Unstructured Mesh Technologies

Presented to
ATPESC 2017 Participants

Tzanio Kolev (LLNL) & Mark Shephard (RPI)

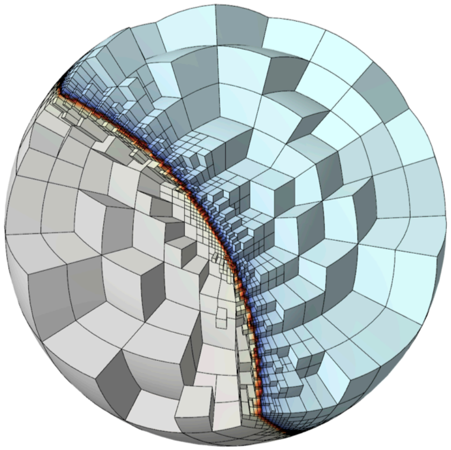
Q Center, St. Charles, IL (USA)
Date 08/07/2017

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

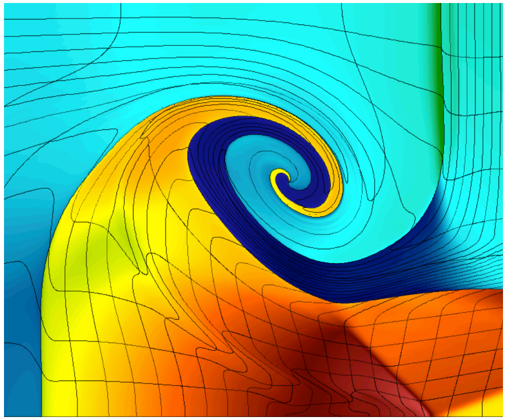
- Backed by well-developed theory.
- Naturally support unstructured and curvilinear grids.
- **High-order finite elements on high-order meshes**
  - Increased accuracy for smooth problems
  - Sub-element modeling for problems with shocks
  - Bridge unstructured/structured grids
  - Bridge sparse/dense 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

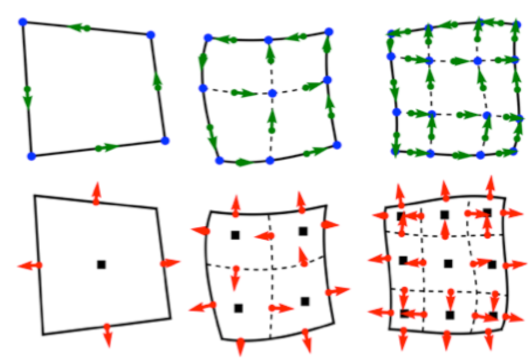


8th order Lagrangian hydro simulation of a shock triple-point interaction

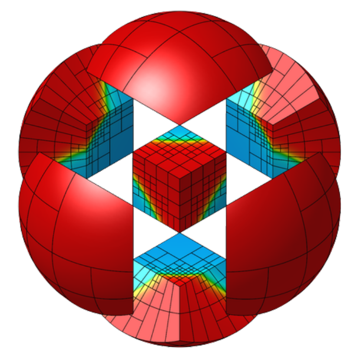
Modular Finite Element Methods (MFEM)

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, STRUMPACK, PUMI (in progress), …*
- 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
(v3.3, Jan/2017)

CEED
EXASCALE DISCRETIZATIONS

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

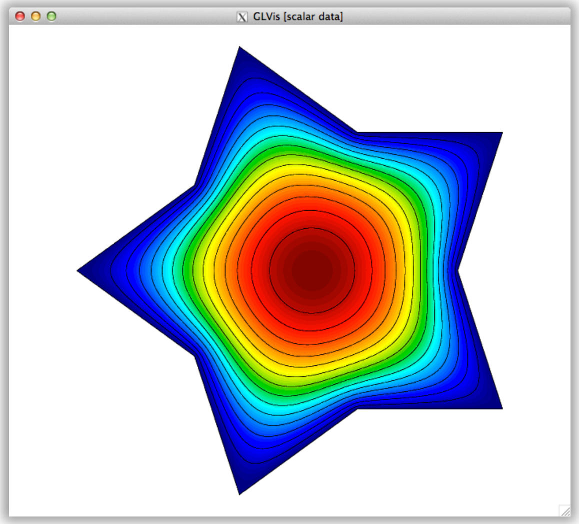
```
   // 5. Set up the linear form b(.) which corresponds to the right-hand side of
   //    the FEM linear system, which in this case is (1,phi_i) where phi_i are
   //    the basis functions in the finite element fespace.
   LinearForm *b = new LinearForm(fespace);
   ConstantCoefficient one(1.0);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

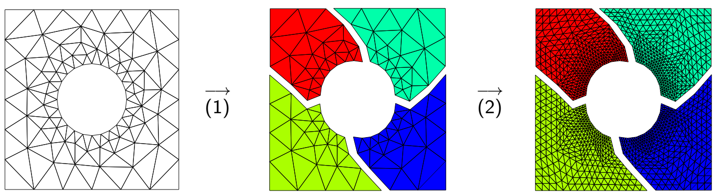
- Visualization

```
   // 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

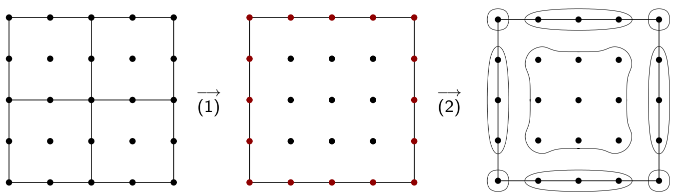
- Parallel mesh

```
// 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();
```

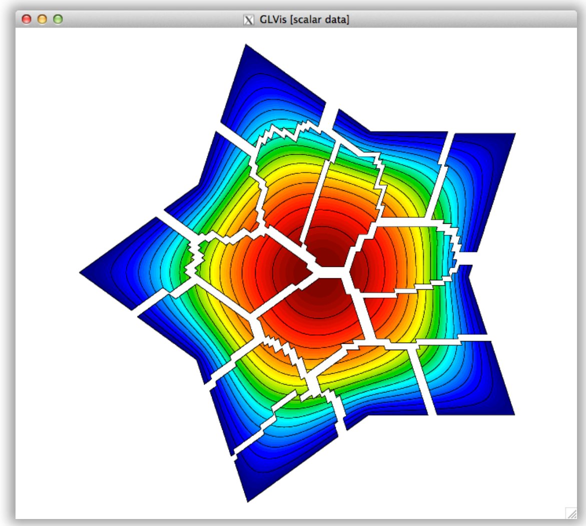
$$A = P^T a P \qquad B = P^T b \qquad x = PX$$

- 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);
```

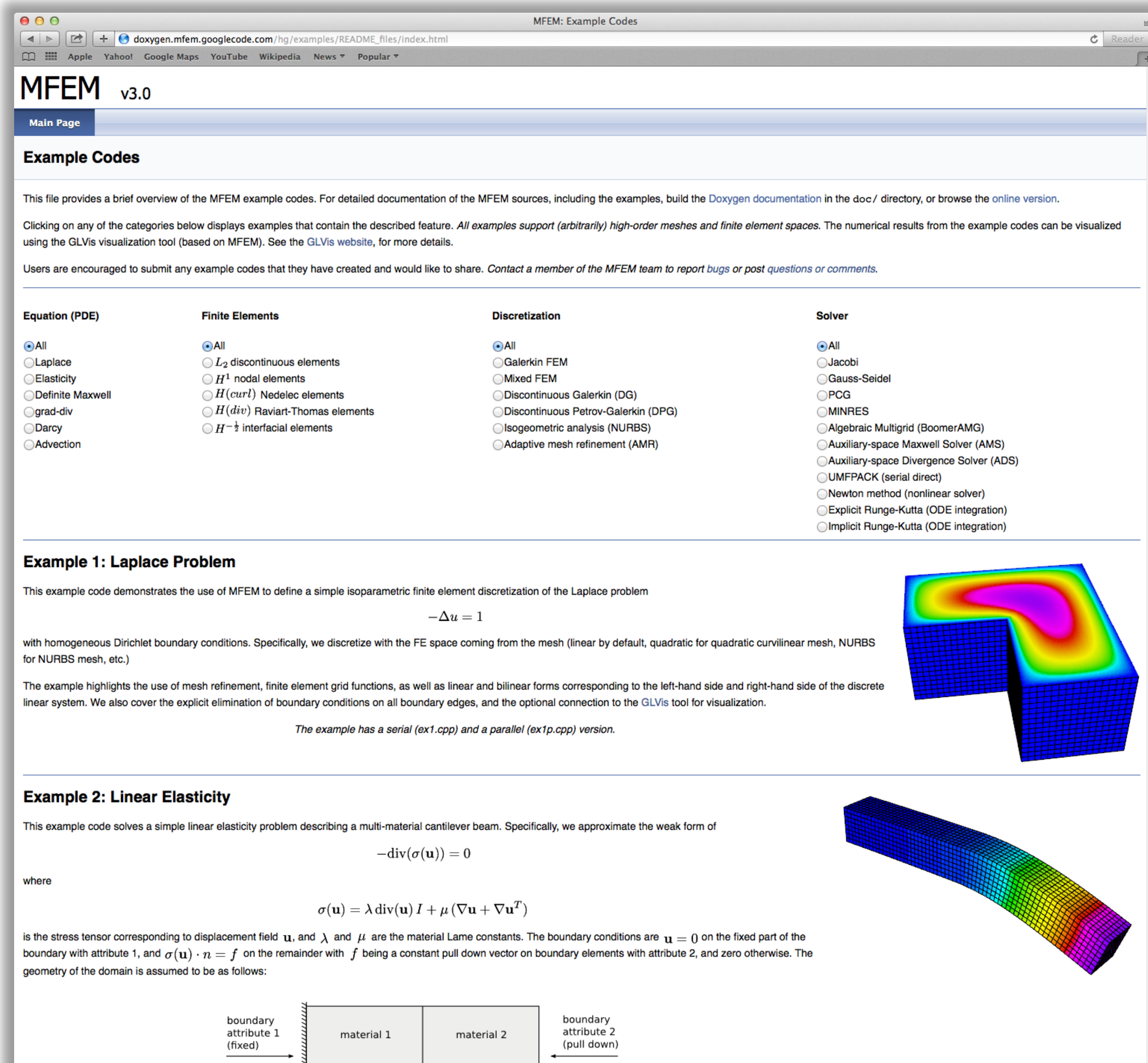
```
// 10. Define the parallel (hypre) matrix and vectors representing a(.,.),
//     b(.) and the finite element approximation.
HypreParMatrix *A = a->ParallelAssemble();
HypreParVector *B = b->ParallelAssemble();
HypreParVector *X = x.ParallelAverage();
```

```
// 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 – demo@6:30pm

http://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:

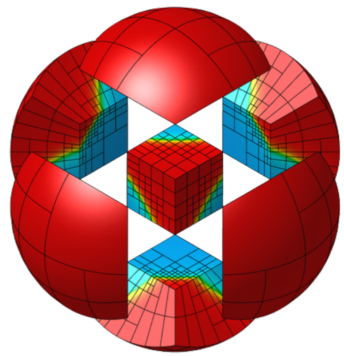
boundary attribute 1 (fixed) | material 1 | material 2 | boundary attribute 2 (pull down)

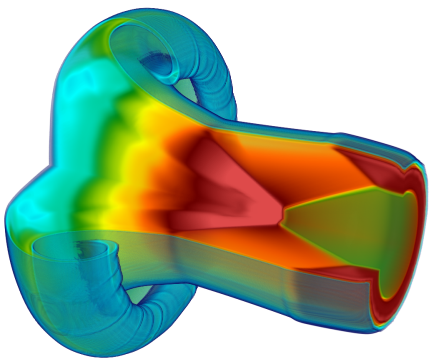
Application to High-order ALE shock hydrodynamics

hypre: *Scalable linear solvers library*

MFEM: *Modular finite element methods library*

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





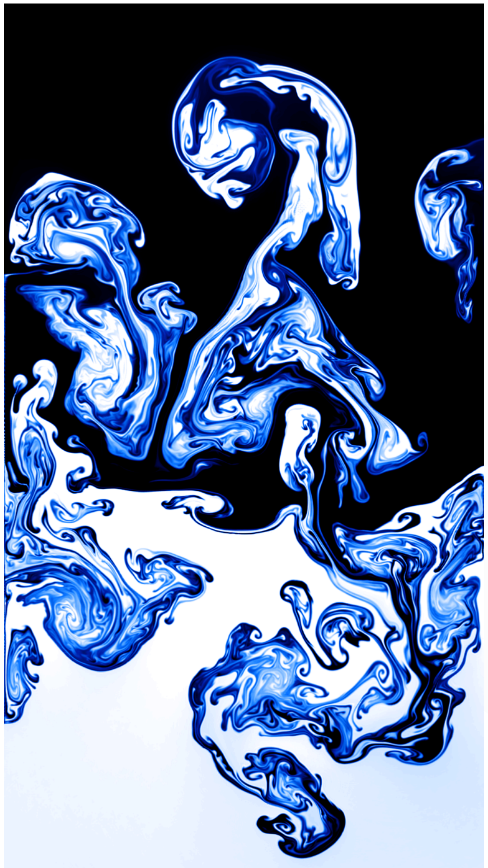
www.llnl.gov/casc/hypre

mfem.org

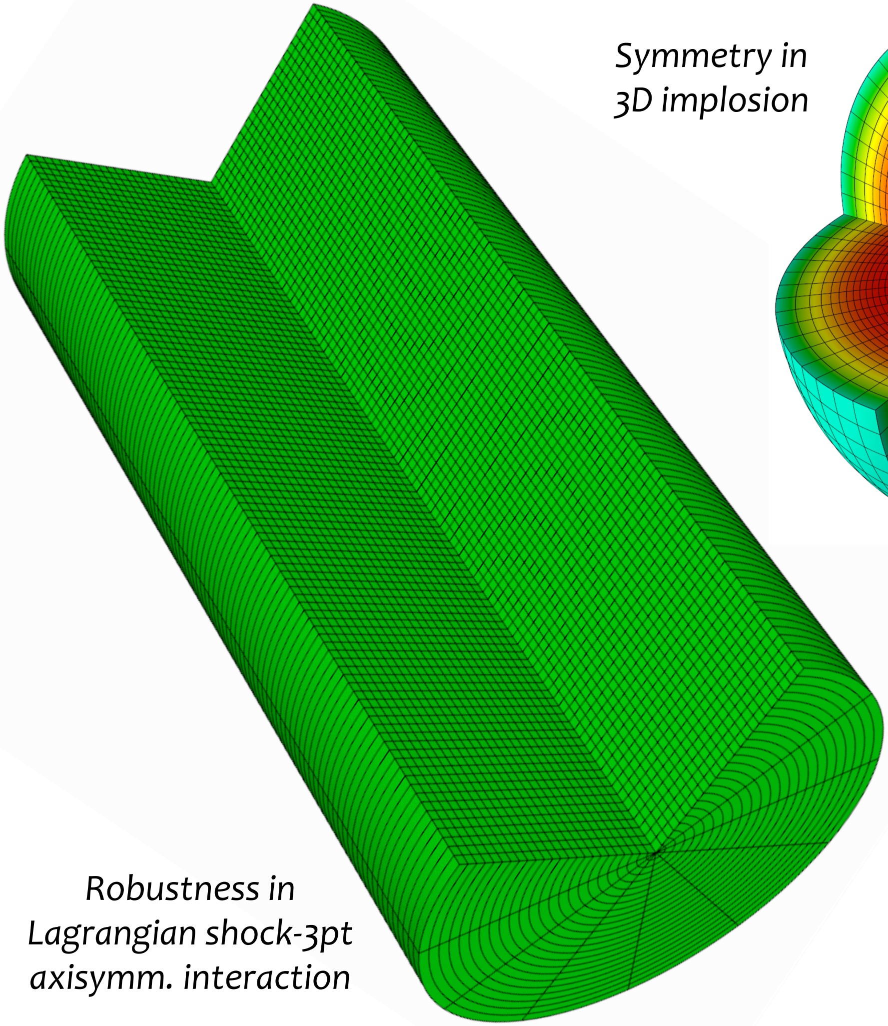
www.llnl.gov/casc/blast

- *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

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

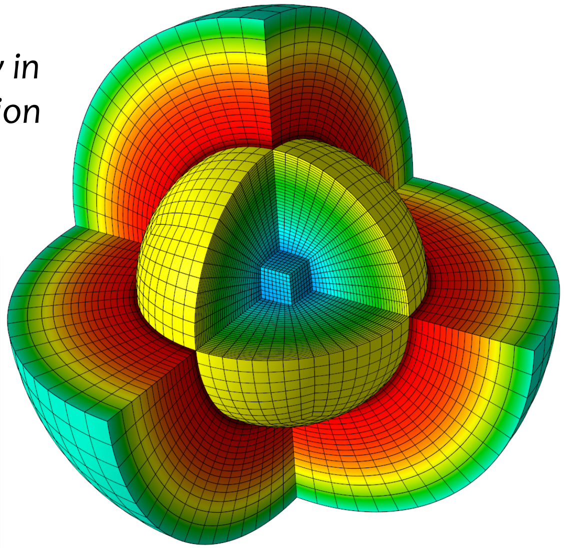


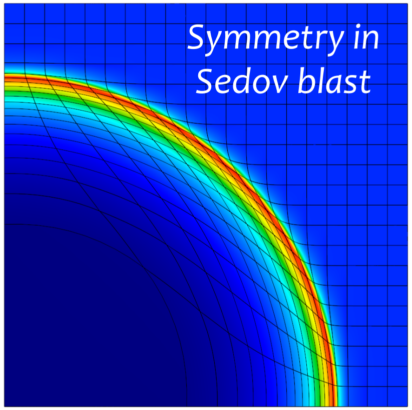
Parallel ALE for Q4 Rayleigh-Taylor instability (256 cores)



Robustness in Lagrangian shock-3pt axisymm. interaction

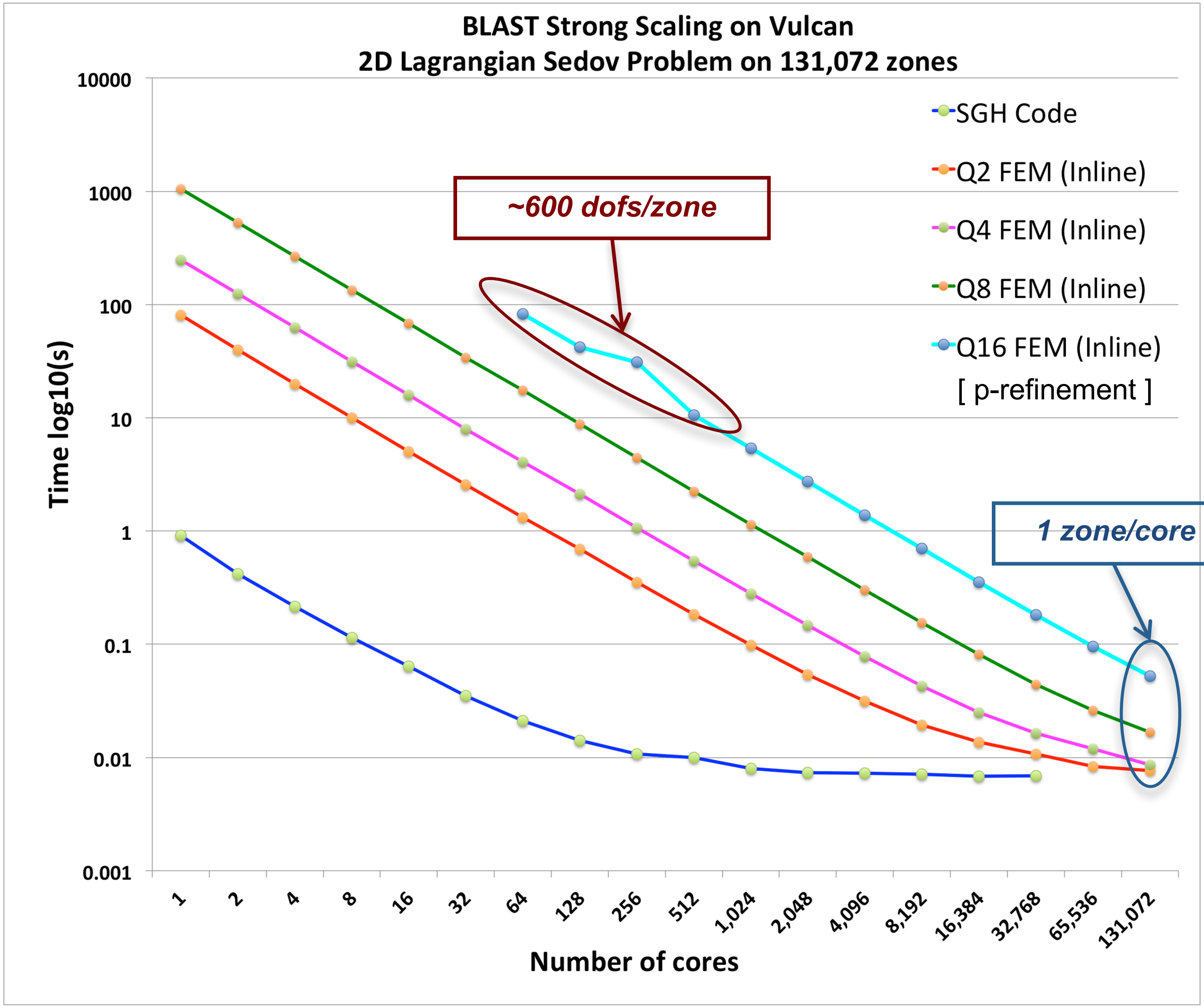
Symmetry in 3D implosion





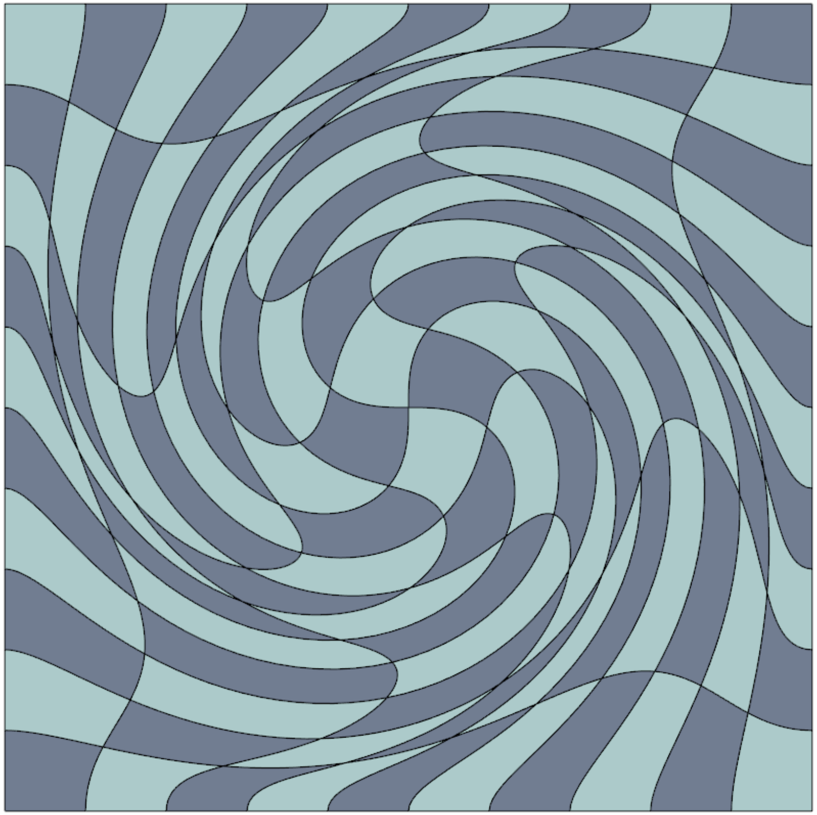
Symmetry in Sedov blast

High-order finite elements have excellent strong scalability

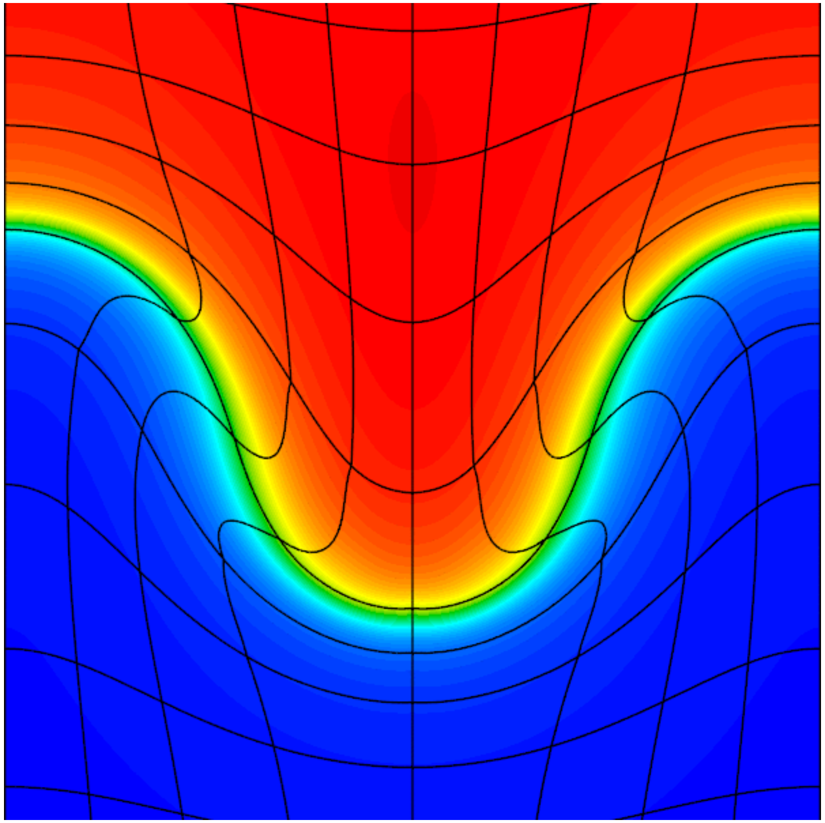


Unstructured Mesh R&D: Mesh optimization and high-quality interpolation between meshes

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



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

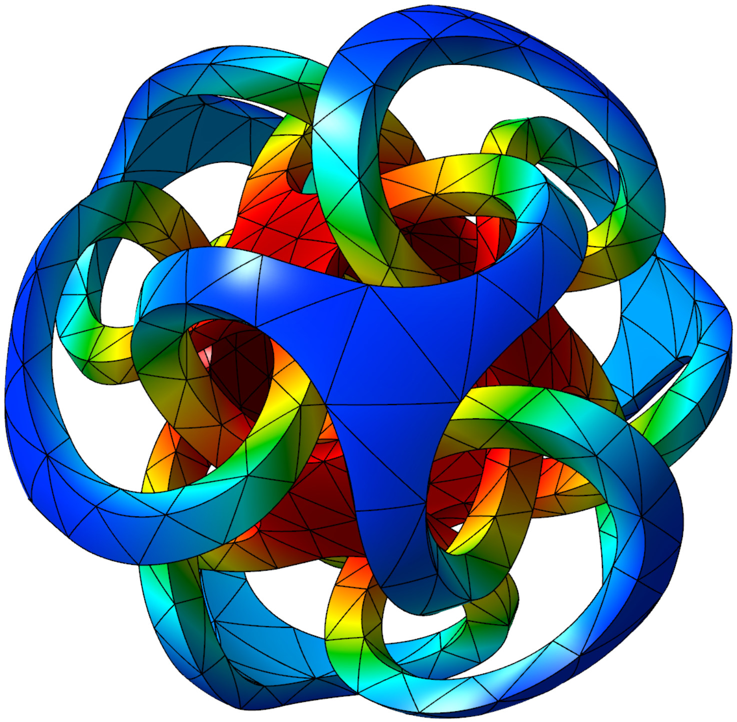


DG advection-based interpolation (ALE remap, Example 9, radiation transport)

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

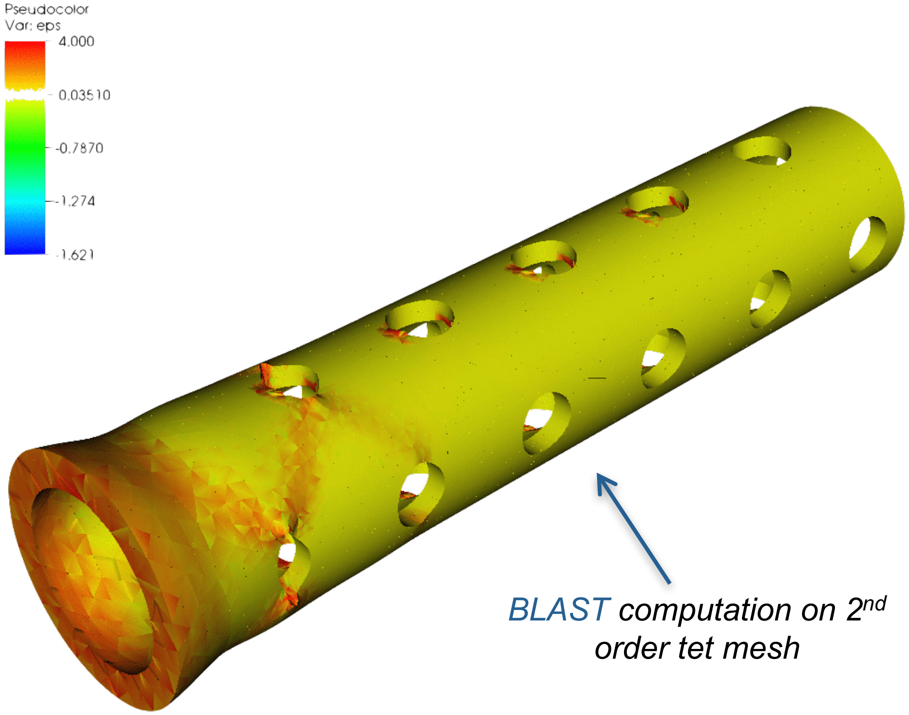
Two visualization options for high-order functions on high-order meshes

GLVis: *native MFEM lightweight OpenGL visualization tool*



glvis.org

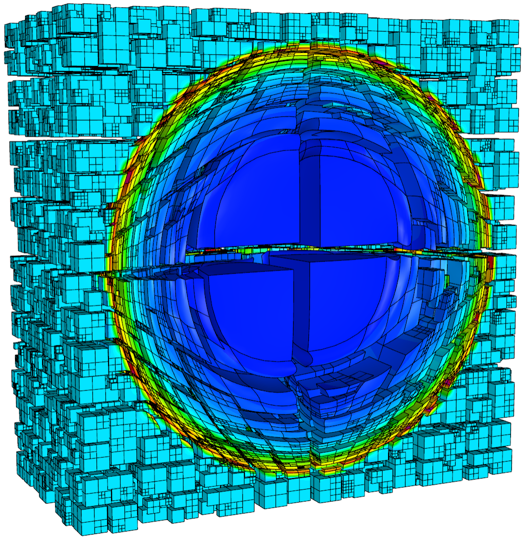
VisIt: *general data analysis tool, MFEM support since version 2.9*



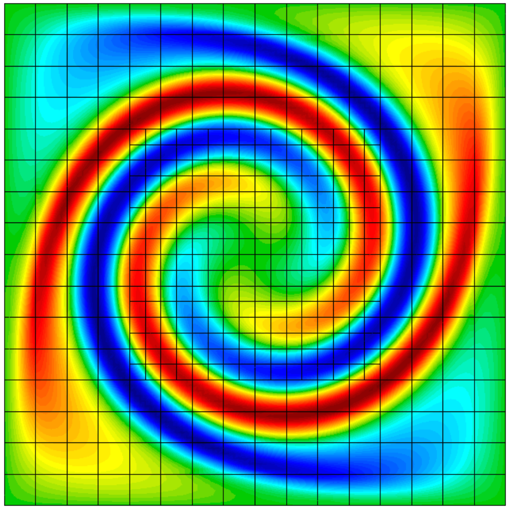
visit.llnl.gov

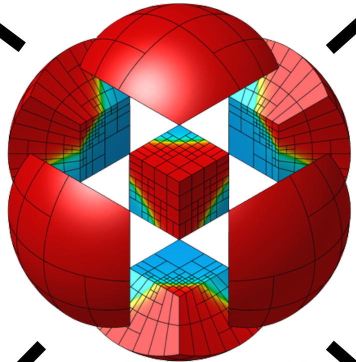
Unstructured Mesh R&D: Library-based AMR algorithms that can be applied to a variety of physics (6:30pm talk)

HO ALE

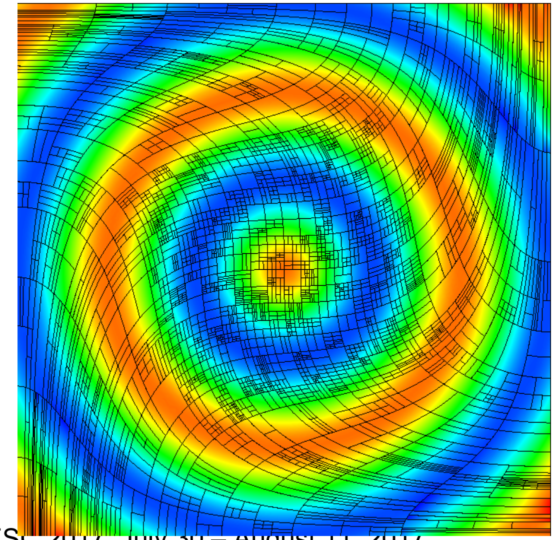


DG advection/ HO transport

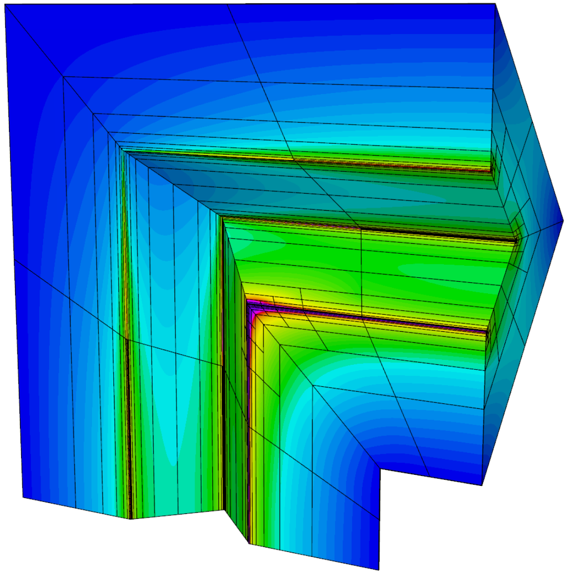




HO flux-based radiation-diffusion



HO MHD

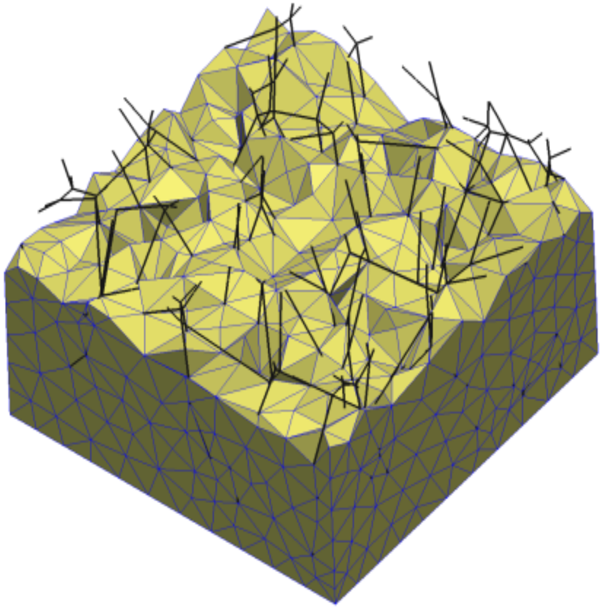


Unstructured Mesh Methods

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

Advantages of unstructured mesh methods

- Fully automated procedures to go from CAD to valid mesh
- Can provide highly effective solutions
  - Easily fitted to geometric features
  - General mesh anisotropy to account for anisotropic physics possible
- Given a complete geometry, with analysis attributes defined on that model, the entire simulation work flow can be automated
- Meshes can easily be adaptively modified



Unstructured Mesh Methods

Disadvantages of unstructured meshes

- More complex data structures than structured meshes
  - Increased program complexity, particularly in parallel
- Can provide the highest accuracy on a per degree of freedom – requires careful method and mesh control
  - The quality of element shapes influences solution accuracy – the degree to which this happens a function of the discretization method
  - Poorly shaped elements increase condition number of global system – iterative solvers increase time to solve
  - Require careful *a priori*, and/or good *a posteriori*, mesh control to obtain good mesh configurations

Unstructured Mesh Methods

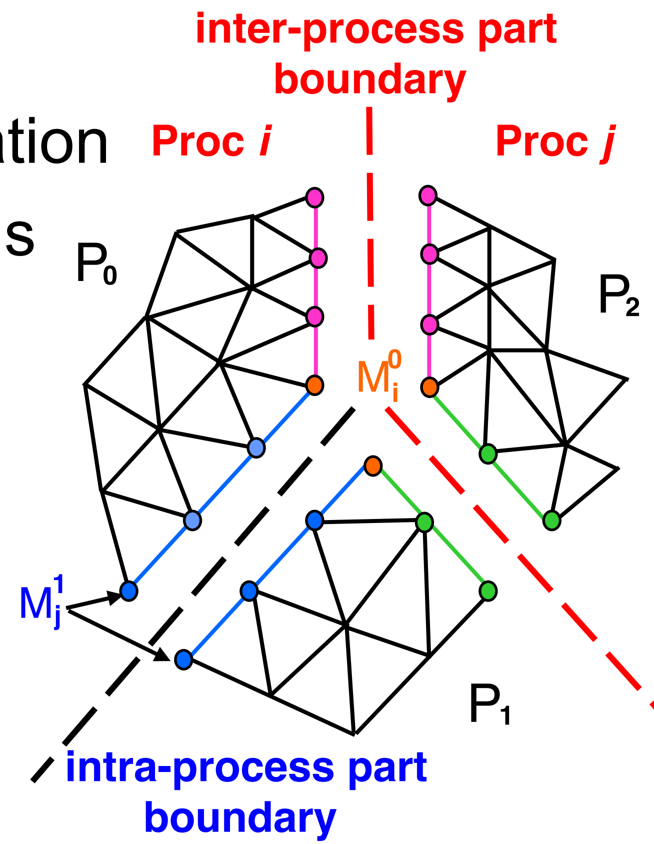
Goal of FASTMath unstructured mesh developments include:

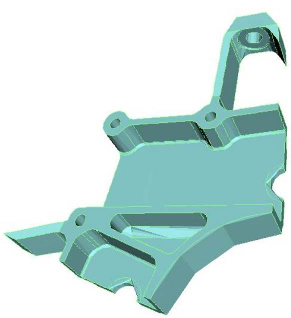
- Provide component-based tools that take full advantage of unstructured mesh methods and are easily used by analysis code developers and users
- Develop those components to operate through multi-level APIs that increase interoperability and ease of integration
- Address technical gaps by developing specific unstructured mesh tools to address needs and eliminate/minimize disadvantages of unstructured meshes
- Work with DOE applications on the integration of these technologies with their tools and to address new needs that arise

Parallel Unstructured Mesh Infrastructure

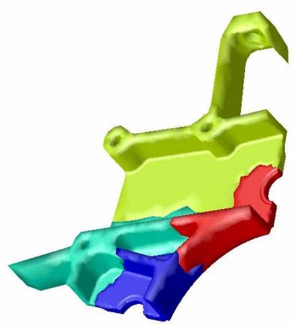
Key unstructured mesh technology needed by applications

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

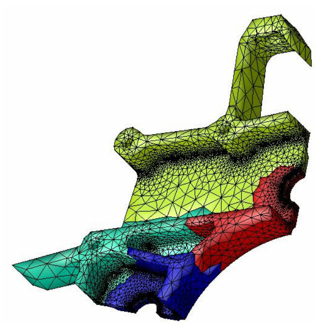




Geometric model



Partition model

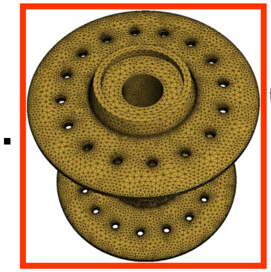


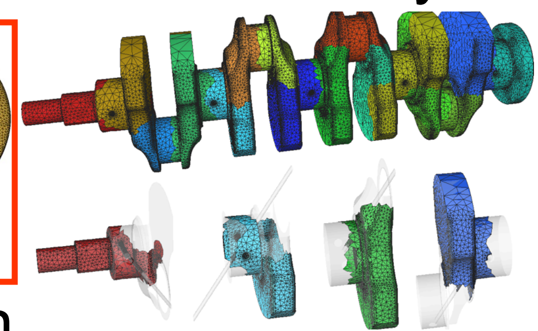
Distributed mesh

Mesh Generation, Adaptation and Optimization

Mesh Generation

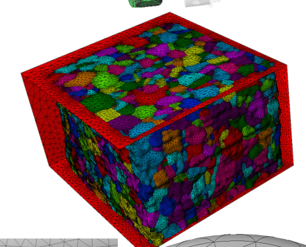
- Automatically mesh complex domains – should work directly from CAD, image data, etc.
- Use tools like Gmsh, Simmetrix, etc.





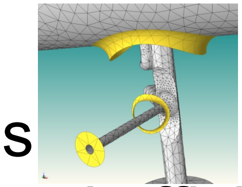
Mesh Adaptation must

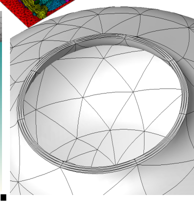
- Use *a posteriori* information to improve mesh
- Account for curved geometry (fixed and evolving)
- Support general, and specific, anisotropic adaptation



Mesh Shape Optimization

- Control element shapes as needed by the various discretization methods for maintaining accuracy and efficiency

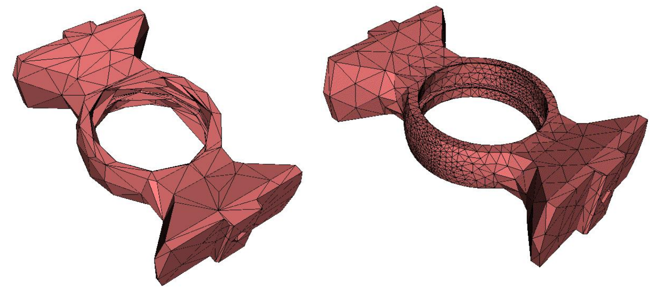


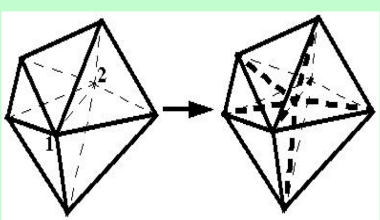


Parallel execution of all three functions critical on large meshes

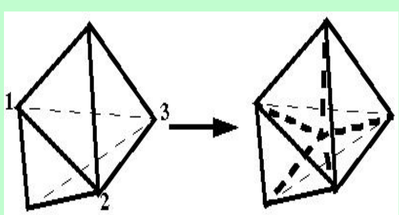
General Mesh Modification for Mesh Adaptation

- Driven by an anisotropic mesh size field that can be set by any combination of criteria
- Employ a “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 constraints (like mass conservation)

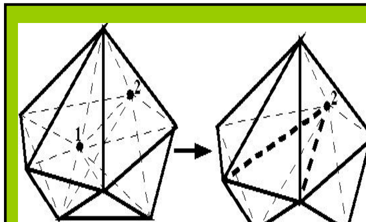




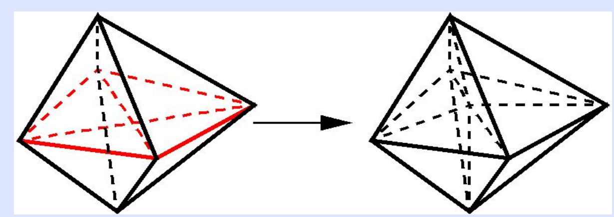
Edge split



face split



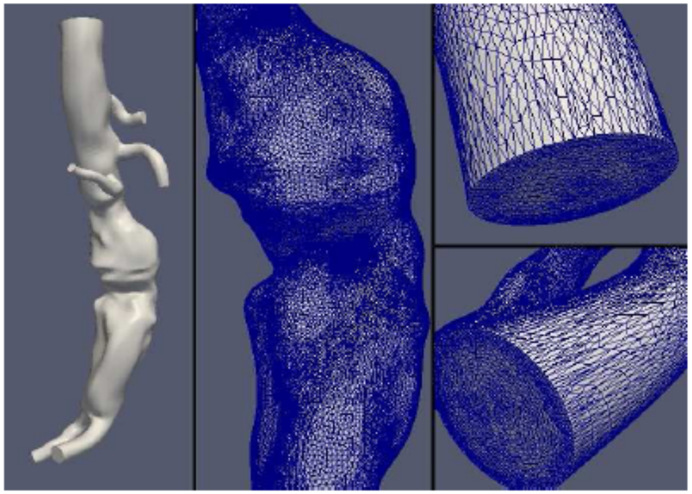
Edge collapse

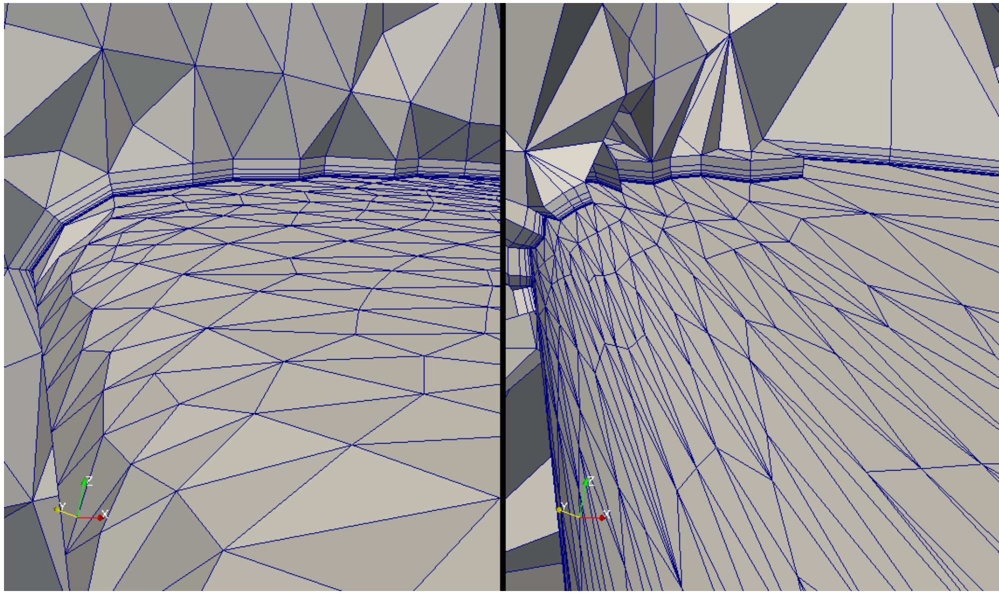


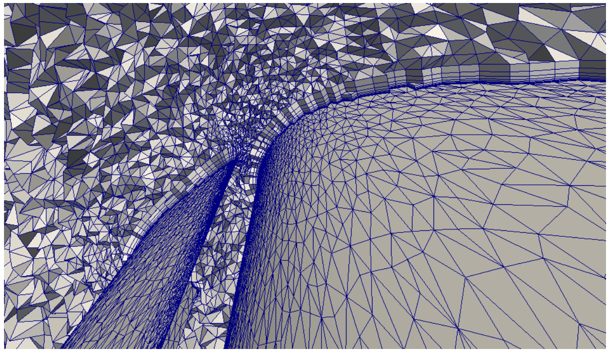
Double split collapse to remove sliver

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

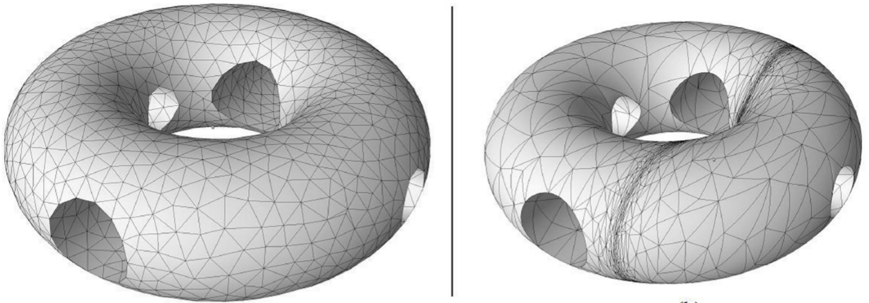




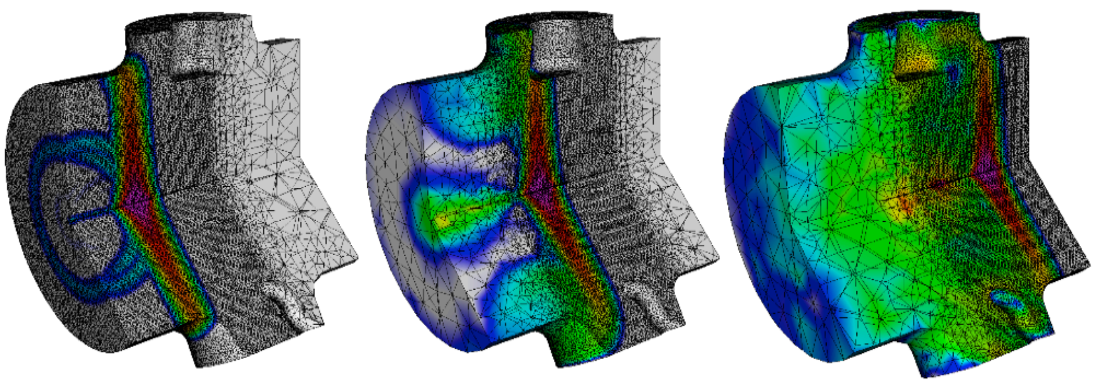


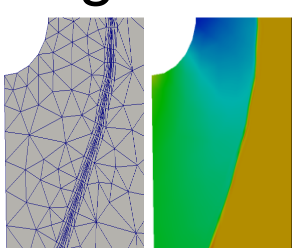
Mesh Adaptation Status

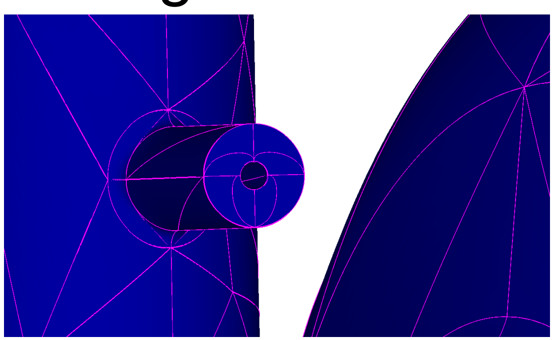
- Supports adaptation of curved elements
- Adaptation based on multiple criteria, examples
  - Level sets at interfaces
  - Tracking particles
  - Discretization errors
  - Controlling element shape in evolving geometry

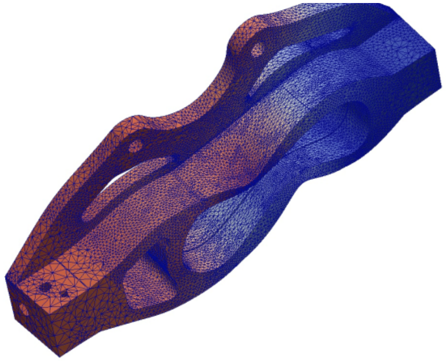


(a) (b)

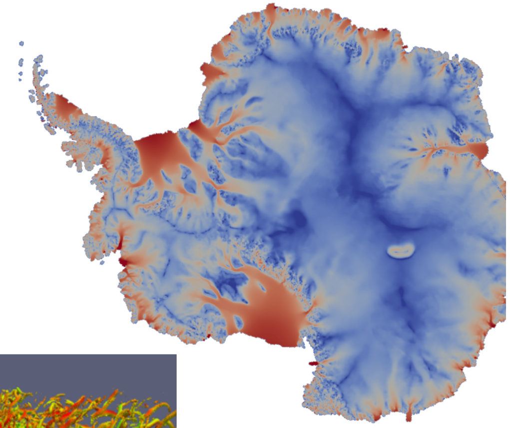




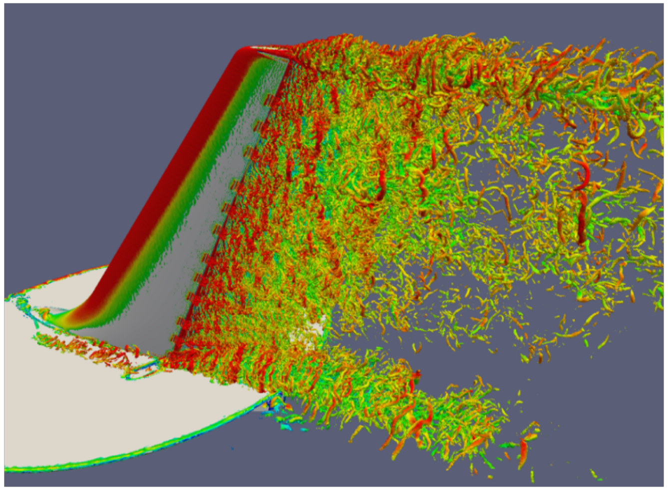




Attached Parallel Fields (APF)



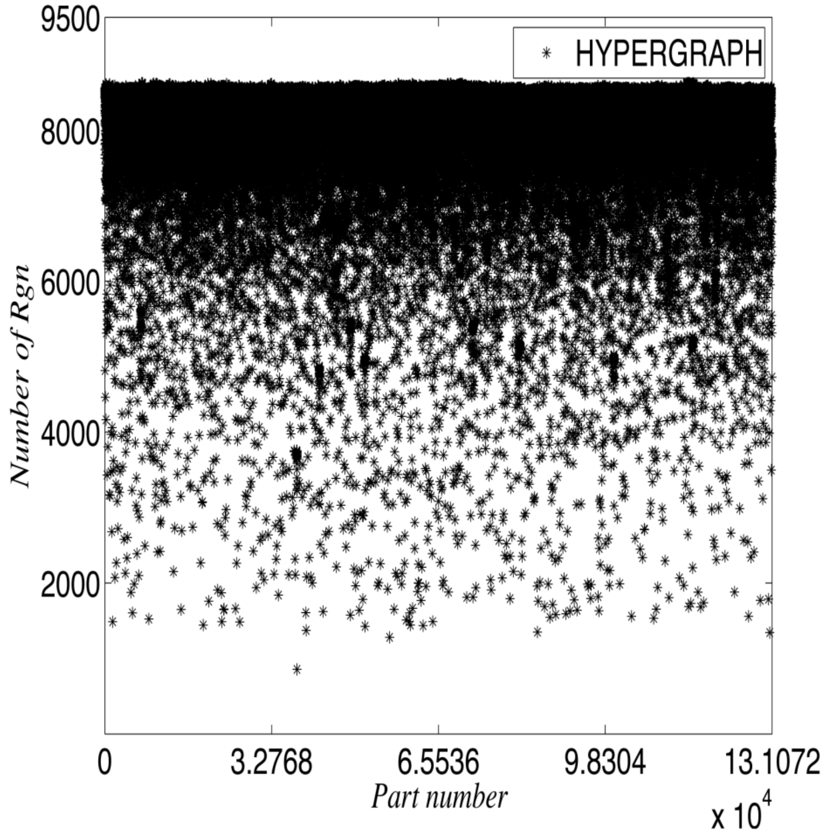
- Attached Parallel Fields (APF)
- Effective storage of solution fields on meshes
- Supports operations on the fields
  - Interrogation
  - Differentiation
  - Integration
  - Interpolation/projection
  - Mesh-to-mesh transfer
  - Local solution transfer



- Recent efforts
  - Adaptive expansion of Fields from 2D to 3D in M3D-C1
  - History-dependent integration point fields for Albany plasticity models

Dynamic Load Balancing

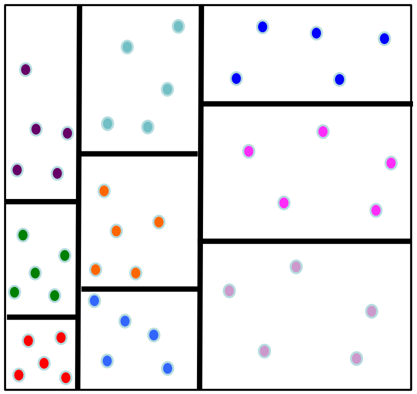
- Purpose: to rebalance load during mesh modification and before each key step in the parallel workflow
  - Equal "work load" with minimum inter-process communications
- FASTMATH load balancing tools
  - Zoltan/Zoltan2 libraries provide multiple dynamic partitioners with general control of partition objects and weights
  - ParMA – Partitioning using mesh adjacencies
  - ParMA and Zoltan2 can use each other's methods



Zoltan/Zoltan2 Toolkits: Partitioners

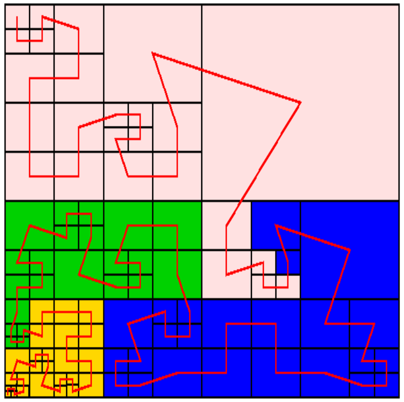
Suite of partitioners supports a wide range of applications; no single partitioner is best for all applications.

Geometric

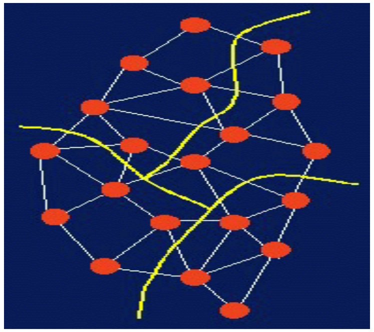


Recursive Coordinate Bisection
Recursive Inertial Bisection
Multi-Jagged Multi-section

Space Filling Curves

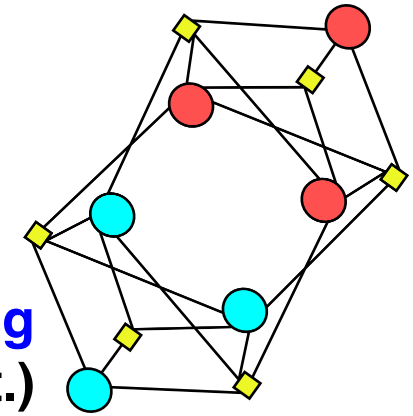


Topology-based



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

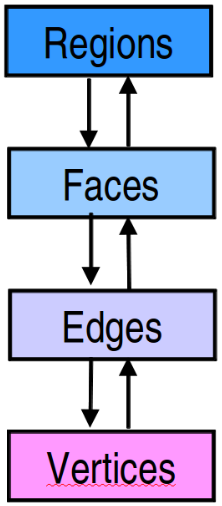
PHG Hypergraph Partitioning
Interface to PaToH (Ohio St.)



ParMA Partition Improvement

Guide partitioning decisions using mesh adjacencies

- All mesh entities can be considered
- Employ diffusive migration
- Well suited to improve partition after graph or geometric partitioning



Example result for PHASTA FE-based CFD code

- 1.6B element mesh from 128K to 1Mi
  Global RIB – 103 sec.,ParMA – 20 sec.
  209% vtx imb reduced to 6%, perfect elm imb increased to 4%, 5.5% reduction in avg vtx per part

Building In-Memory Parallel Workflows

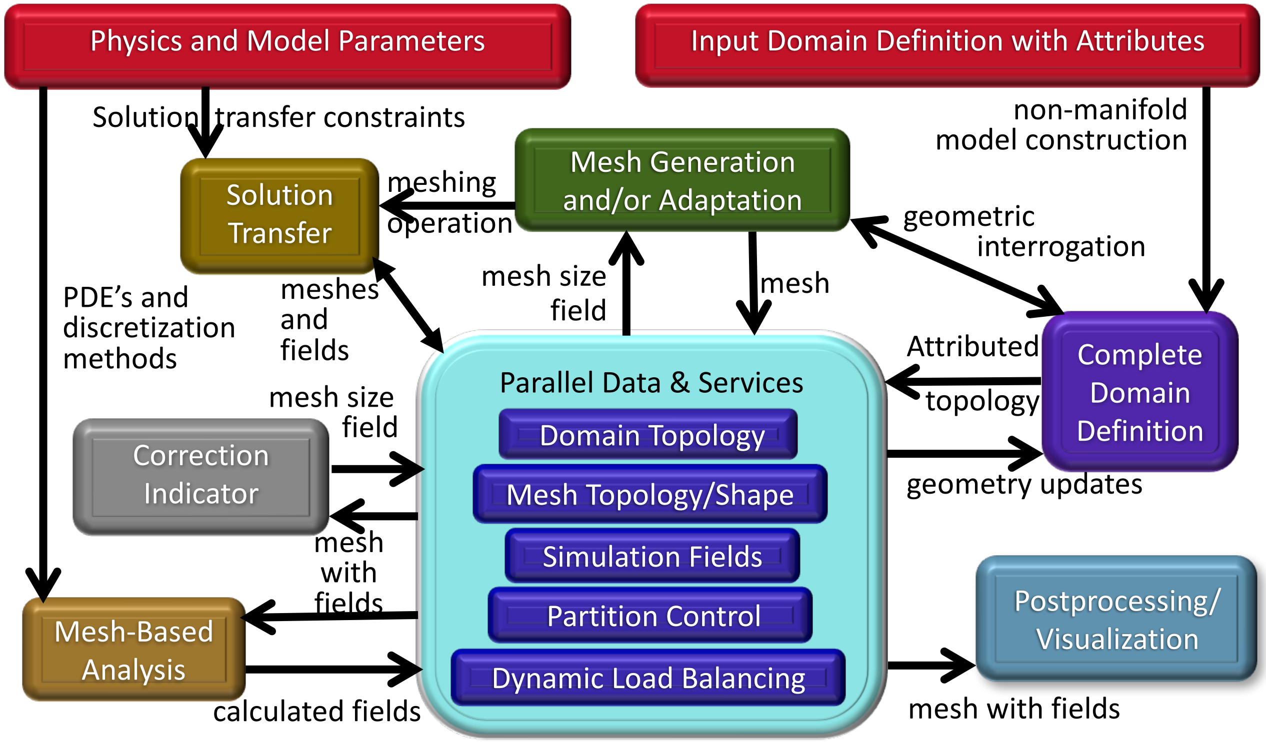
A scalable workflow requires effective component coupling

- Avoid file-based information passing
  - On massively parallel systems I/O dominates power consumption
  - Parallel filesystem technologies lag behind performance and scalability of processors and interconnects
  - Unlike compute nodes, the file system resources are almost always shared and performance can vary significantly
- Use APIs and data-streams to keep inter-component information transfers and control in on-process memory
  - When possible, don't change horses
  - Component implementation drives the selection of an in-memory coupling approach
  - Link component libraries into a single executable

Creation of Parallel Adaptive Loops

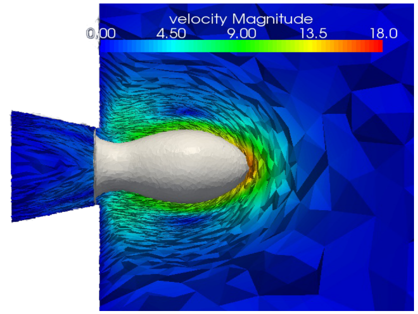
Parallel data and services are the core

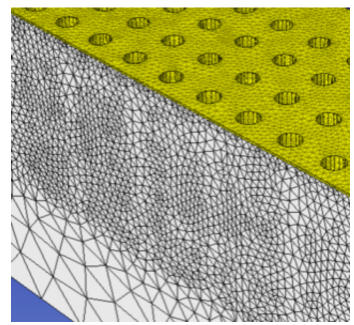
- Geometric model topology for domain linkage
- Mesh topology – it must be distributed
- Simulation fields distributed over geometric model and mesh
- Partition control
- Dynamic load balancing required at multiple steps
- API's to link to
  - CAD
  - Mesh generation and adaptation
  - Error estimation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

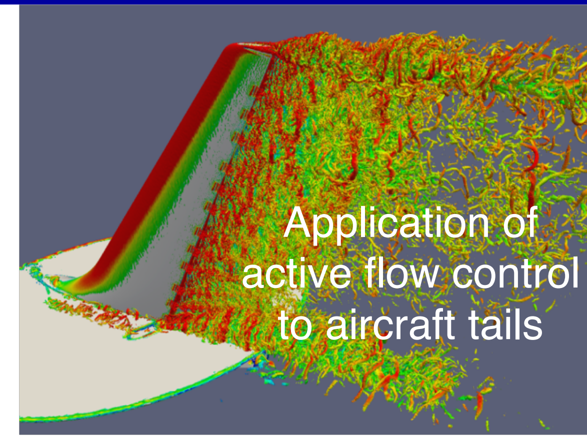


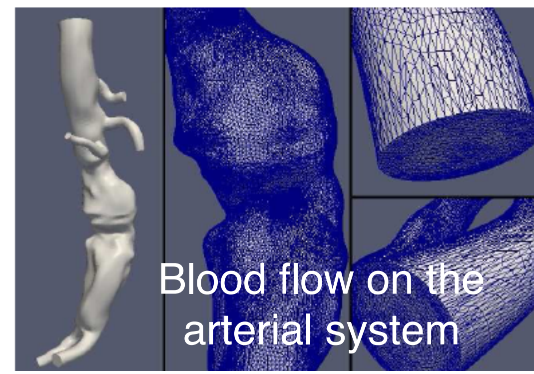
Parallel Adaptive Simulation Workflows

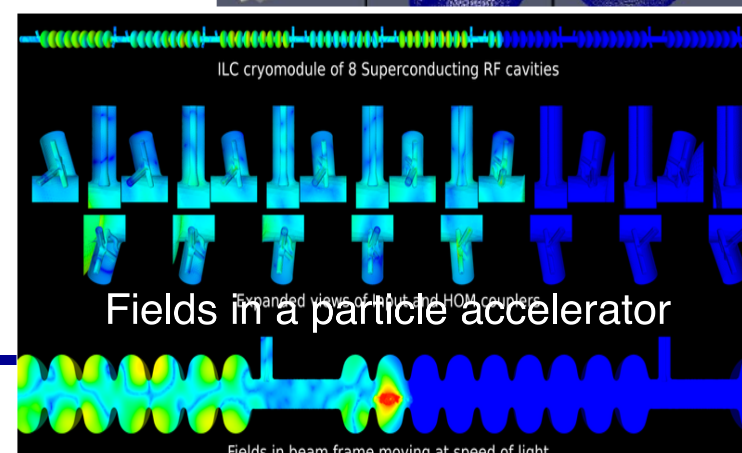
- Automation and adaptive methods critical to reliable simulations for both scientific and industrial applications
- In-memory examples
  - MFEM – FE framework
  - PHASTA – FE for NS
  - FUN3D – FV CFD
  - Proteus – multiphase FE
  - Albany/Trilinos – FE Solid mechanics
  - ACE3P – High order FE electromagnetics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











PUMI Software Pointers

Resources for PUMI:

- Overview: scorec.rpi.edu/pumi/
- Design, concepts, and applications: (TOMS journal paper) scorec.rpi.edu/REPORTS/2014-9.pdf
- Intro and user's guide: scorec.rpi.edu/pumi/pumi_intro.pdf, scorec.rpi.edu/pumi/PUMI.pdf
- APIs: scorec.rpi.edu/~seol/scorec/doxygen/
- Build instructions: github.com/SCOREC/core/wiki/General-Build-instructions
- Nightly regression: my.cdash.org/index.php?project=SCOREC
- Much more: github.com/SCOREC/core/wiki

Recent PUMI advances (its running on the latest Phi's at Argonne and NERSC, there is also a GPU version):

- Thesis on array-based implementation using manycore & GPUs: scorec.rpi.edu/reports/view_report.php?id=710
- See Ibanez or Smith 2015 - 2017 papers: scorec.rpi.edu/reports/