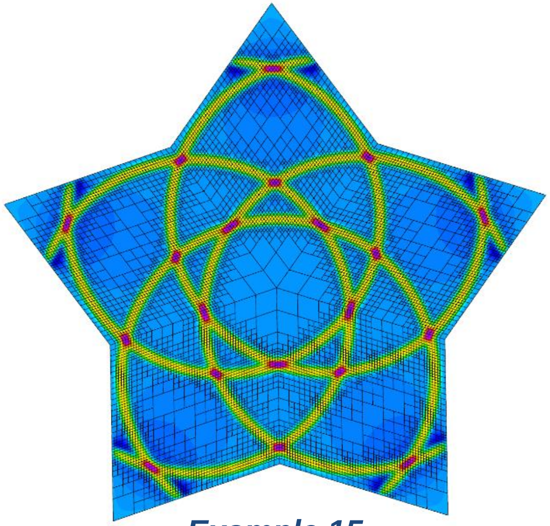
MFEM's unstructured AMR infrastructure

Adaptive mesh refinement on library level:

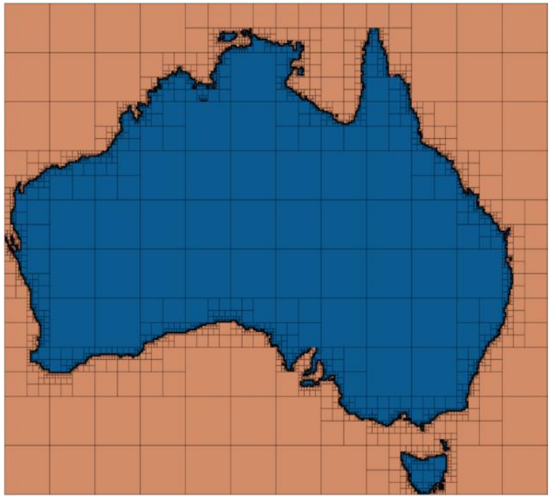
- Conforming local refinement on simplex meshes
- ***Non-conforming refinement for quad/hex meshes***
- h-refinement with fixed p

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)

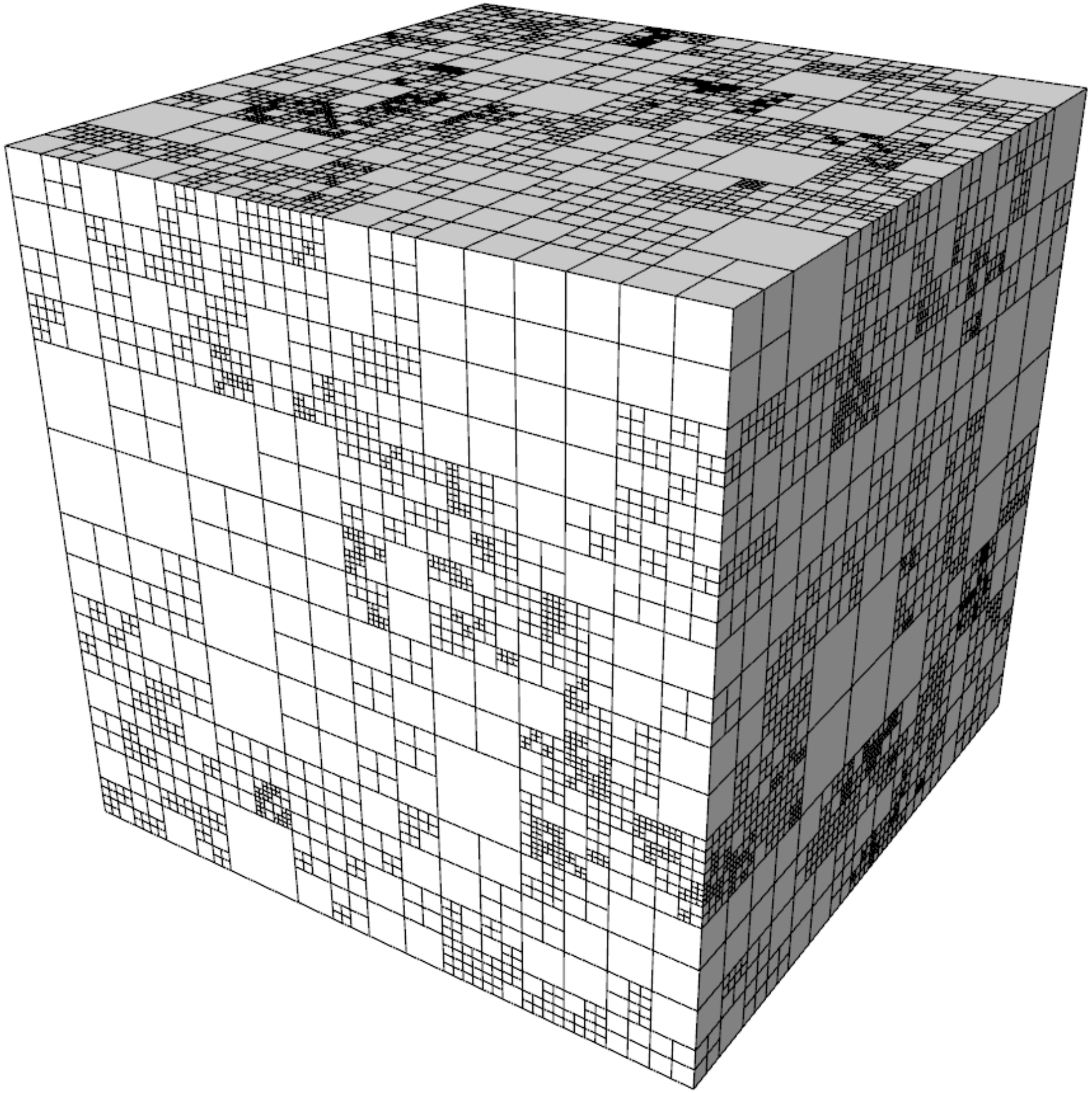


Example 15

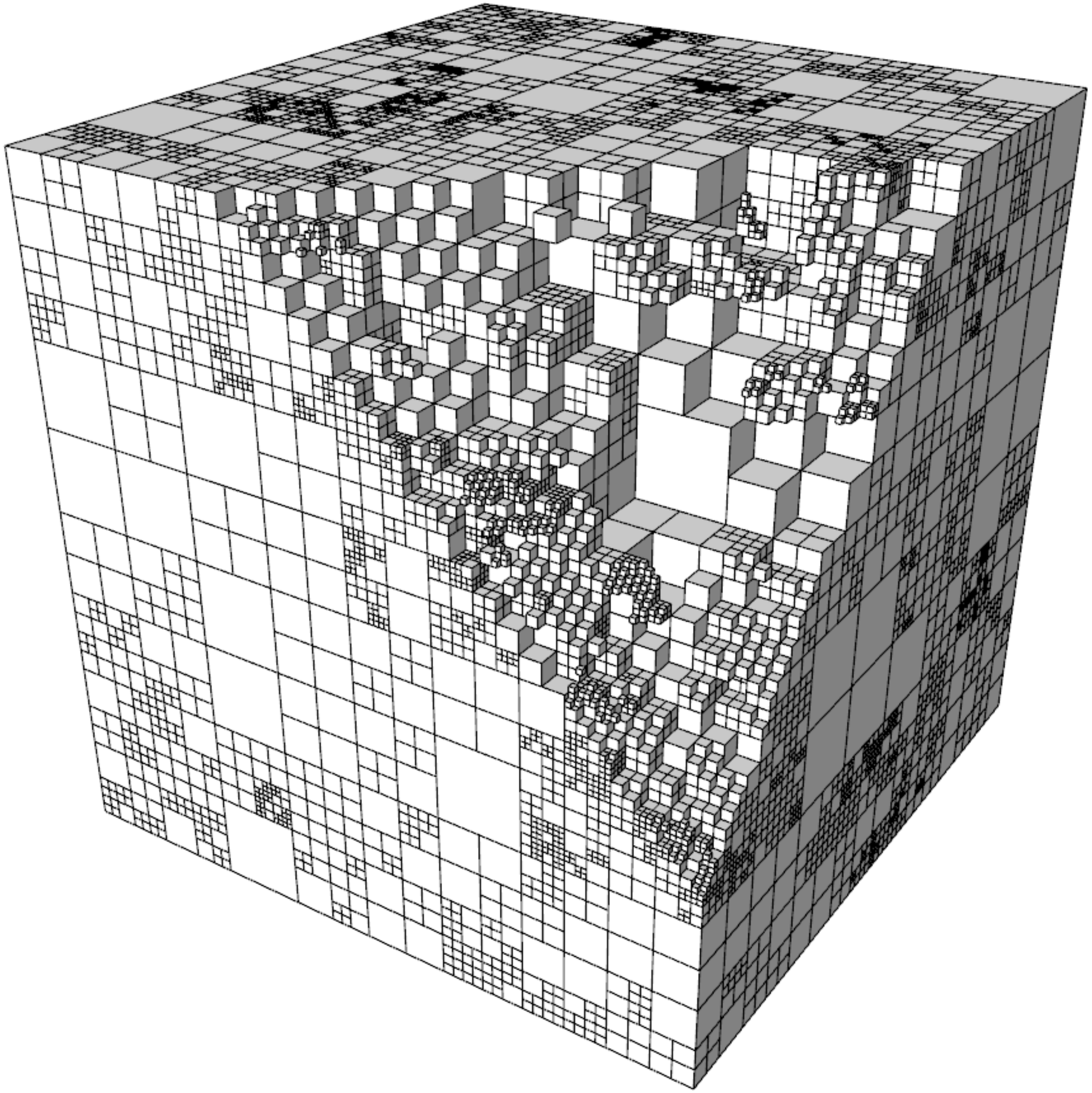


Shaper miniapp

Nonconforming variational restriction

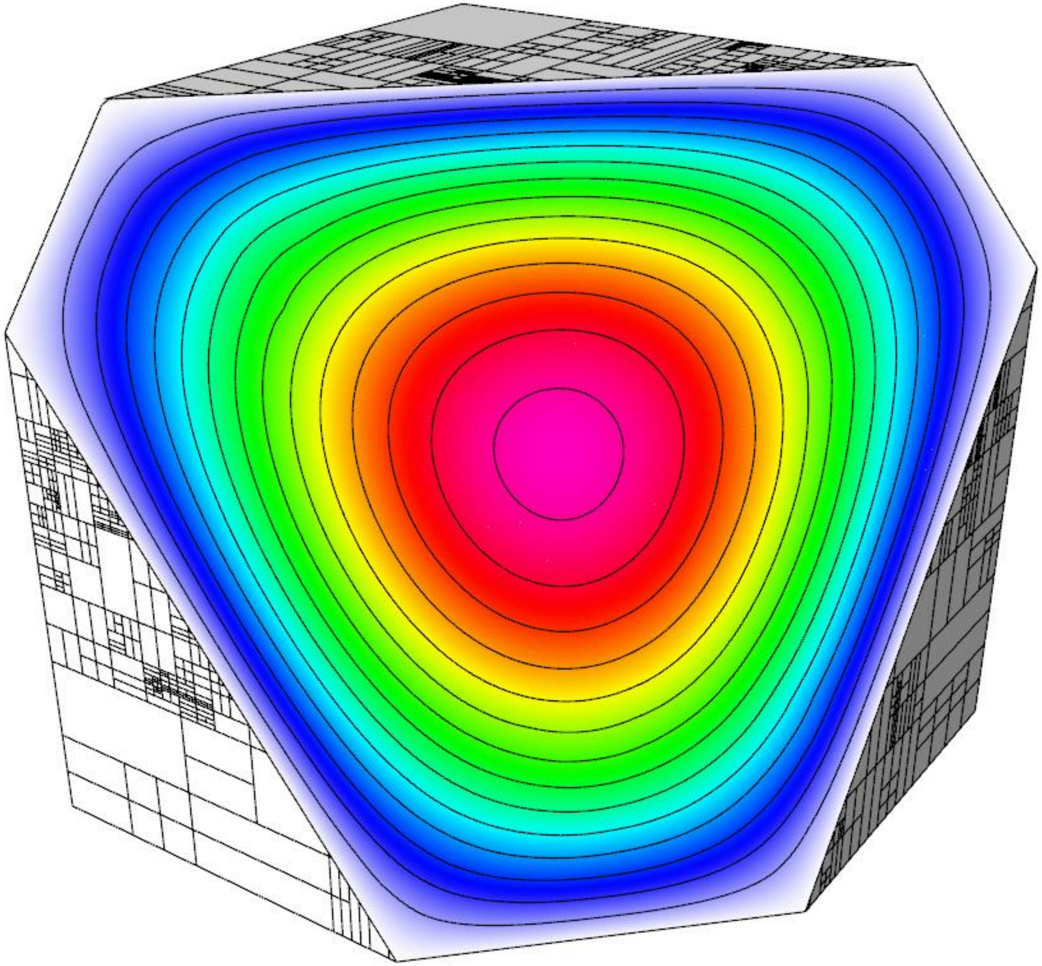


Nonconforming variational restriction



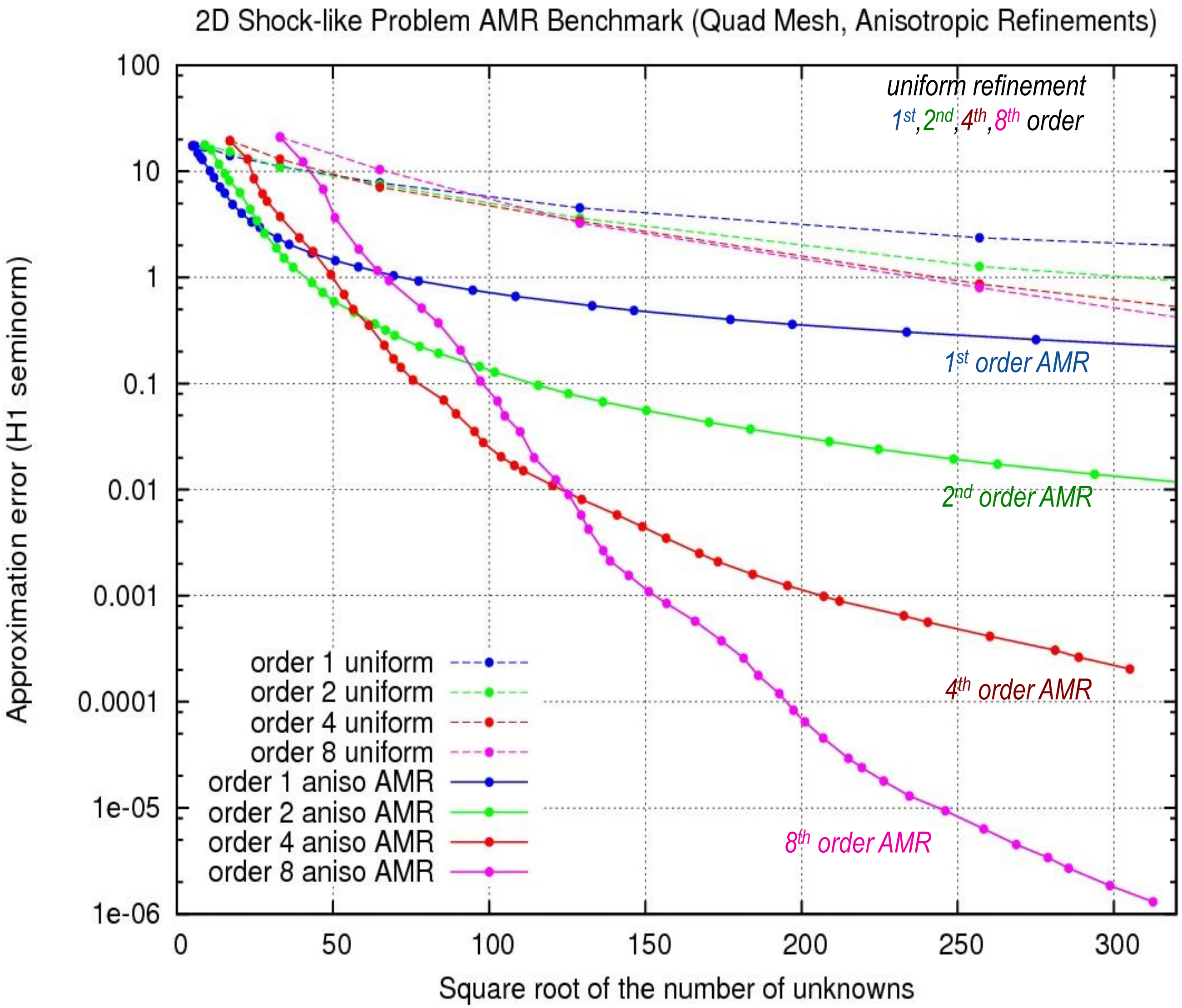
Regular assembly of A on the elements of the (cut) mesh

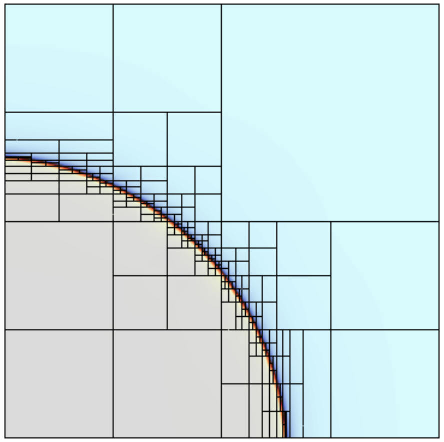
Nonconforming variational restriction

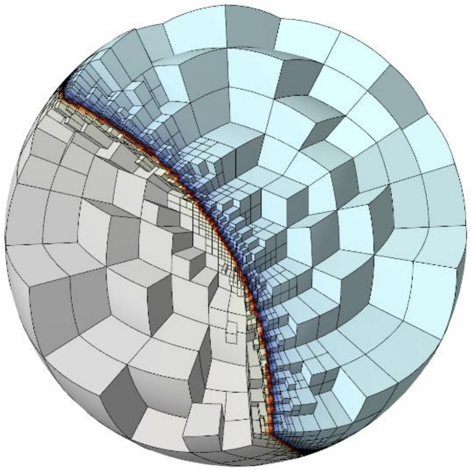


Conforming solution $y = P\,x$

AMR = smaller error for same number of unknowns

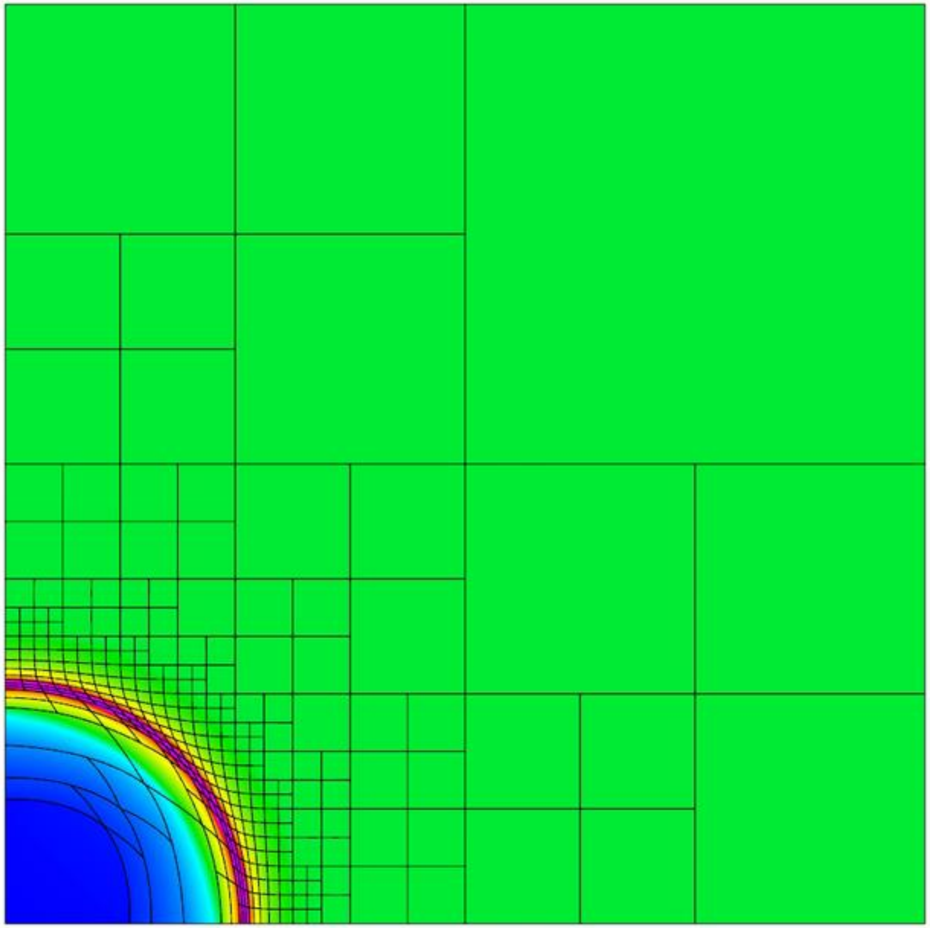




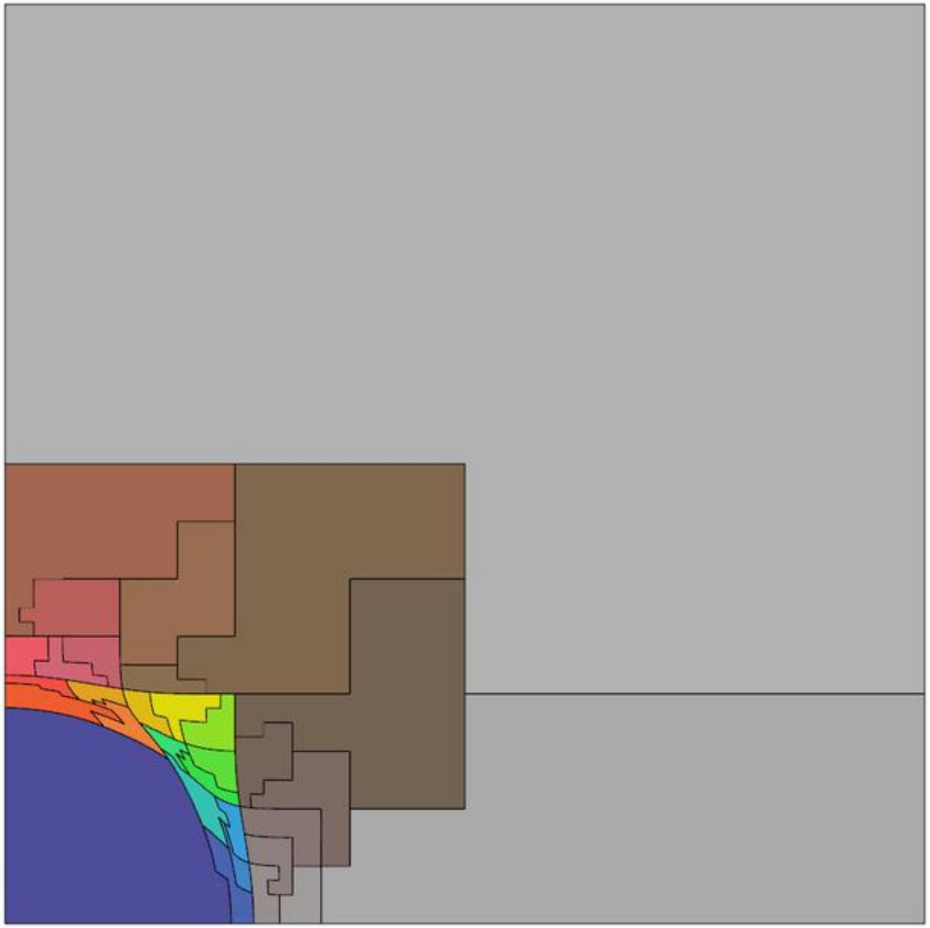


Anisotropic adaptation to shock-like fields in 2D & 3D

Parallel dynamic AMR, Lagrangian Sedov problem

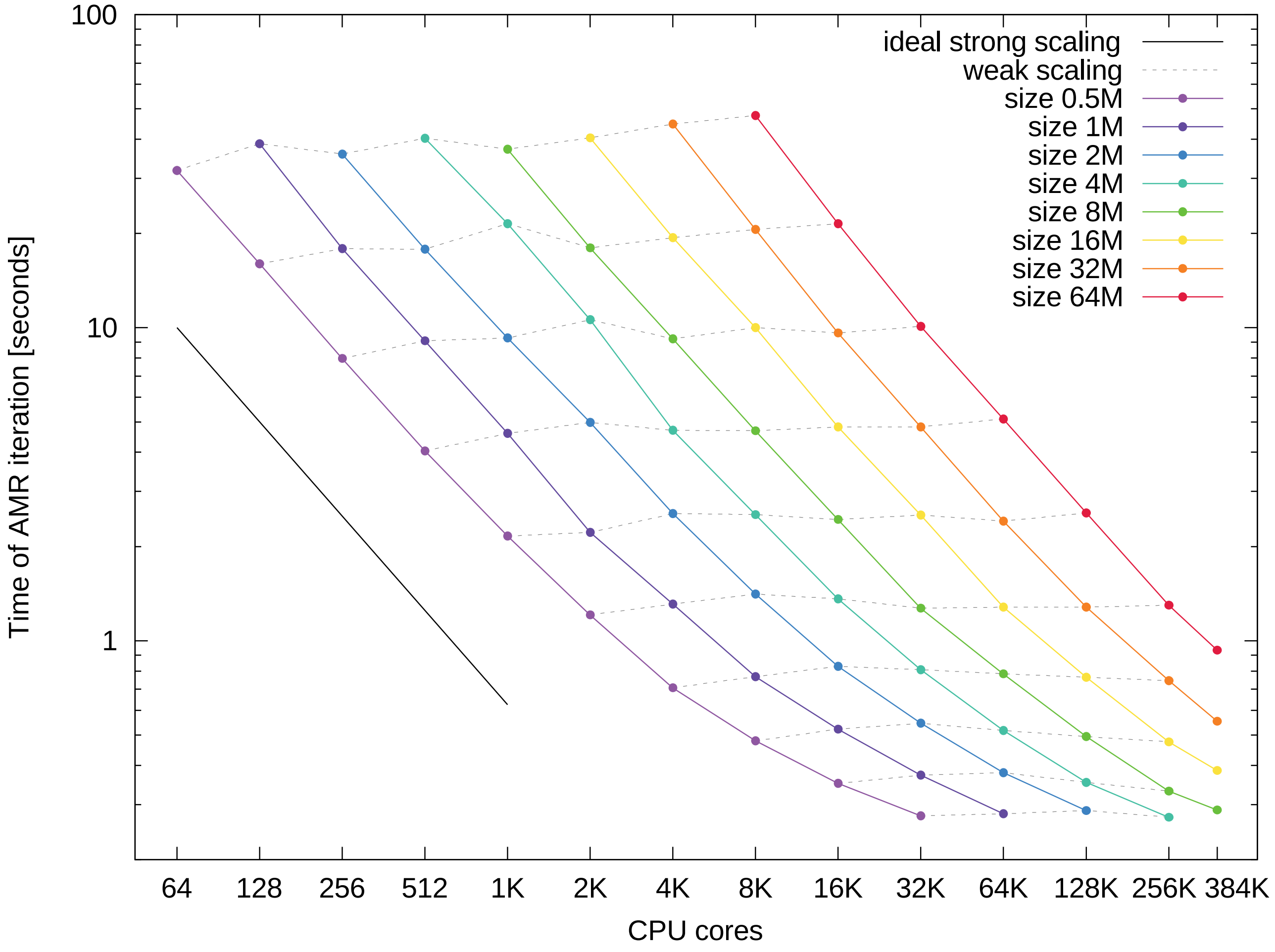


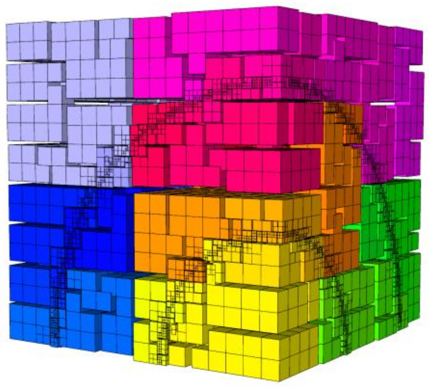
Adaptive, viscosity-based refinement and derefinement. 2nd order Lagrangian Sedov



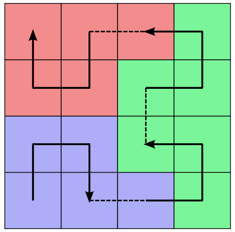
Parallel load balancing based on space-filling curve partitioning, 16 cores

Parallel AMR scaling to ~400K MPI tasks



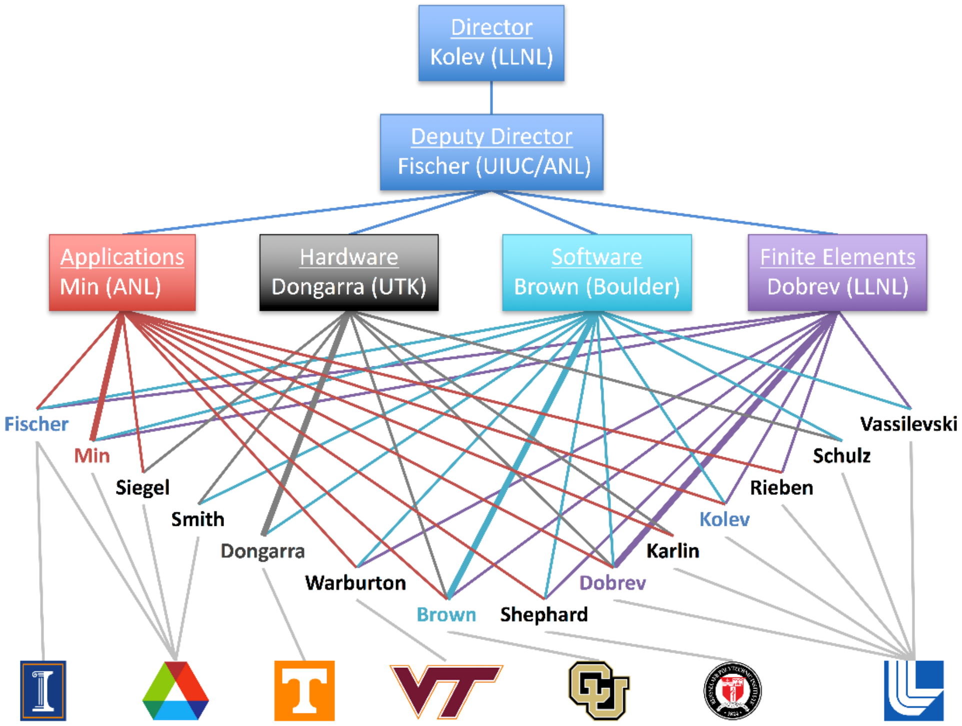


Parallel decomposition (2048 domains shown)



Parallel partitioning via Hilbert curve

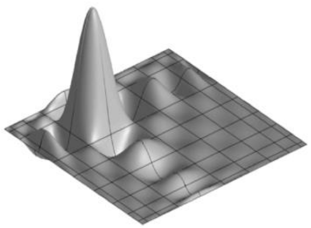
- weak+strong scaling up to ~400K MPI tasks on BG/Q
- **measure AMR only components**: interpolation matrix, assembly, marking, refinement & rebalancing (no linear solves, no “physics”)



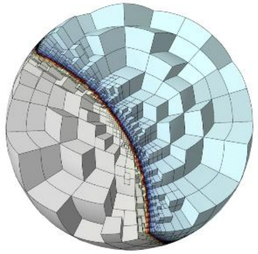
2 Labs, 5 Universities, 30+ researchers

- PDE-based simulations on **unstructured grids**
- **high-order** and **spectral** finite elements

  ✓ *any order space on any order mesh* ✓ *curved meshes,*
  ✓ *unstructured AMR* ✓ *optimized low-order support*



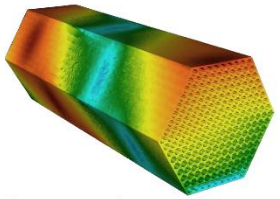
10th order basis function



non-conforming AMR, 2nd order mesh

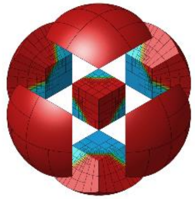
- state-of-the art CEED **discretization libraries**

  ✓ *better exploit the hardware to deliver significant performance gain over conventional methods*
  ✓ *based on MFEM/Nek, low & high-level APIs*



nek5000.mcs.anl.gov

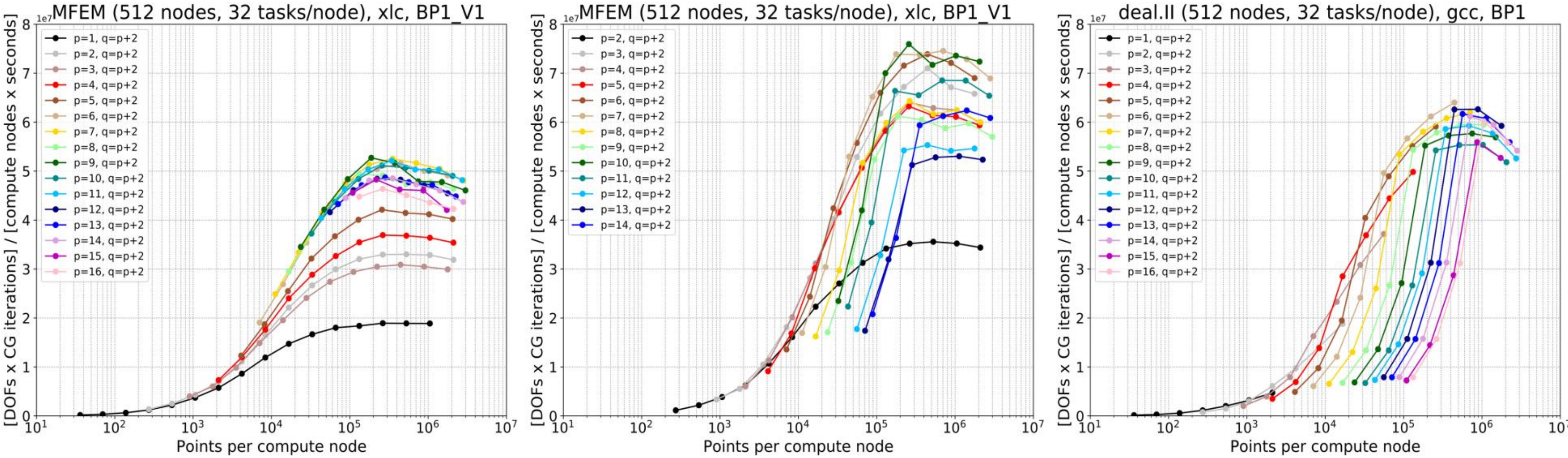
High-performance spectral elements



mfem.org

Scalable high-order finite elements

CEED Bake-off Problem 1 on CPU



(a) BP1 MFEM-before (b) BP1 MFEM-after (c) BP1 deal.II

- All runs done on BG/Q (for repeatability), 8192 cores in C32 mode. 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.
- Preliminary results – paper in preparation
- Cooperation/collaboration is what makes the bake-offs rewarding.

High-order methods show promise for high-quality & performance 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**
  - Porting to GPUs: Summit and Sierra
  - Efficient high-order methods on simplices
  - Matrix-free scalable preconditioners

Q4 Rayleigh-Taylor single-material ALE on 256 processors

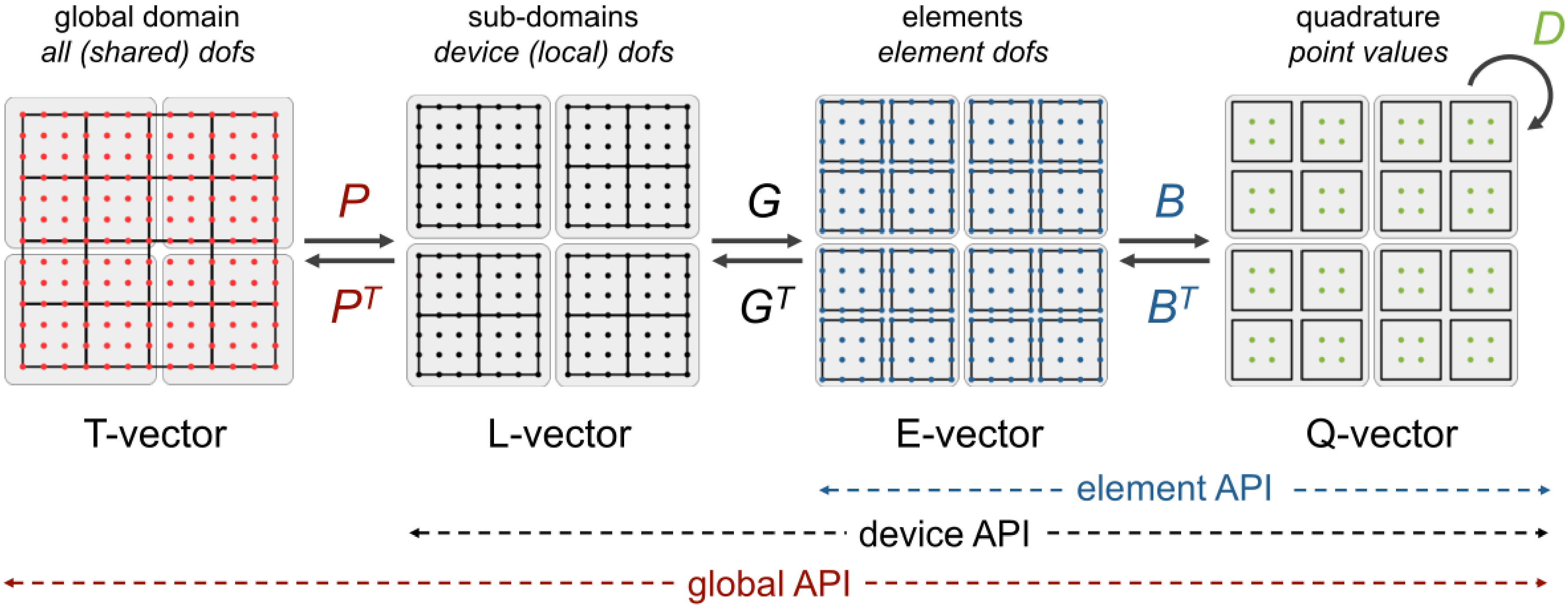
Lawrence Livermore National Laboratory

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

Fundamental finite element operator decomposition

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

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



- ***partial assembly*** = store only D, evaluate B (tensor-product structure)
- better representation than A: *optimal memory*, *near-optimal FLOPs*
- purely algebraic, applicable to many apps

CEED high-order benchmarks (BPs)

- CEED's *bake-off problems* (BPs) are high-order kernels/benchmarks designed to test and compare the performance of high-order codes.

  BP1: Solve {Mu=f}, where {M} is the mass matrix, q=p+2

  BP2: Solve the vector system $\{Mu_i=f_i\}$ with {M} from BP1, q=p+2

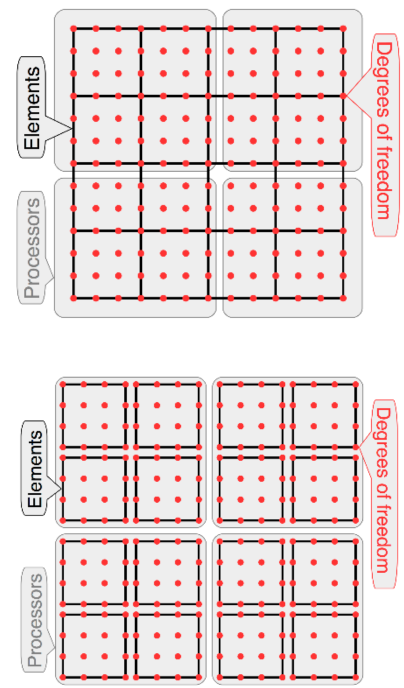
  BP3: Solve {Au=f}, where {A} is the Poisson operator, q=p+2

  BP4: Solve the vector system $\{Au_i=f_i\}$ with {A} from BP3, q=p+2

  BP5: Solve {Au=f}, where {A} is the Poisson operator, q=p+1

  BP6: Solve the vector system $\{Au_i=f_i\}$ with {A} from BP3, q=p+1

- Compared Nek and MFEM implementations on BG/Q, KNLs, GPUs.
- Community involvement – deal.ii, interested in seeing *your* results.
- Goal is to learn from each other, benefit all CEED-enabled apps.



BP terminology: T- and E-vectors of HO dofs

github.com/ceed/benchmarks

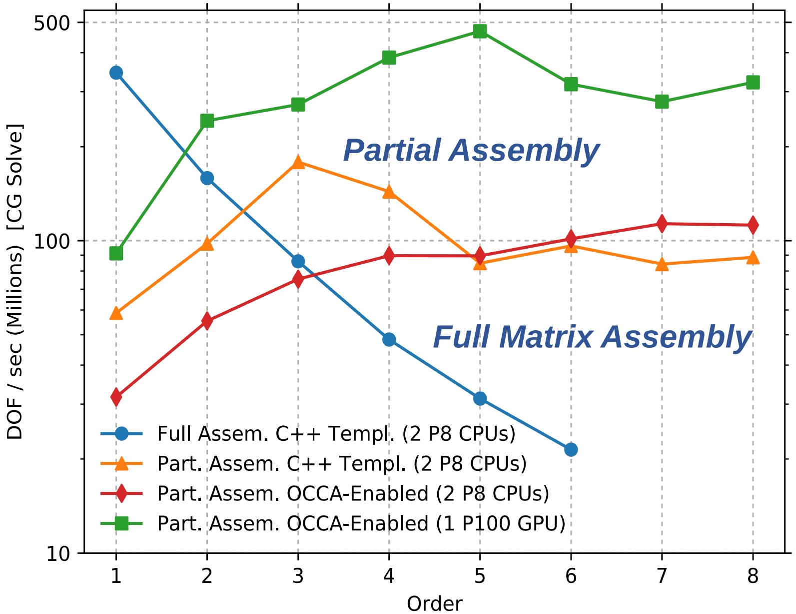
Tensorized partial assembly

$$B_{ki} = \varphi_i(q_k) = \varphi_{i_1}^{1d}(q_{k_1})\varphi_{i_2}^{1d}(q_{k_2}) = B_{k_1 i_1}^{1d} B_{k_2 i_2}^{1d} \qquad U_{k_1 k_2} = B_{k_1 i_1}^{1d} B_{k_2 i_2}^{1d} V_{i_1 i_2} \mapsto U = B^{1d} V (B^{1d})^T$$

p – order, d – mesh dim, $O(p^d)$ – dofs

Method	Memory	Assembly	Action
Full Matrix Assembly	$O(p^{2d})$	$O(p^{3d})$	$O(p^{2d})$
Partial Assembly	$O(p^d)$	$O(p^d)$	$O(p^{d+1})$

Storage and floating point operation scaling for different assembly types



Poisson CG solve performance with different assembly types (higher is better)

Full matrix performance drops sharply at high orders while partial assembly scales well!

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

FASTMath Unstructured Mesh Technologies

K.D. Devine[1], V. Dobrev, D.A. Ibanez[1], T. Kolev[2], K.E. Jansen[3],

O. Sahni[3], A.G. Salinger[1], S. Seol[4], M.S. Shephard[4], C.W. Smith[4]

[1]Sandia National Laboratories

[2]Lawrence Livermore National Laboratory

[3]University of Colorado

[4]Rensselaer Polytechnic Institute

U.S. DEPARTMENT OF ENERGY | Office of Science

ATPESC Numerical Software Track

EXASCALE COMPUTING PROJECT

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 proper geometric model, with analysis attributes defined on that model, the entire simulation work flow can be automated

Disadvantages

- More complex data structures and increased program complexity, particularly in parallel
- Requires careful mesh quality control (level required a function of the unstructured mesh analysis code)
- Poorly shaped elements increase condition number of global system – makes matrix solves harder

Unstructured Mesh Methods

Goal of FASTMath unstructured mesh technologies:

- Component-based tools that take full advantage of unstructured mesh methods and are easily used by analysis code developers and users
- Components operate through multi-level APIs that increase interoperability and ease integration
- Unstructured mesh tools to address needs and eliminate/minimize disadvantages of unstructured meshes
- Integration of these technologies with their tools to address application needs that arise

FASTMath Unstructured Mesh Developments

Areas of Technology development:

- Unstructured Mesh Analysis Codes – Support application's PDE solution needs – MFEM is a key example code
- 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 the effective execution of applications on heterogeneous systems
- Unstructured Mesh for PIC – Tools to support PIC on unstructured meshes
- Unstructured Mesh for UQ – Bringing unstructured mesh adaptation to UQ
- In Situ Vis and Data Analytics – Tools to gain insight as simulations execute

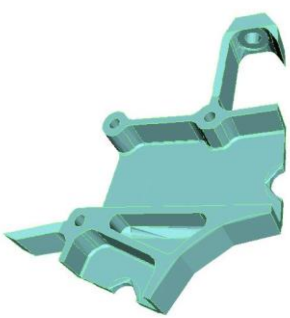
Performant Unstructured Meshes

- Goal
  - Unstructured meshing technologies that execute on exascale systems
  - Develop versions of tools that run on accelerators (GPUs)
  - Strive to have all operations on execute on GPUs
- Developing GPU based versions of
  - Unstructured mesh solvers (MFEM, etc.)
  - Mesh adaptation (Omega_h)
  - PIC operations on Unstructured Meshes
- Relevant Software Tools
  - MFEM
  - Omega_h (https://github.com/ibaned/omega_h)
  - PUMIpic

Parallel Unstructured Mesh Infrastructure

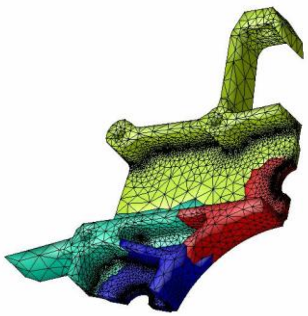
Key unstructured mesh technology needed by applications

- Effective parallel representation for adaptive mesh control and geometry interaction provided by PUMI and Omega_h
- Base parallel functions
  - Partitioned mesh control and modification
  - Read only copies for application needs
  - Associated data, grouping, etc.
- Attached fields supported

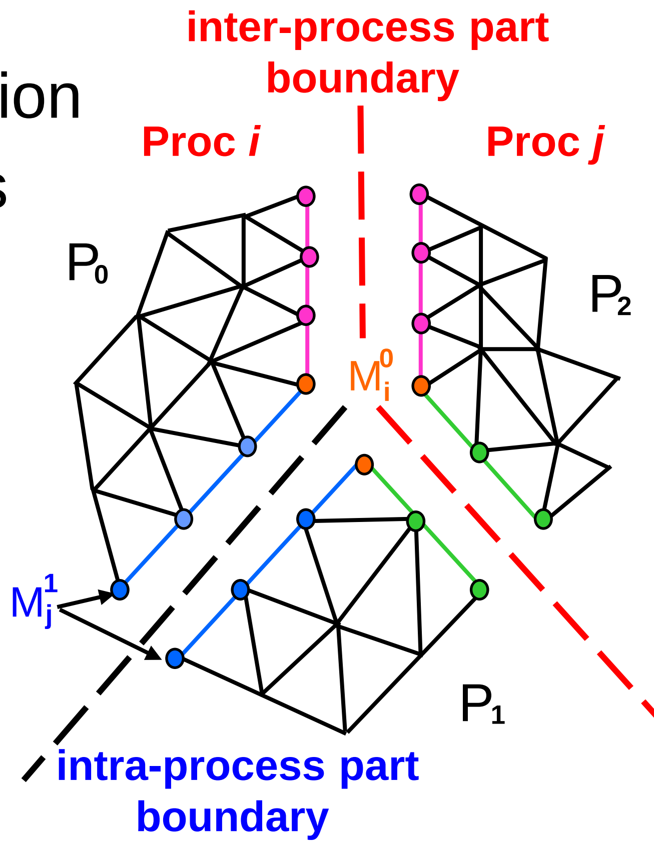


Geometric model

Partition model



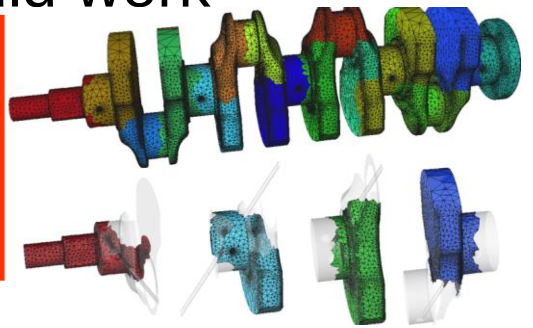
Distributed mesh



Mesh Generation, Adaptation and Optimization

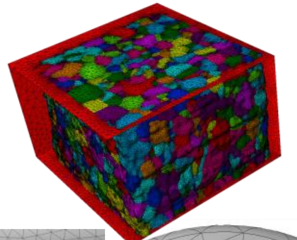
Mesh Generation

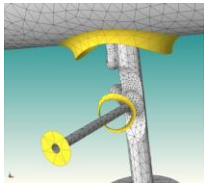
- Automatically mesh complex domains – should work directly from CAD, image data, etc.
- Use tools like Gmsh, Simmetrix, etc.

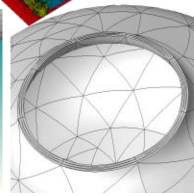


Mesh Adaptation must

- Use *a posteriori* information to improve mesh based on discretization errors or user supplied solution based criteria
- Account for curved geometry (fixed and evolving)
- Support general anisotropic adaptation
- Support some forms of mixed mesh adaptation



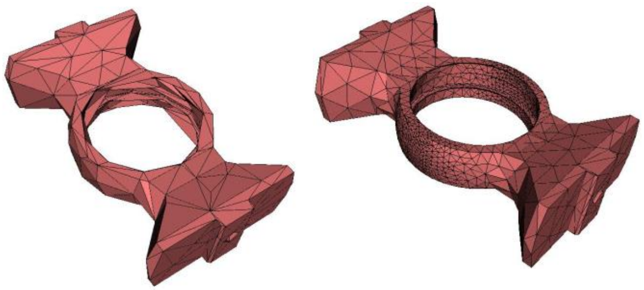


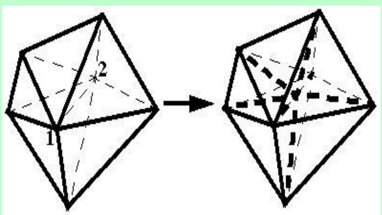


Parallel execution of all functions critical on large meshes

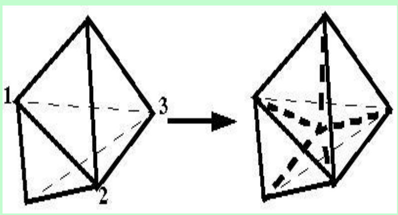
General Mesh Modification for Mesh Adaptation

- Driven by an anisotropic mesh size field that can be set by any combination of criteria
- Employ a “general set” of mesh modification operations to alter the mesh into one that matches the given mesh size field
- Advantages
  - Supports 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

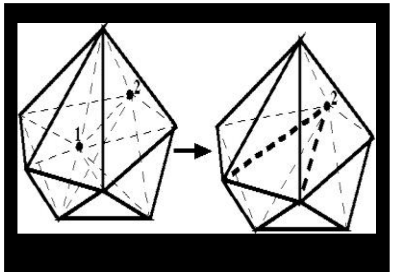


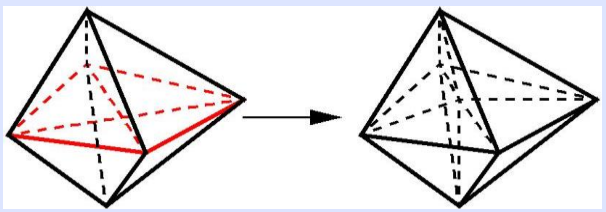


Edge split



face split

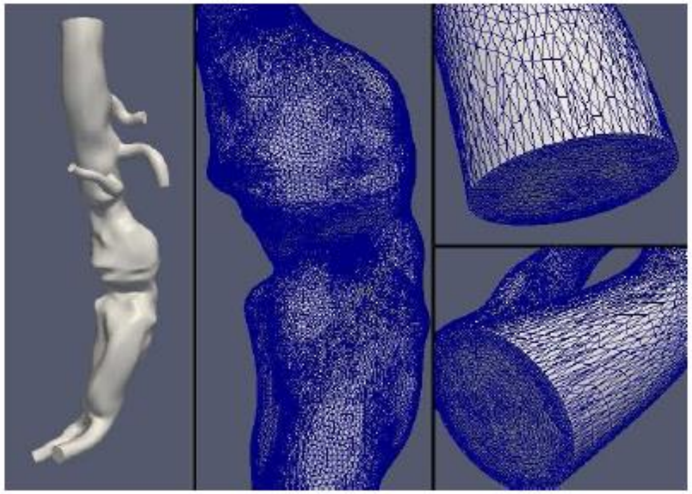


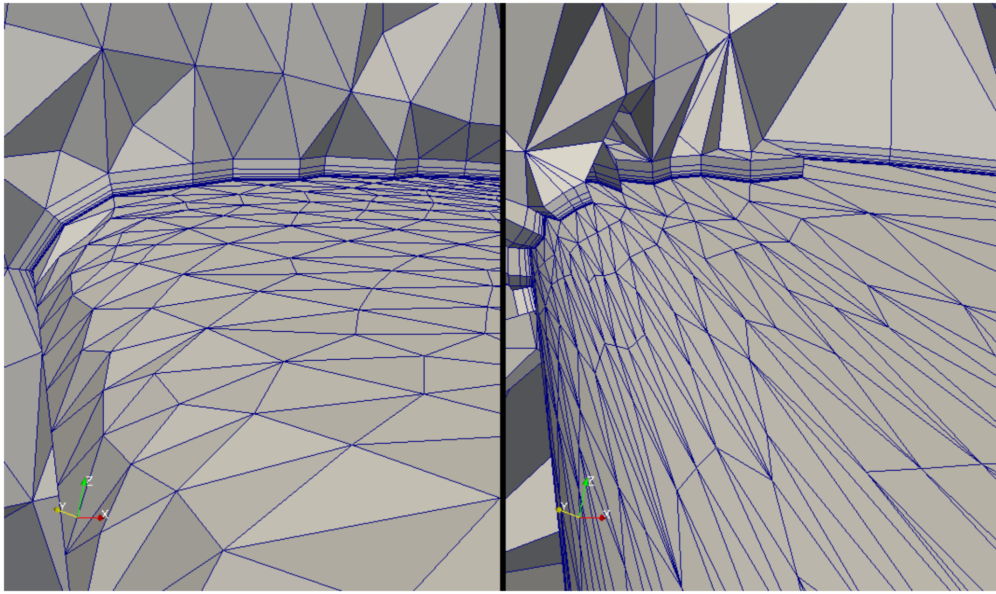


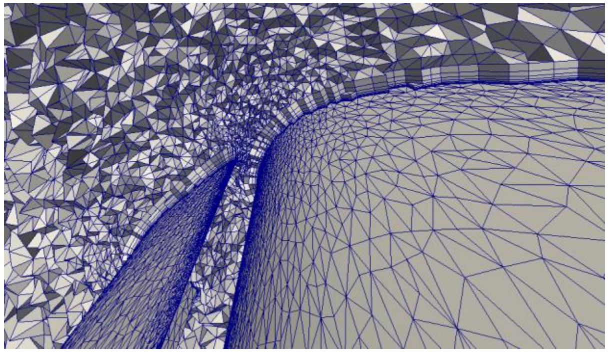
Double split collapse to remove sliver

Mesh Adaptation Status

- 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

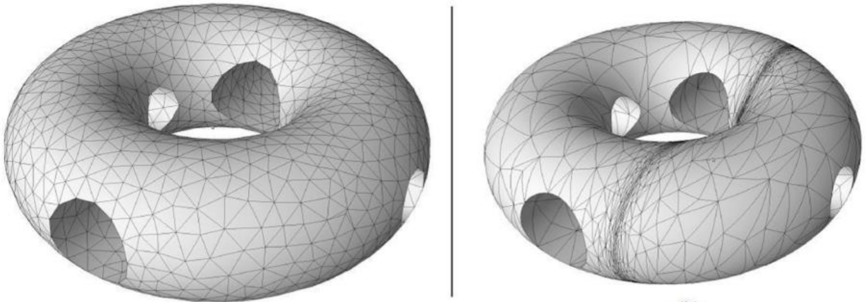




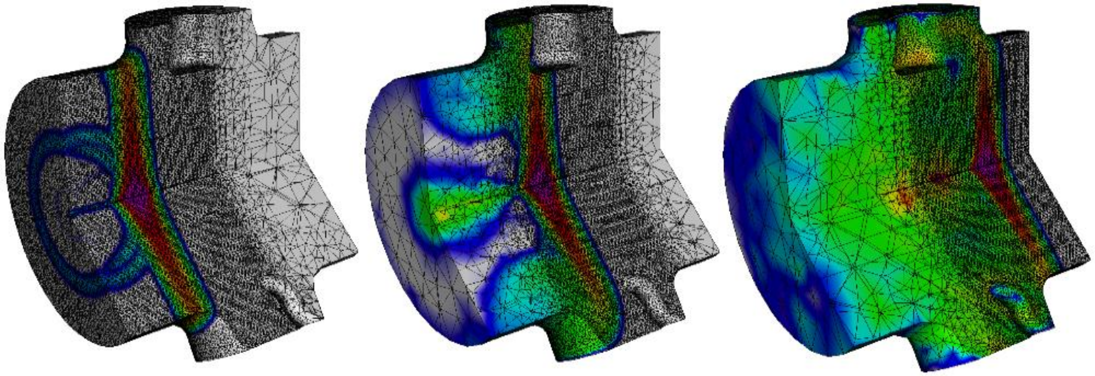


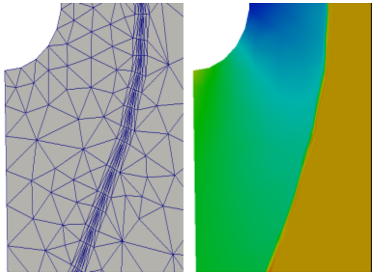
Mesh Adaptation Status

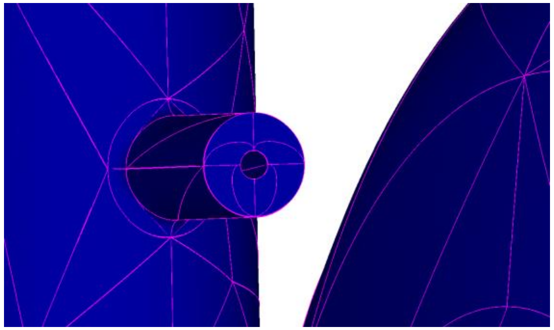
- Supports adaptation of curved elements
- Adaptation based on multiple criteria, examples
  - Level sets at interfaces
  - Tracking particles
  - Discretization errors
  - Controlling element shape in evolving geometry problems

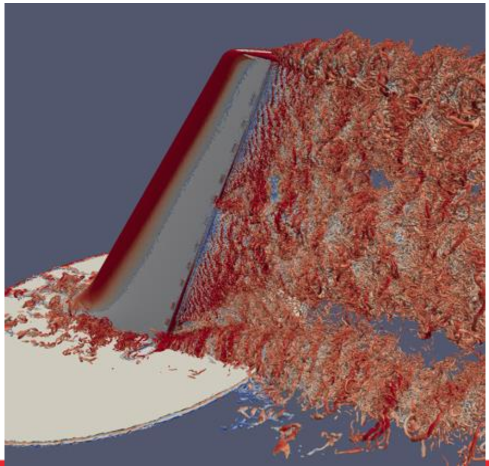


(a) (b)



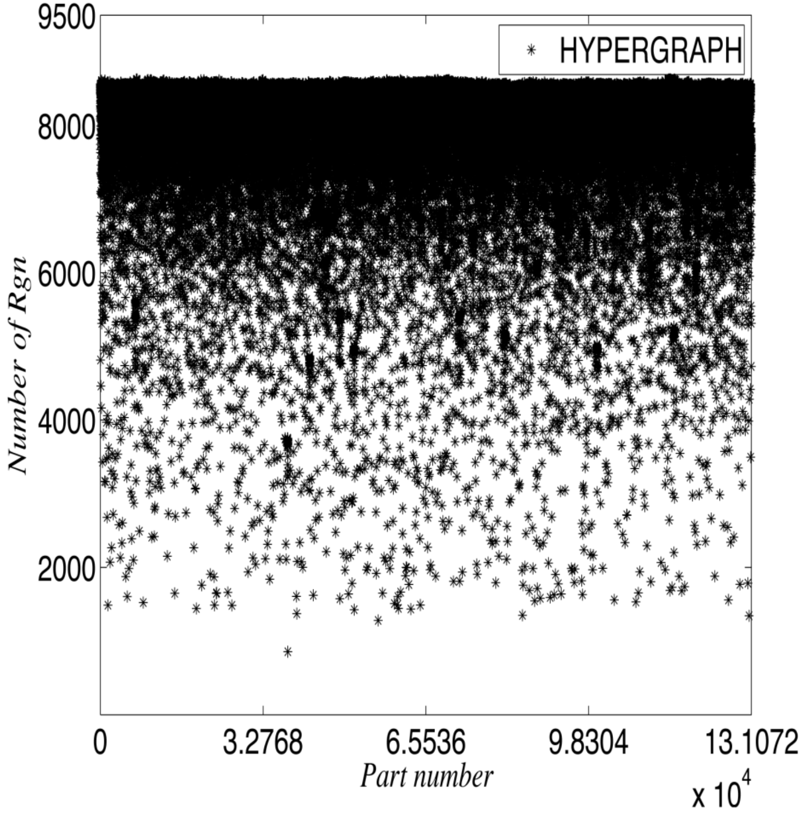






Dynamic Load Balancing

- Purpose: to rebalance load during an evolving simulation (mesh adaptation, particle moving through mesh, etc.)
  - Goal is equal “work load” with minimum inter-process communications
- FASTMath load balancing tools
  - Zoltan/Zoltan2 libraries that provide multiple dynamic partitioners with general control of partition objects and weights
  - EnGPar diffusive multi-criteria partition improvement



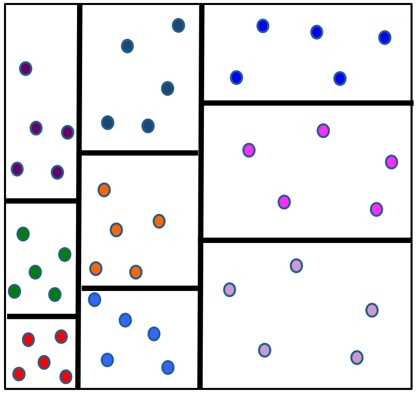
Architecture-aware partitioning and task mapping reduce application communication time at extreme scale

- *Partitioning and load balancing*: assign work to processes in ways that avoid process idle time and minimize communication
- *Task mapping*: assign processes to cores in ways that reduce messages distances and network congestion
- *Important in extreme-scale systems*:
  - Small load imbalances can waste many resources
  - Large-scale networks can cause messages to travel long routes and induce congestion
- *Challenge* to develop algorithms that…
  - account for underlying architectures & hierarchies
  - run effectively side-by-side with application across many platforms (multicore, GPU)

Zoltan/Zoltan2 Toolkits: Partitioners

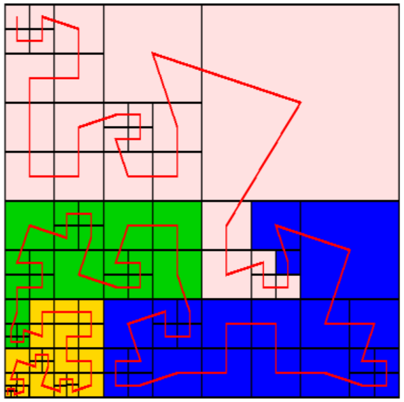
Suite of partitioners supports a wide range of applications; no single partitioner is best for all applications.

Geometric

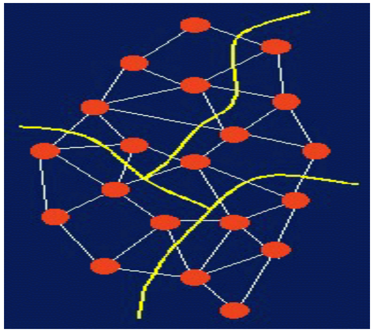


Recursive Coordinate Bisection
Recursive Inertial Bisection
Multi-Jagged Multi-section

Space Filling Curves

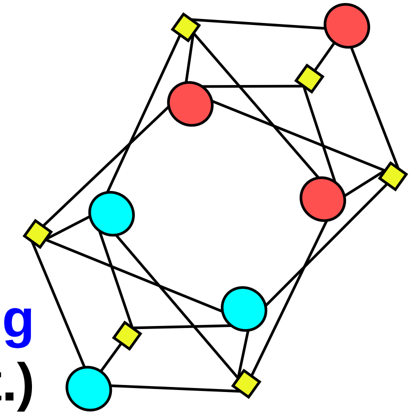


Topology-based



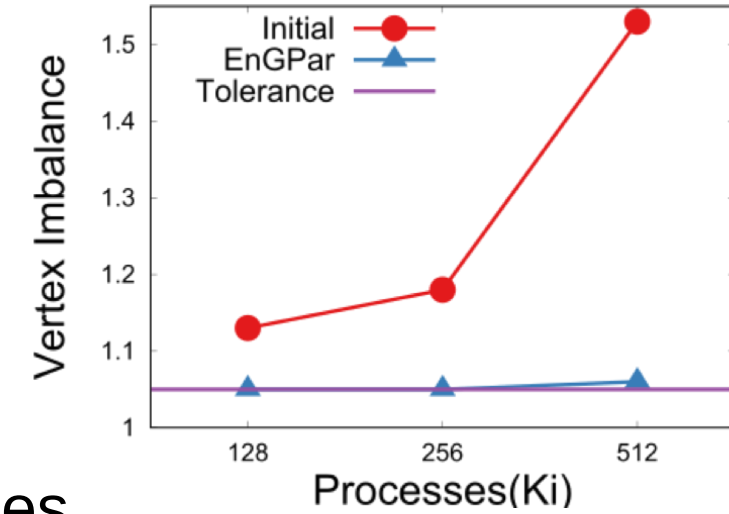
PHG Graph Partitioning
Interface to ParMETIS (U. Minnesota)
Interface to PT-Scotch (U. Bordeaux)

PHG Hypergraph Partitioning
Interface to PaToH (Ohio St.)



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

- Multi-(hyper)graph supports representing multiple types of dependencies between application work items
- Loop over application defined list of edge types
- Diffusion sends boundary edges from heavily loaded parts to lighter parts
  - Bias selection towards edges that are far from the graph ‘center’
  - Multiple traversals of boundary with increasing limit of edge degree
  - Receiver cancels send if it imbalances higher priority edge type

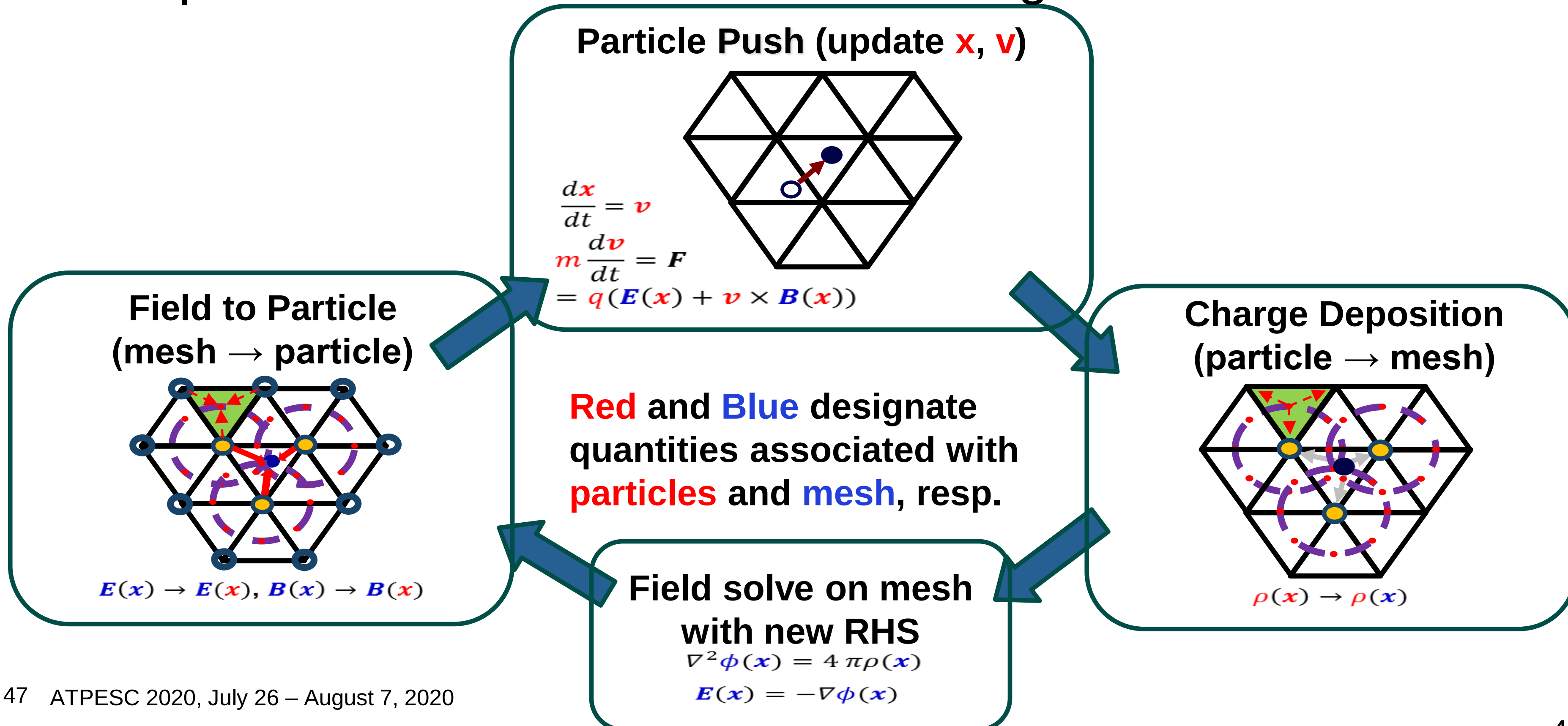


- On a 1.3B element mesh EnGPar reduced a 53% vtx imbalance to 6%, elm imbalance of 5%, edge cut increase by 1% (took 8 seconds)

Parallel Unstructured Mesh PIC – PUMIpic

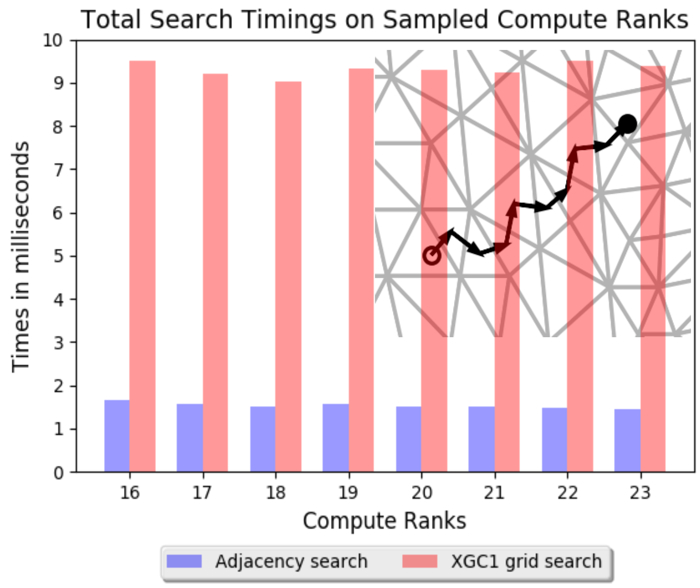
Current approaches have copy of entire mesh on each process

PUMIpic supports a distributed mesh
- Employ large overlaps to avoid communication during push
- All particle information accessed through the mesh



Parallel Unstructured Mesh PIC – PUMIpic

- Components interacting with mesh
  - Mesh distribution
  - Particle migration
  - Adjacency search
  - Charge-to-mesh mapping
  - Field-to-Particle mapping
  - Dynamic load balancing
  - Continuum solve
- Builds on parallel unstructured mesh infrastructure
- Developing set of components to be integrated into applications
  - XGC – Gyrokinetic Code
  - GITR - Impurity Transport



Require knowledge of element that particle is in after push

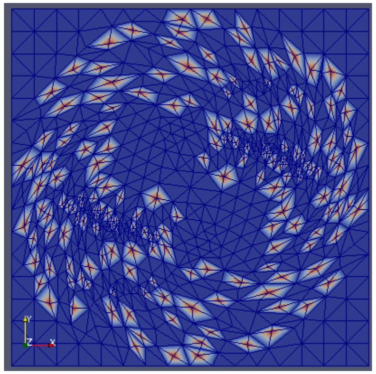
- Particle motion “small” per time step
- Using mesh adjacencies on distributed mesh
- Overall 4 times improvement

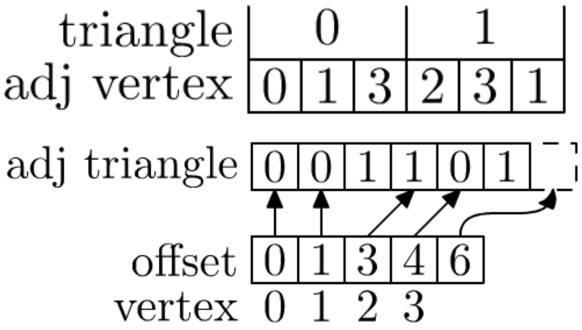
PUMIpic Data Structures

- The layout of particles in memory is critical for high performance push, scatter, and gather operations on GPUs.
- Mesh data structure requirements:
  - Provide required adjacency information on GPU
  - Reduce irregular memory accesses by building arrays of mesh field information needed for particles.
- Particle data structure requirements:
  - Optimizes push, scatter, and gather operations
  - Associates particles with mesh elements
  - Changes in the number of particles per element
  - Evenly distributes work with a range of particle distributions (e.g. uniform, Gaussian, exponential, etc.)
  - Stores a lot of particles per GPU – low overhead

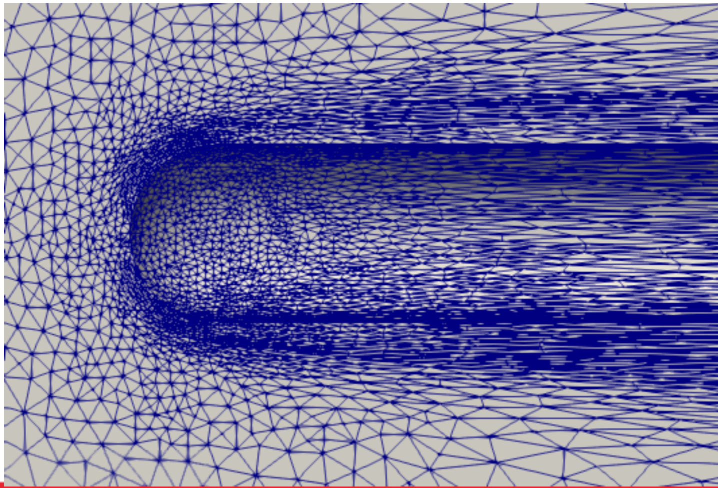
Mesh Data – Omega_h

- Omega_h features
  - Compact arrays ordered so that adjacent entities are aligned
  - BFS-like algorithms for effective local serial
  - Space filling curves to support parallelization
  - Independent set construction (currently for mesh adaptation)
  - On-node OpenMP or CUDA parallelism using Kokkos



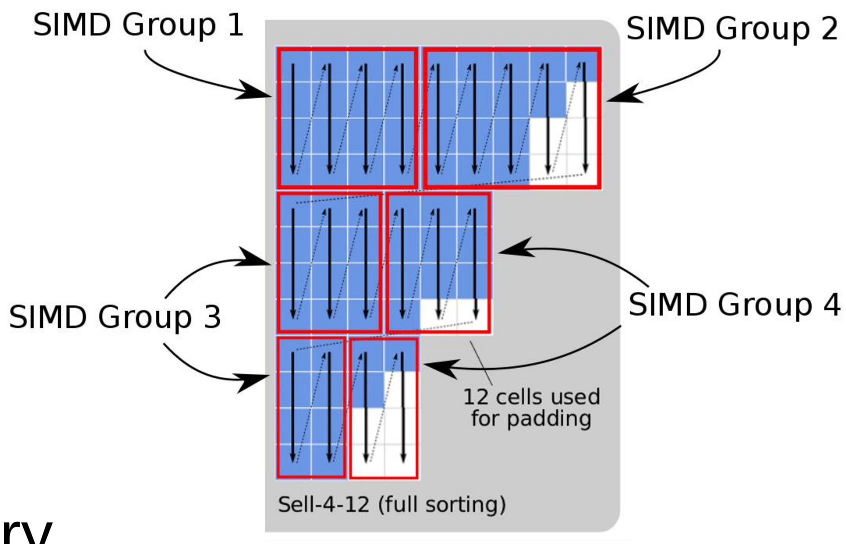


github.com/ibaned/omega_h



Particle Data Structures (cont.)

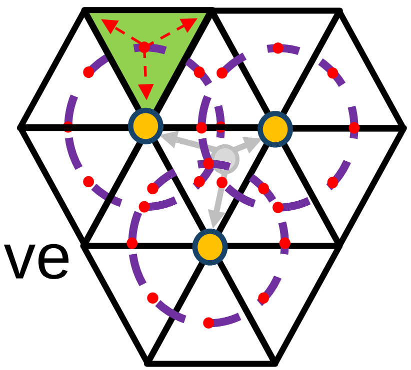
- Particles associated with elements in mesh
- Sell-C-σ (SCS) structure selected
  - Layout: rotated and sorted CSR, a row has the particles of an element
  - Pros – Fast push, lower memory usage for scatter/gather
  - Cons – Complexity
- Demonstrated good strong scaling for required PIC operations of 4096 nodes (24567 GPU’s) on Summit

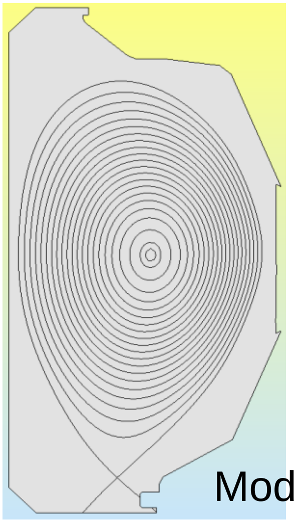


SCS with vertical slicing (bottom)
Besta, Marending, Hoefler, IPDPS 2017

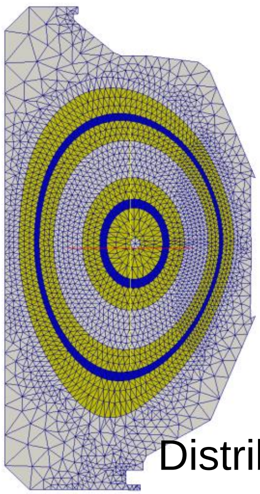
PUMIpic for XGC Gyrokinetic Code

- XGC uses a 2D poloidal plane mesh considering particle paths
  - Mesh distribution takes advantage of physics defined model/mesh
  - Separate parallel field solve on each poloidal plane
- XGC gyro-averaging for Charge-to-Mesh
- PETSc used for field solve
  - Solves on each plane
  - Mesh partitioned over N_{ranks}/N_{planes} ranks
  - Ranks for a given plane form MPI sub-communicators

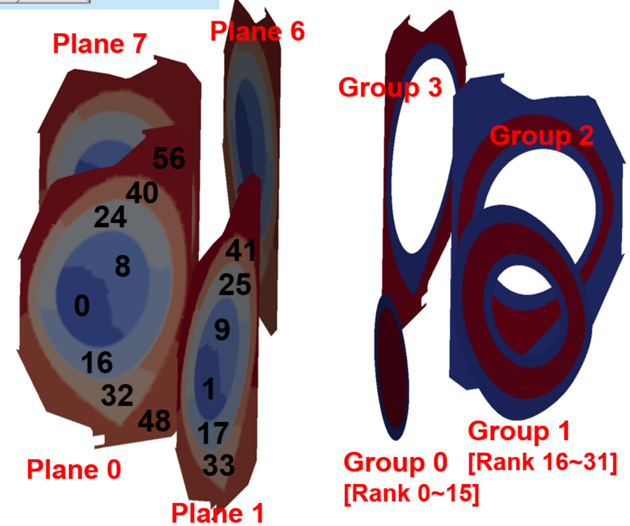




Model



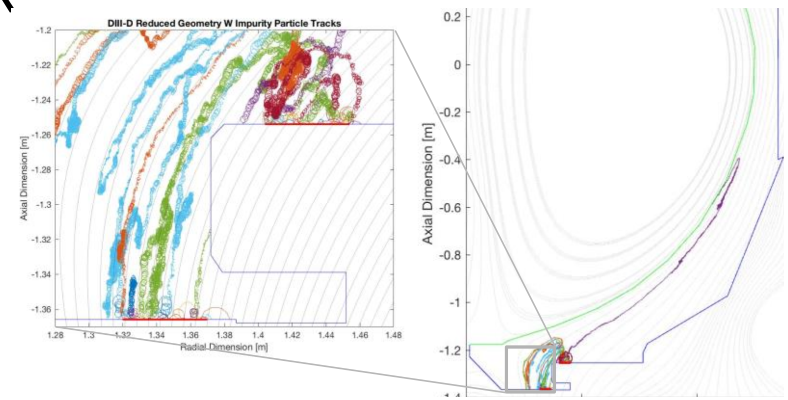
Mesh Distribution



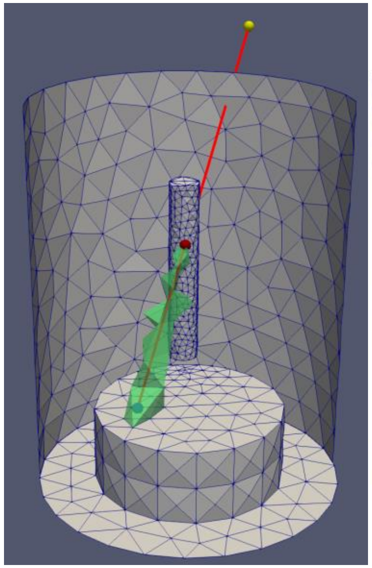
Two-level partition for solver (left) and particle push (right)

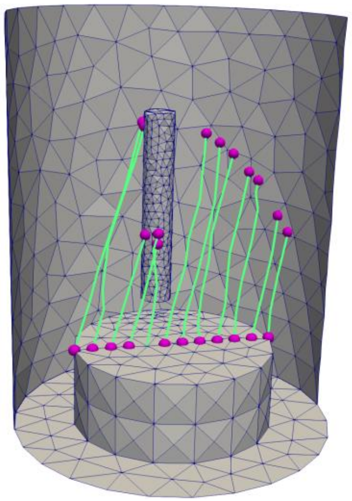
Impurity Transport Code - GITR

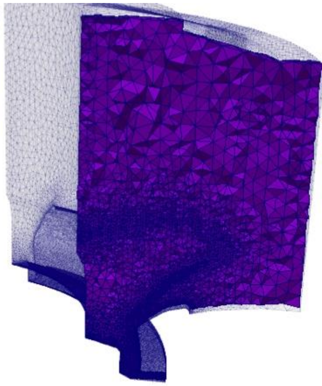
- PUMIpic capabilities needed for GITR
  - Fully 3D graded/adapted meshes based on particle distribution
  - Wall interactions
  - Plan on supporting the future case where the fields evolve based on particle position
- Development of 3D mesh version of GITR initiated
  - Based on PUMIpic
  - Efforts focused on GPU based on-node operations
  - Complete version available, performance improvement underway



Particle traces from original GITR





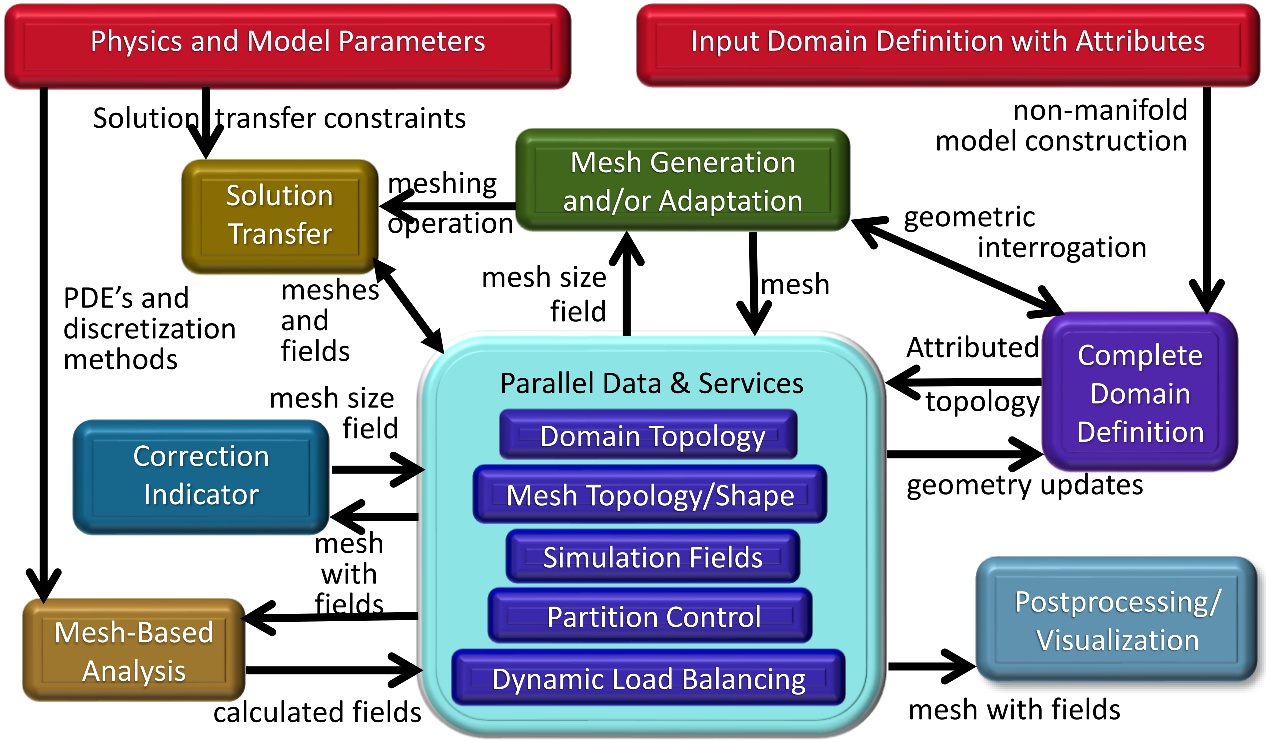


Possible graded mesh

Creation of Parallel Adaptive Loops

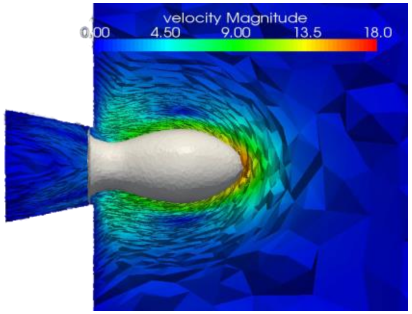
Parallel data and services are at the core

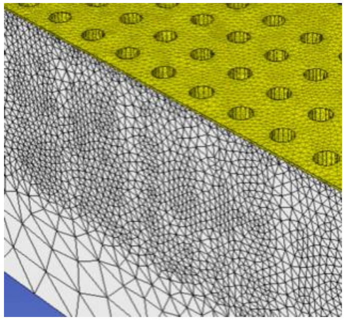
- Geometric model topology for domain linkage
- Mesh topology – it must be distributed
- Simulation fields distributed over geometric model and mesh
- Partition control
- Dynamic load balancing required at multiple steps
- API's to link to
  - CAD
  - Mesh generation and adaptation
  - Error estimation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

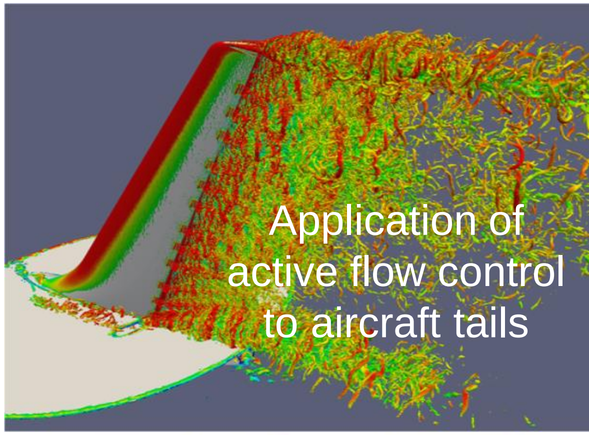


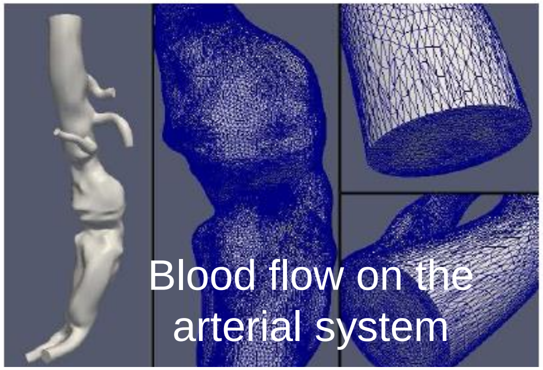
Parallel Adaptive Simulation Workflows

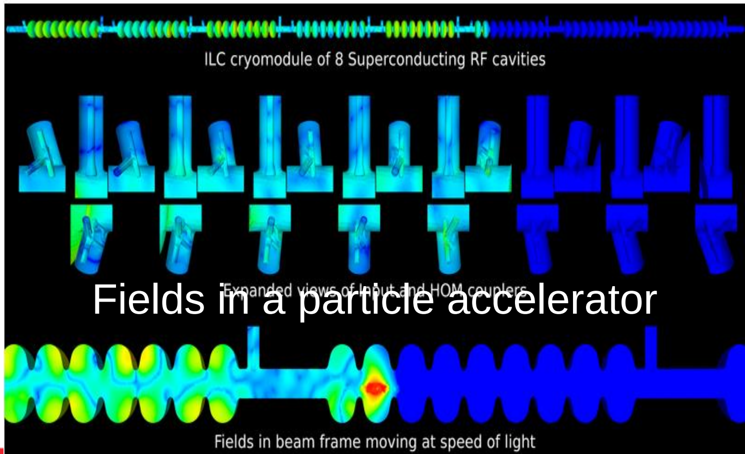
- Automation and adaptive methods critical to reliable simulations
- In-memory examples
  - MFEM – High order FE framework
  - PHASTA – FE for NS
  - FUN3D – FV CFD
  - Proteus – multiphase FE
  - Albany – FE framework
  - ACE3P – High order FE electromagnetics
  - M3D-C1 – FE based MHD
  - Nektar++ – High order FE flow





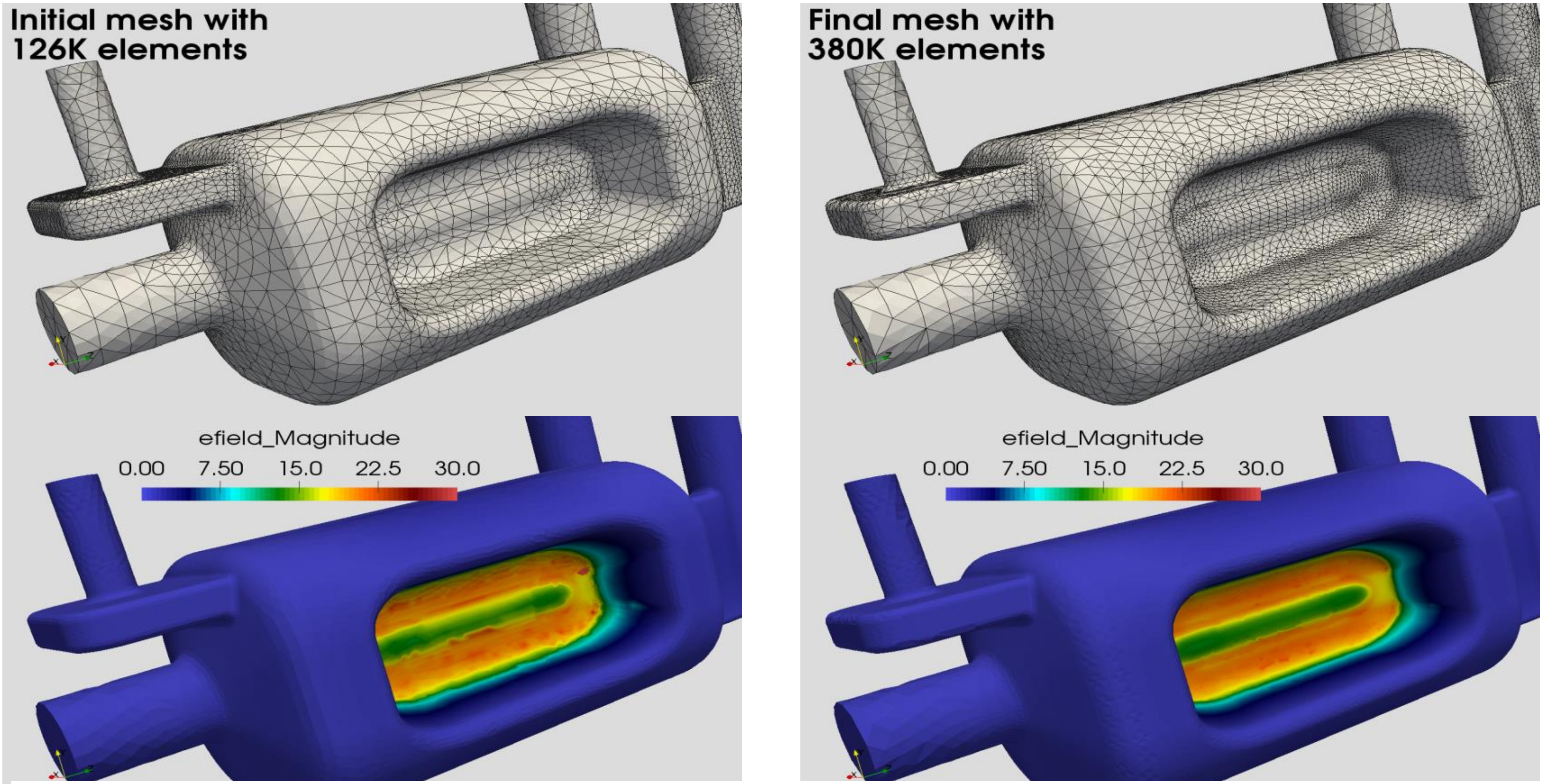






Application interactions – Accelerator EM

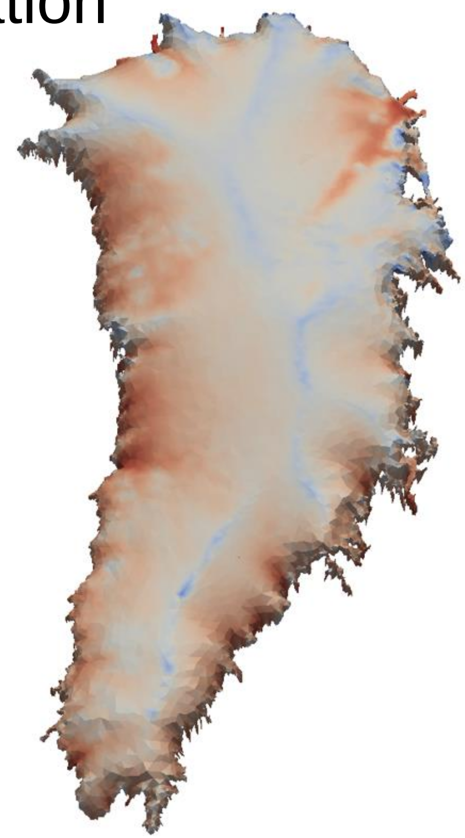
Omega3P Electro Magnetic Solver (second-order curved meshes)

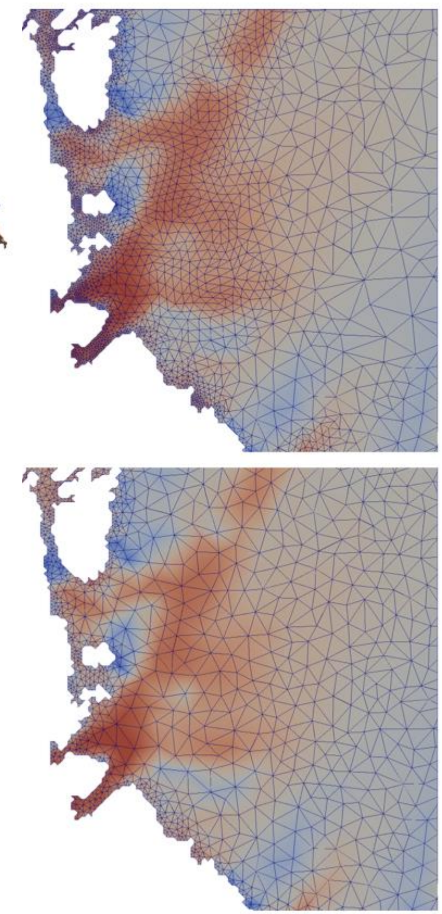


This figure shows the adaptation results for the CAV17 model. (top left) shows the initial mesh with ~126K elements, (top right) shows the final (after 3 adaptation levels) mesh with ~380K elements, (bottom left) shows the first eigenmode for the electric field on the initial mesh, and (bottom right)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

Application interactions – Land Ice

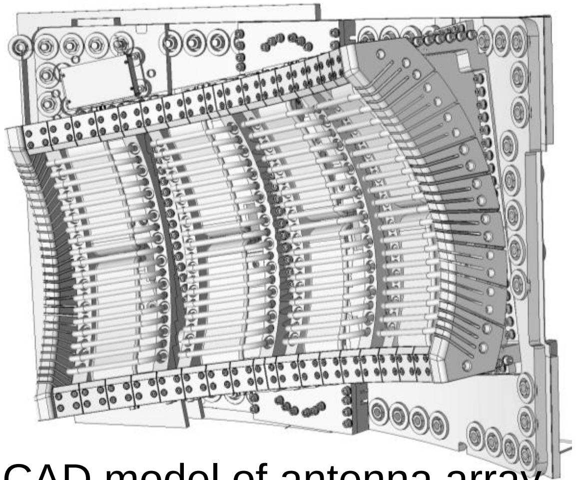
- FELIX, a component of the Albany framework is the analysis code
- Omega_h parallel mesh adaptation is integrated with Albany to do:
  - Estimate error
  - Adapt the mesh
- Ice sheet mesh is modified to minimize degrees of freedom
- Field of interest is the ice sheet velocity





Application interactions – RF Fusion

- Accurate RF simulations require
  - Detailed antenna CAD geometry
  - CAD geometry defeaturing
  - Extracted physics curves from EFIT
  - Faceted surface from coupled mesh
  - Analysis geometry combining CAD, physics geometry and faceted surface
  - Well controlled 3D meshes for accurate FE calculations in MFEM
  - Conforming mesh adaptation with PUMI



CAD model of antenna array