Unstructured Meshing Technologies

Presented to
ATPESC 2018 Participants

Tzanio Kolev (LLNL), Mark S. Shephard (RPI), and Cameron W. Smith (RPI)

Q Center, St. Charles, IL (USA)
Date 08/06/2018

U.S. DEPARTMENT OF ENERGY | Office of Science

ATPESC Numerical Software Track

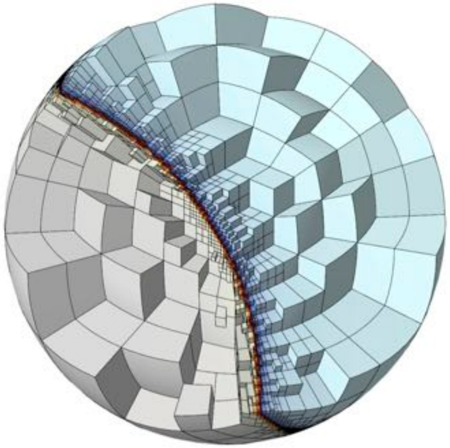
EXASCALE COMPUTING PROJECT

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

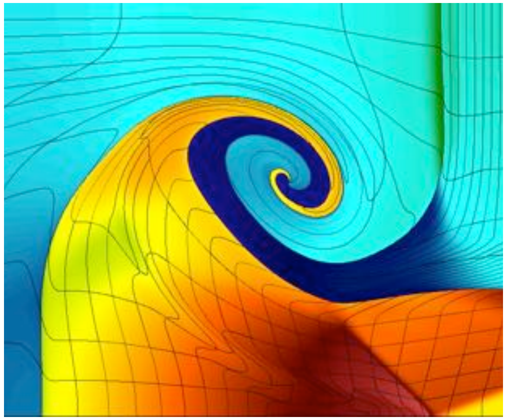
- Backed by well-developed theory.
- Naturally support unstructured and curvilinear grids.
- **High-order finite elements on high-order meshes**
  - Increased accuracy for smooth problems
  - Sub-element modeling for problems with shocks
  - Bridge unstructured/structured grids
  - Bridge sparse/dense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

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

$H(grad)$	$H(curl)$	$H(div)$	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

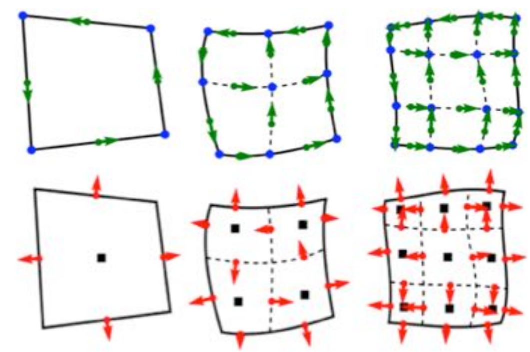


8th order Lagrangian hydro simulation of a shock triple-point interaction

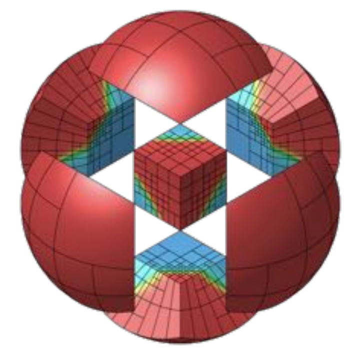
Modular Finite Element Methods (MFEM)

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

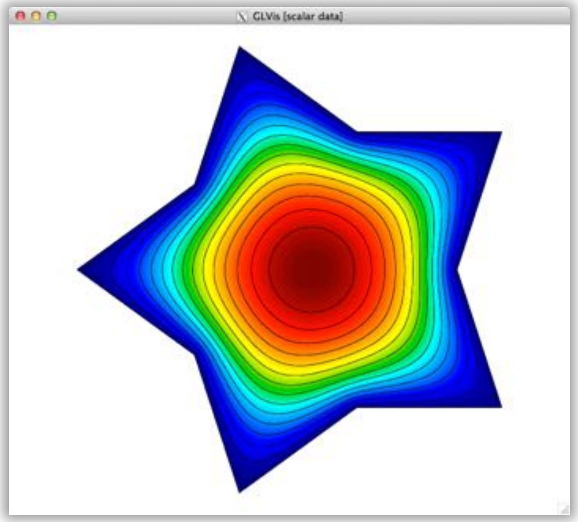
```
// 5. Set up the linear form b(.) which corresponds to the right-hand side of
//    the FEM linear system, which in this case is (1,phi_i) where phi_i are
//    the basis functions in the finite element fespace.
LinearForm *b = new LinearForm(fespace);
ConstantCoefficient one(1.0);
b->AddDomainIntegrator(new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

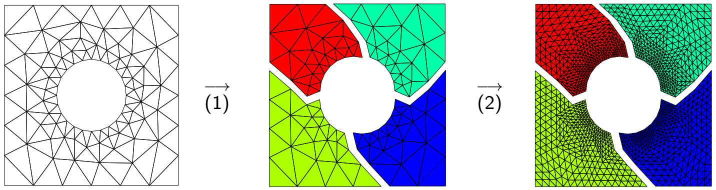
- Visualization

```
   // 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

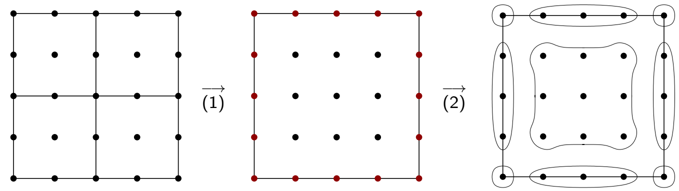
- Parallel mesh

```
// 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();
```

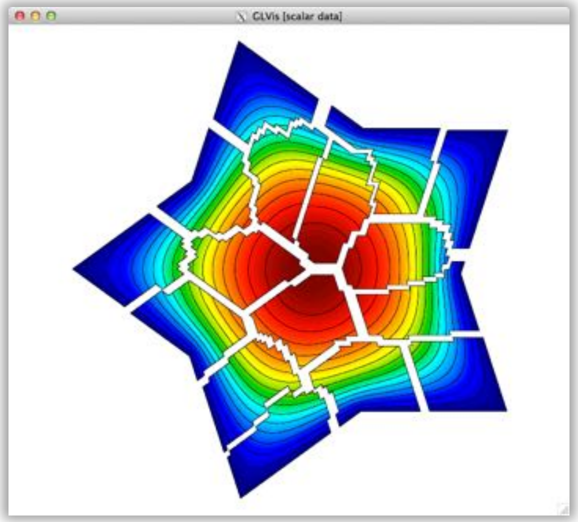
$$A = P^T a P \qquad B = P^T b \qquad x = PX$$

- 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

Example 1 – parallel Laplace equation

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

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

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

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

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

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

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

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

MFEM example codes – mfem.org/examples

Discretization Demo & Lesson

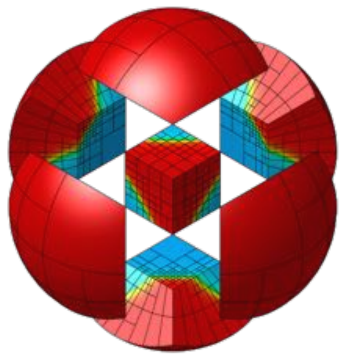
https://xsdk-project.github.io/ATPESC2018HandsOnLessons/lessons/mfem_convergence/

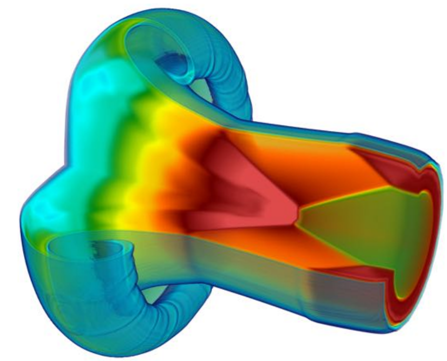
Application to high-order ALE shock hydrodynamics

***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

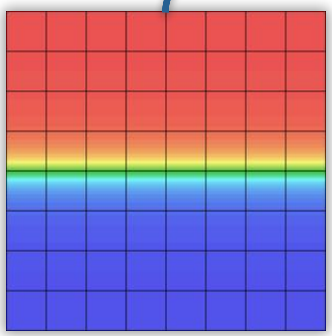
- *hypre* provides scalable algebraic multigrid solvers
- MFEM provides finite element discretization abstractions
  - uses *hypre*'s parallel data structures, provides finite element info to solvers
- BLAST solves the Euler equations using a high-order ALE framework
  - combines and extends MFEM's objects

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

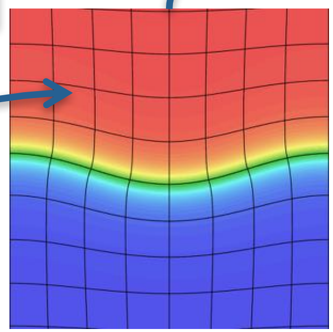
Lagrange phase

Physical time evolution

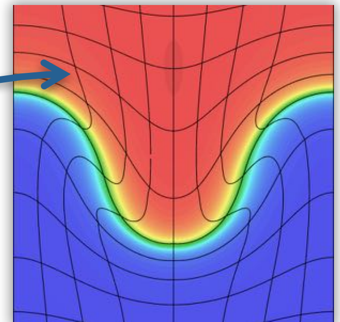
Based on physical motion



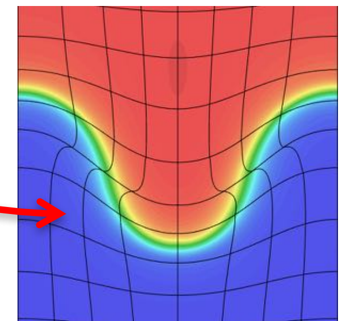
$t = 0$



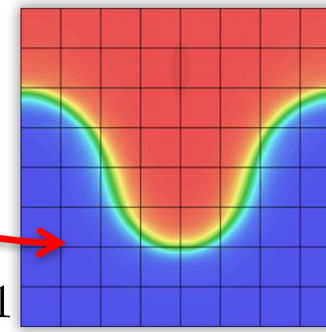
$t = 1.5$



$t = 3.0$ $\quad \tau = 0$



$\tau = 0.5$



$\tau = 1$

Remap phase

Pseudo-time evolution

Based on mesh motion

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

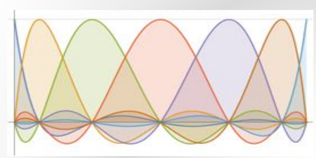
Momentum Conservation: $\rho \dfrac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \nabla \cdot \sigma$

Mass Conservation: $\dfrac{\mathrm{d}\rho}{\mathrm{d}t} = -\rho \nabla \cdot \vec{v}$

Energy Conservation: $\rho \dfrac{\mathrm{d}e}{\mathrm{d}t} = \sigma : \nabla \vec{v}$

Equation of Motion: $\dfrac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{v}$

- Galerkin FEM



Gauss-Lobatto basis

- Discont. Galerkin

Bernstein basis

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

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

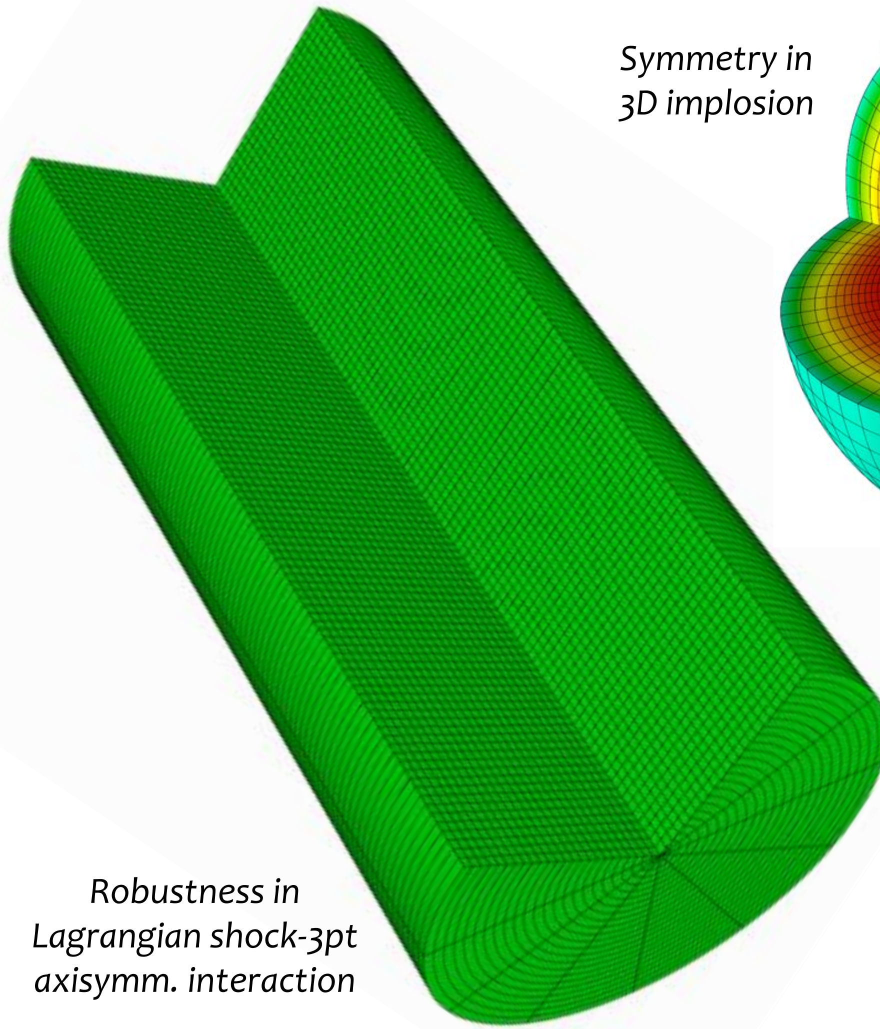
Mass Conservation: $\dfrac{\mathrm{d}\rho}{\mathrm{d}\tau} = \vec{v}_m \cdot \nabla \rho$

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

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

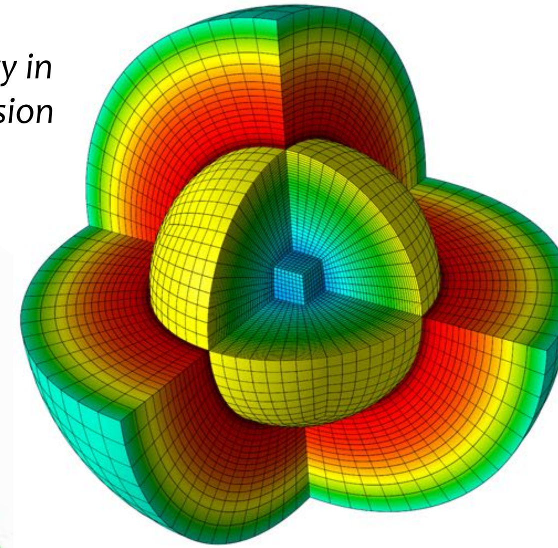
High-order finite elements lead to more accurate, robust and reliable hydrodynamic simulations

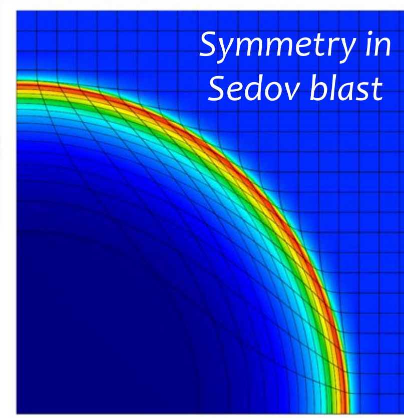
Parallel ALE for Q4 Rayleigh-Taylor instability (256 cores)



Robustness in Lagrangian shock-3pt axisymm. interaction

Symmetry in 3D implosion

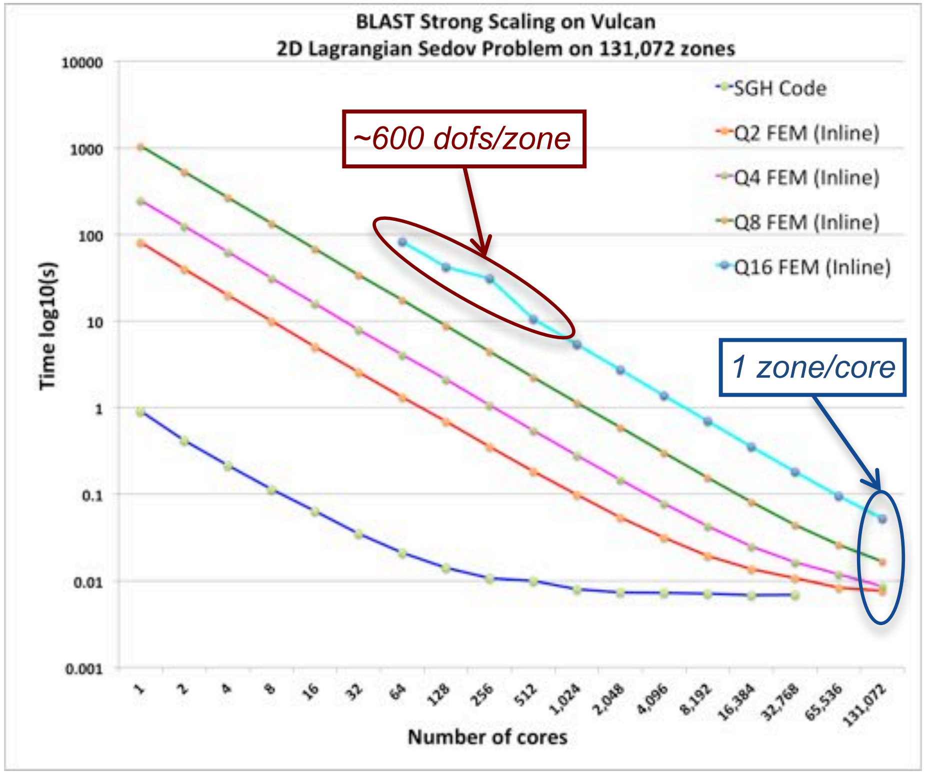




Symmetry in Sedov blast

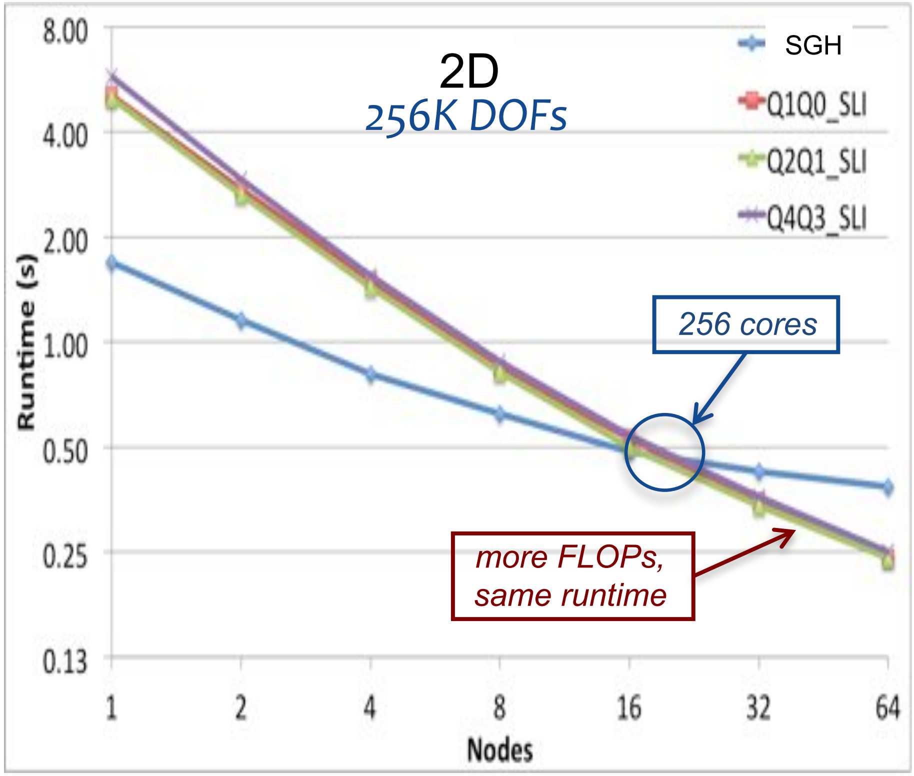
High-order finite elements have excellent strong scalability

Strong scaling, p-refinement



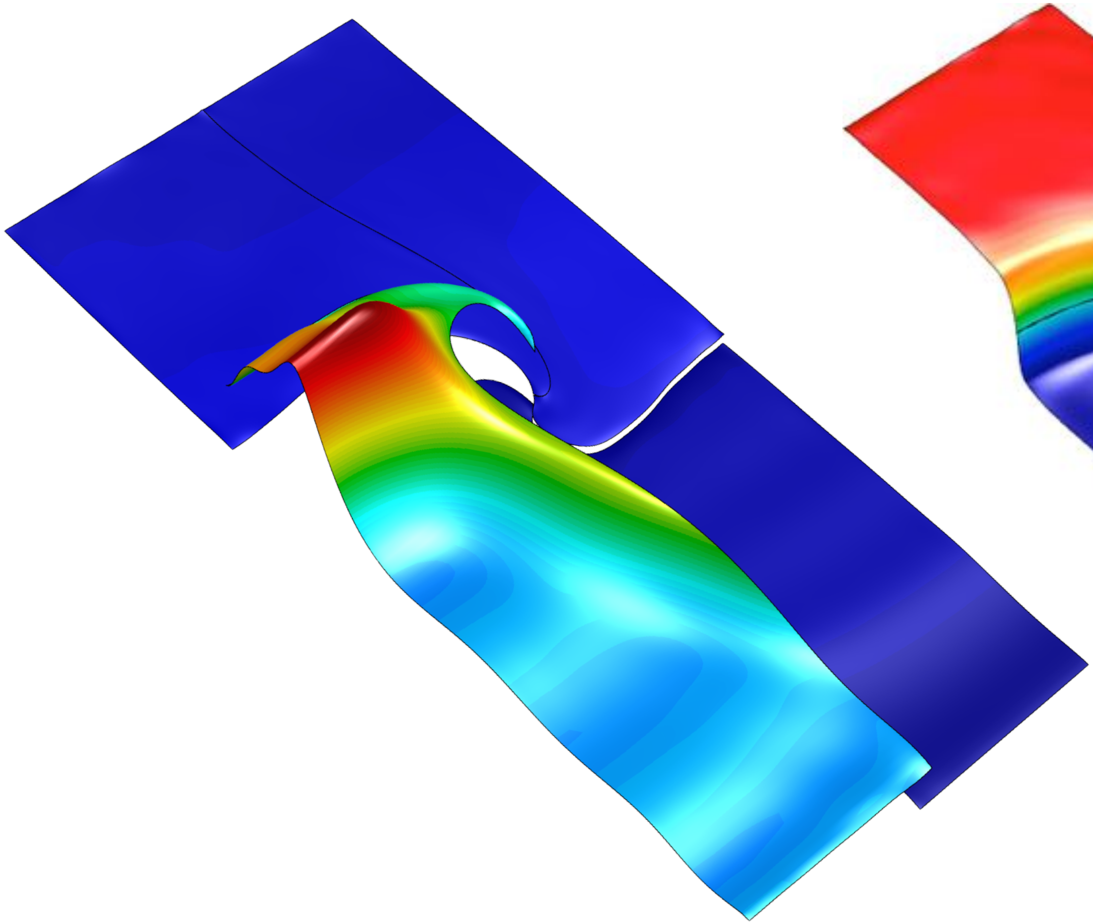
Finite element partial assembly

Strong scaling, fixed #dofs

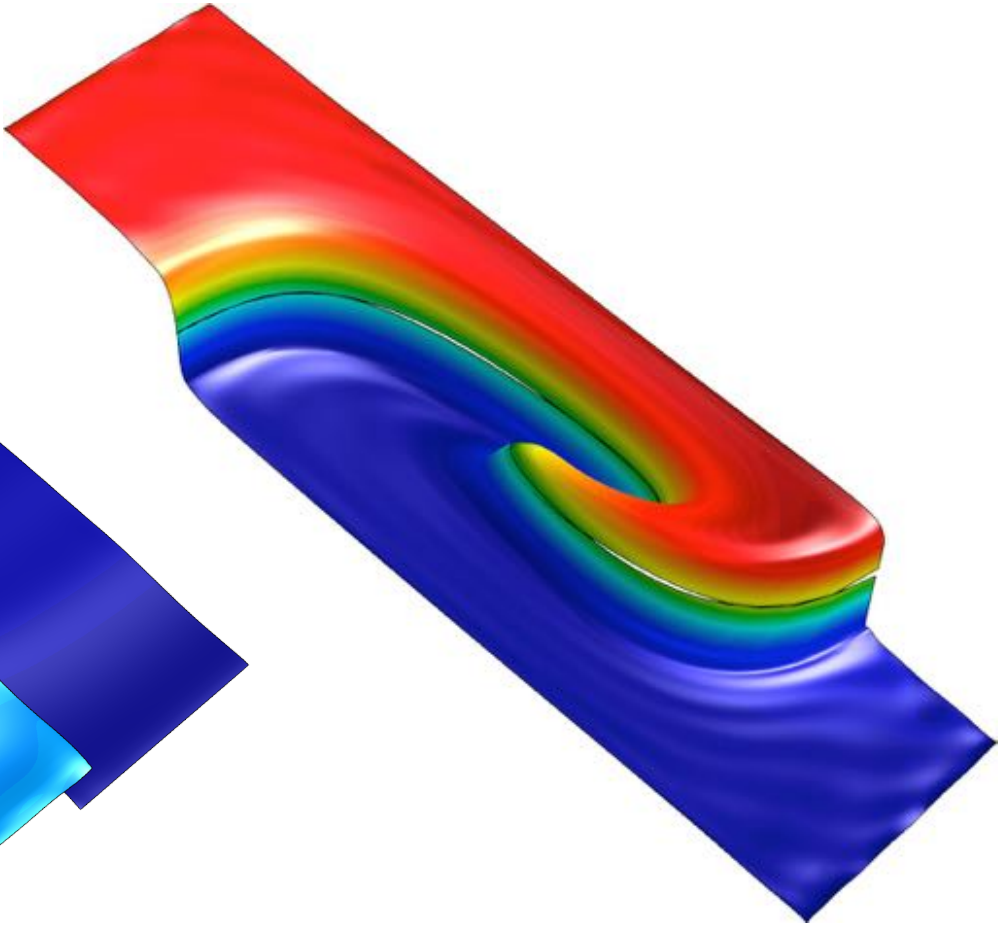


FLOPs increase faster than runtime

High-order discretizations pose unique challenges



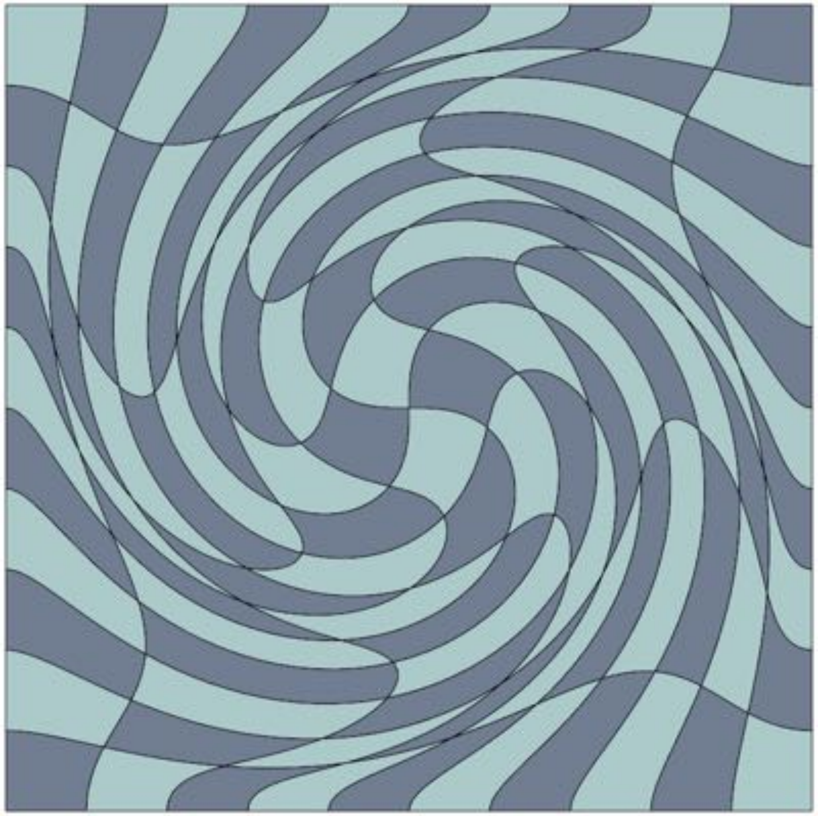
Shock triple-point interaction (4 elements)



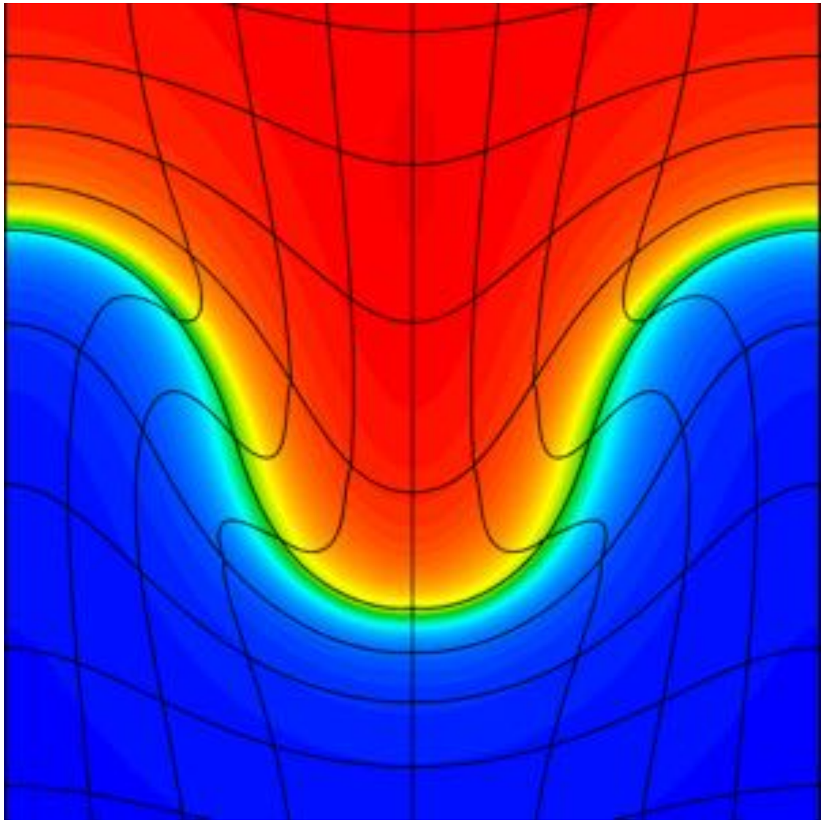
Smooth RT instability (2 elements)

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)

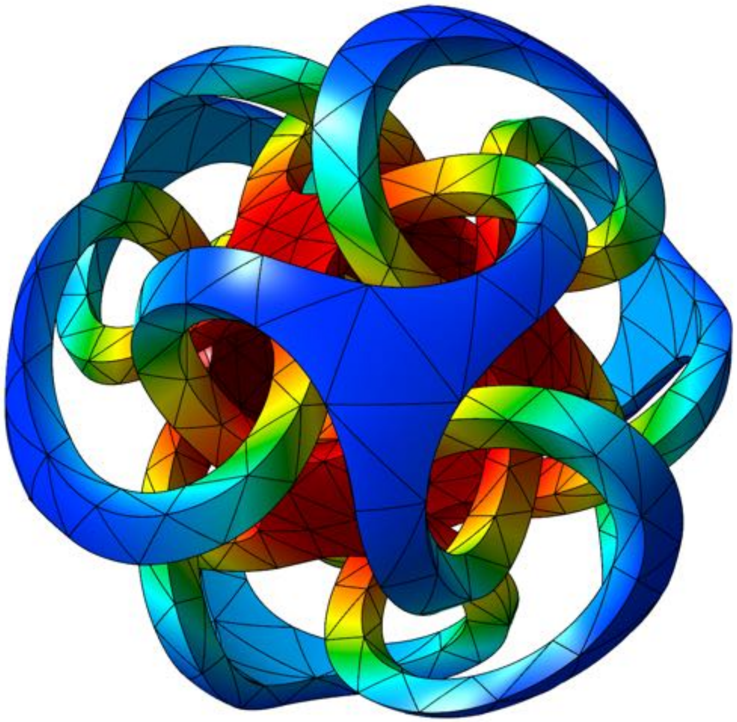


DG advection-based interpolation (ALE remap, Example 9, radiation transport)

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

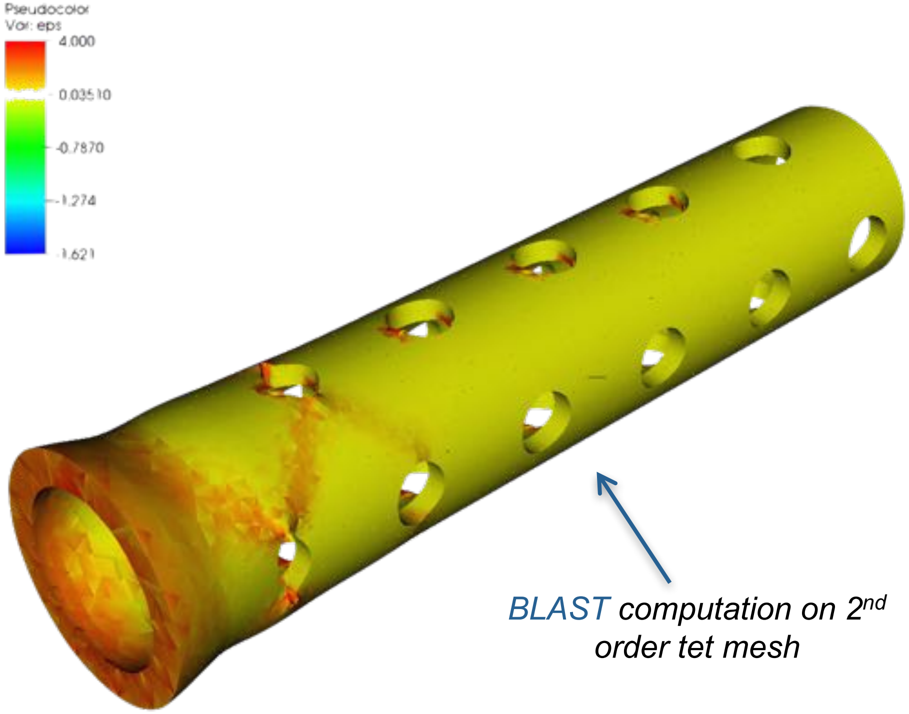
Two visualization options for high-order functions on high-order meshes

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

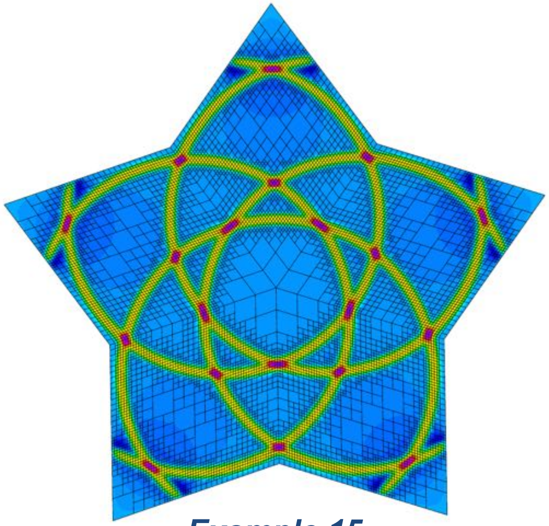
MFEM's unstructured AMR infrastructure

Adaptive mesh refinement on library level:

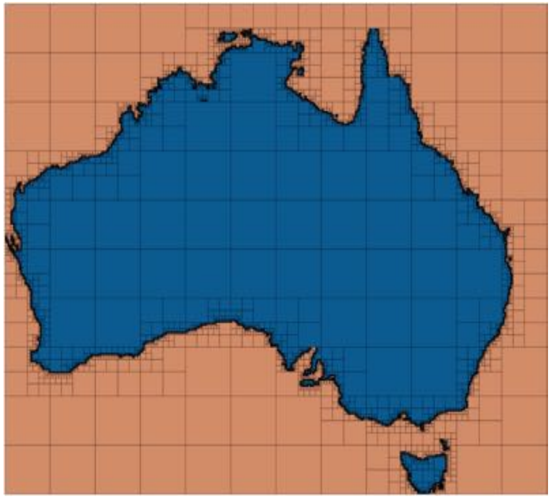
- Conforming local refinement on simplex meshes
- ***Non-conforming refinement for quad/hex meshes***
- h-refinement with fixed p

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)



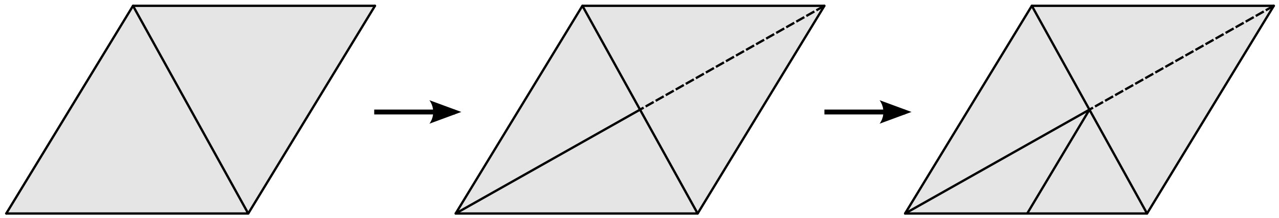
Example 15



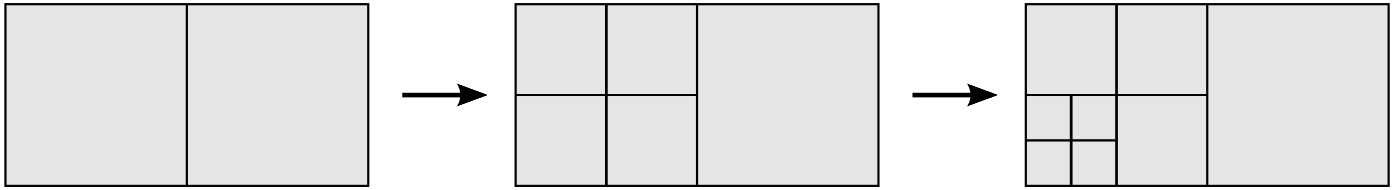
Shaper miniapp

Conforming & Nonconforming Mesh Refinement

- Conforming refinement



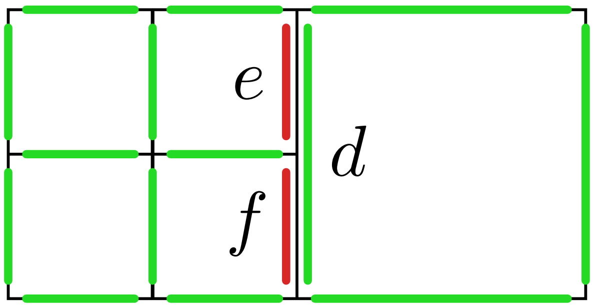
- Nonconforming refinement



  - Natural for quadrilaterals and hexahedra

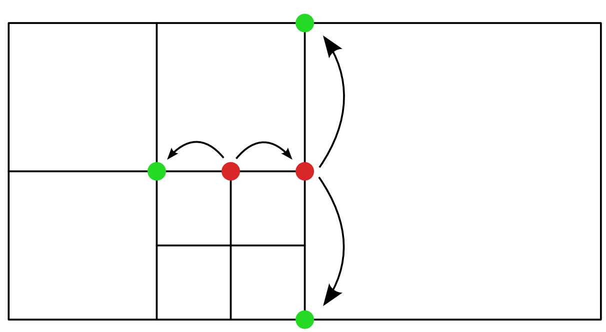
General nonconforming constraints

H(curl) elements



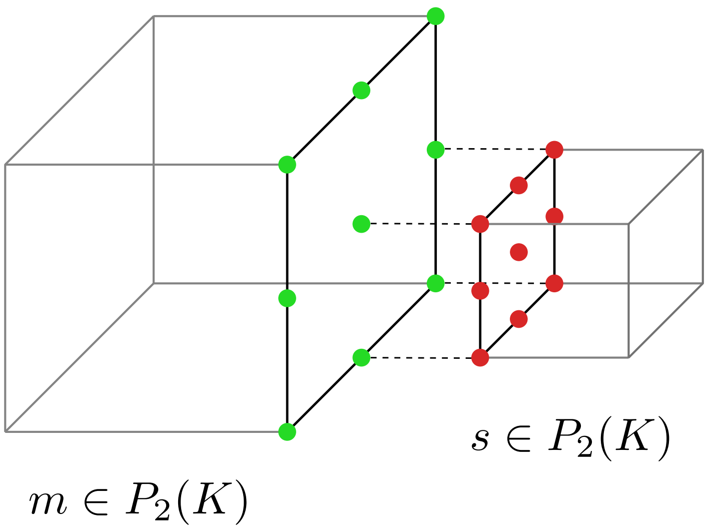
Constraint: *e = f = d/2*

Indirect constraints



More complicated in 3D…

High-order elements



Constraint: *local interpolation matrix*

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

Nonconforming variational restriction

- General constraint:

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

x – conforming space DOFs,
y – nonconforming space DOFs (unconstrained + slave),

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

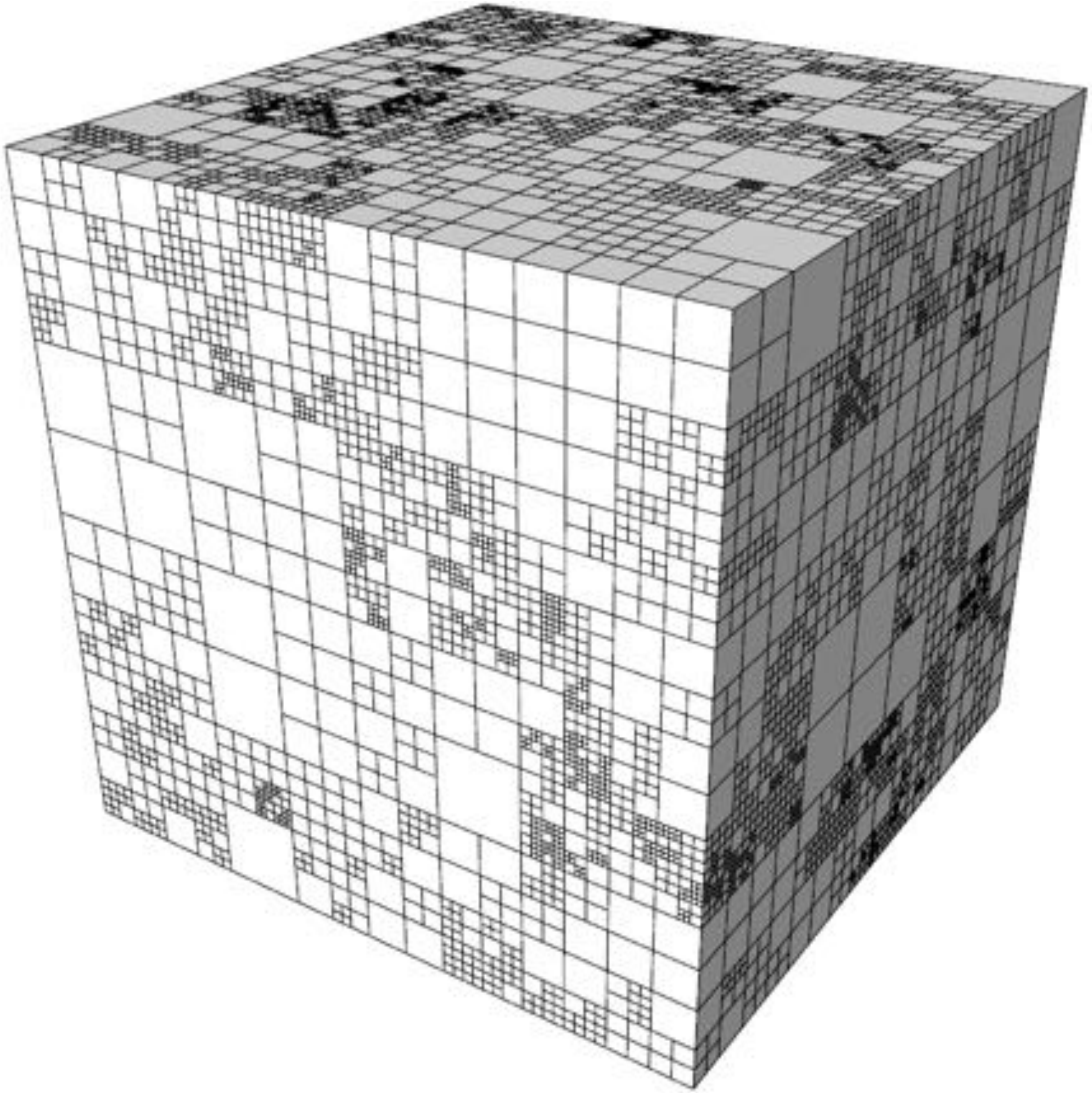
W – interpolation for slave DOFs

- Constrained problem:

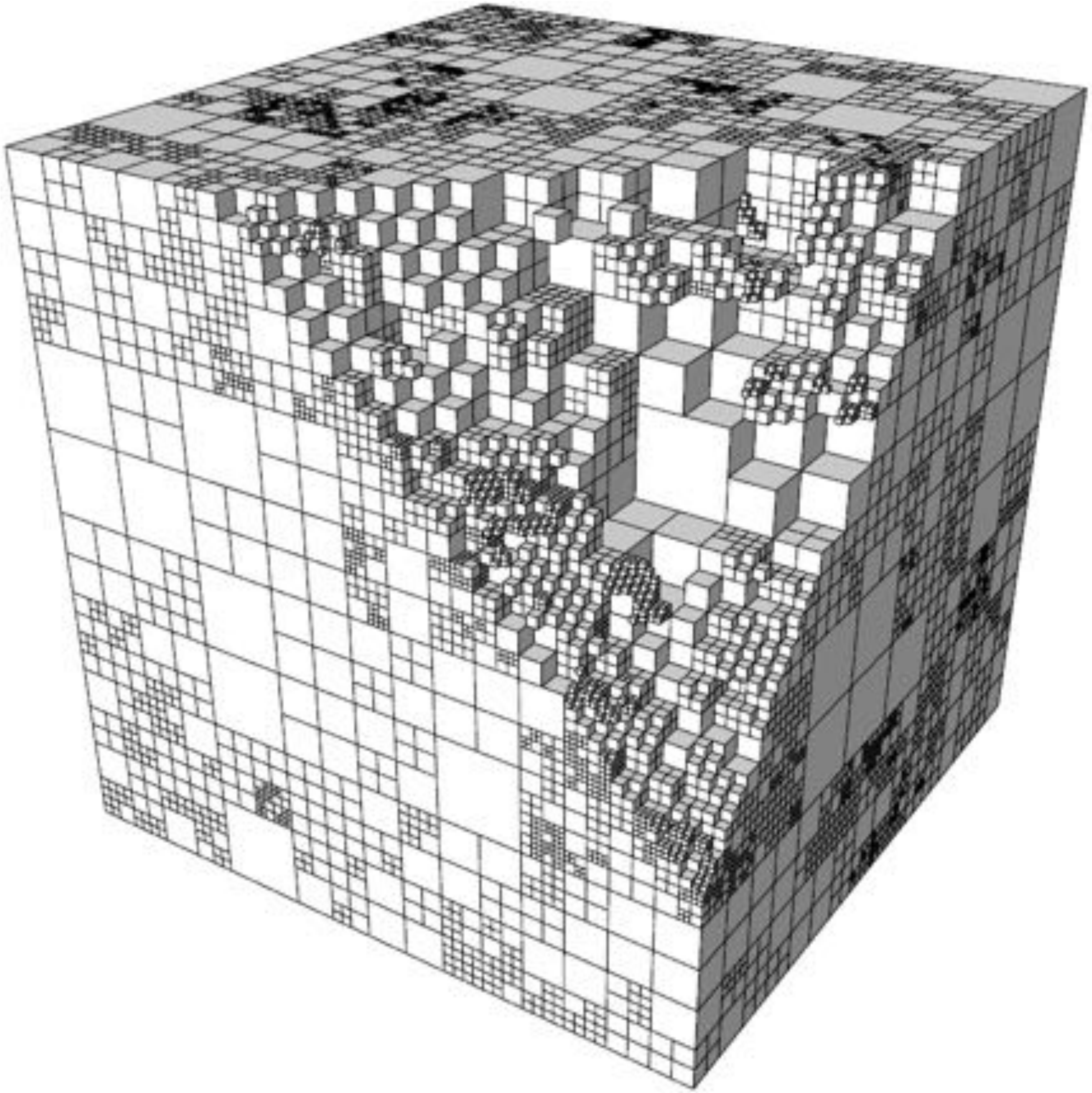
$$P^T APx = P^T b,$$

$$y = Px.$$

Nonconforming variational restriction

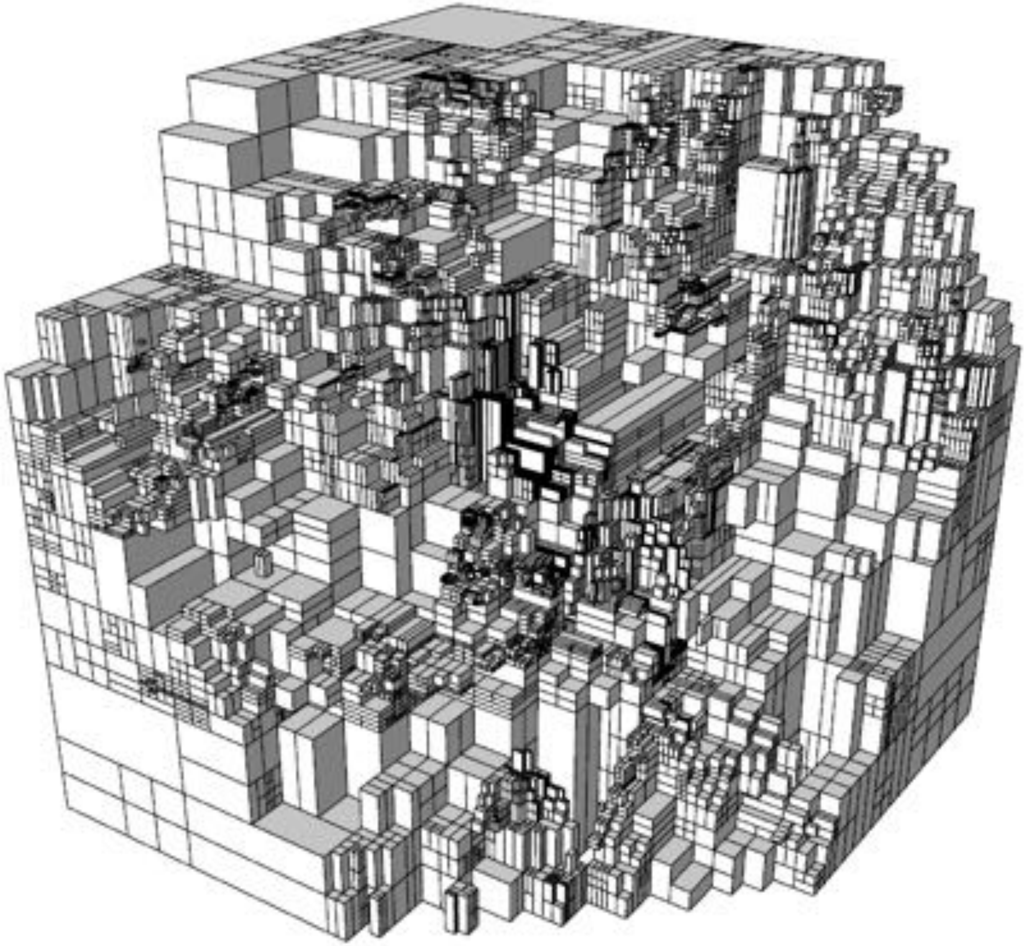


Nonconforming variational restriction



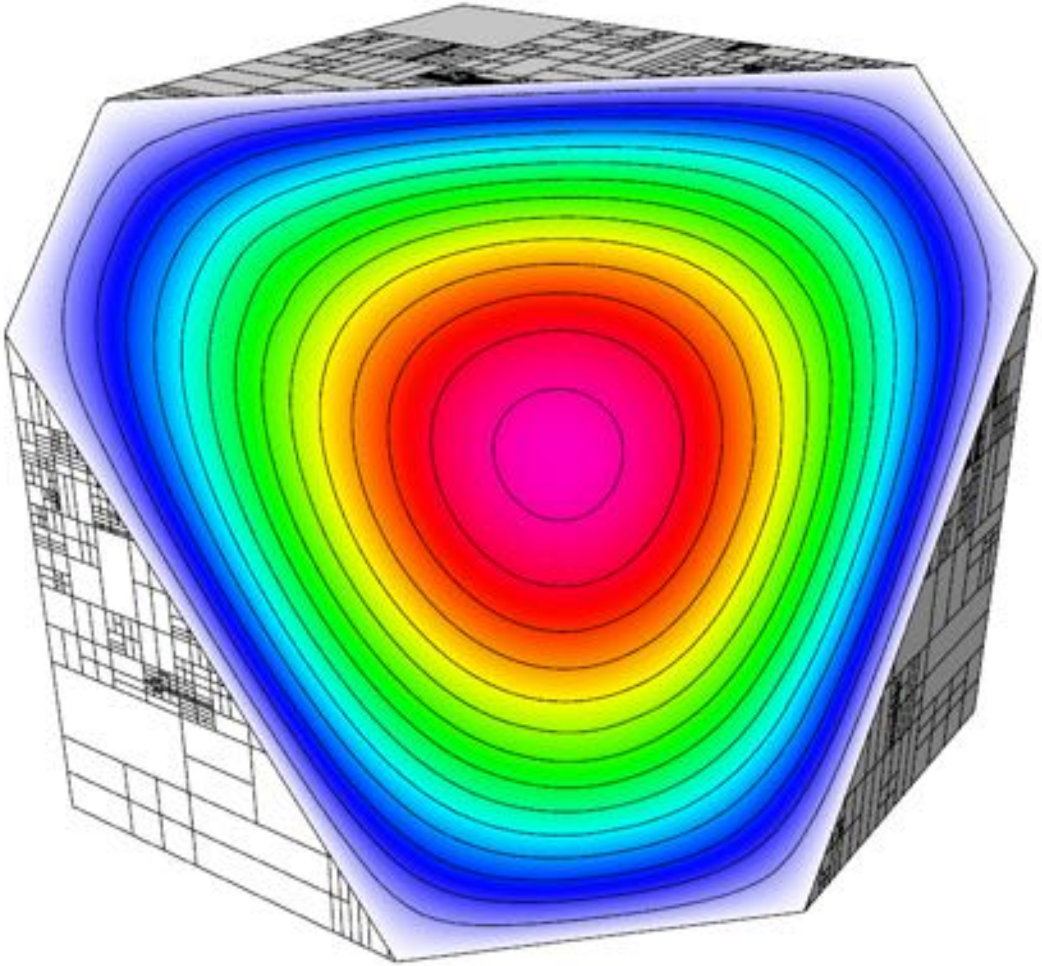
Regular assembly of A on the elements of the (cut) mesh

Nonconforming variational restriction



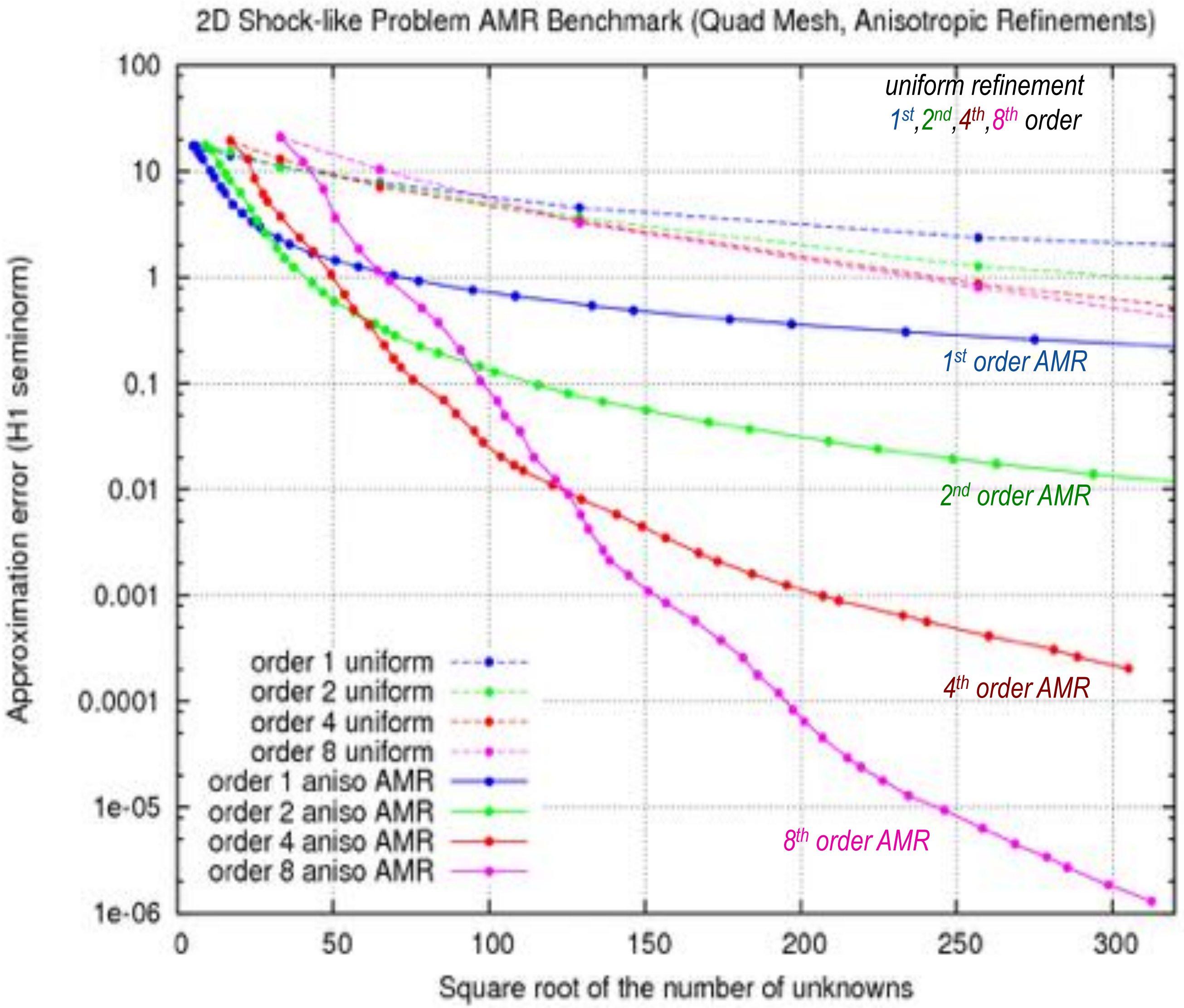
Regular assembly of A on the elements of the (cut) mesh

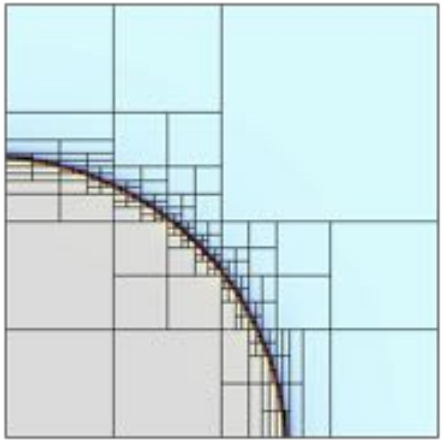
Nonconforming variational restriction

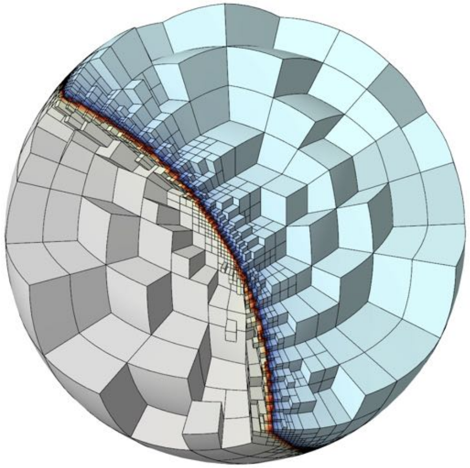


Conforming solution y = P x

AMR = smaller error for same number of unknowns



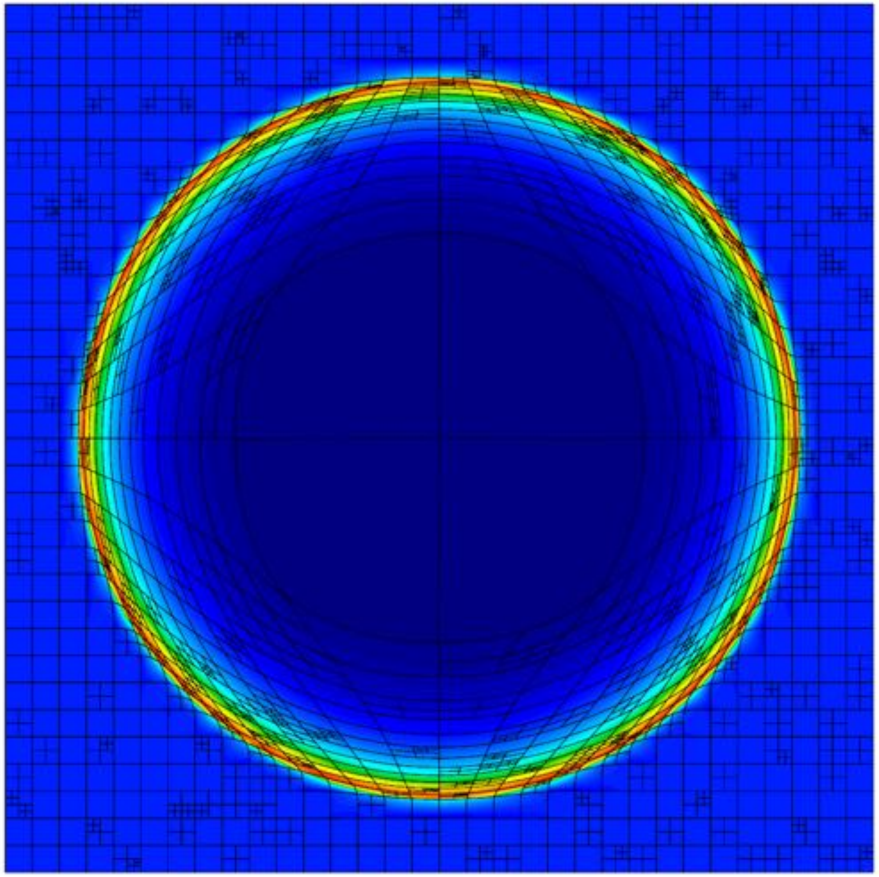




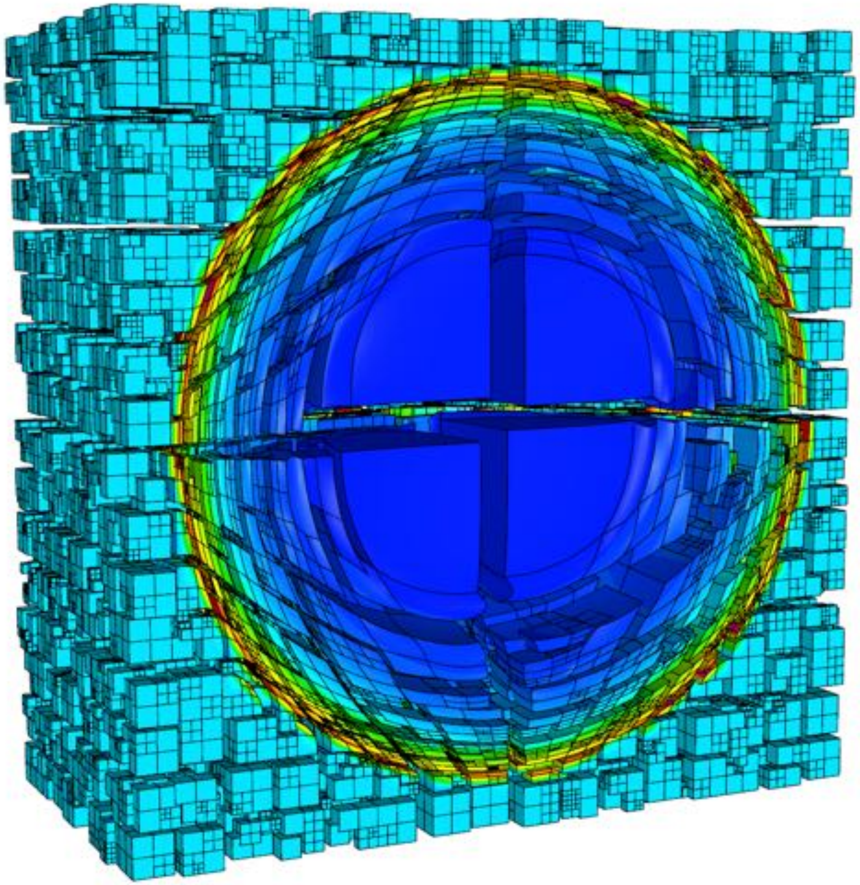
Anisotropic adaptation to shock-like fields in 2D & 3D

Static parallel refinement, Lagrangian Sedov problem

8 cores, random non-conforming ref.

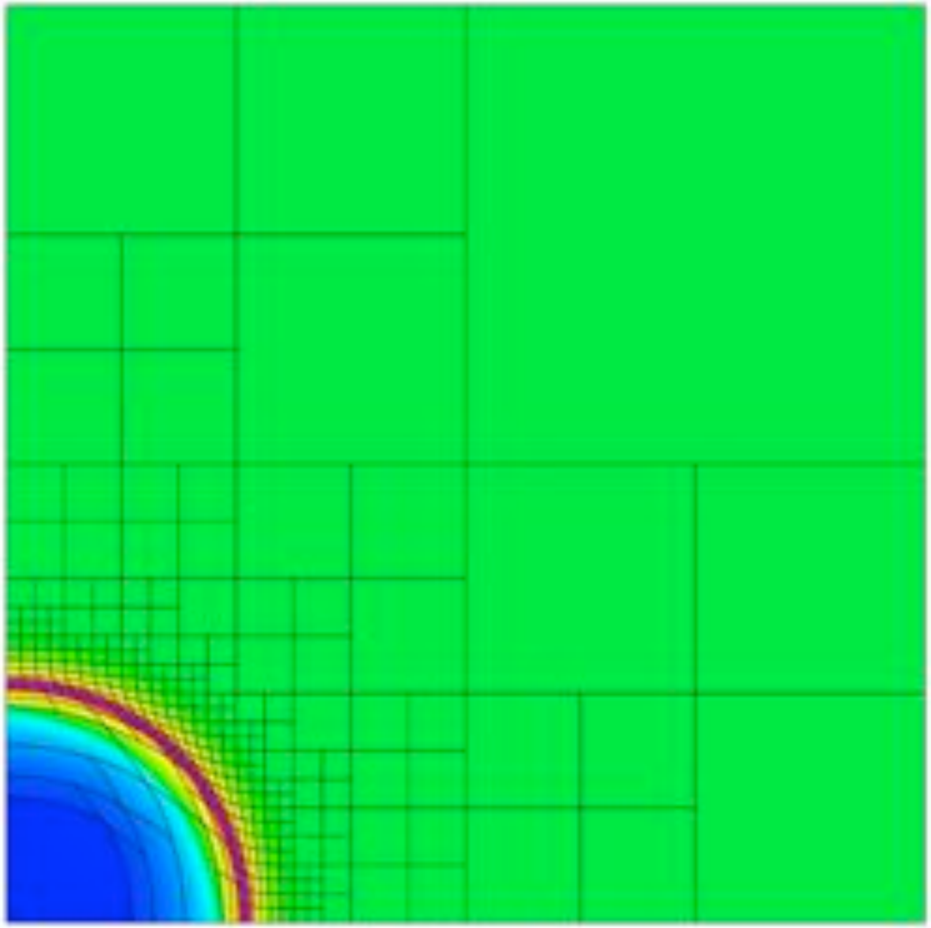


4096 cores, random non-conforming ref.

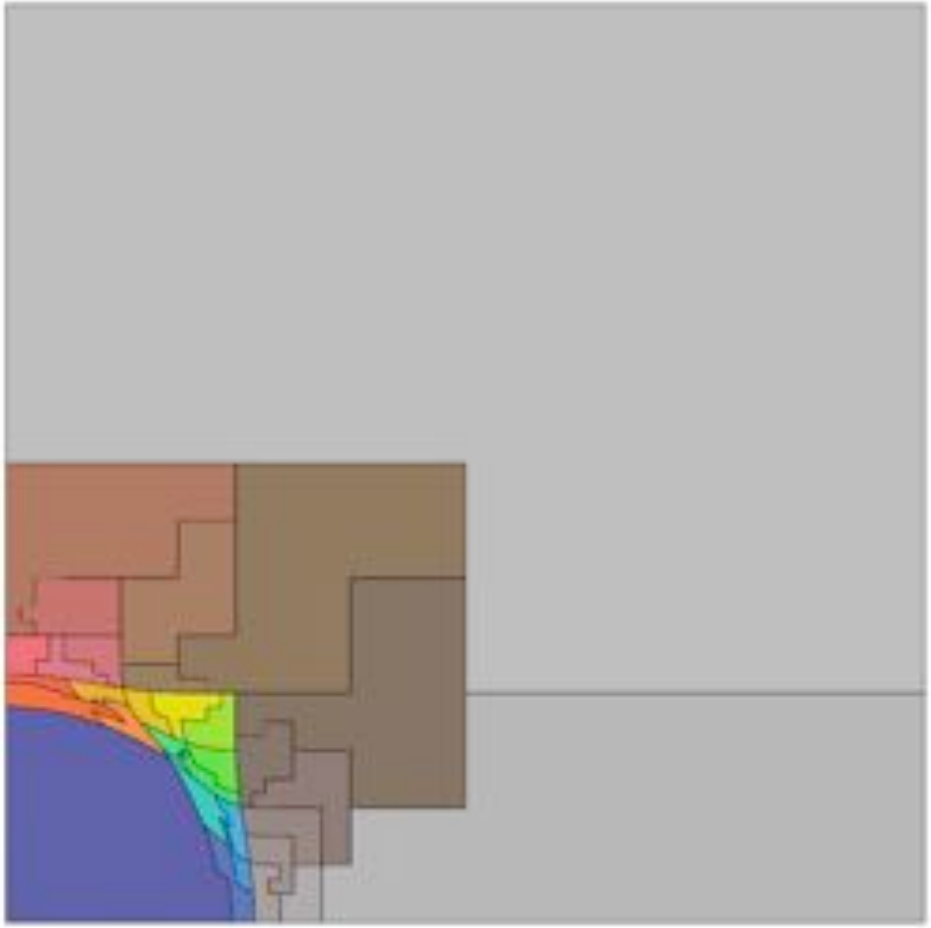


Shock propagates through non-conforming zones without imprinting

Parallel dynamic AMR, Lagrangian Sedov problem

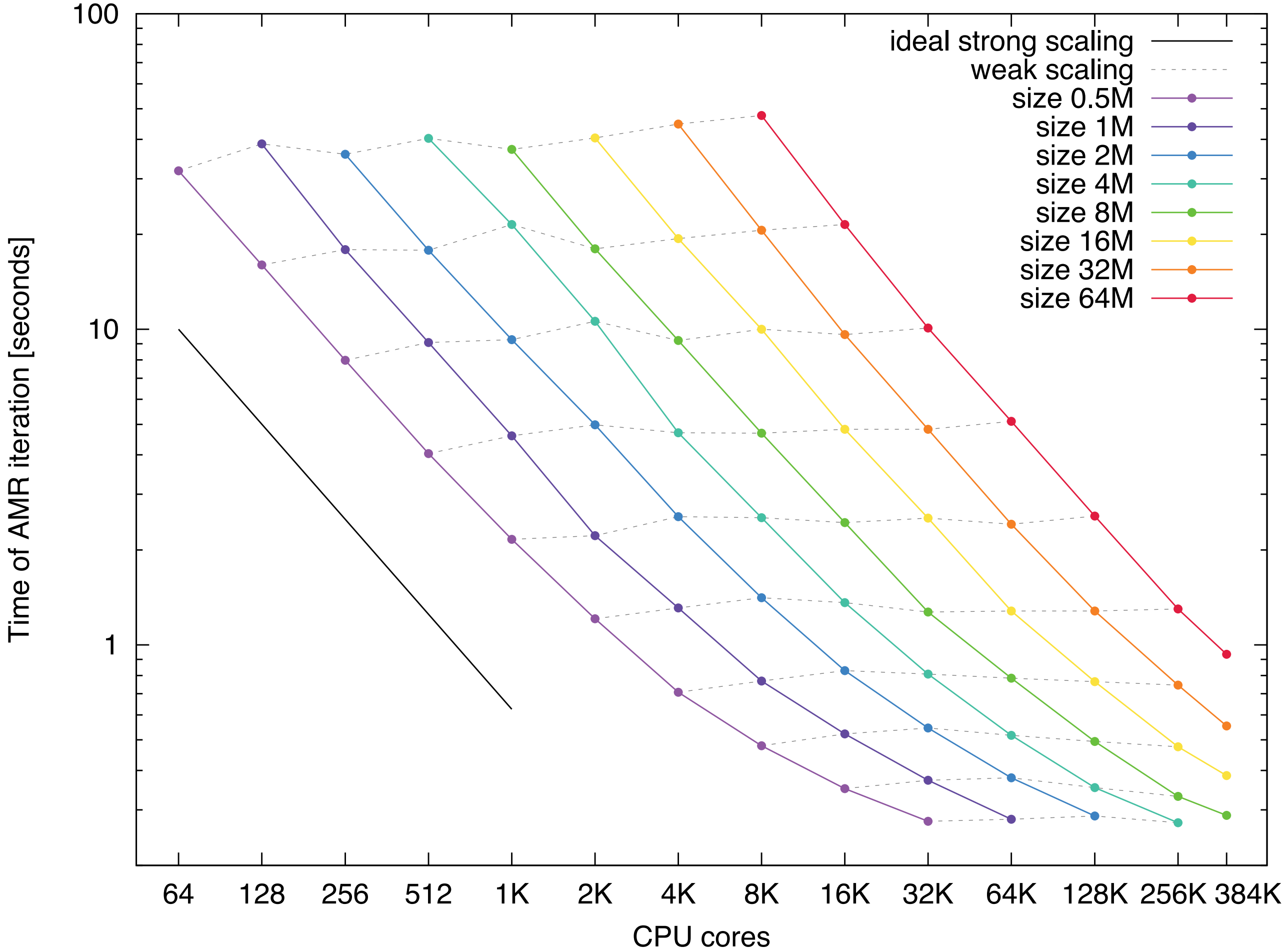


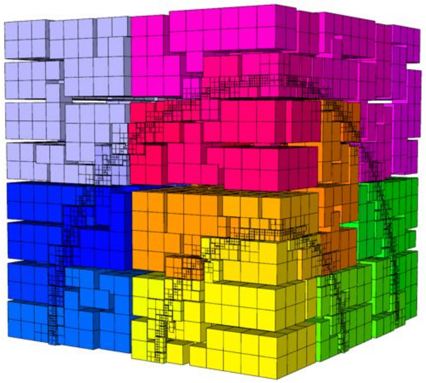
Adaptive, viscosity-based refinement and derefinement. 2nd order Lagrangian Sedov



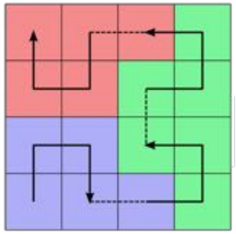
Parallel load balancing based on space-filling curve partitioning, 16 cores

Parallel AMR scaling to ~400K MPI tasks



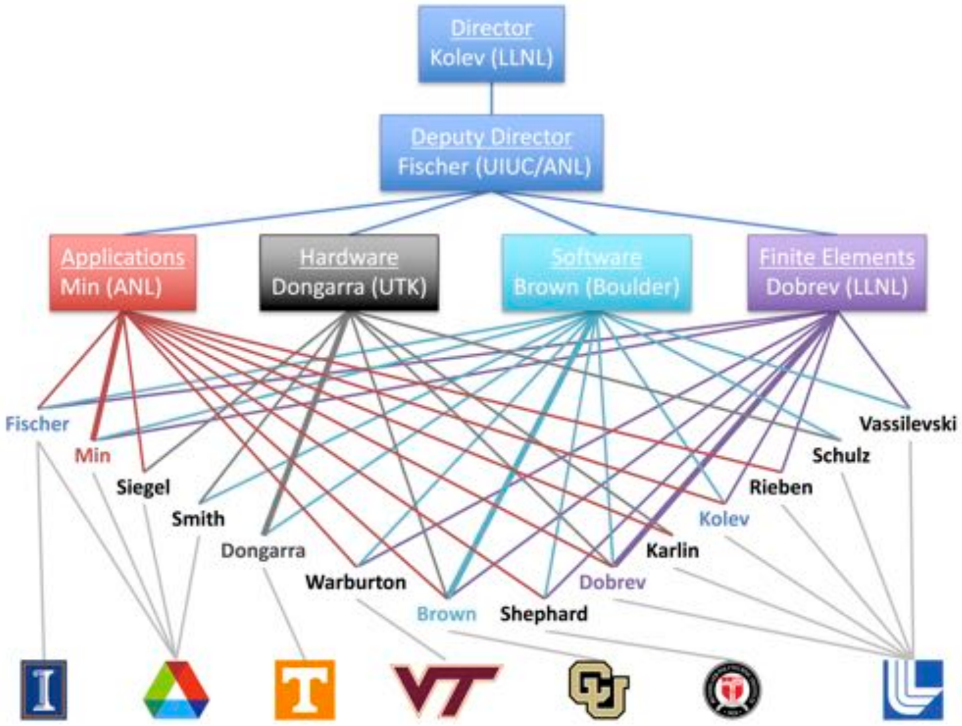


Parallel decomposition (2048 domains shown)



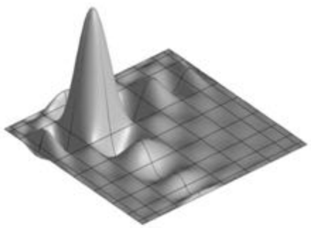
Parallel partitioning via Hilbert curve

- 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")

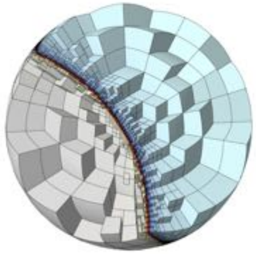


2 Labs, 5 Universities, 30+ researchers

- 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*

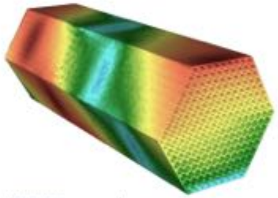


10th order basis function



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*



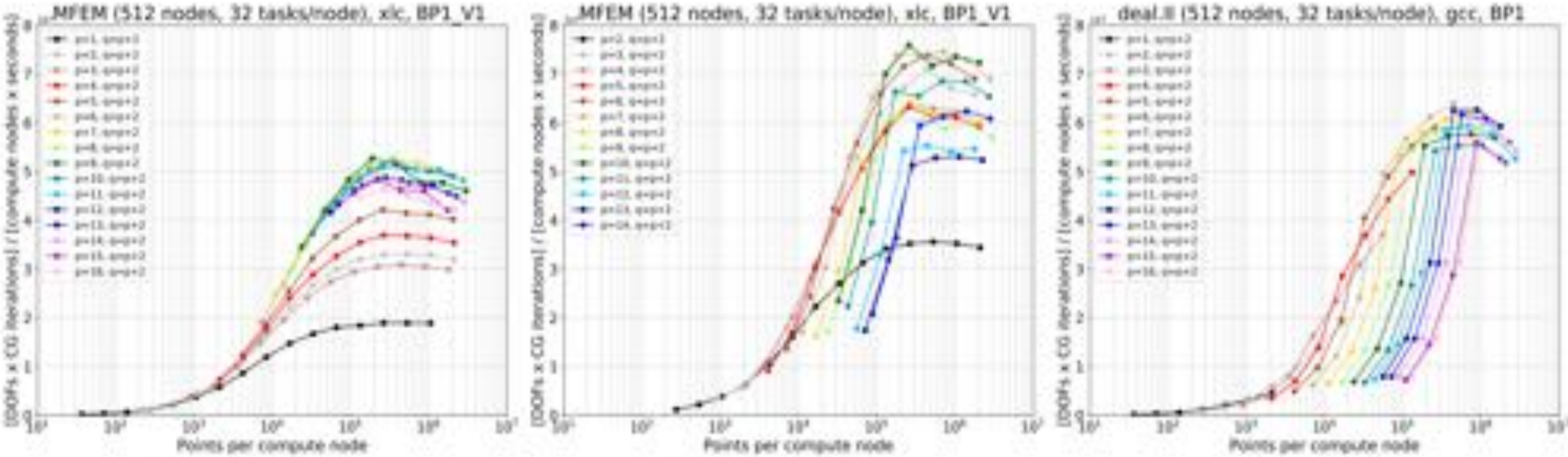
nek5000.mcs.anl.gov

High-performance spectral elements

mfem.org

Scalable high-order finite elements

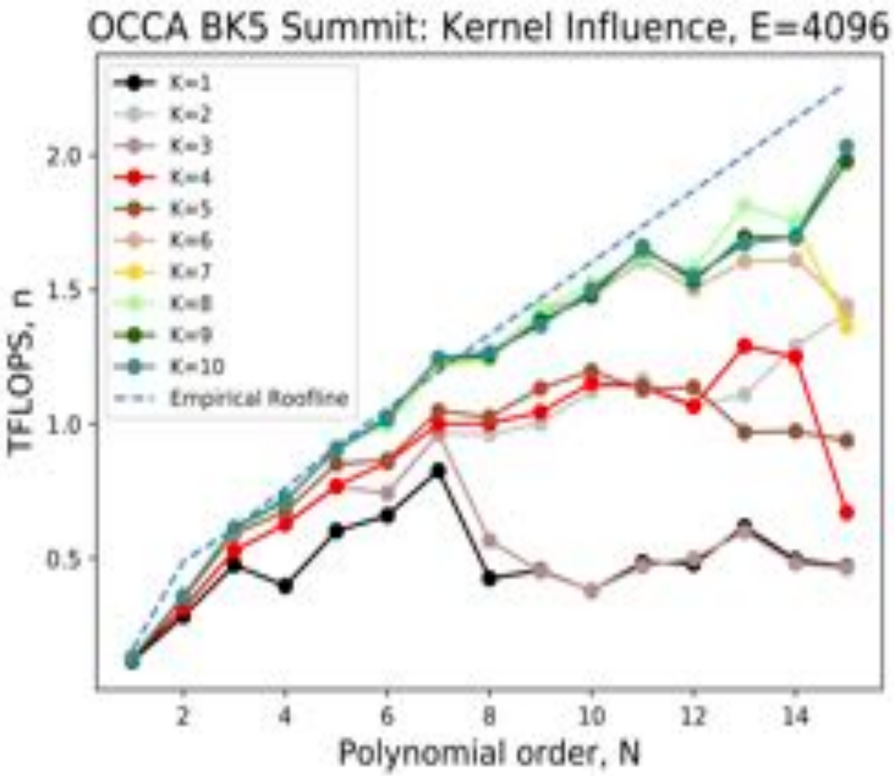
CEED Bake-off Problem 1 on CPU

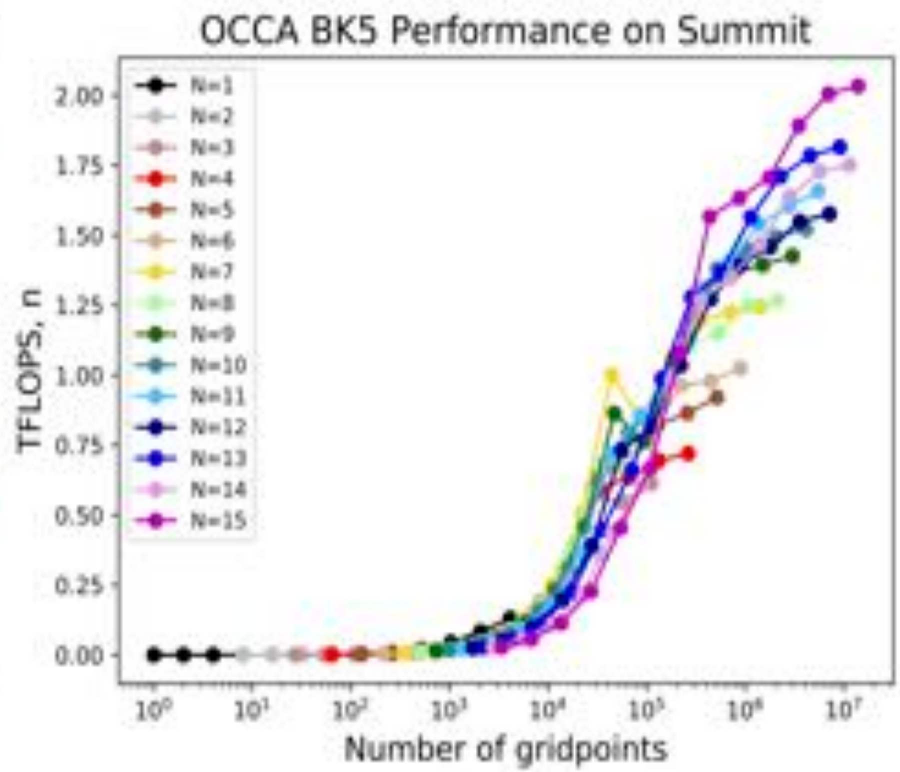


(a) BP1 MFEM-before (b) BP1 MFEM-after (c) BP1 deal.II

- 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.

CEED Bake-off Kernel 5 on GPU





- BK5 – BP5 kernel, just local (unassembled) matvec with E-vectors
- OCCA-based kernels with a lot of sophisticated tuning
- > 2 TFLOPS on single V100 GPU

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

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], G. Slota[4], C.W. Smith[4]

[1]Sandia National Laboratories

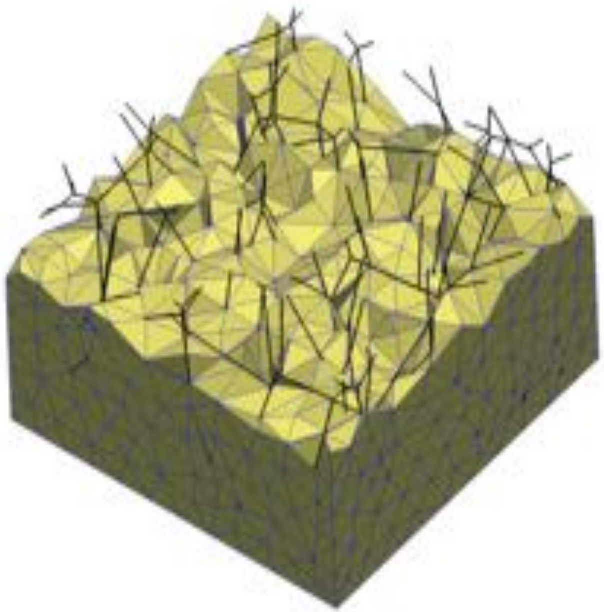
[2]Lawrence Livermore National Laboratory

[3]University of Colorado

[4]Rensselaer Polytechnic Institute

Unstructured Mesh Technologies – To Be Covered

- Background
- Summary of FASTMath development efforts
- Discussion of core parallel mesh support tools (the things other that the unstructured mesh analysis code)
  - Parallel mesh infrastructure
  - Mesh generation/adaptation
  - Dynamic load balancing
  - Unstructured mesh infrastructure for particle-in-cell codes
- Some ongoing applications
- Hands-on demonstration



Unstructured Mesh Methods

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

Advantages

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

Disadvantages

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

FASTMATH

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 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

FASTMATH

FASTMath Unstructured Mesh Developments

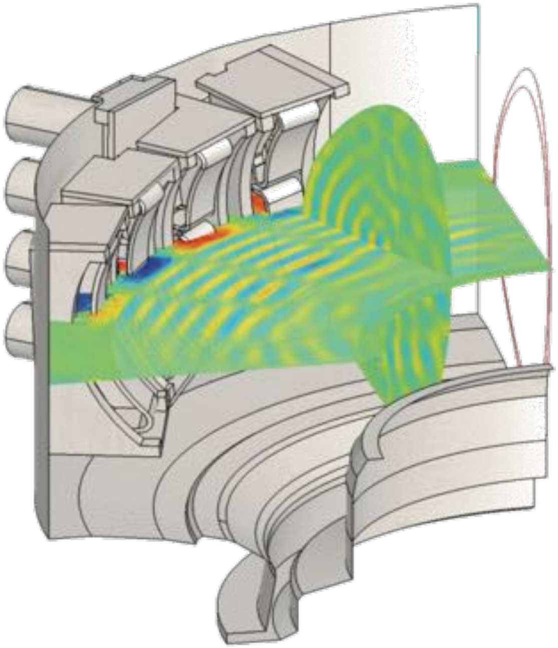
Technology development areas:

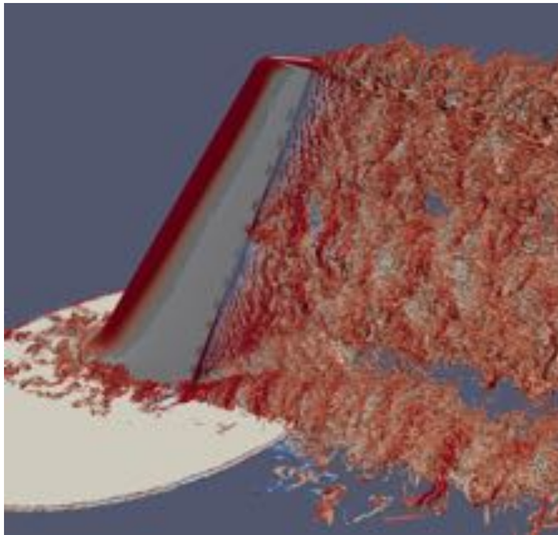
- Unstructured Mesh Analysis Codes – Support application's PDE solution needs
- Performant Mesh Adaptation – Parallel mesh adaptation to integrate into analysis codes to ensure solution accuracy
- Dynamic Load Balancing and Task Management – Technologies to ensure load balance and effectively execute operations by optimal task placement
- 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

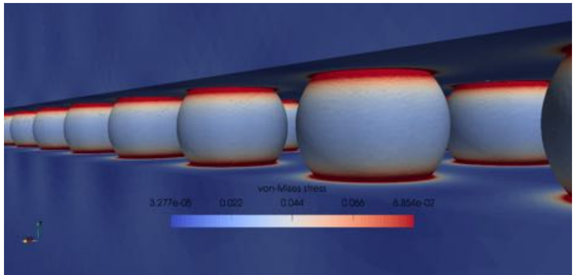
Unstructured Mesh Analysis Codes

Advanced unstructured mesh analysis codes

- MFEM – High-order F.E. framework
  - Arbitrary order curvilinear elements
  - Applications include shock hydrodynamics, Electromagnetic fields in fusion reactors, etc.
- ALBANY – Generic F.E. framework
  - Builds on Trilinos components
  - Applications include ice modeling, non-linear solid mechanics, quantum device modeling, etc.
- PHASTA – Navier Stokes Flow Solver
  - Highly scalable code including turbulence models
  - Applications include nuclear reactors, multiphase flows, etc.



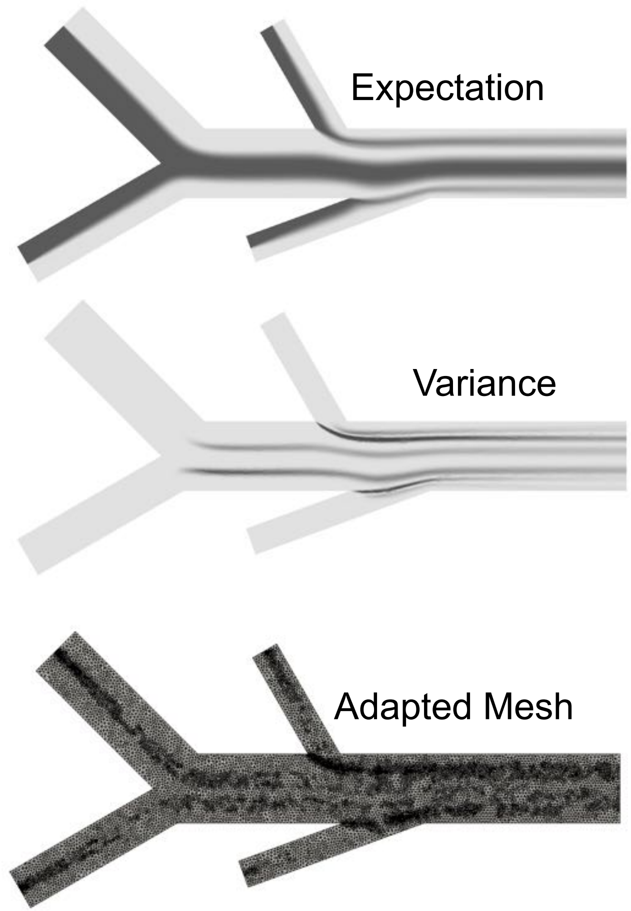




FASTMATH

Unstructured Mesh for Uncertainty Quantification

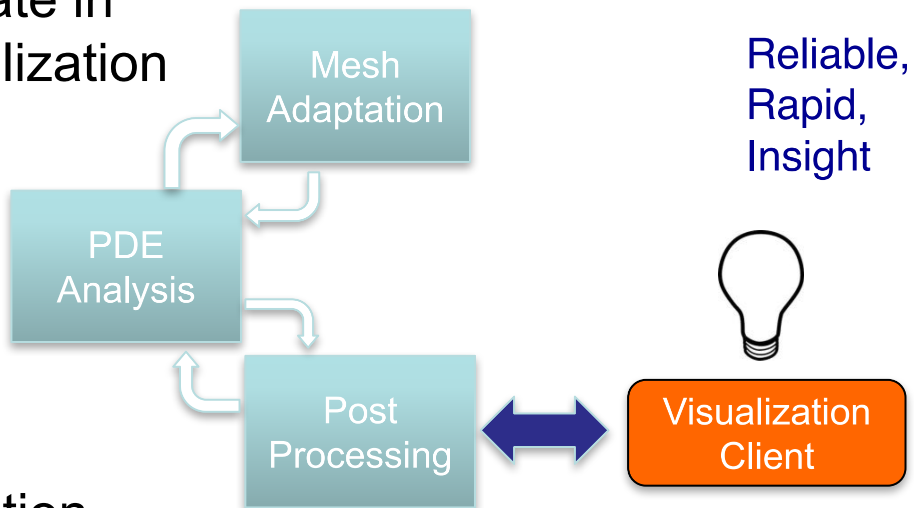
- Adaptive control of discretization a prerequisite for the effective application of UQ operations
- Substantial potential for joint adaptivity in the physical and stochastic domains
  - Preliminary study mesh adaptivity in the physical space with spectral/p-adaptivity in the stochastic space
  - Target of consideration of geometric uncertainty where unstructured meshes will be critical
- Developments
  - Stochastic space error estimators
  - Basis and sample reduction strategies
  - UQ driven load balancing



FASTMATH

In Situ Visualization and Data Analytics

- Solvers scaled to 3M processes producing 10TB/s need in situ tools to gain insight to avoid the high cost involved with saving data
  - Substantial progress made to date in live, reconfigurable, in situ visualization
  - Effort now focused on user steering and data analytics
- Target in situ operations
  - Live, reconfigurable in situ data analytics
  - Live, analyst-guided grid adaptation
  - Scalable data reduction techniques
  - Live, reconfigurable problem definition, including geometry
  - Live, parameter sensitivity analysis for immersive simulation

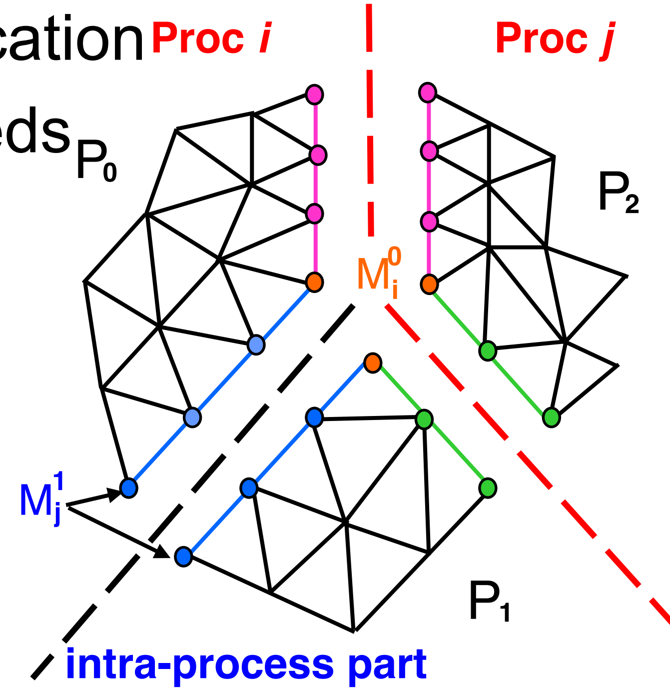


FASTMATH

Parallel Unstructured Mesh Infrastructure

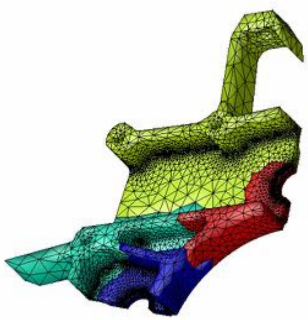
Key unstructured mesh technology needed by applications

- 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

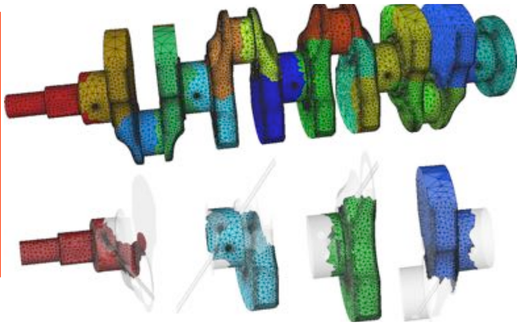

Distributed mesh

FASTMATH

Mesh Generation, Adaptation and Optimization

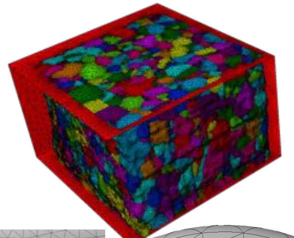
Mesh Generation

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



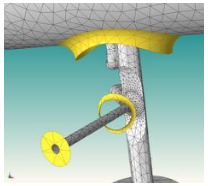
Mesh Adaptation must

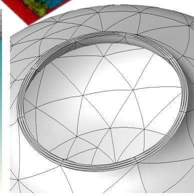
- Use *a posteriori* information to improve mesh
- Account for curved geometry (fixed and evolving)
- Support general, and specific, anisotropic adaptation



Mesh Shape Optimization

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



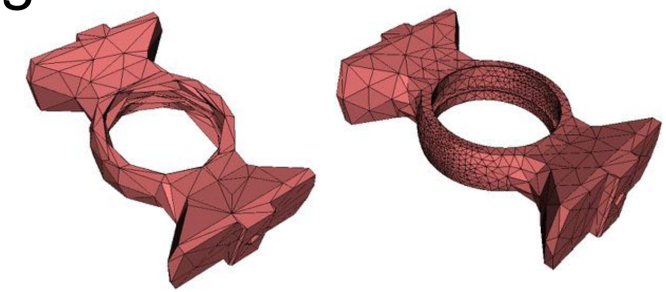


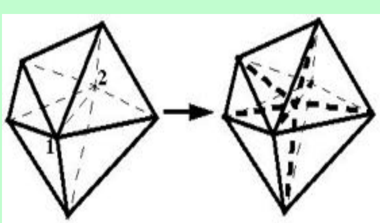
Parallel execution of all three functions critical on large meshes

FASTMATH

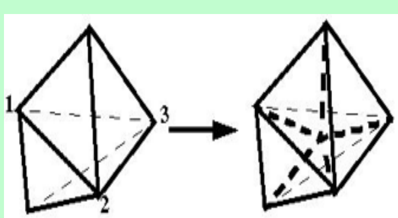
General Mesh Modification for Mesh Adaptation

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

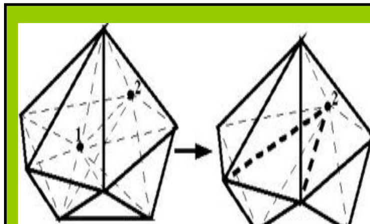




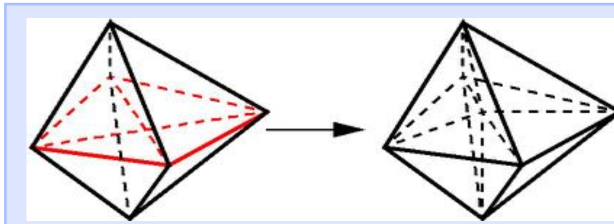
Edge split



face split



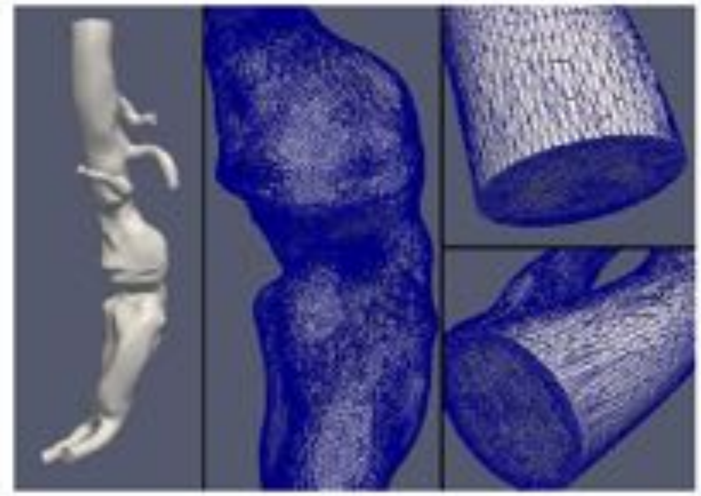
Edge collapse

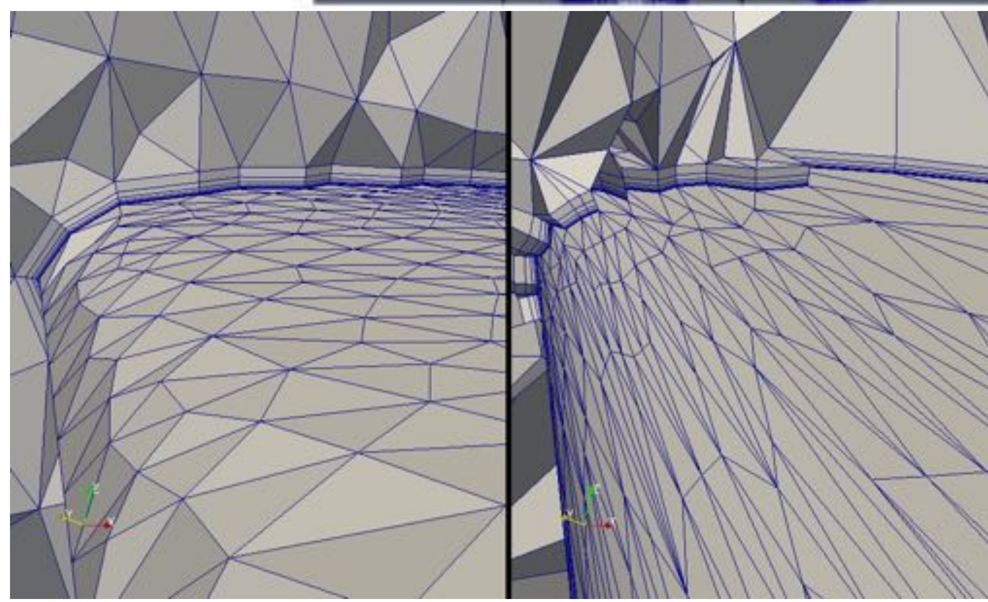


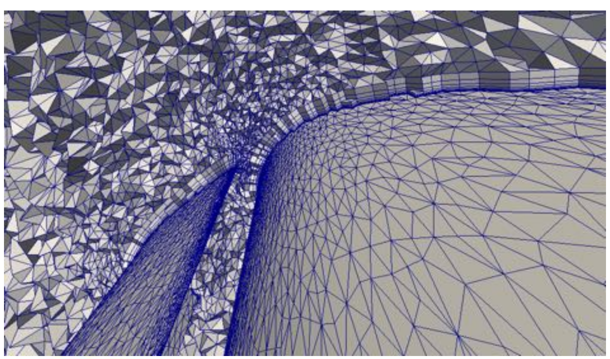
Double split collapse to remove sliver

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



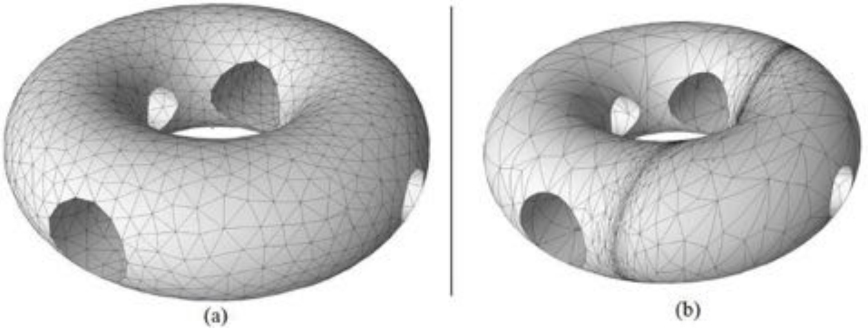


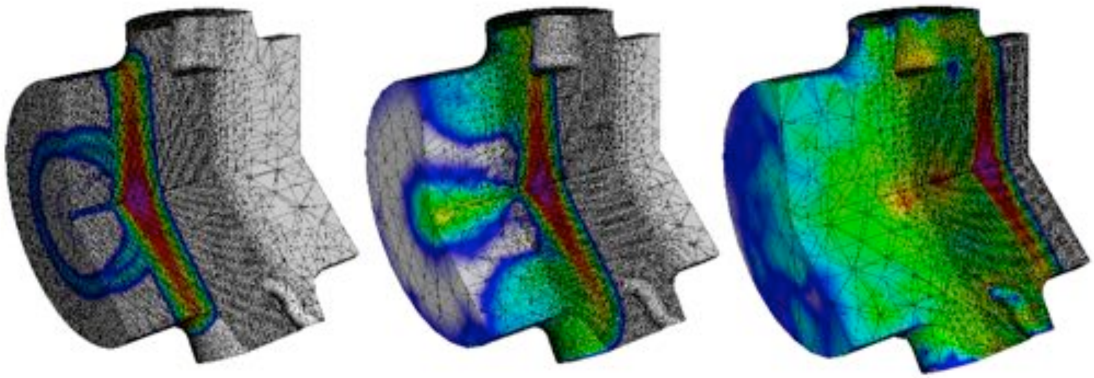


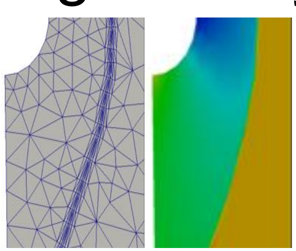
FASTMATH

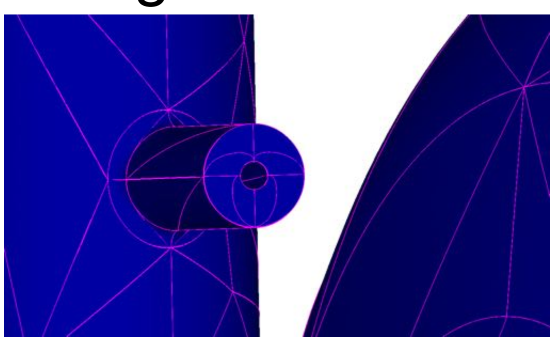
Mesh Adaptation Status

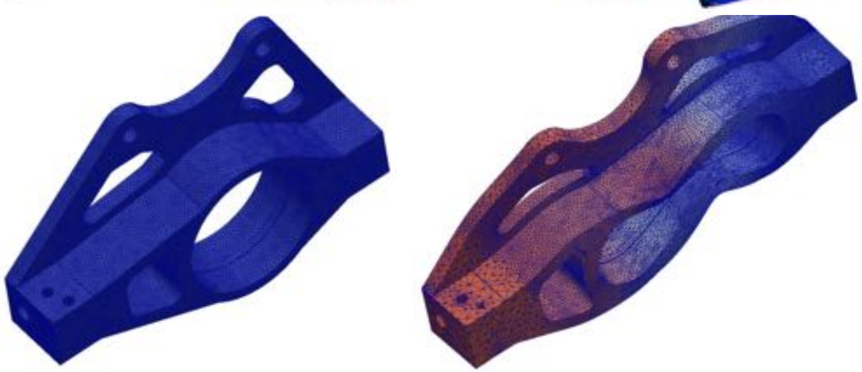
- Supports adaptation of curved elements
- Adaptation based on multiple criteria, examples
  - Level sets at interfaces
  - Tracking particles
  - Discretization errors
  - Controlling element shape in evolving geometry





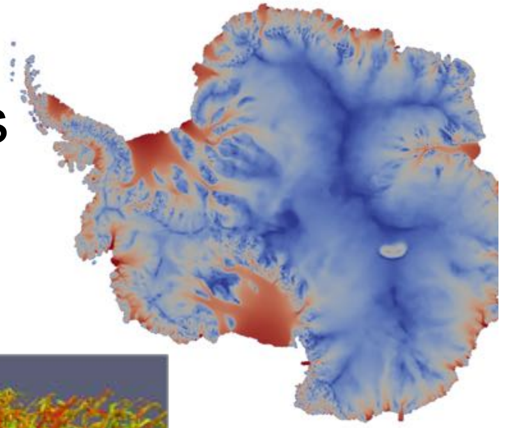


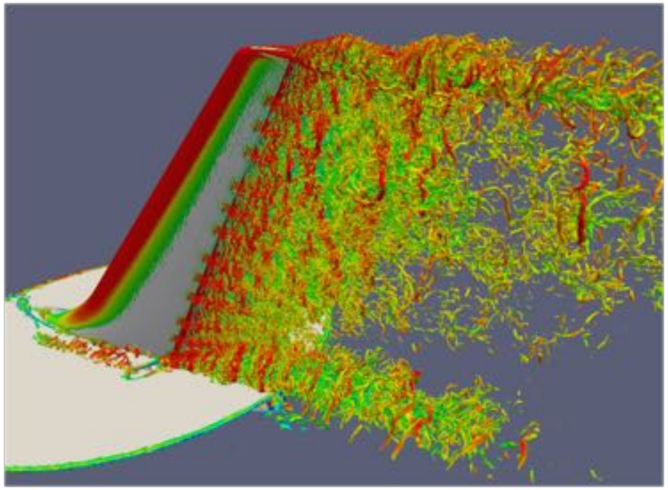




Attached Parallel Fields (APF)

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



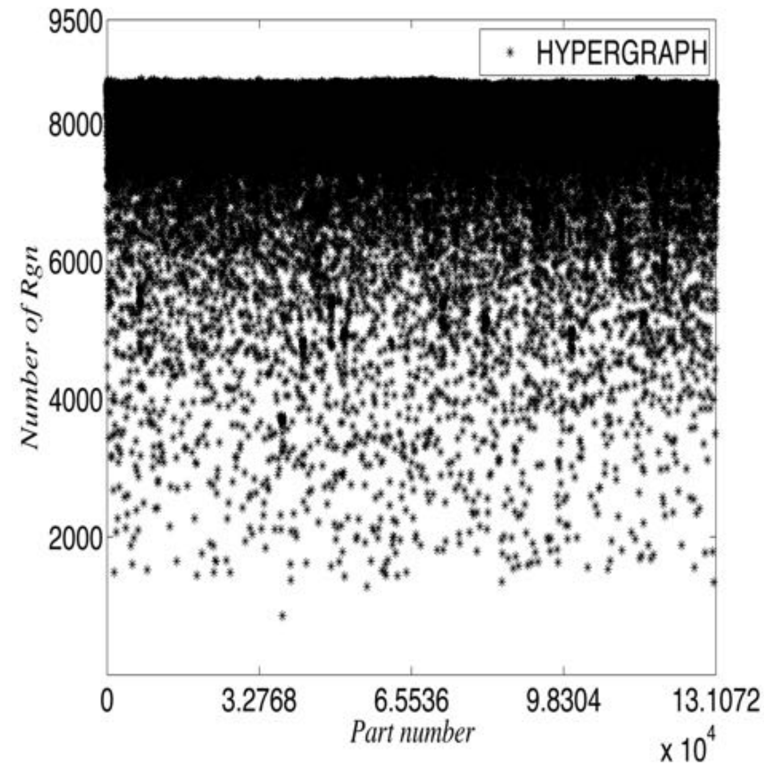


- Example operations
  - Adaptive expansion of Fields from 2D to 3D in M3D-C1
  - History-dependent integration point fields for Albany plasticity models

FASTMATH

Dynamic Load Balancing

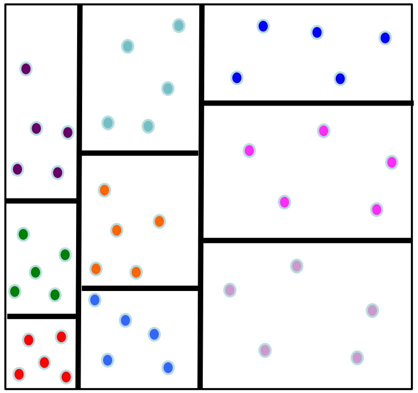
- Purpose: to rebalance load during mesh modification and before each key step in the parallel workflow
  - Equal "work load" with minimum inter-process communications
- FASTMath load balancing tools
  - Zoltan/Zoltan2 libraries provide multiple dynamic partitioners with general control of partition objects and weights
  - EnGPar diffusive multi-criteria partition improvement
  - XtraPuLP scalable graph partitioning



Number of mesh elements in each of 128Ki parts

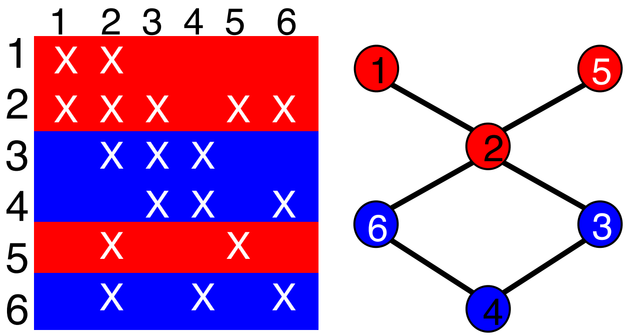
Zoltan/Zoltan2 suite of partitioners supports a wide range of applications

- Geometric: parts contain physically close objects
  - Fast to compute → good for dynamic load balancing
  - Applications: Particle methods, contact detection, adaptive mesh refinement, architecture-aware task mapping
  - Recursive Coordinate/Inertial Bisection, MultiJagged, Space Filling Curve



MultiJagged partition of a particle simulation

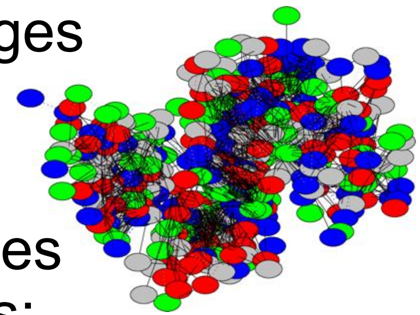
- Topology-based: parts contain topologically connected objects
  - Explicitly model communication costs → higher quality partitions
  - Applications: Mesh-based methods, linear systems, circuits, social networks
  - Graph (interfaces to XtraPuLP, ParMETIS, Scotch)
  - Hypergraph

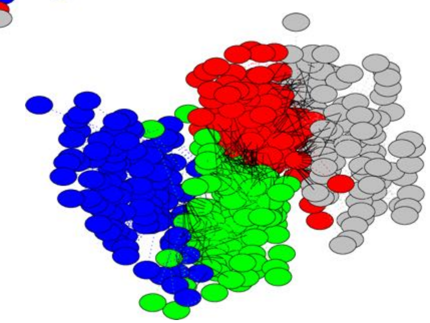


Row-based partition of a sparse matrix via graph partitioning

PuLP / XtraPuLP provide scalable graph partitioning for multicore and distributed memory systems

- PuLP: Shared-memory multi-objective/constraint partitioning
- XtraPuLP: Distributed implementation of PuLP for large-scale and distributed graph processing applications
- Designed to …
  - balance both graph vertices and edges
  - minimize total and maximum communication





- Effective for irregular graphs and meshes containing latent ‘community’ properties; network analysis; information graph processing
- Interface in Zoltan2
- Library and source at: https://github.com/HPCGraphAnalysis/PuLP

FASTMATH

Dynamic Load Balancing for Adaptive Workflows

At >16Ki ranks, existing tools providing multi-level graph methods consume too much memory and fail; geometric methods have high cuts and are inefficient for analysis.

An approach that combines existing methods with **ParMA** diffusive improvement accounts for multiple criteria:

- Accounts for DOF on any mesh entity
- Analysis and partitioning is quicker

Goal of current **EnGPar** developments is to generalize methods

- Take advantage of graph methods and new hardware
- Broaden the areas of application to new applications (mesh based and others)

Partitioning to 1M Parts

Multiple tools needed to maintain partition quality at scale

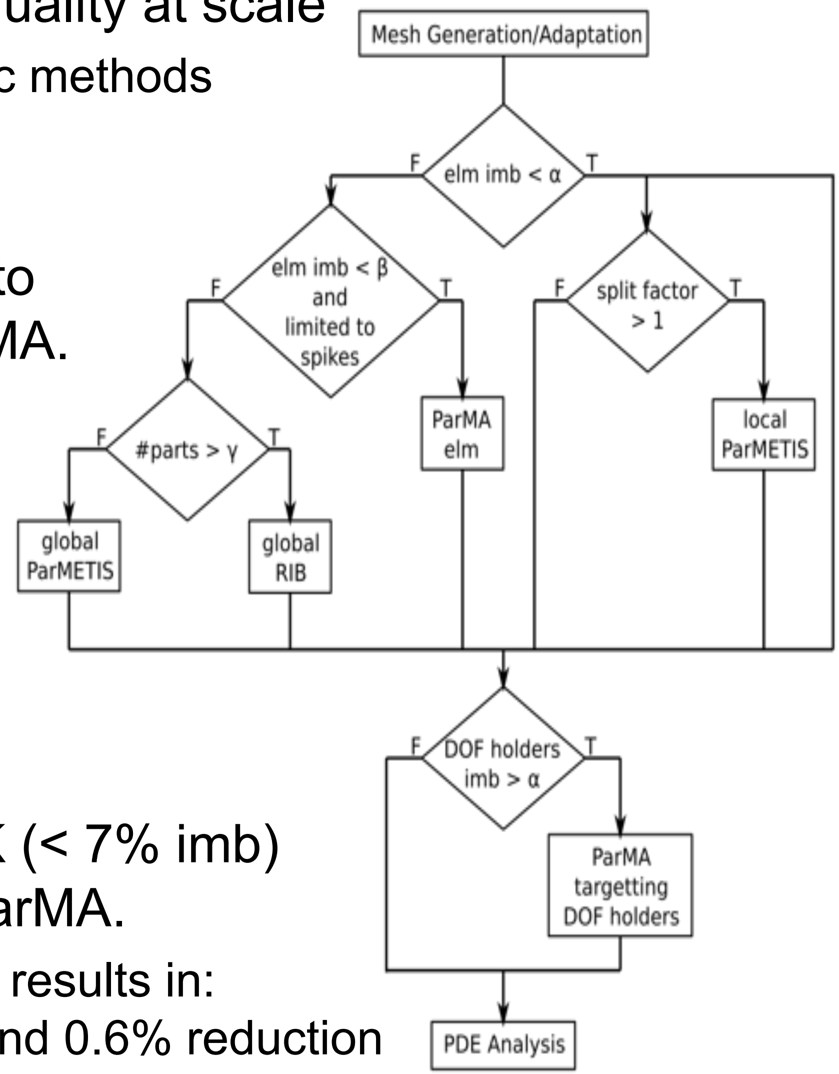
- Local and global topological and geometric methods
- ParMA quickly reduces large imbalances and improves part shape

Partitioning 1.6B element mesh from 128K to 1M parts (1.5k elms/part) then running ParMA.

- Global RIB - 103 sec, ParMA - 20 sec: 209% vtx imb reduced to 6%, elm imb up to 4%, 5.5% reduction in avg vtx per part
- Local ParMETIS - 9.0 sec, ParMA - 9.4 sec results in: 63% vtx imb reduced to 5%, 12% elm imb reduced to 4%, and 2% reduction in avg vtx per part

Partitioning 12.9B element mesh from 128K (< 7% imb) to 1Mi parts (12k elms/part) then running ParMA.

- Local ParMETIS - 60 sec, ParMA - 36 sec results in: 35% vtx imb to 5%, 11% elm imb to 5%, and 0.6% reduction in avg vtx per part



FASTMATH

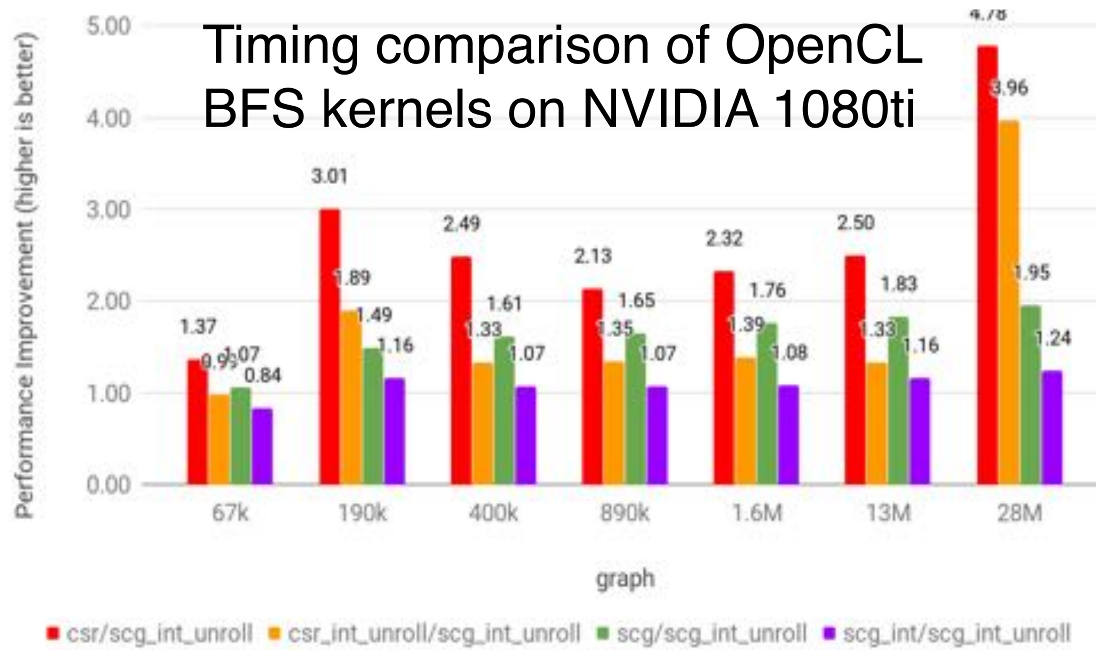
Operation on Accelerator Supported Systems

EnGPar based on more standard graph operations than ParMA

- GPU based breath first traversals

scg_int_unroll is 5 times faster than csr on 28M graph and up to 11 times faster than serial push on Intel Xeon (not shown).

Timing comparison of OpenCL BFS kernels on NVIDIA 1080ti



Developments:

- Different layouts (CSR, Sell-C-Sigma), support migration
- Accelerate selection using coloring
- Focus on pipelined kernel implementations for FPGAs

FASTMATH

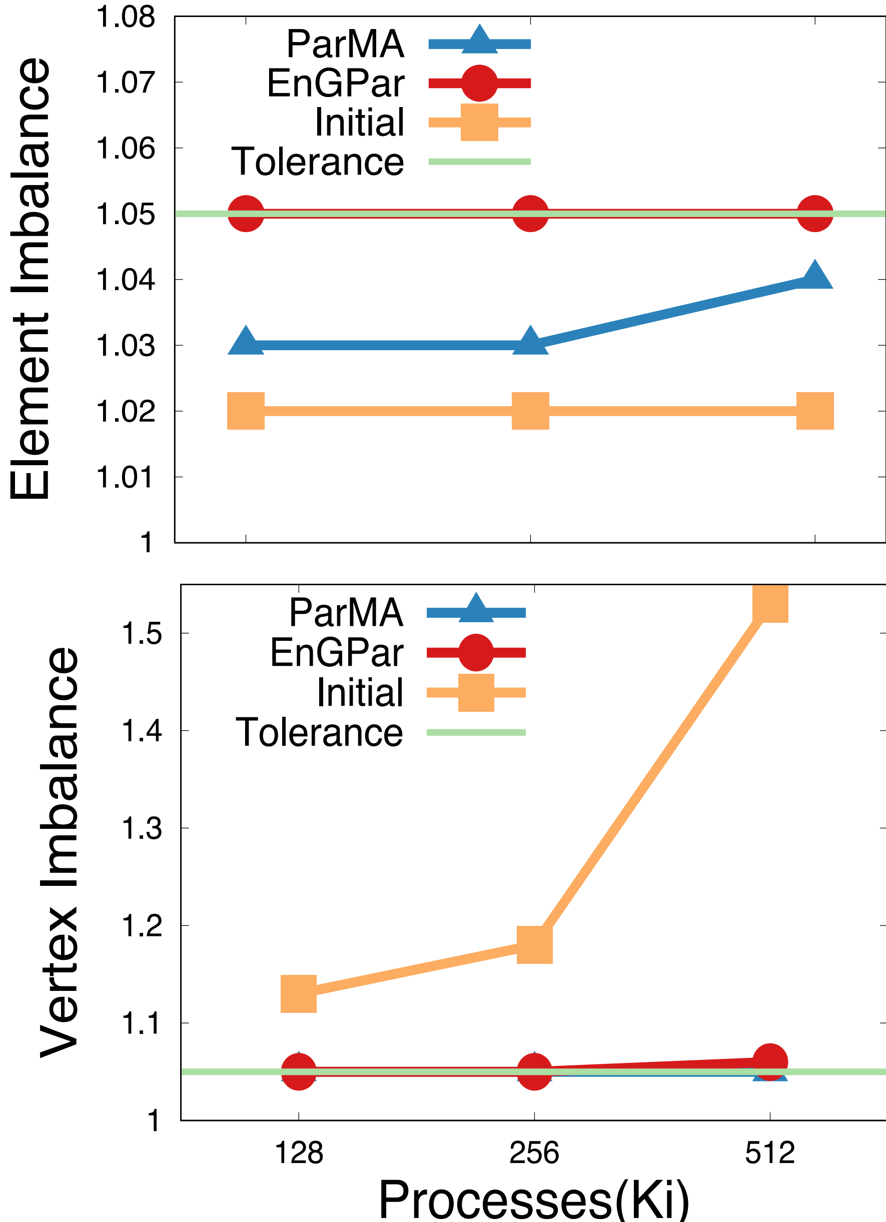
EnGPar for Conforming Meshes

Tests run on billion element mesh on Mira BlueGene/Q

- Global ParMETIS part k-way to 8Ki
- Local ParMETIS part k-way from 8Ki to 128Ki, 256Ki, and 512Ki parts

Imbalances after running EnGPar vtx>elm are shown

- Creating the 512Ki partition from 8Ki parts takes 147 seconds with ParMETIS (including migration)
- EnGPar reduces a 53% vertex imbalance to 5% in 7 seconds on 512Ki processes. ParMA requires 17 seconds.



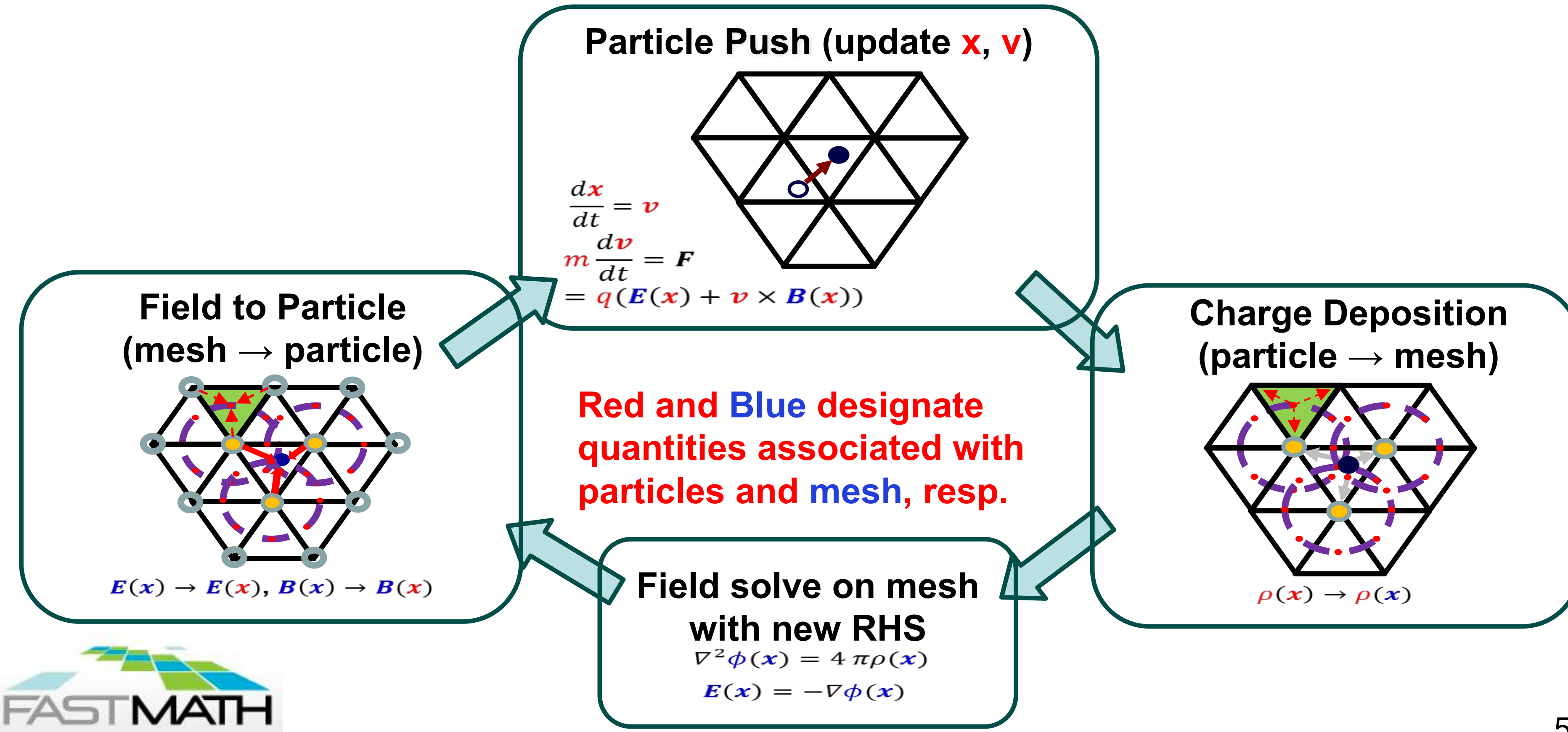
FASTMATH

Parallel Unstructured Mesh PIC – PUMIpic

Current approaches have copy of entire mesh on each process

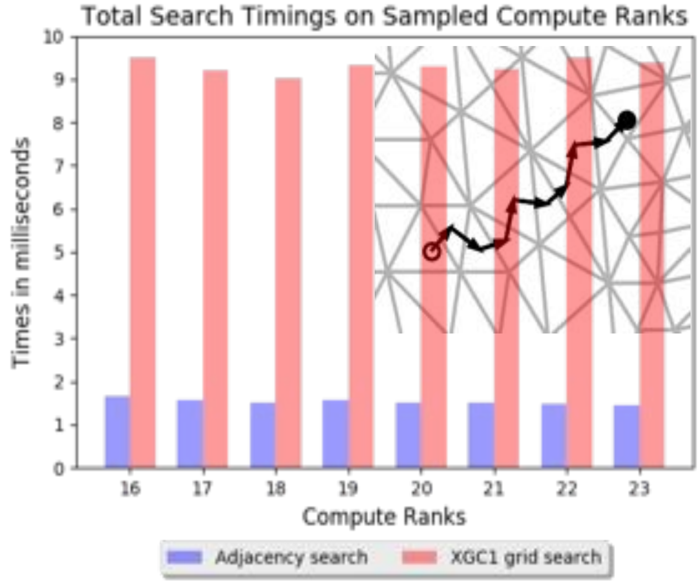
PUMIpic supports a distributed mesh

- 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
  - M3D-C1 – Core Plasma

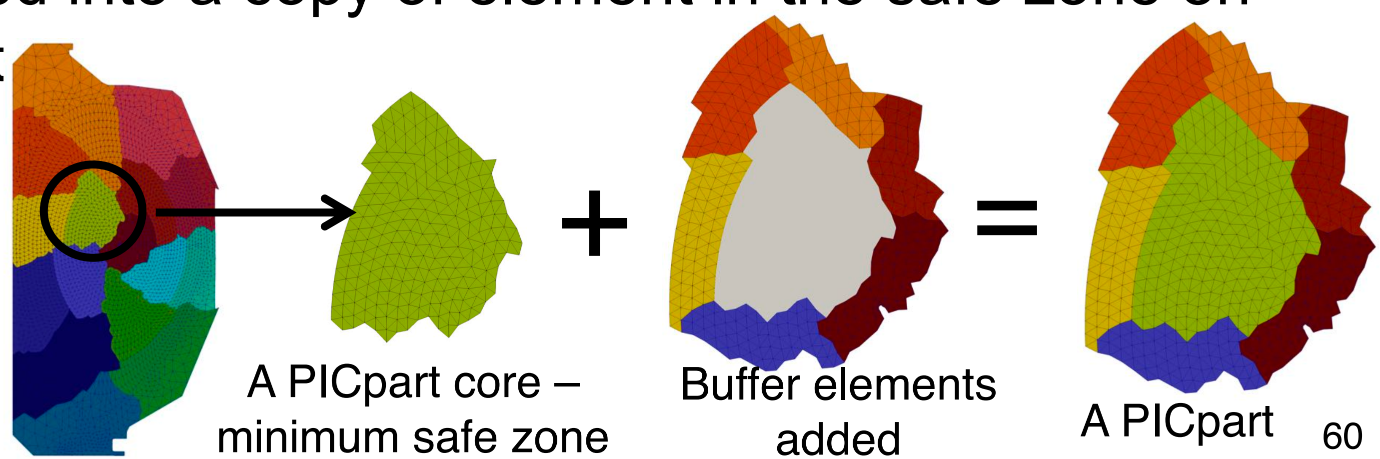


Require knowledge of element that particle is in after push

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

Construction of Distributed Mesh

- Steps to construct PICparts:
  - Define non-overlapping mesh partition considering the needs of the physics/numerics of the PIC code
  - Add overlap to safely ensure particles remain on PICpart during a push
  - Evaluate PICpart safe zone: Defined as elements for which particles are “safe” for next push (no communication) – must be at least original core, preferably larger
- After a Push particles that move out of a safe zone element must be migrated into a copy of element in the safe zone on another PICpart

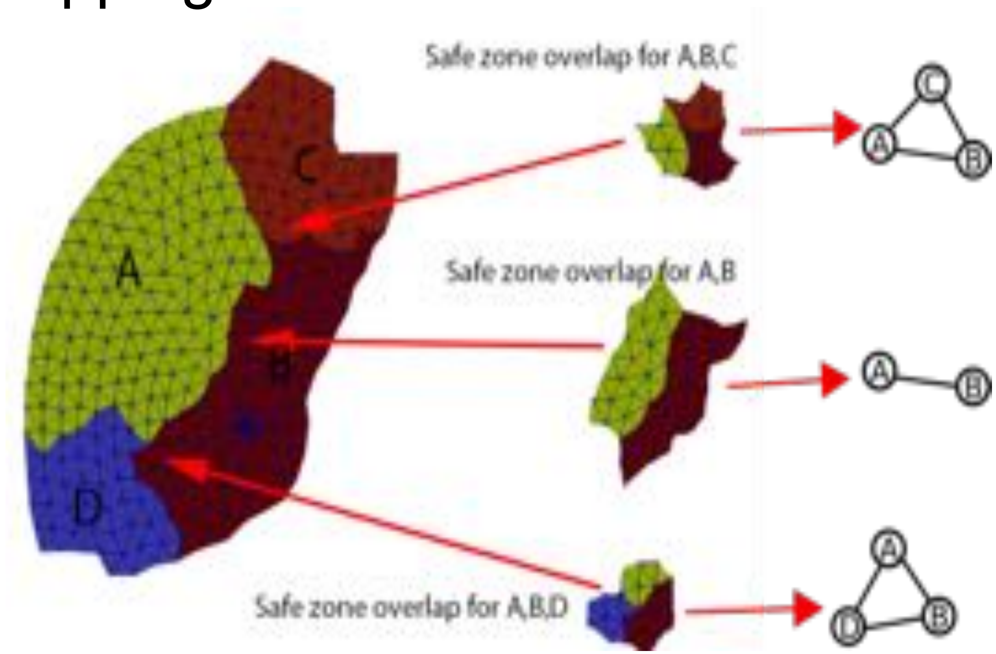


Dynamic Load balancing

Load balance can be lost as particles migrate

Use EnGPar to migrate particles for better load balance

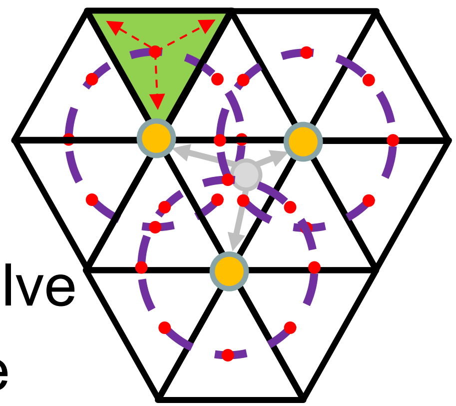
- Construct subgraphs connecting processes for each overlapping safe zone
- Set the weights of vertices to be the number of particles in the elements for the overlapping safe zone
- Diffusively migrate weight (# of particles) in each subgraph until processes are balanced

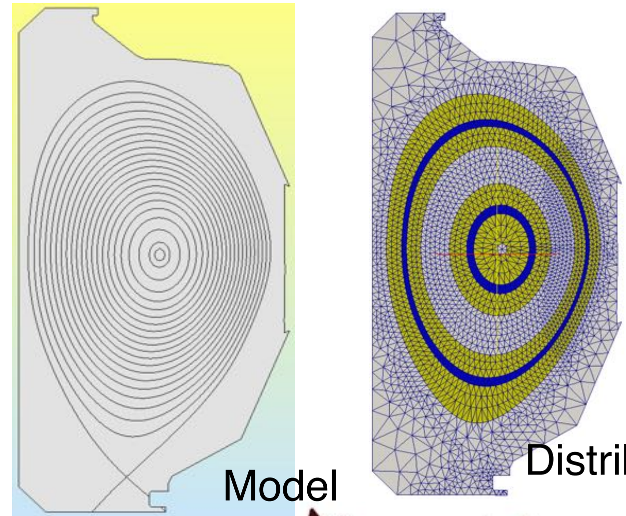


Overlapping safe zones between parts A,B,C, and D near the A-B boundary

PUMIpic for XGC Gyrokinetic Code

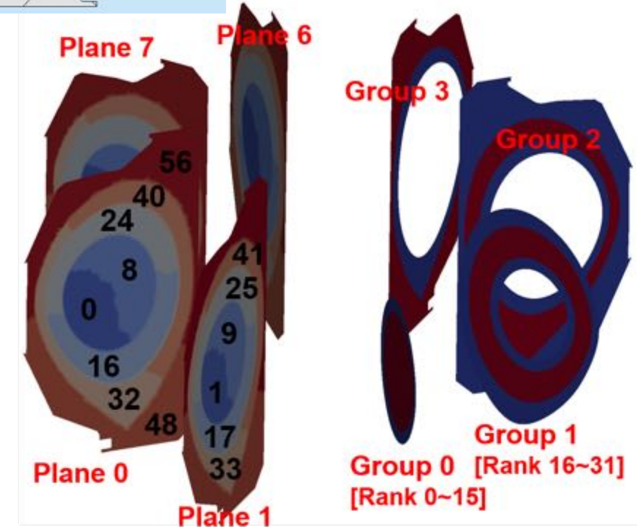
- XGC uses a 2D poloidal plane mesh considering particle paths
  - Mesh distribution takes advantage of physics defined model/mesh
  - Separate parallel field solve on each poloidal plane
- XGC gyro-averaging for Charge-to-Mesh
- PETSc used for field solve
  - Solves on each plane
  - Mesh partitioned 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)

FASTMATH

Building In-Memory Parallel Workflows

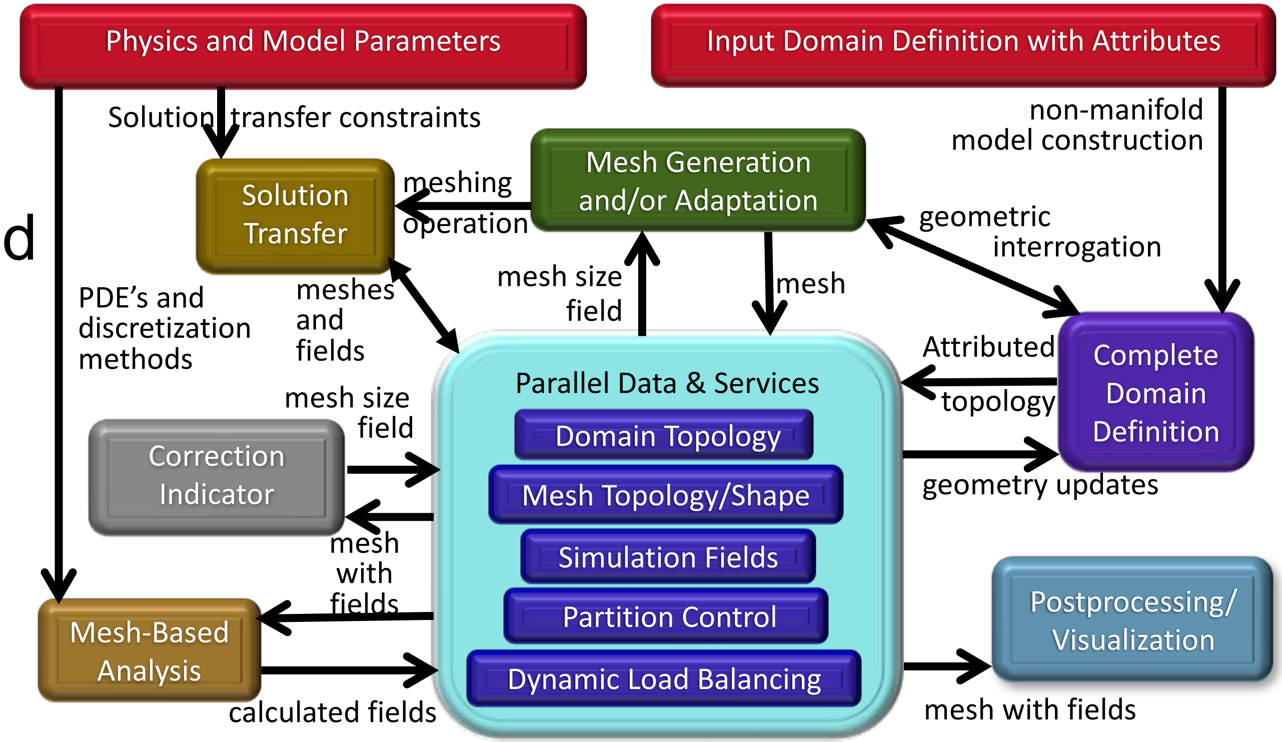
A scalable workflow requires effective component coupling

- Avoid file-based information passing
  - On massively parallel systems I/O dominates power consumption
  - Parallel file system technologies lag behind performance of processors and interconnects
  - Unlike compute nodes, the file system resources are shared and performance can vary significantly
- Use APIs and data-streams to keep inter-component information transfers and control in on-process memory
  - Component implementation drives the selection of an in-memory coupling approach
  - Link component libraries into a single executable

Creation of Parallel Adaptive Loops

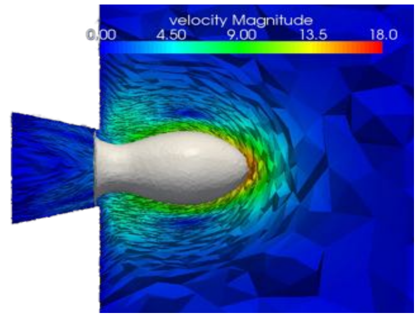
Parallel data and services are the core

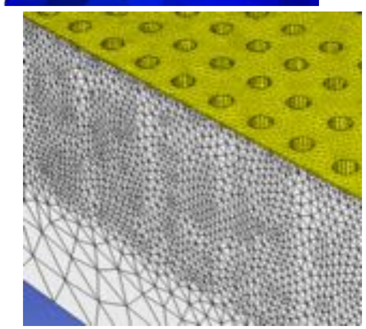
- Geometric model topology for domain linkage
- Mesh topology – it must be distributed
- Simulation fields distributed over geometric model and mesh
- Partition control
- Dynamic load balancing required at multiple steps
- API's to link to
  - CAD
  - Mesh generation and adaptation
  - Error estimation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

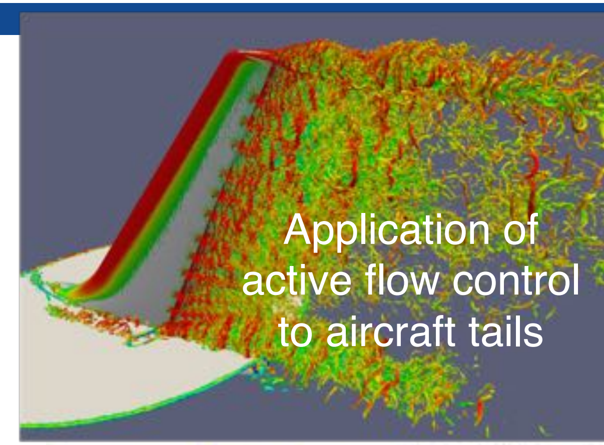


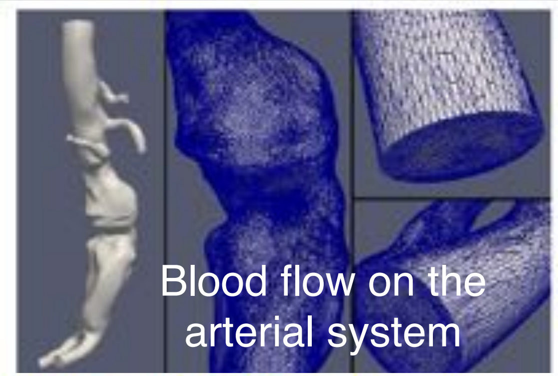
Parallel Adaptive Simulation Workflows

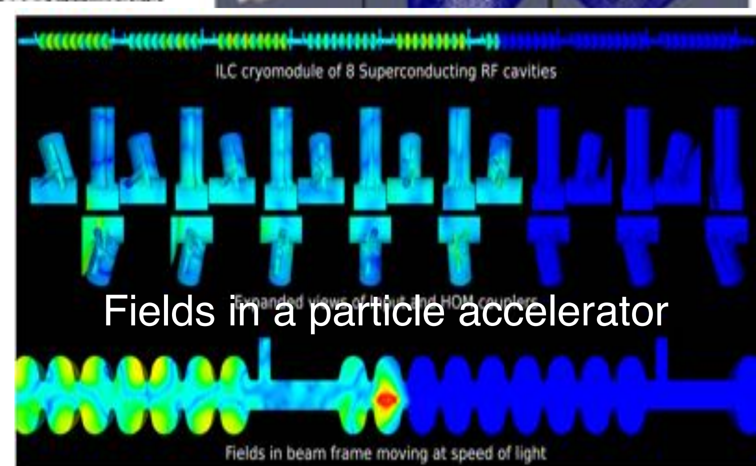
- Automation and adaptive methods critical to reliable simulations
- In-memory examples
  - MFEM – High order FE framework
  - PHASTA – FE for NS
  - FUN3D – FV CFD
  - Proteus – multiphase FE
  - Albany – FE framework
  - ACE3P – High order FE electromagnetics
  - M3D-C1 – FE based MHD
  - Nektar++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





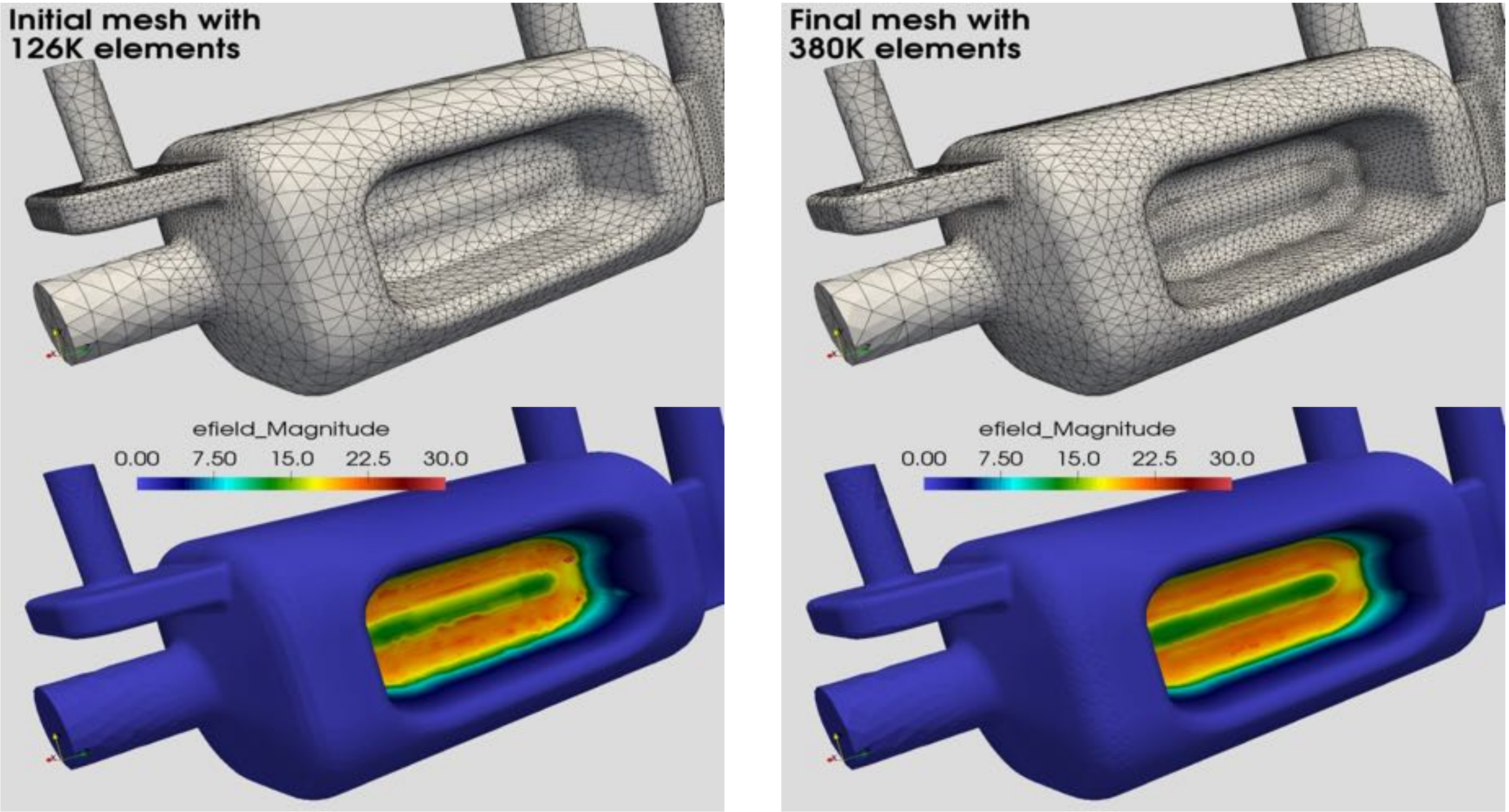






Application interactions – Accelerator EM

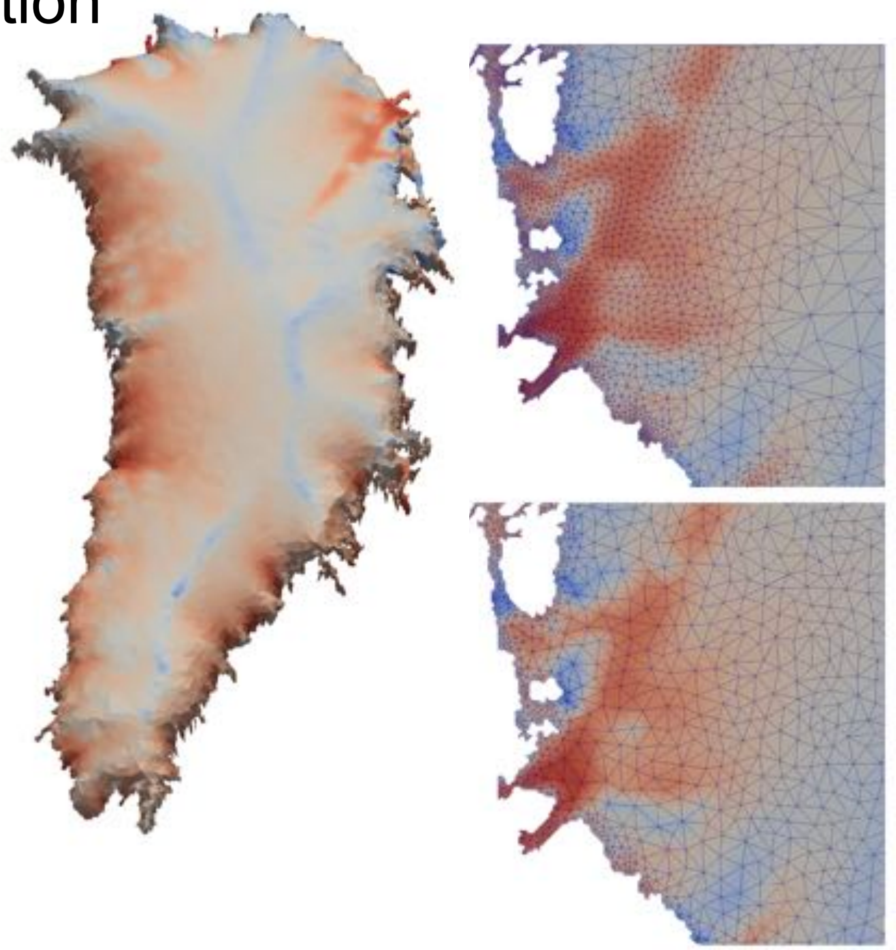
Omega3P Electro Magnetic Solver (second-order curved meshes)



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

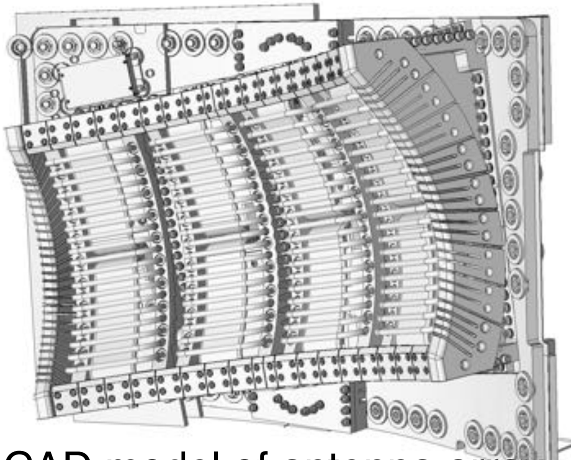
- FELIX, a component of the Albany framework is the analysis code
- Omega_h parallel mesh adaptation is integrated with Albany to do:
  - Estimate error
  - Adapt the mesh
- Ice sheet mesh is modified to minimize degrees of freedom
- Field of interest is the ice sheet velocity



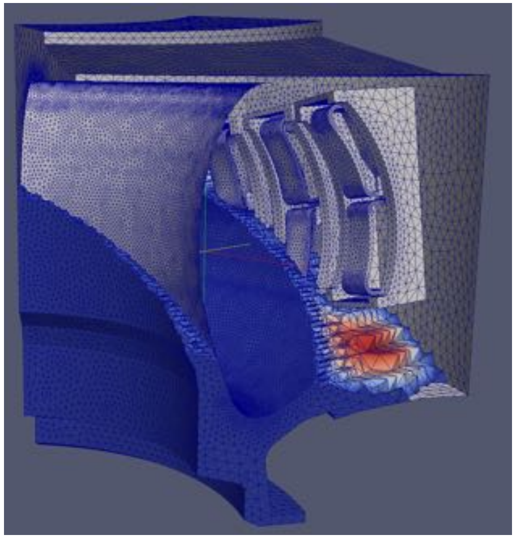
FASTMATH

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
  - Integration with up-stream and down-stream simulation codes



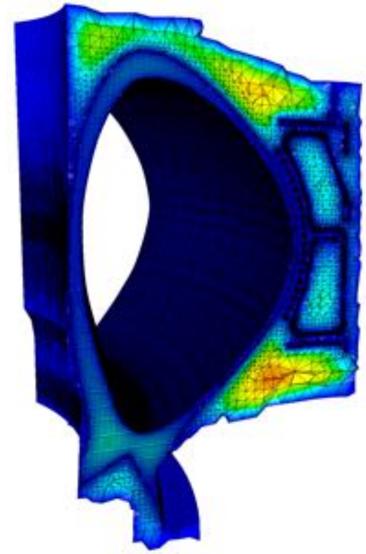
CAD model of antenna array



Simplified antenna array and plasma surface merged into reactor geometry and meshed

FASTMATH

Integration of PUMI/MeshAdapt into MFEM



MFEM ideally suited to address RF simulation needs

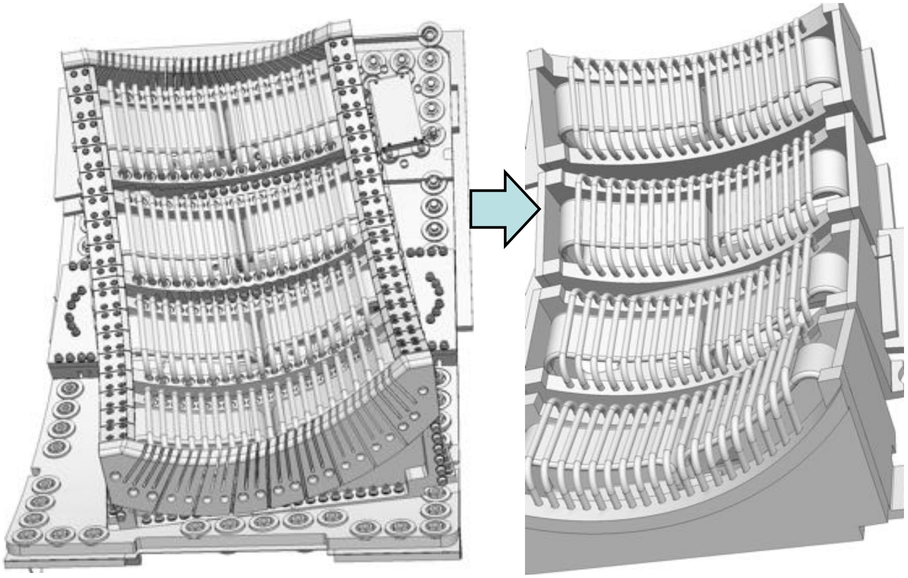
- Higher convergence rates of high-order methods can effectively deliver needed level of accuracy
- Well demonstrated scalability
- Frequency domain EM solver developed

Components integrated

- Curve straight sided meshes – includes mesh topology modification – just curving often yields invalid elements)
- Element geometry inflation up to order 6
- PUMI parallel mesh management
- Curved mesh adaptation based on mesh modification
- EngPar for mesh partition improvement

FASTMATH

Geometry and Meshing for RF Simulations

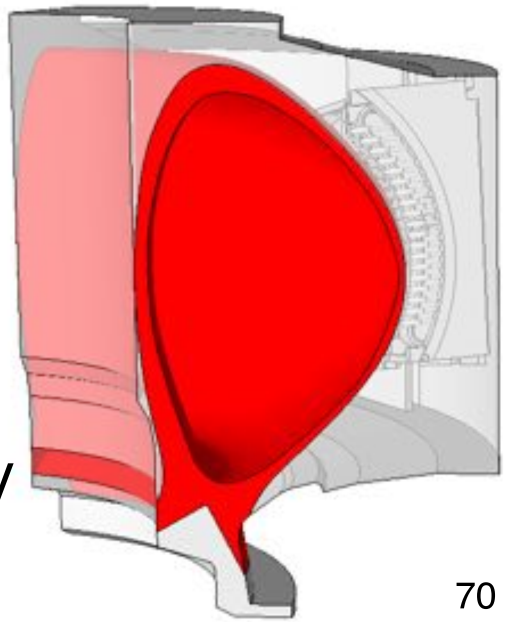


De-featuring Antenna CAD:

- Models have unneeded details
- SimModeler provides tools to “de-feature” CAD models
- Bolts, mounts & capping holes removed

Combining Geometry:

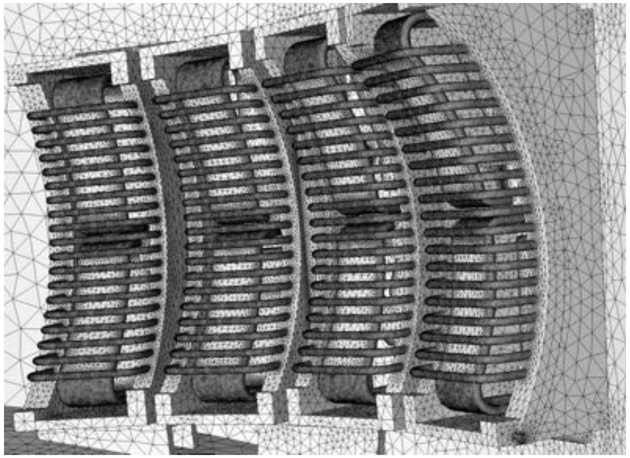
- Import components:
  - De-featured CAD assemblies
  - EFIT curves for SOL (psi = 1.05)
  - TORIC outer surface mesh
- Create rotated surfaces from cross section
- Assemble components into analysis geometry



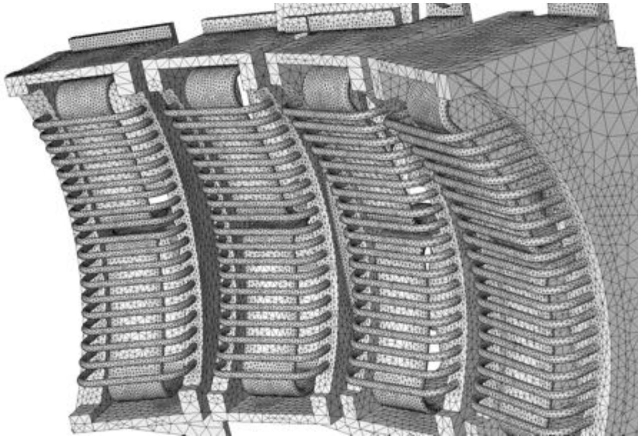
FASTMATH

Geometry and Meshing for RF Simulations

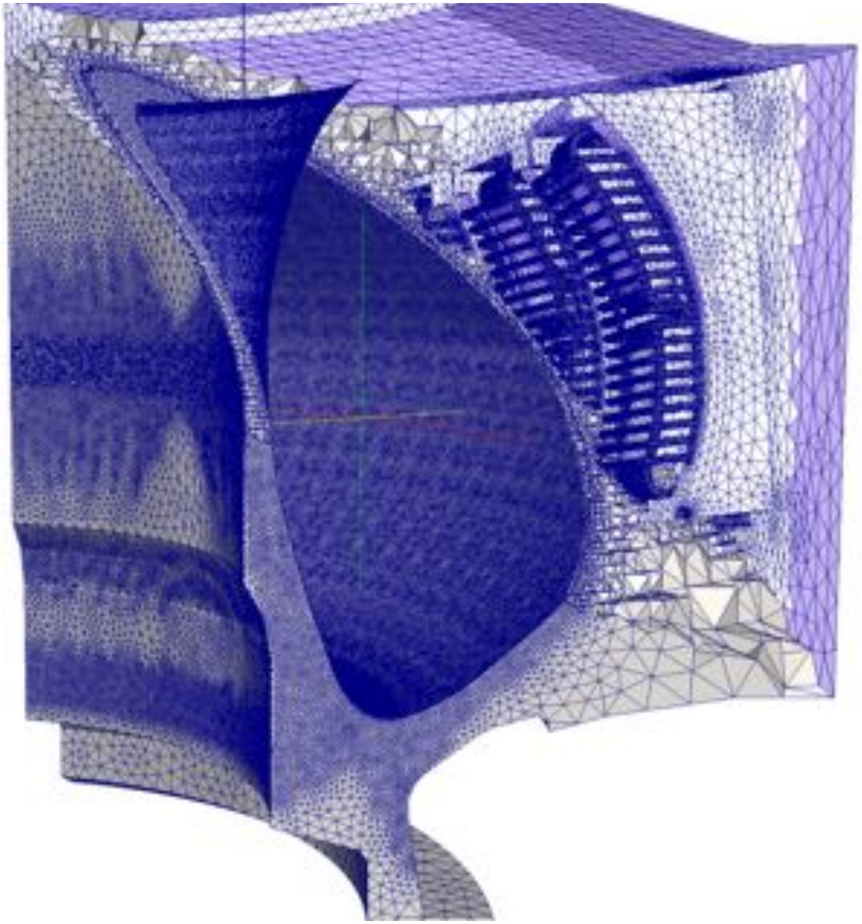
- Mesh controls set on Analysis Geometry
- Mesh generation – linear or or quadratic curved meshed
- Order inflation up to 6th order



Linear mesh
8M elements



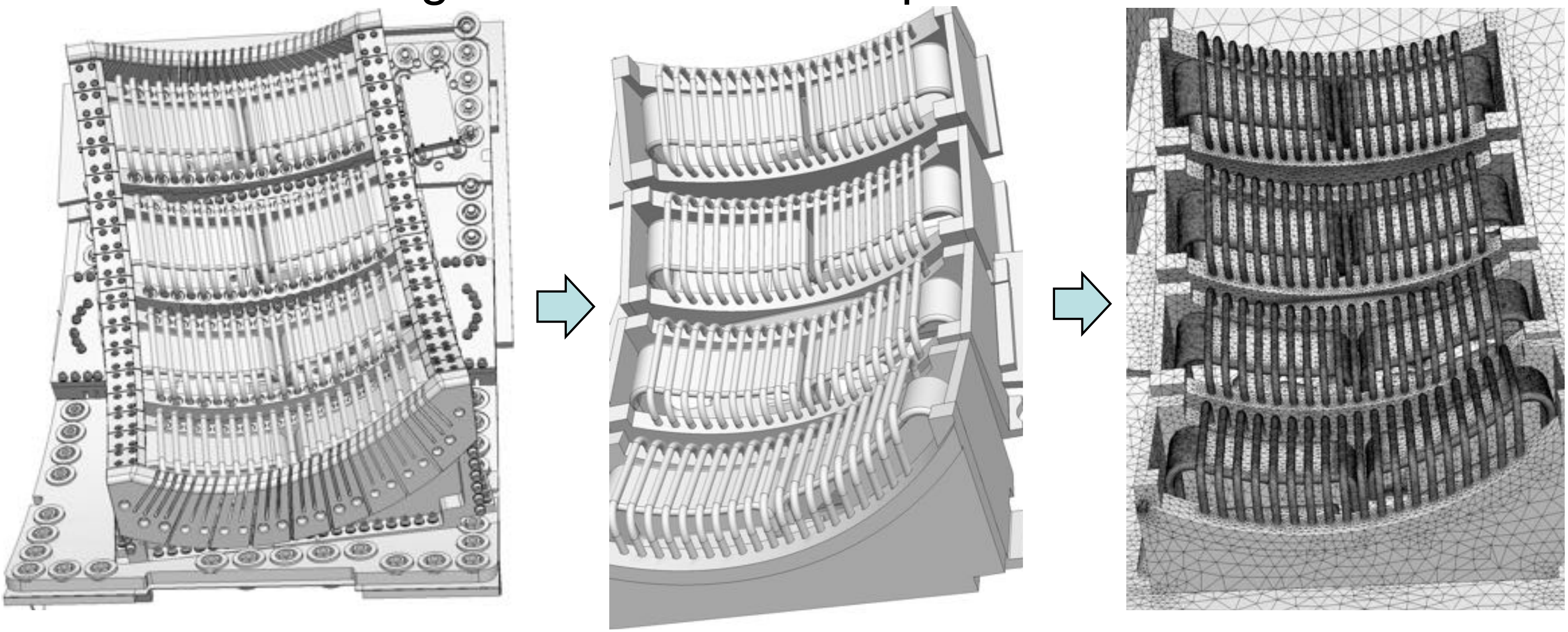
Quadratic mesh
2.5M elements



8M elements mesh
with refined SOL

Hands-on Exercise: Workflow Introduction

Exercising Simmetrix and PUMI tools for model preparation and mesh generation on a complex CAD model



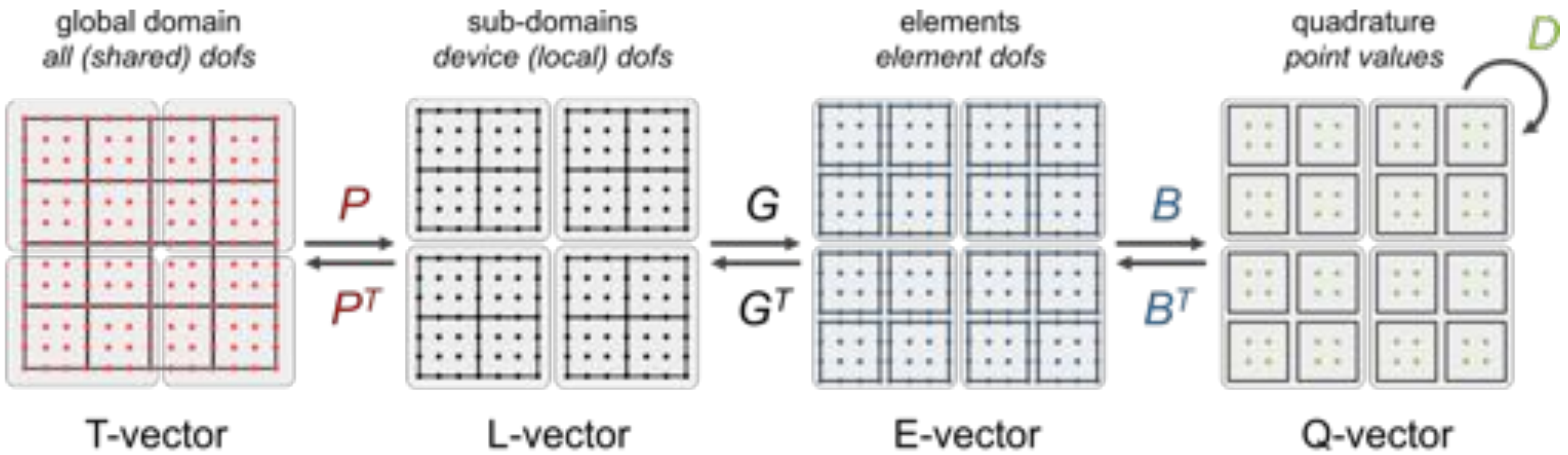
https://xsdk-project.github.io/ATPESC2018HandsOnLessons/lessons/pumi/

MFEM – Extra Slides

Fundamental finite element operator decomposition

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

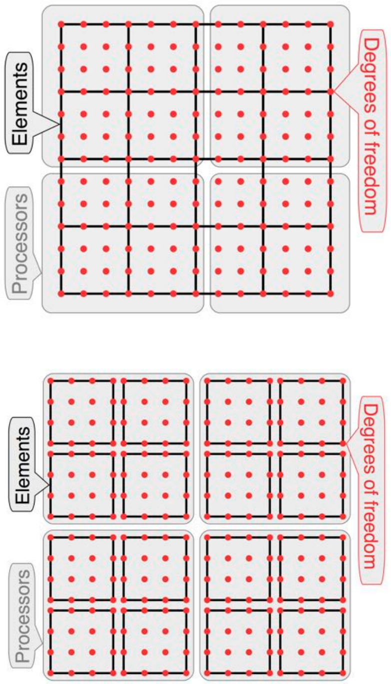
$$\boxed{A = P^T G^T B^T D B G P}$$



- ***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

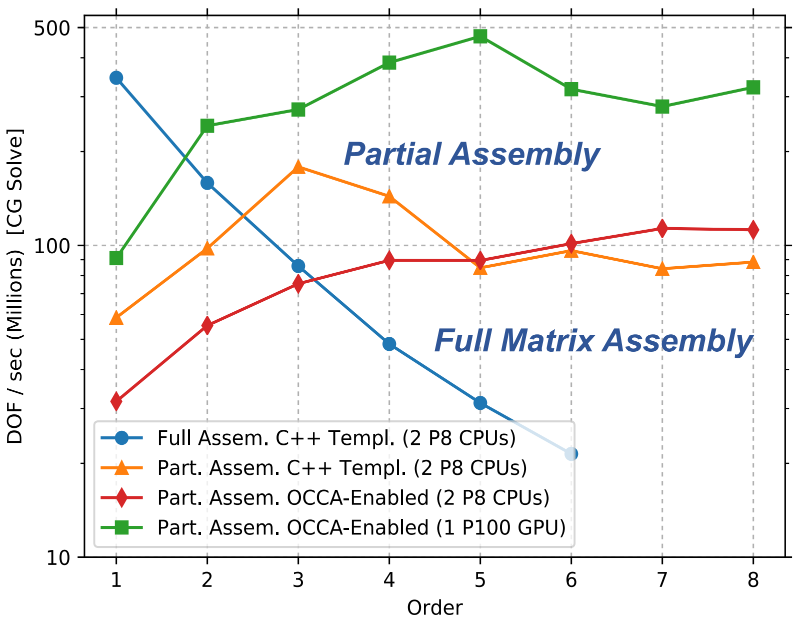
Tensorized partial assembly

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

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!