FASTMATH

FASTMath Unstructured Mesh Technologies

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

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

Argonne National Laboratory · Berkeley Lab · Oak Ridge National Laboratory · Sandia National Laboratories · NREL National Renewable Energy Laboratory · Sustainable Horizons Institute

Massachusetts Institute of Technology · Rensselaer · SMU · USC University of Southern California · The University of Tennessee Knoxville

Unstructured Mesh Methods

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

Advantages

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

Disadvantages

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

Unstructured Mesh Methods

Goal of FASTMath unstructured mesh developments include:

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

FASTMath Unstructured Mesh Development Areas

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

FASTMath Unstructured Mesh Tools and Components

- FE Analysis codes:
  - MFEM (https://mfem.org/): High-order exascale finite element library.
  - LGR (https://github.com/SNLComputation/lgrtk): Tool Kit for Lagrangian grid reconnection.
  - PHASTA (https://github.com/phasta/phasta): Stabilized finite element fluid dynamics code.
- Load balancing, task placement:
  - Jet (https://github.com/sandialabs/Jet-Partitioner/): Parallel graph partitioner that runs on most CPU and GPU systems.
  - Zoltan (https://github.com/sandialabs/Zoltan): Dynamic load balancing library.
  - Zoltan2 (https://github.com/trilinos/Trilinos/tree/master/packages/zoltan2): A package of combinatorial algorithms for scientific computing.
  - PUMI-Balance (http://scorec.github.io/EnGPar/): A hyper-graph based parallel unstructured mesh dynamic partition improvement component

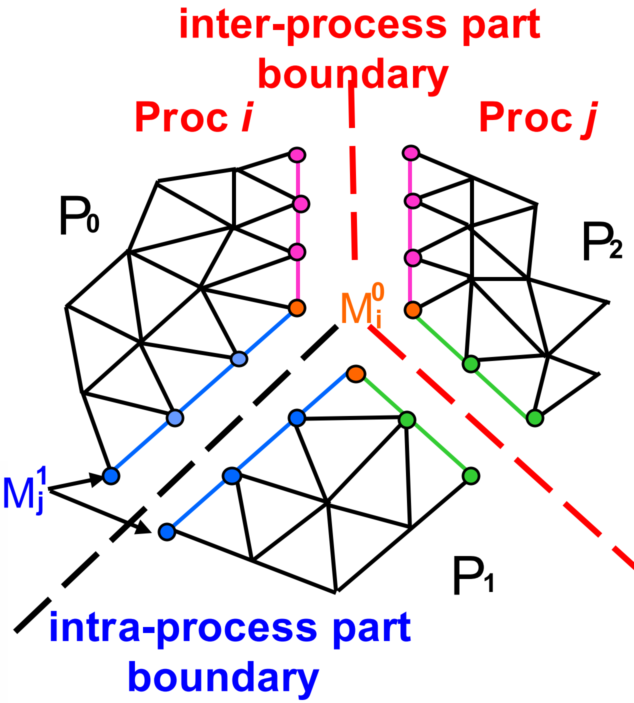
Parallel Unstructured Mesh Infrastructure

- Unstructured Mesh Infrastructure:
  - PUMI-General-Adapt (https://github.com/SCOREC/core): CPU based scalable conforming mesh adaptation based on local mesh modification.
  - PUMI-Portable-Adapt (https://github.com/SCOREC/omega_h): Perfromant (CPUs and GPUs currently) infrastructure supporting scalable conforming mesh adaptation based on local mesh modification.
  - PUMI-Pic (https://github.com/SCOREC/pumi-pic): Performance portable (CPUs and GPUs) with scalable mesh and particle data structures and operators to support particle-in-cell simulation codes.
  - PUMI-Tally (to be released soon): Performance portable (CPUs and GPUs) infrastructure with scalable mesh and particle data structures and operators to support Monte Carlo neutral particle transport.
  - PUMI-Balance (http://scorec.github.io/EnGPar/): A hyper-graph based parallel unstructured mesh dynamic partition improvement component.
- General code coupling:
  - PCMS (https://github.com/SCOREC/pcms): Code coupling library for exascale applications (from file based to parallel in memory).

Parallel Unstructured Mesh Infrastructure

Support unstructured mesh interactions on exascale systems

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

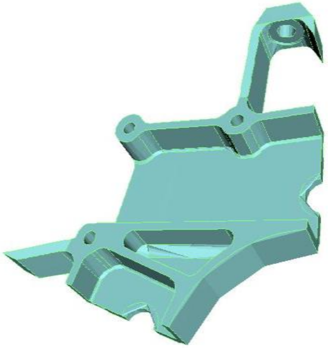


inter-process part boundary

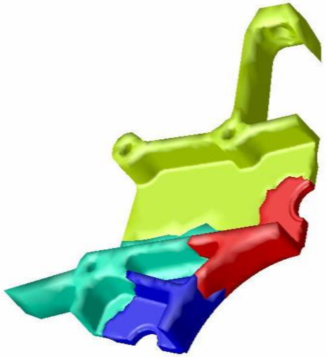
Proc *i*

Proc *j*

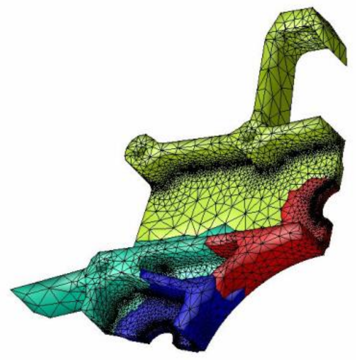
intra-process part boundary



Geometric model



Partition model

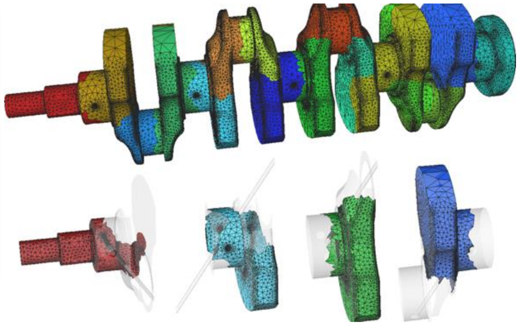


Distributed mesh

Mesh Generation and Control

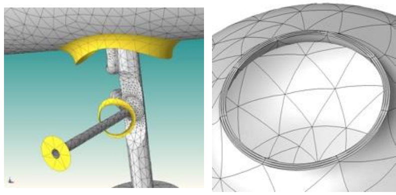
Mesh Generation:

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

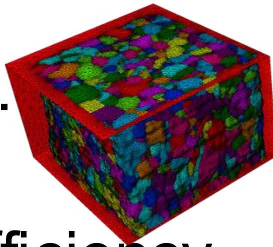


Mesh control:

- Use *a posteriori* information to improve mesh.
- Curved geometry and curved mesh entities.



- Support full range of mesh modifications – vertex motion, mesh entity curving, cavity based refinement and coarsening, etc. anisotropic adaptation.

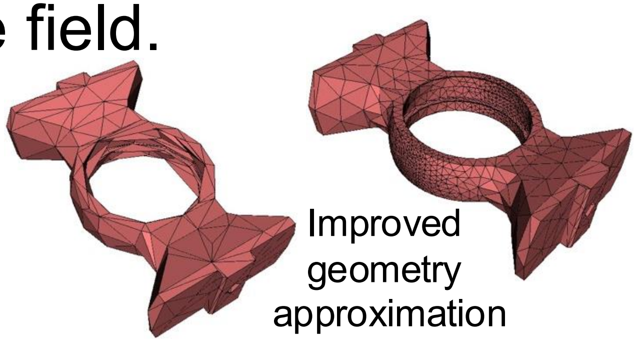


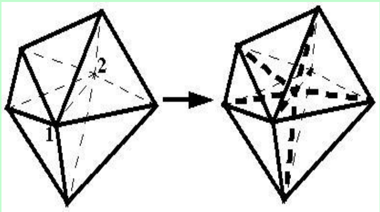
- Control element shapes as needed by the various discretization methods for maintaining accuracy and efficiency.

Parallel execution of all functions is critical on large meshes.

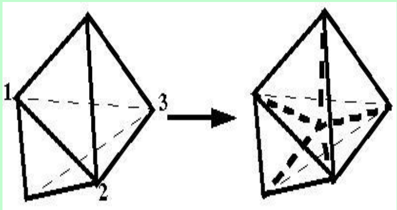
General Mesh Modification for Mesh Adaptation

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

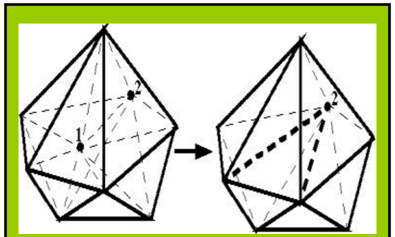

Improved geometry approximation



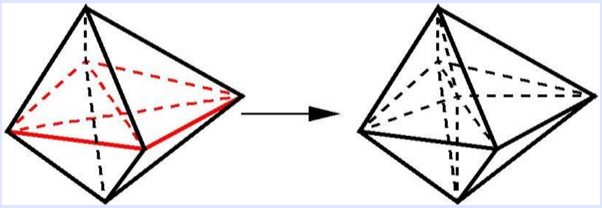
Edge split



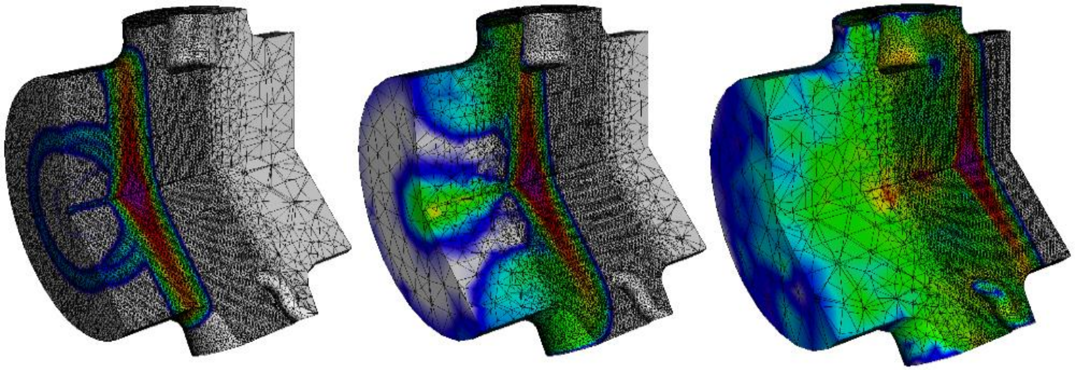
face split



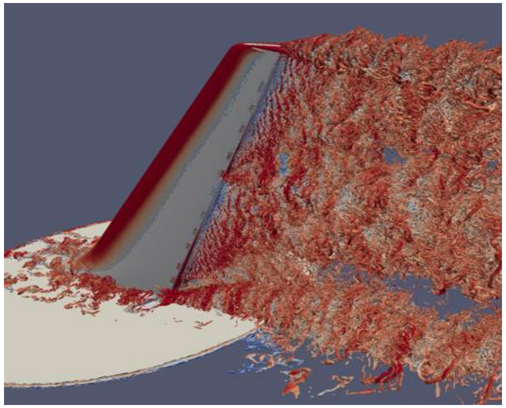
Edge collapse


Double split collapse to remove sliver

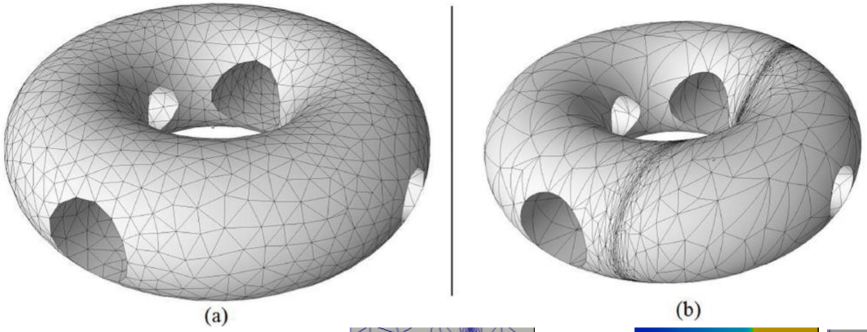
Mesh Adaptation Capabilities



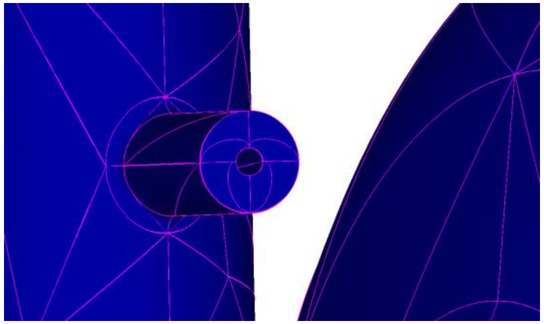
Tracking evolving solution features



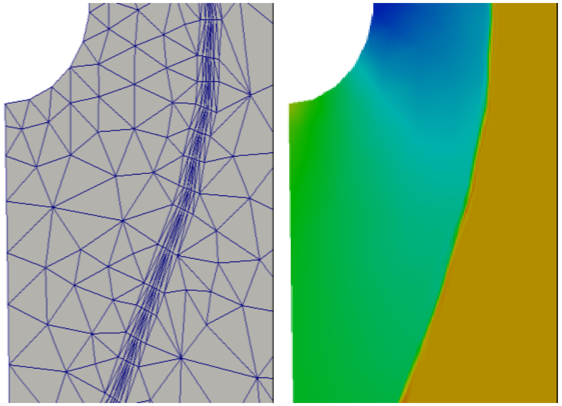
Parallel adaptation of mesh with > 90 billion elements

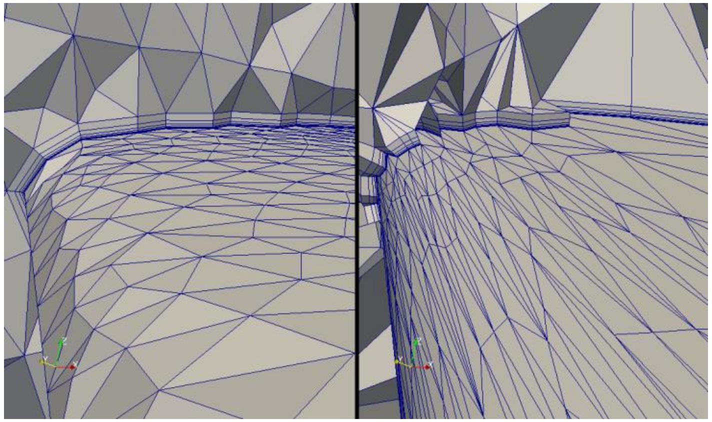


Adaptation of curved high-order elements



Automatic detection and isolation of solution features





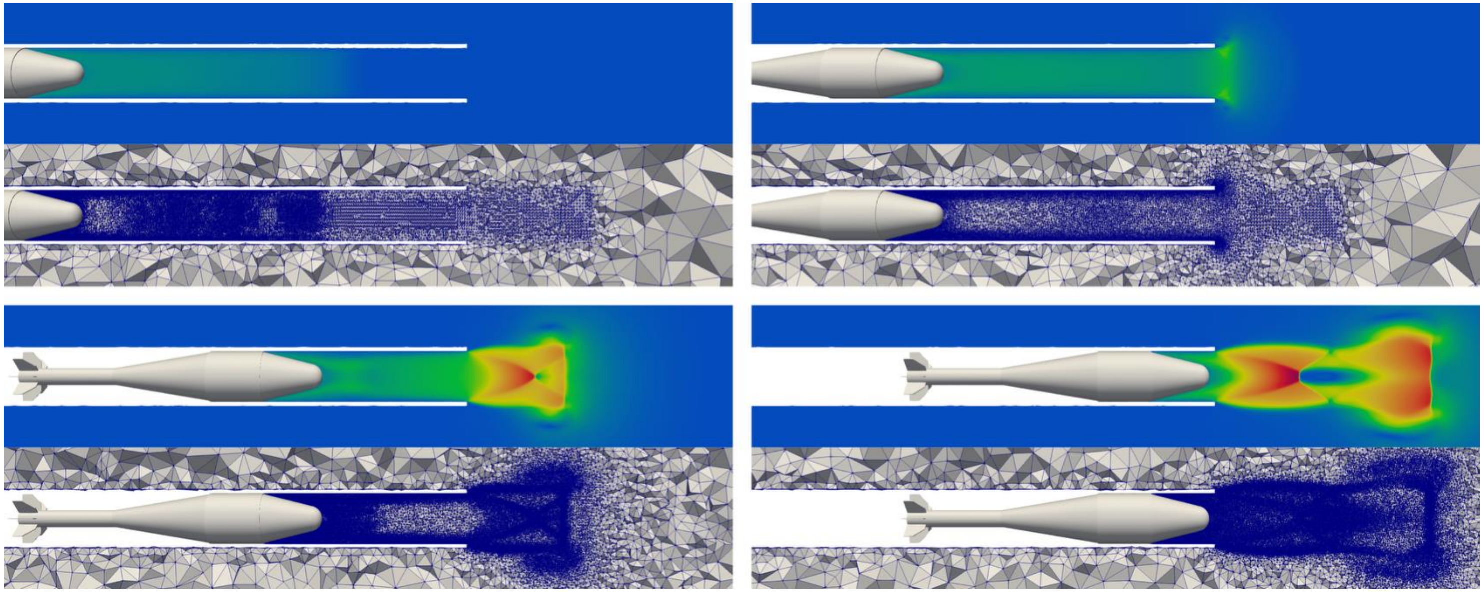
Mixed element topology boundary layer mesh adaptation

Mesh Adaptation of Evolving Geometry Problems

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

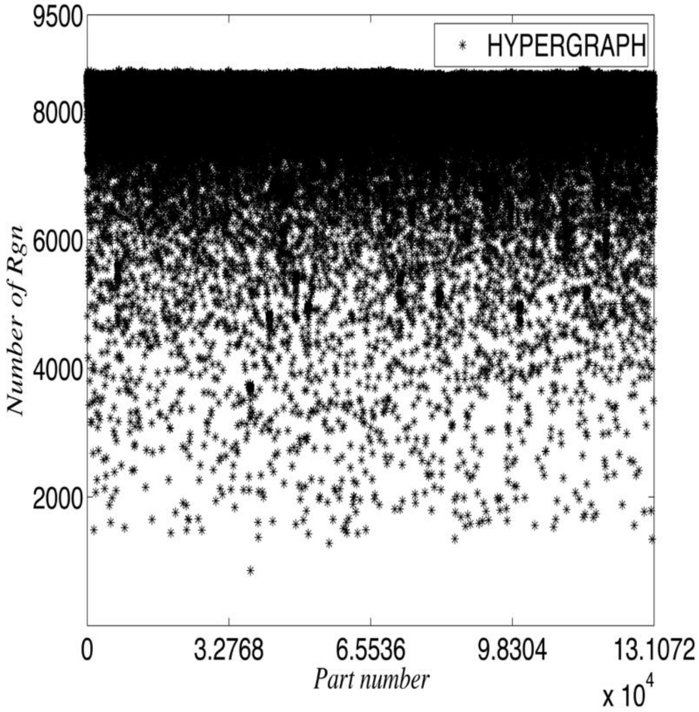
Adaptive loop:

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



Load Balancing, Dynamic Load balancing

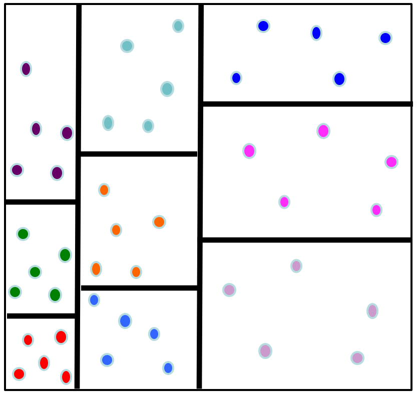
- Purpose: Balance or rebalance computational load while controlling communications.
  - Equal “work load” with minimum inter-process communications.
- FASTMath load balancing tools:
  - Jet library is a multilevel graph partitioner that runs on a GPU (distributed mesh version under development).
  -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



Zoltan/Zoltan2 Toolkits: Partitioners

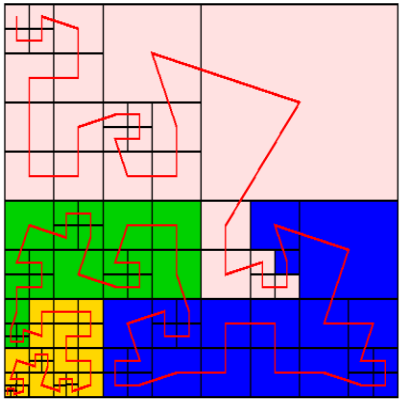
Suite of partitioners supports a wide range of applications; no single partitioner is best for all applications.

Geometric

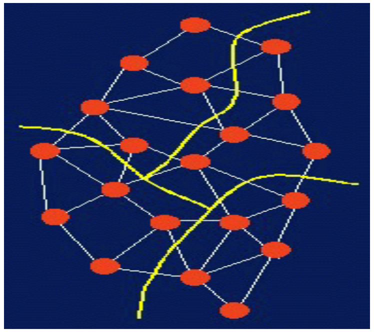


Recursive Coordinate Bisection
Recursive Inertial Bisection
Multi-Jagged Multi-section

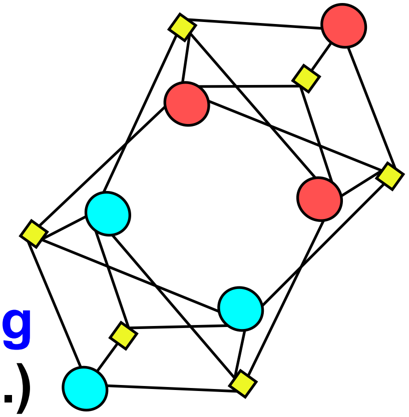
Space Filling Curves



Topology-based



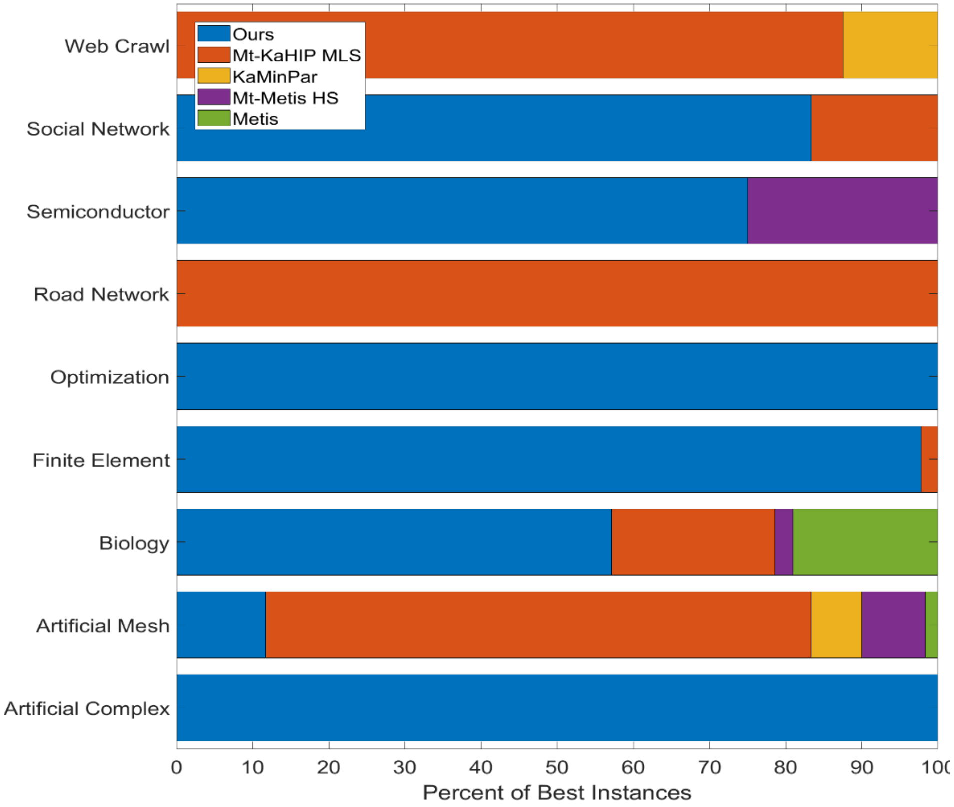
PHG Graph Partitioning
Interface to ParMETIS **(U. Minnesota)**
Interface to PT-Scotch (U. Bordeaux)



PHG Hypergraph Partitioning
Interface to PaToH (Ohio St.)

A New Graph Partitioner for GPU: Jet

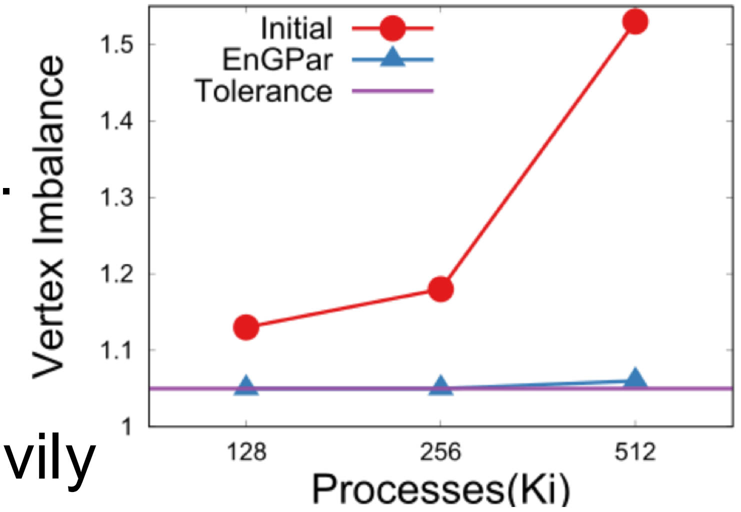
- Multilevel graph partitioner on GPU.
- Uses new label propagation refinement algorithm.
- Results (blue bars) slightly better than Metis/Parametis, but significant speedup due to GPU execution.
- Best partitions for 98% of the test graphs from finite element meshes.

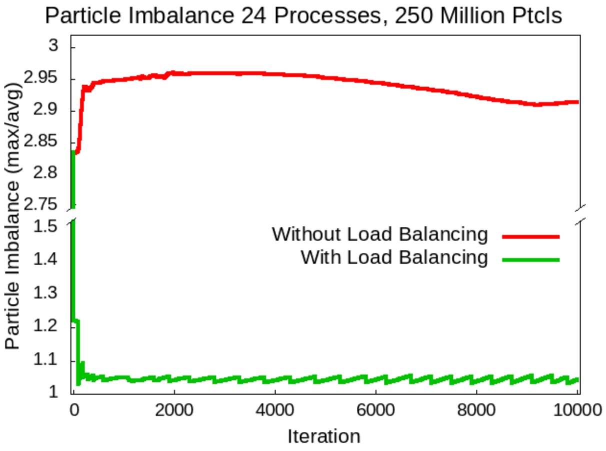


- Currently single GPU (up to ~1B edges)
- Multi GPU - distributed memory version is under development.

PUMI-Balance Reduces Hyper-Graph Imbalance

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



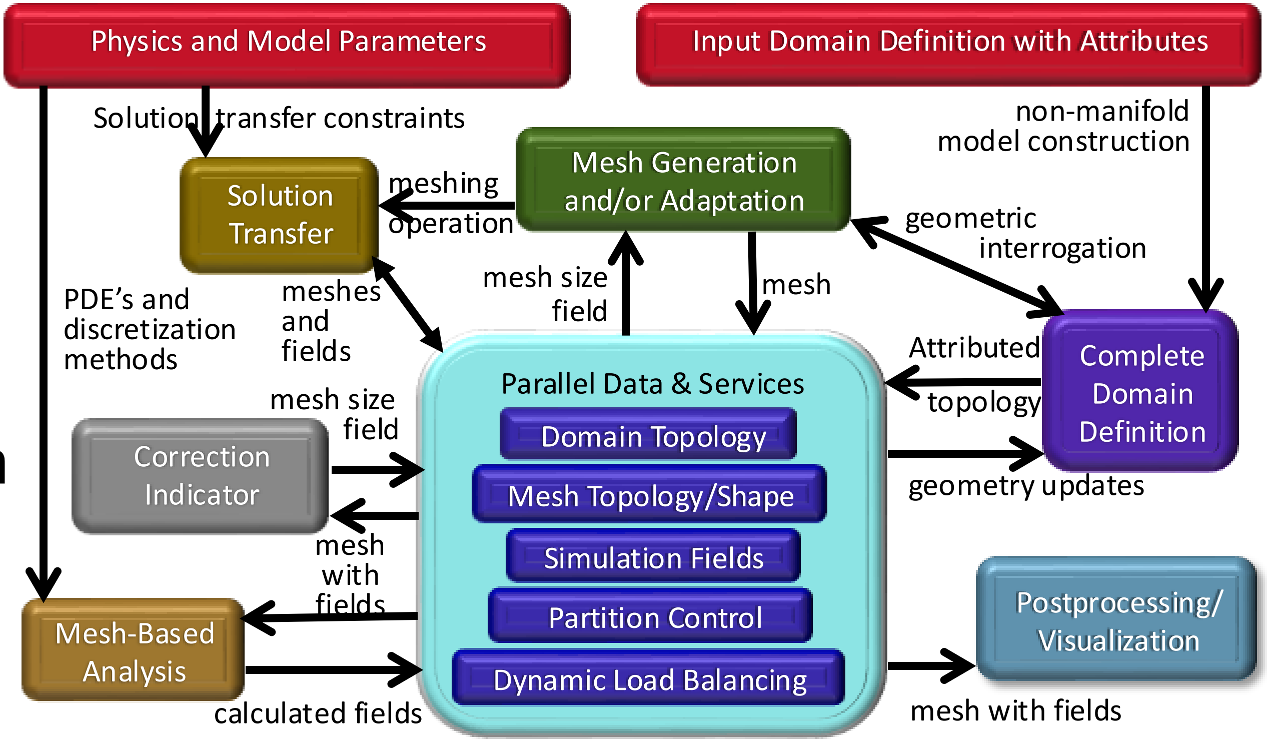


Application of EnGPar particle dynamic load balancing in a GITRm impurity transport simulation

Creation of Parallel Adaptive Loops

Parallel data and services used to develop adaptive simulations:

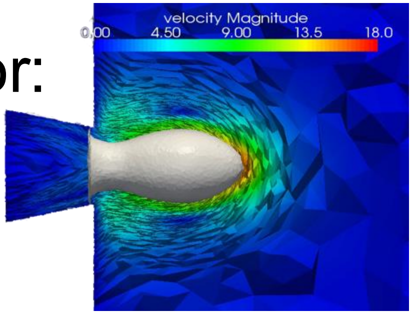
- Geometric model topology for domain linkage.
- Mesh topology – it must be distributed.
- Simulation fields distributed over geometric model and mesh.
- Partition control.
- Dynamic load balancing required at multiple steps.
- API's to link to:
  - CAD/Geometry
  - Mesh generation and adaptation.
 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

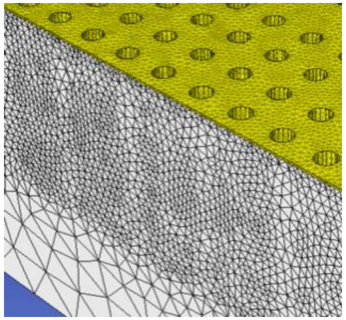


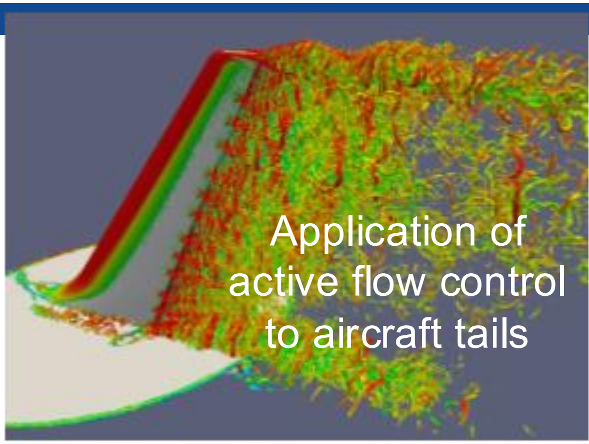
FASTMATH

Parallel Adaptive Simulation Workflows

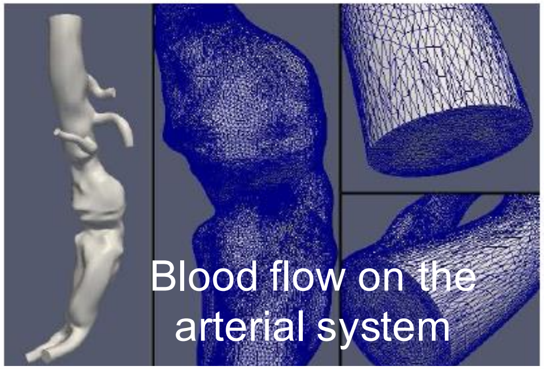
- In memory adaptive loops support effective data movement.
- In-memory adaptive loops for:
  - MFEM – High order FE framework
  - PHASTA – FE for NS
  - FUN3D – FV CFD
  - Proteus – multiphase FE
  - Albany – FE framework
  - ACE3P – High order FE electromagnetics
  - M3D-C1 – FE based MHD
  - Nektar++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



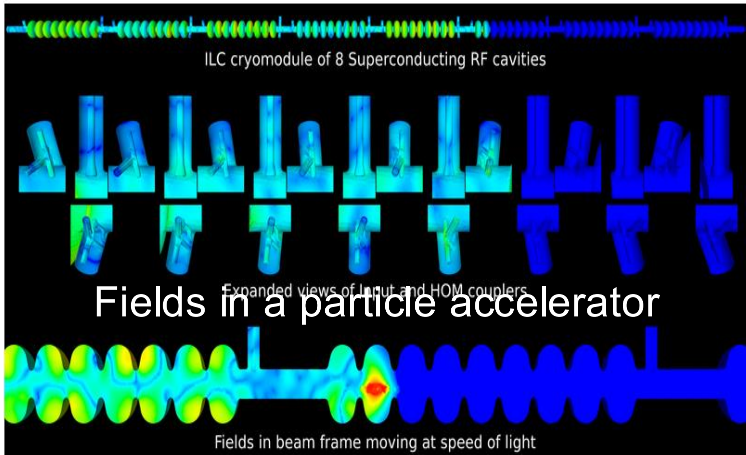




Application of active flow control to aircraft tails



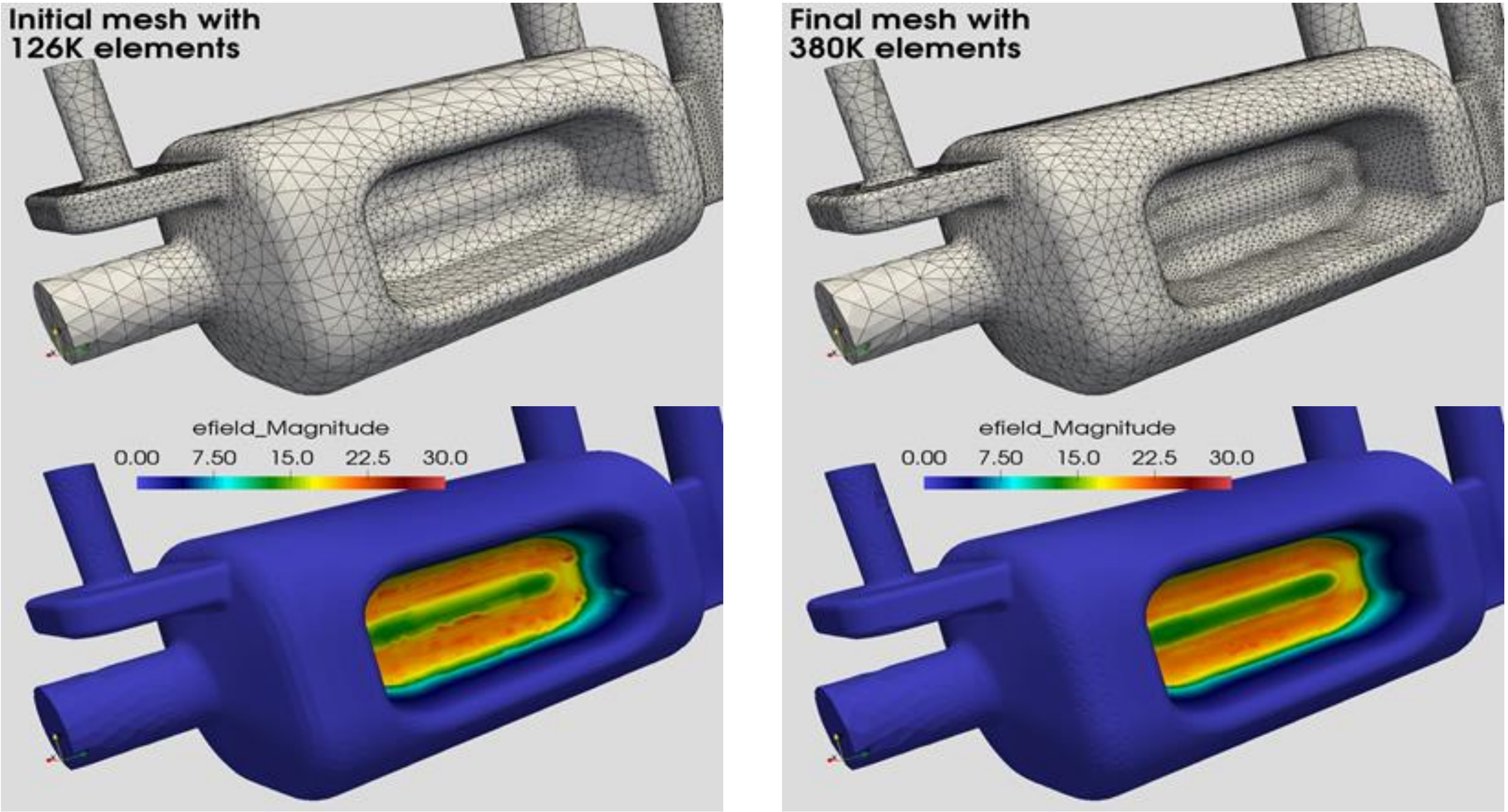
Blood flow on the arterial system


ILC cryomodule of 8 Superconducting RF cavities
Expanded views of Input and HOM couplers
Fields in beam frame moving at speed of light

Fields in a particle accelerator

Application interactions – Accelerator EM

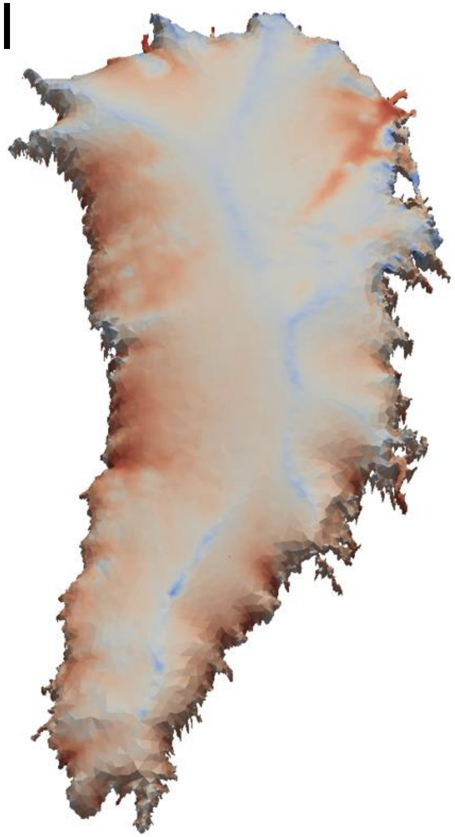
Omega3P Electro Magnetic Solver (second-order curved meshes)

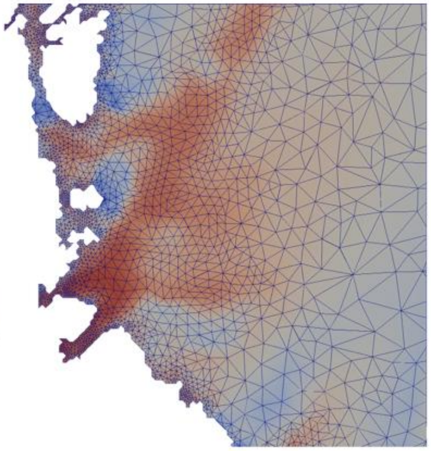


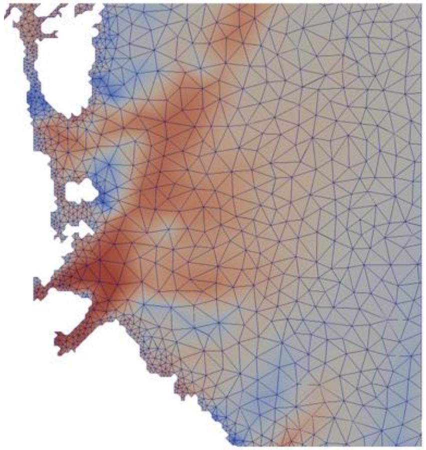
This figure shows the adaptation results for the CAV17 model. (top left) shows the initial mesh with ~126K elements, (top right) shows the final (after 3 adaptation levels) mesh with ~380K elements, (bottom left) shows the first eigenmode for the electric field on the initial mesh, and (bottom right)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

Application interactions – Land Ice

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

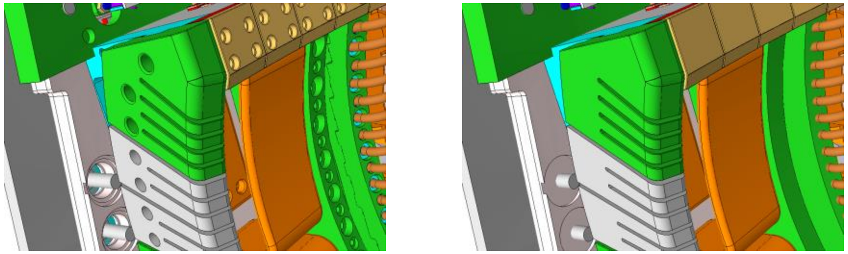




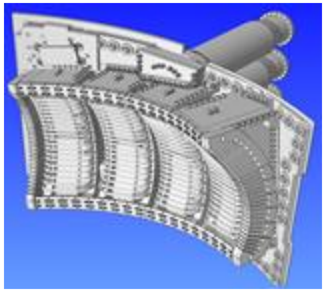


Application interactions – RF Fusion

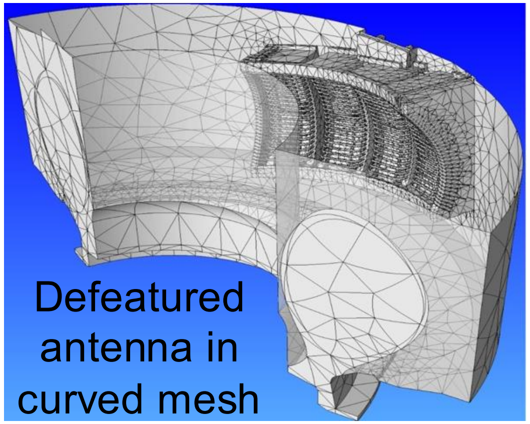
- Accurate RF simulations require:
  - Detailed antenna CAD geometry.
  - CAD geometry defeaturing.
  - Extracted physics curves from GEQDSK equilibrium file.
  - Analysis geometry combines CAD, and physics geometry.
  - 3D meshes for accurate FE calculations in MFEM.
  - Projection based error estimator.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



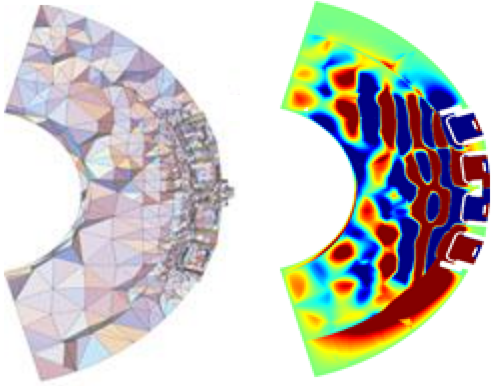
Fast elimination of unwanted features



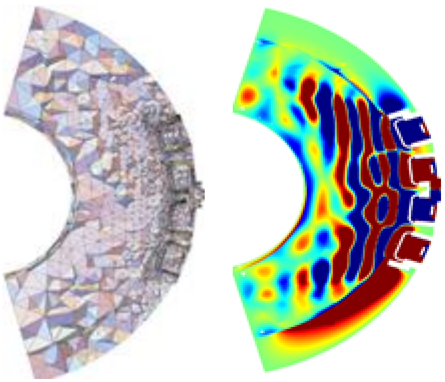
CAD of antenna array



Defeatured antenna in curved mesh



Initial Mesh

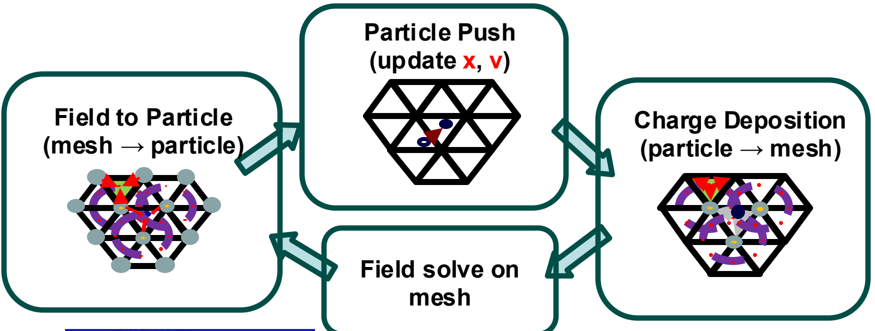


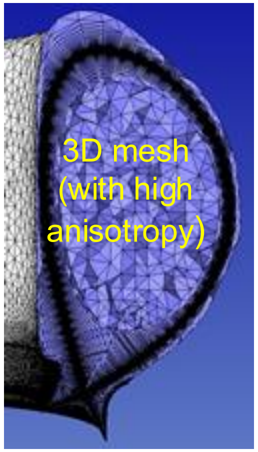
Final Adapted Mesh

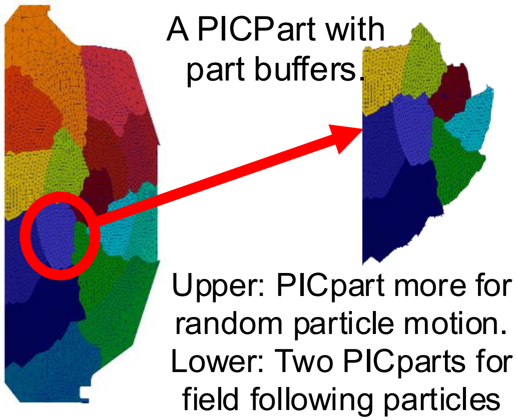
Supporting Unstructured Mesh for Particle-in-Cell Calculations

PUMI-Pic data structures are mesh centric:

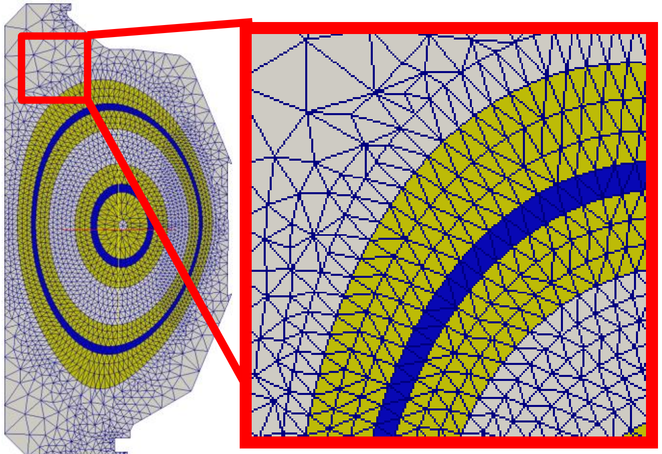
- Mesh is distributed as needed by the application in terms of PICparts.
- Mesh can be graded and anisotropic.
- Particle data associated with elements.
- Operations take advantage of distributed mesh topology.
- Mesh relation to geometry used to speed calculation for near surface physics.
- Mesh distributed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





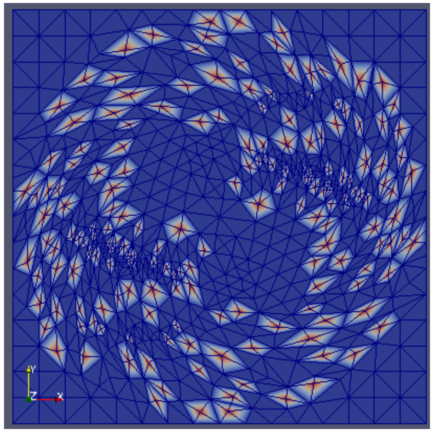

A PICPart with part buffers.

Upper: PICpart more for random particle motion.
Lower: Two PICparts for field following particles

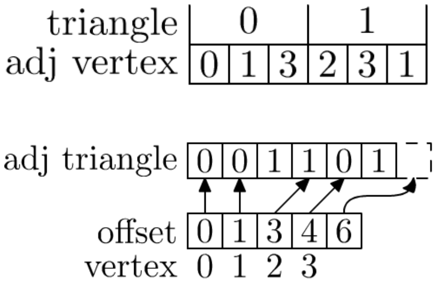


Mesh Data Structure for Heterogeneous Systems

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



Adaptation following rotating flow field.

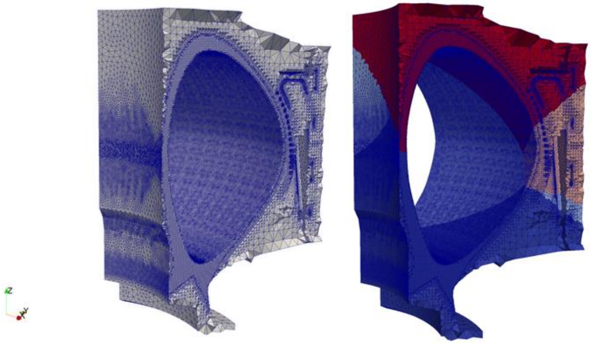


triangle 0 1
adj vertex 0 1 3 2 3 1

adj triangle 0 0 1 1 0 1

offset 0 1 3 4 6
vertex 0 1 2 3

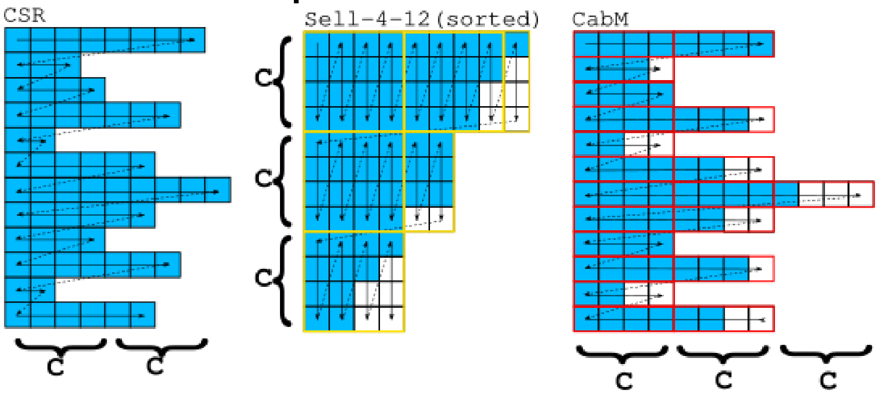
Mesh entity adjacency arrays.



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

PUMI-Pic Particle Data Structures

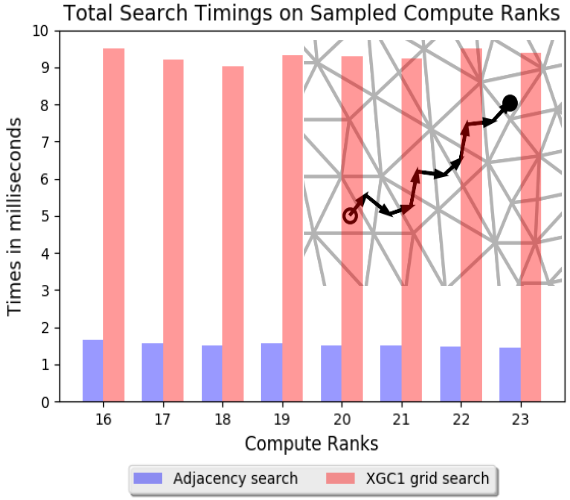
- Layout of particles in memory is critical to performance:
  - Optimizes push (sort/rebuild), scatter, and gather operations.
  - Associate particles with elements for large per element particle cases.
  - Support changes in the number of particles per element.
  - Evenly distributes work under a range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



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

PIC Operations Supported by PUMI-Pic

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



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

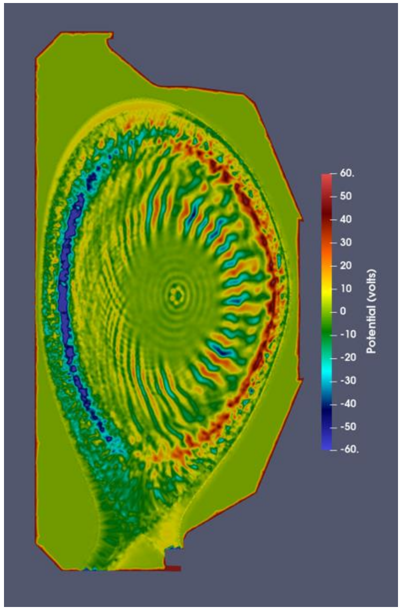
PUMI-Pic based XGCm Edge Plasma Code

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

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

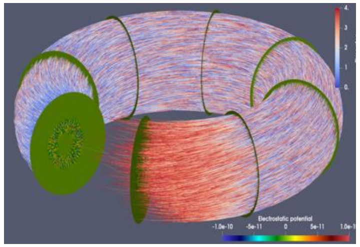
Testing of PUMI-Pic for use in XGC like push:

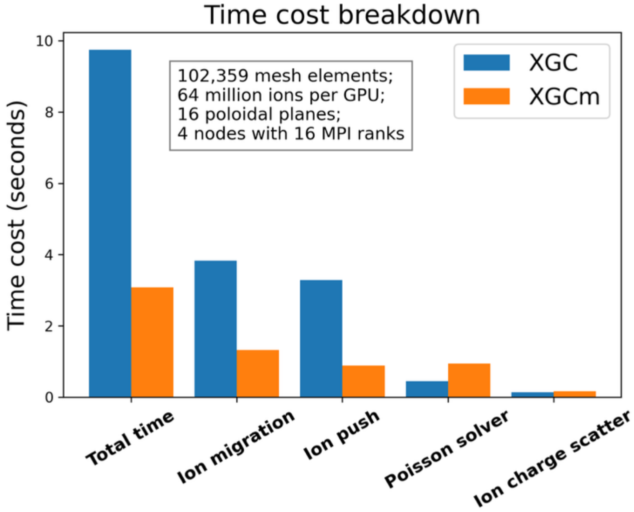
- 2M elements, 1M vertices, 2 to 128 poloidal planes.
- Pseudo push and particle-to-mesh gyro scatter.
- Tested on up to 24,576 GPUs with 1.1 trillion particles, for 100 iterations: push, adjacency.
- PUMI-Pic weak scaling up to 24,576 GPUs (4096 nodes) with 48 million particles per GPU.



Total time comparison:

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

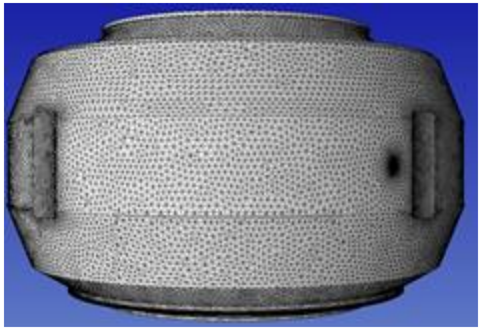


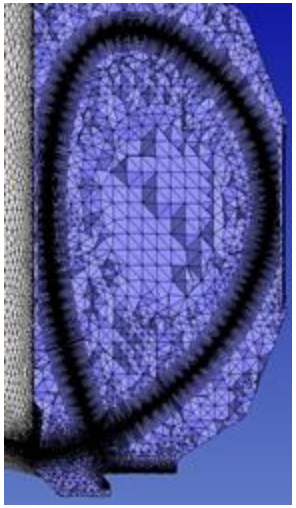


FASTMATH

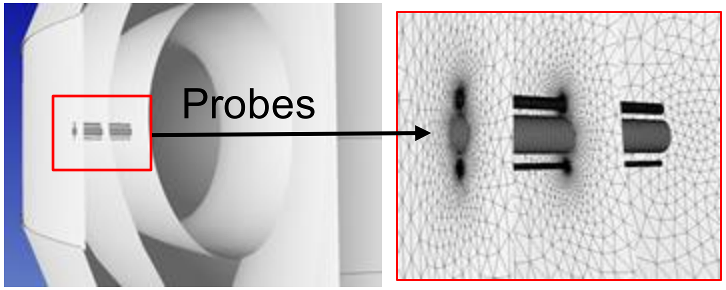
PUMIPic based GITRm Impurity Transport Code

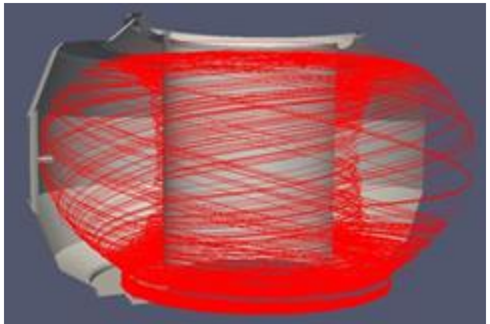
- Incorporates impurity transport capabilities of GITR.
- 3D mesh for cases including divertors, tiles, limiters, specific diagnostics/probes etc.

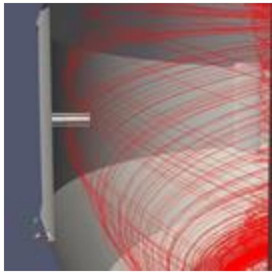




- Status:
  - Physics equivalent to GITR.
  - Efficient Multi-species capabilities.
  - Anisotropy mesh for accurate field transfer.
  - Field transfer from SOLPS to 3D mesh.
  - Non-uniform particle distribution – evolves quickly in time.
  - Load balancing particles via PUMI-Balance.
  - Distance to boundary for sheath E field.
  - Post-processing on 3D unstructured mesh.



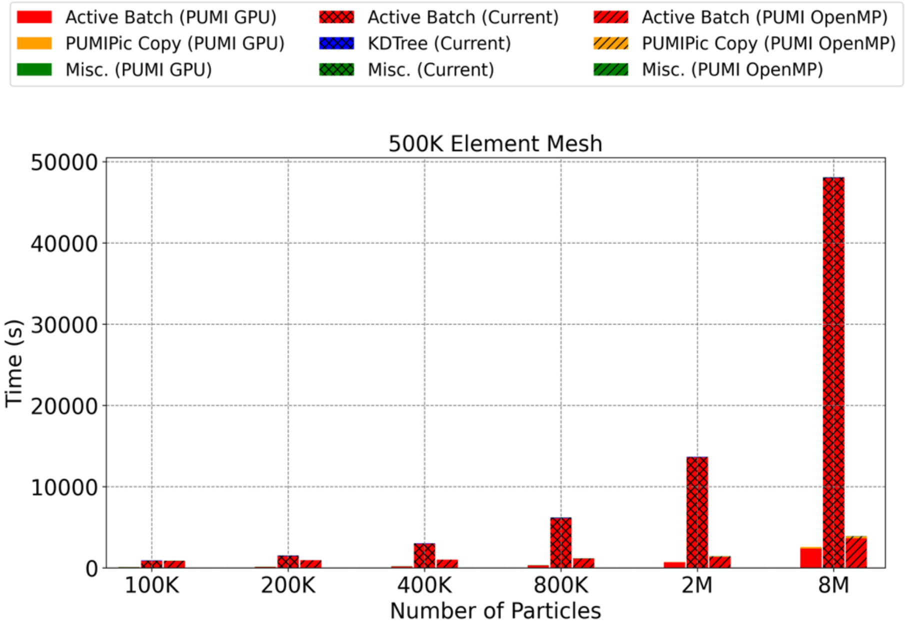




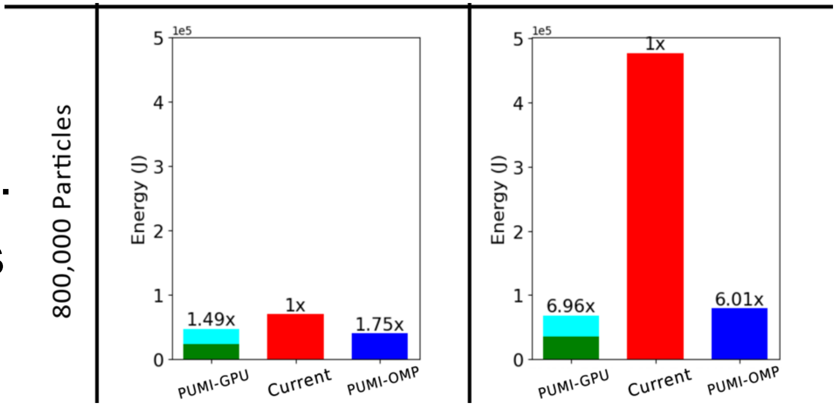
FASTMATH

PUMI-Tally - GPU Acceleration of Monte Carlo Tallies

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



Computation Time

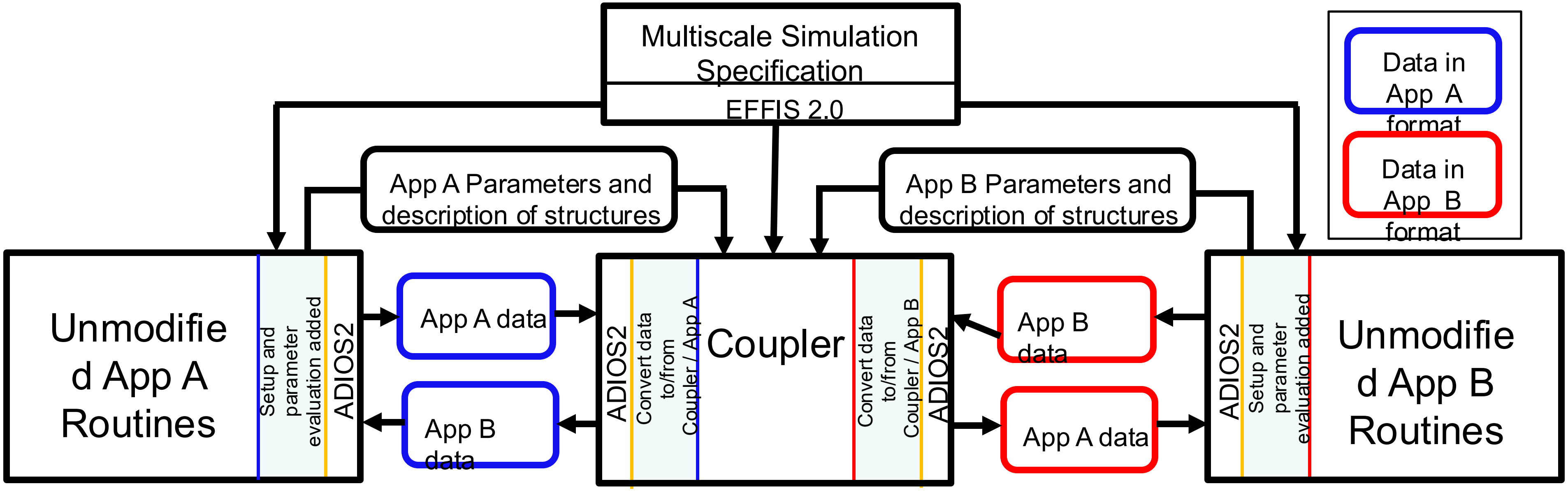


Energy consumption

Parallel Coupler for Multimodel Simulations (PCMS)

Goals of PCMS:

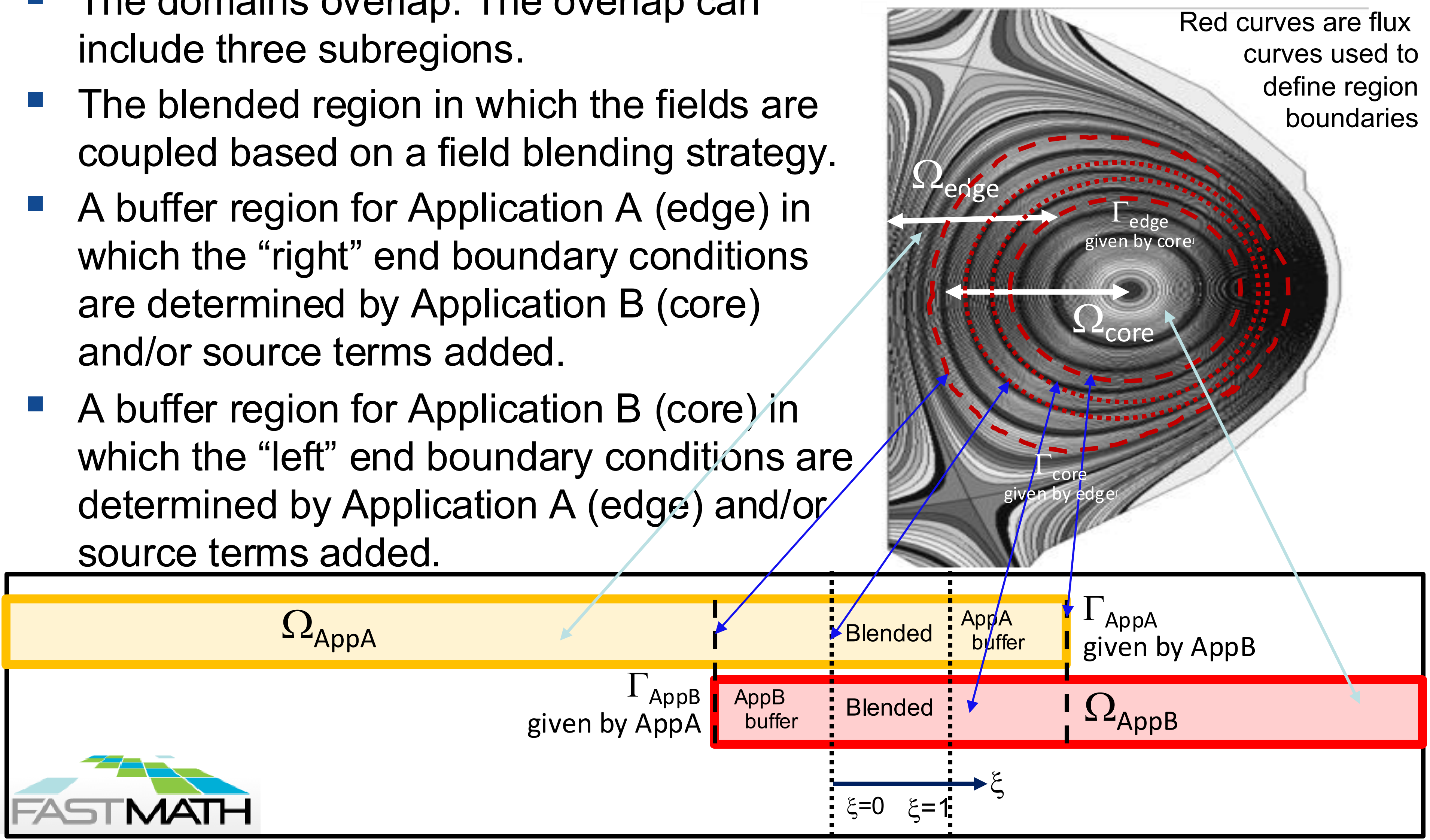
- Keep applications clean. Only modification to application codes is access its data structures.
- General structures and functions for coupling operations.
- Make effective use of the massively parallel, heterogeneous computing systems.


Multiscale Simulation Specification
EFFIS 2.0
App A Parameters and description of structures
App B Parameters and description of structures
Data in App A format
Data in App B format
Unmodified App A Routines
Setup and parameter evaluation added
ADIOS2
App A data
App B data
ADIOS2
Convert data to/from Coupler / App A
Coupler
Convert data to/from Coupler / App B
ADIOS2
App B data
App A data
ADIOS2
Setup and parameter evaluation added
Unmodified App B Routines

In-Memory Coupling of Fusion Codes

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

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



Red curves are flux curves used to define region boundaries

“Rendezvous” Algorithm to Control Coupler Domain Partition

Challenge:

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

Approach:

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

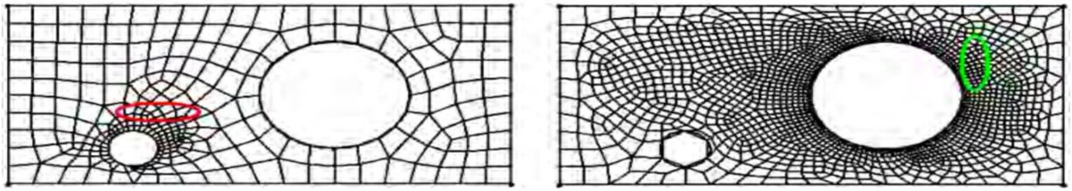


Fig. 4. Thermal (left) and stress (right) grids. The colored ovals are clumps of grid cells the same processor owns in the two grids.

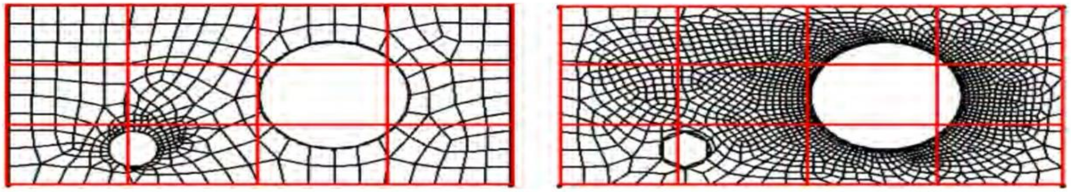
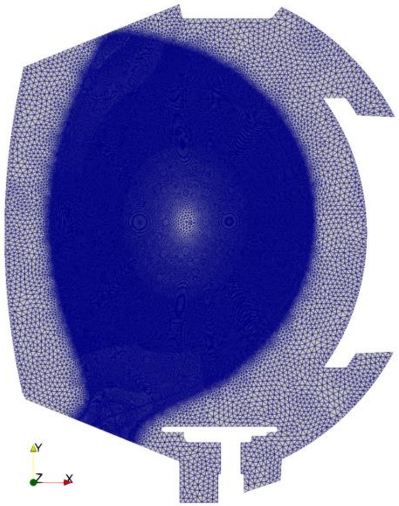
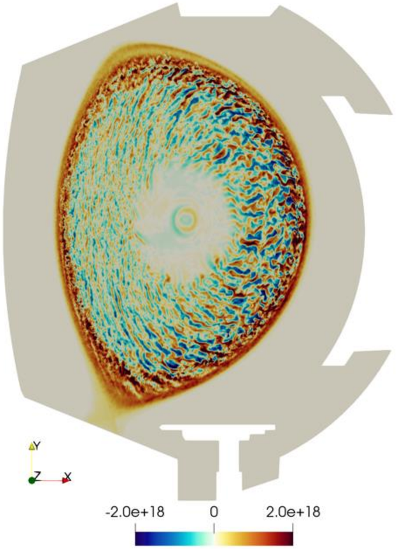


Fig. 5. Rendezvous decomposition overlaid on thermal and stress grids.

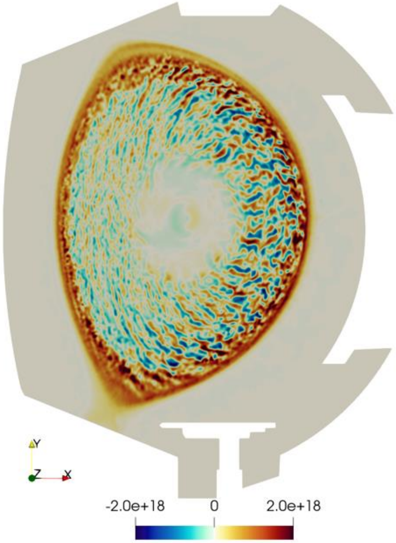
Example Field Transfer with RBF on LCPP



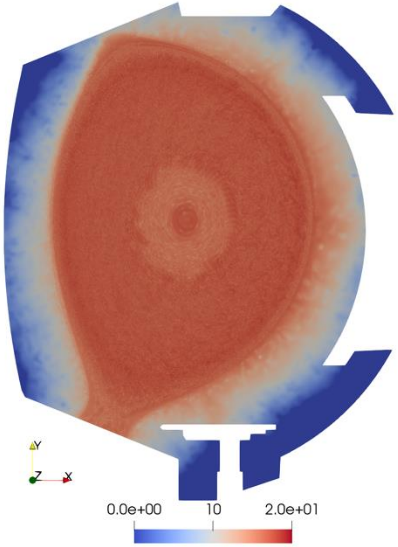
XGC mesh with 611,359 elements and 306, 002 nodes



$idensity$ loaded as the **exact/initial field** in the coupler app



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

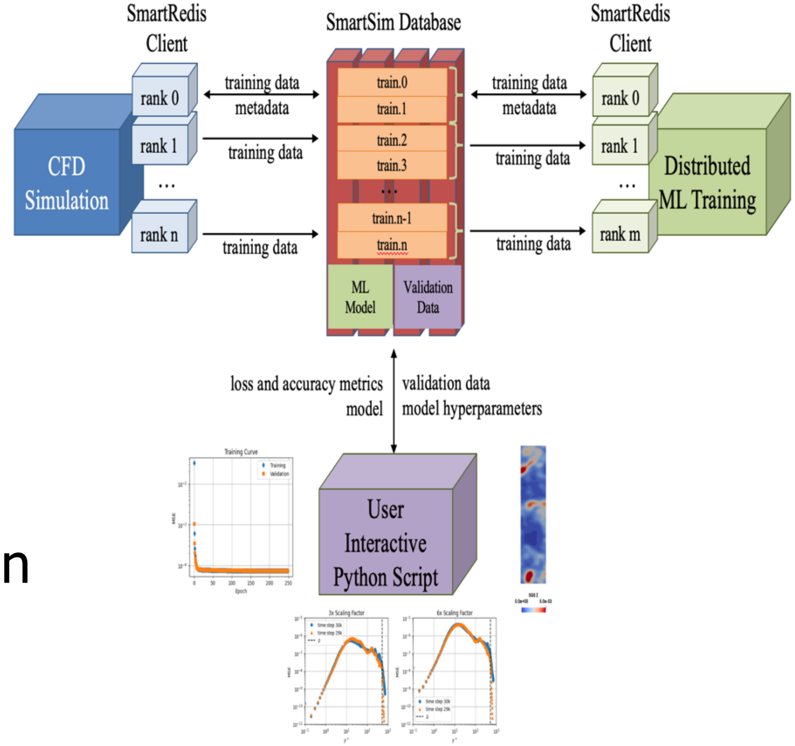


$idensity$ error field after 10 **MLS-RBF** interpolation operation: cell to node and node to cell in each iteration. $\boldsymbol{\rho = 0.007}$

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

In Situ Machine Learning During Simulations

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



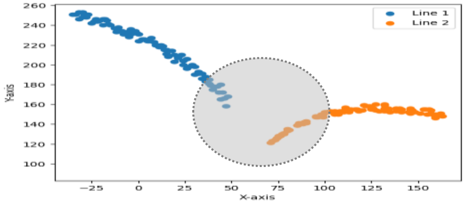
Online training with user interactive script

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

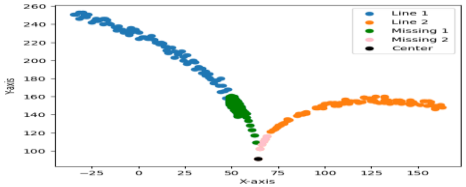
Mesh Related AI/ML Developments

Robust feature detection/processing:

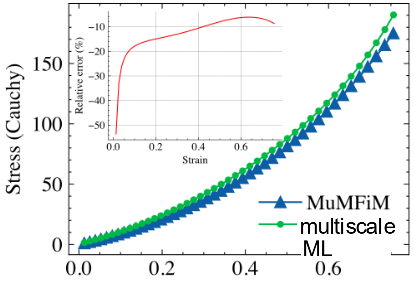
- Physics-based and AI/ML sensors: neural network processes fragmented/noisy data.
- Application: anisotropic diffusion transport in fusion devices, ground line dynamics in ice sheets.



Improve sensor data

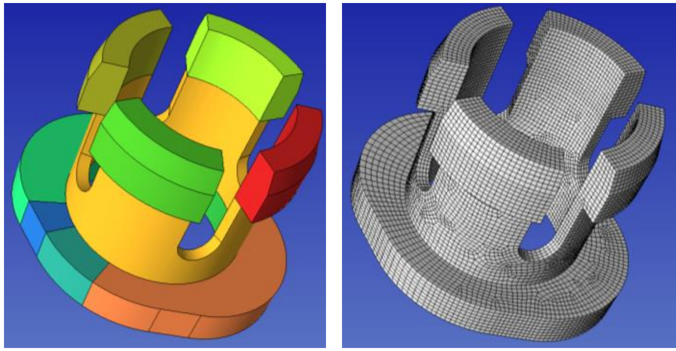


- Physics informed material models
- Machine learned constitutive models can reduce runtime cost of upscaling FEM simulations by orders of magnitude while retaining 80% of accuracy.



Comparing ML to Multiscale

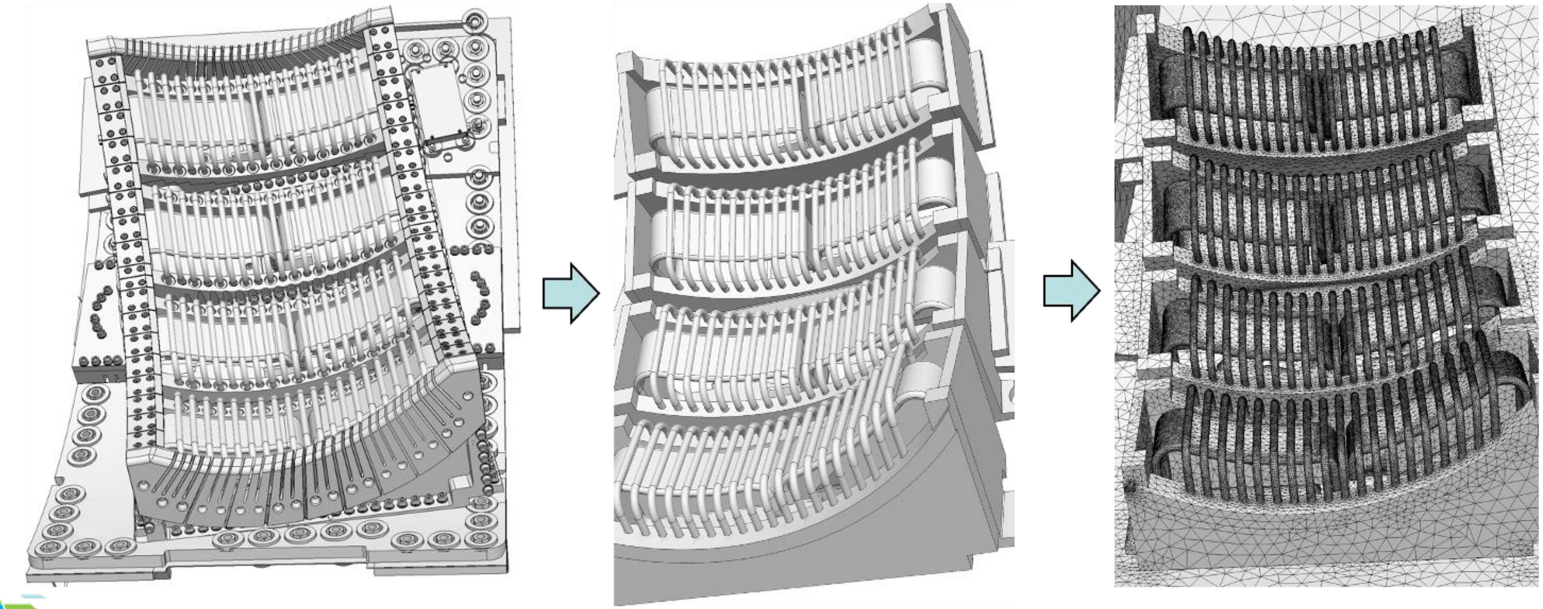
- ML Agents for automated hex meshing
- Reinforced learning agent decomposes CAD modes into regions suitable for hexahedral mesh generation.



Run the latest Simmetrix and PUMI software on RPI systems

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

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



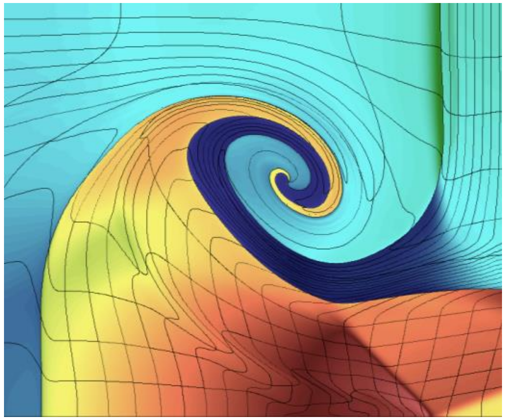
Finite elements are a good foundation for large-scale simulations on current and future architectures

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

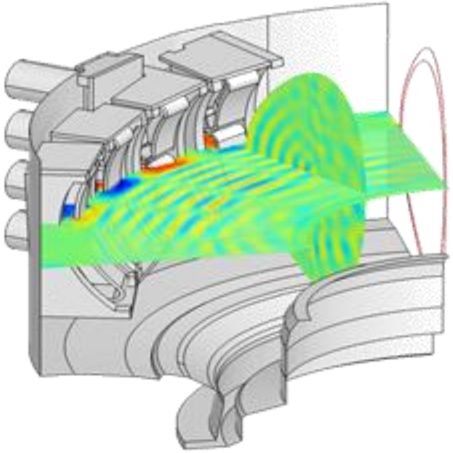
$$\boxed{H(grad)} \xrightarrow{\nabla} \boxed{H(curl)} \xrightarrow{\nabla\times} \boxed{H(div)} \xrightarrow{\nabla\cdot} \boxed{L_2}$$

“nodes”	“edges”	“faces”	“elems”
High-order kinematics	**High-order MHD**	**High-order rad. diffusion**	**High-order thermodynamics**

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



8th order Lagrangian simulation of shock triple-point interaction



Core-Edge tokamak EM wave propagation

Modular Finite Element Methods (MFEM)

Flexible discretizations on unstructured grids

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

High-order methods and scalability

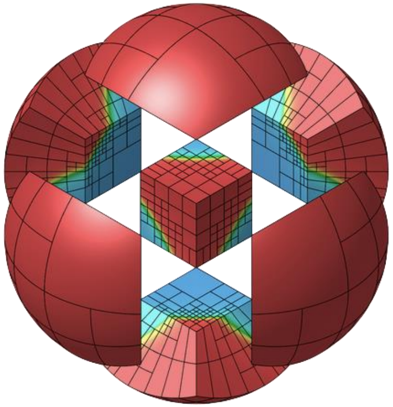
- Arbitrary-order H1, H(curl), H(div)- and L2 elements
- Arbitrary order curvilinear meshes
- MPI scalable to millions of cores + GPU accelerated
- Enables development from laptops to exascale machines.

Solvers and preconditioners

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

Open-source software

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

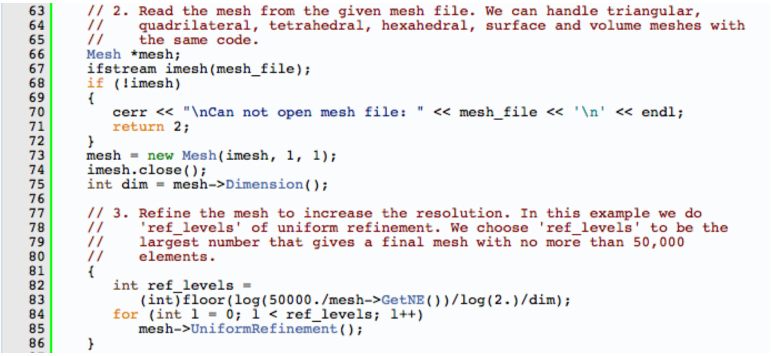


mfem.org

(v4.8, April 2025)

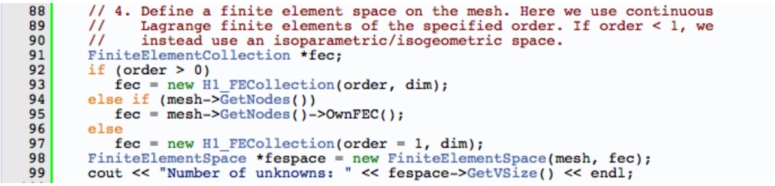
Example 1 – Laplace equation

- Mesh



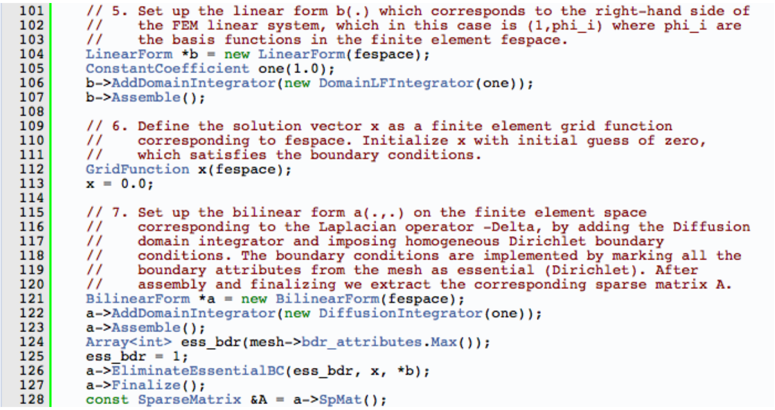
```
   // 2. Read the mesh from the given mesh file. We can handle triangular,
   //    quadrilateral, tetrahedral, hexahedral, surface and volume meshes with
   //    the same code.
   Mesh *mesh;
   ifstream imesh(mesh_file);
   if (!imesh)
   {
      cerr << "\nCan not open mesh file: " << mesh_file << '\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



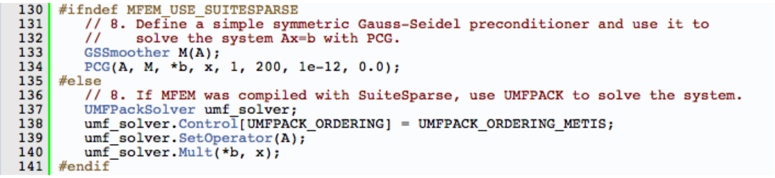
```
   // 4. Define a finite element space on the mesh. Here we use continuous
   //    Lagrange finite elements of the specified order. If order < 1, we
   //    instead use an isoparametric/isogeometric space.
   FiniteElementCollection *fec;
   if (order > 0)
      fec = new H1_FECollection(order, 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



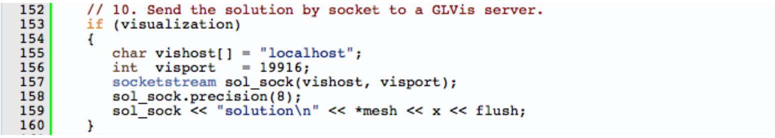
```
   // 5. Set up the linear form b(.) which corresponds to the right-hand side of
   //    the FEM linear system, which in this case is (1,phi_i) where phi_i are
   //    the basis functions in the finite element fespace.
   LinearForm *b = new LinearForm(fespace);
   ConstantCoefficient one(1.0);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

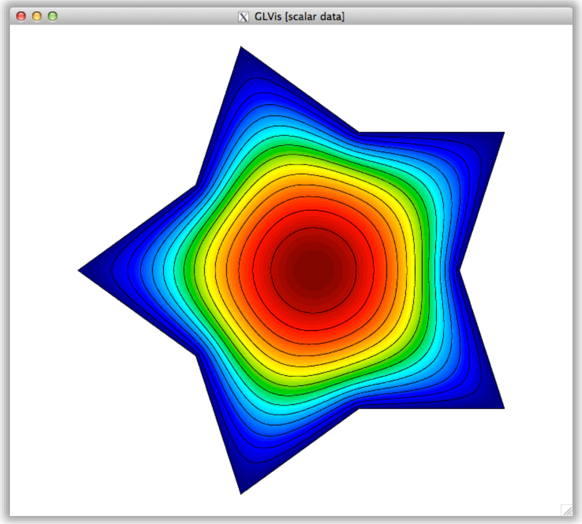


```
#ifndef MFEM_USE_SUITESPARSE
   // 8. Define a simple symmetric Gauss-Seidel preconditioner and use it to
   //    solve the system Ax=b with PCG.
   GSSmoother M(A);
   PCG(A, M, *b, x, 1, 200, 1e-12, 0.0);
#else
   // 8. If MFEM was compiled with SuiteSparse, use UMFPACK to solve the system.
 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

- Visualization



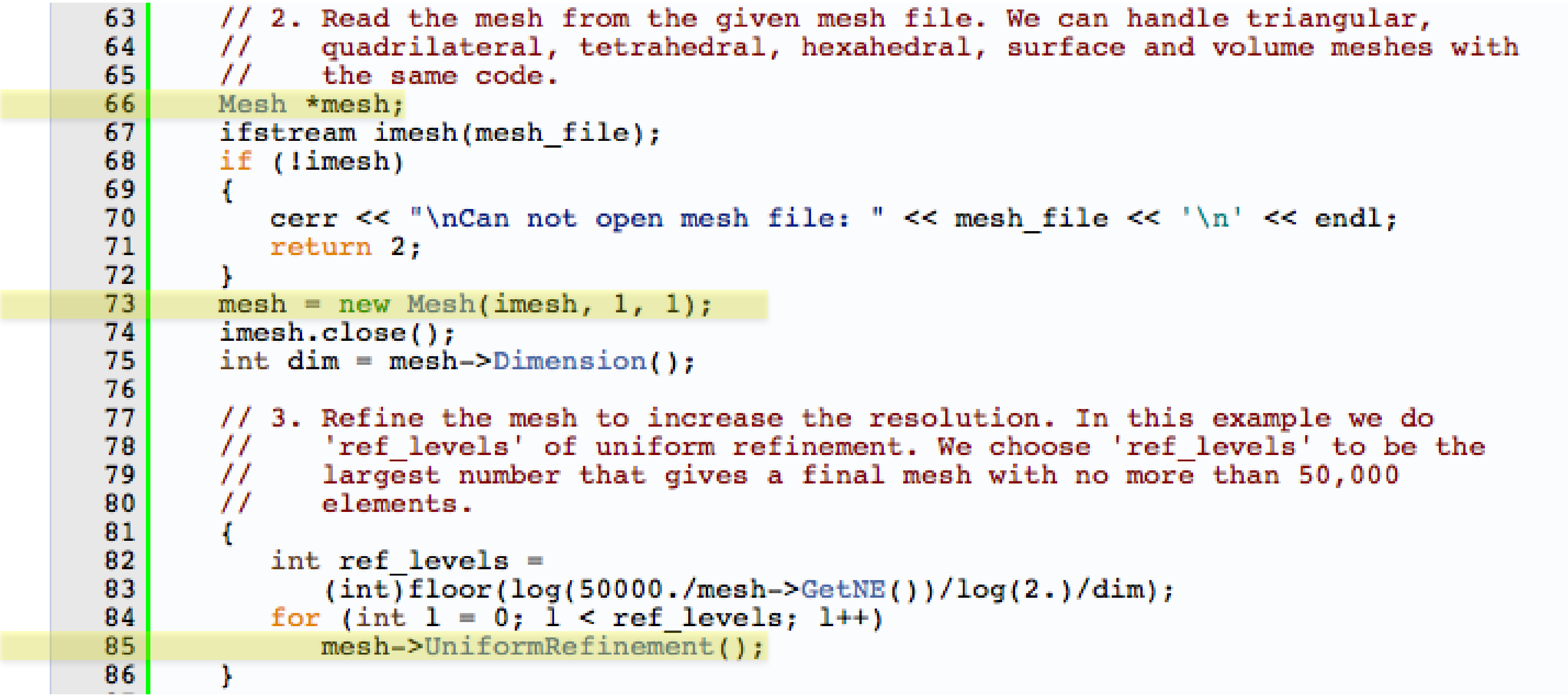
```
   // 10. Send the solution by socket to a GLVis server.
   if (visualization)
   {
      char vishost[] = "localhost";
      int  visport   = 19916;
      socketstream sol_sock(vishost, visport);
      sol_sock.precision(8);
      sol_sock << "solution\n" << *mesh << x << flush;
   }
```



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

Example 1 – Laplace equation

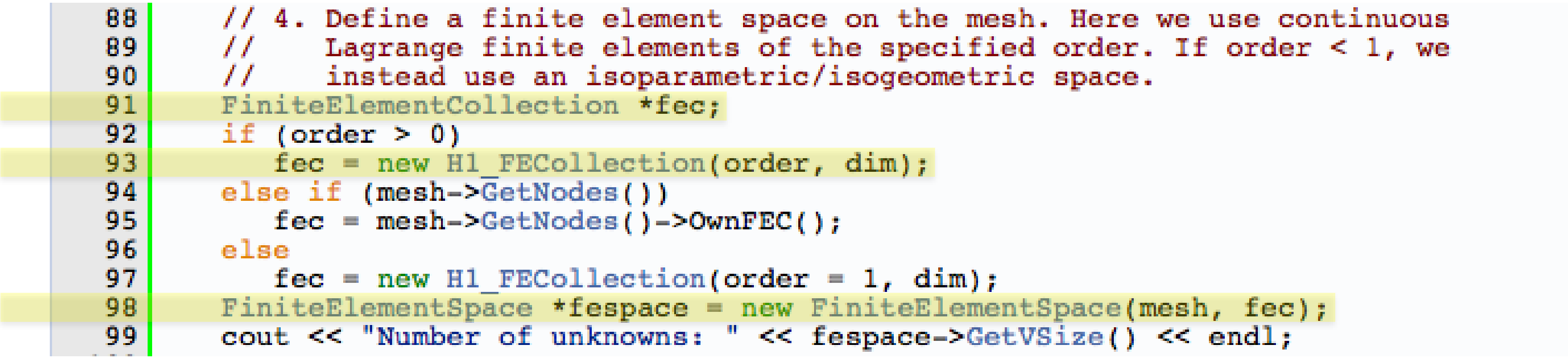
- Mesh



```
// 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

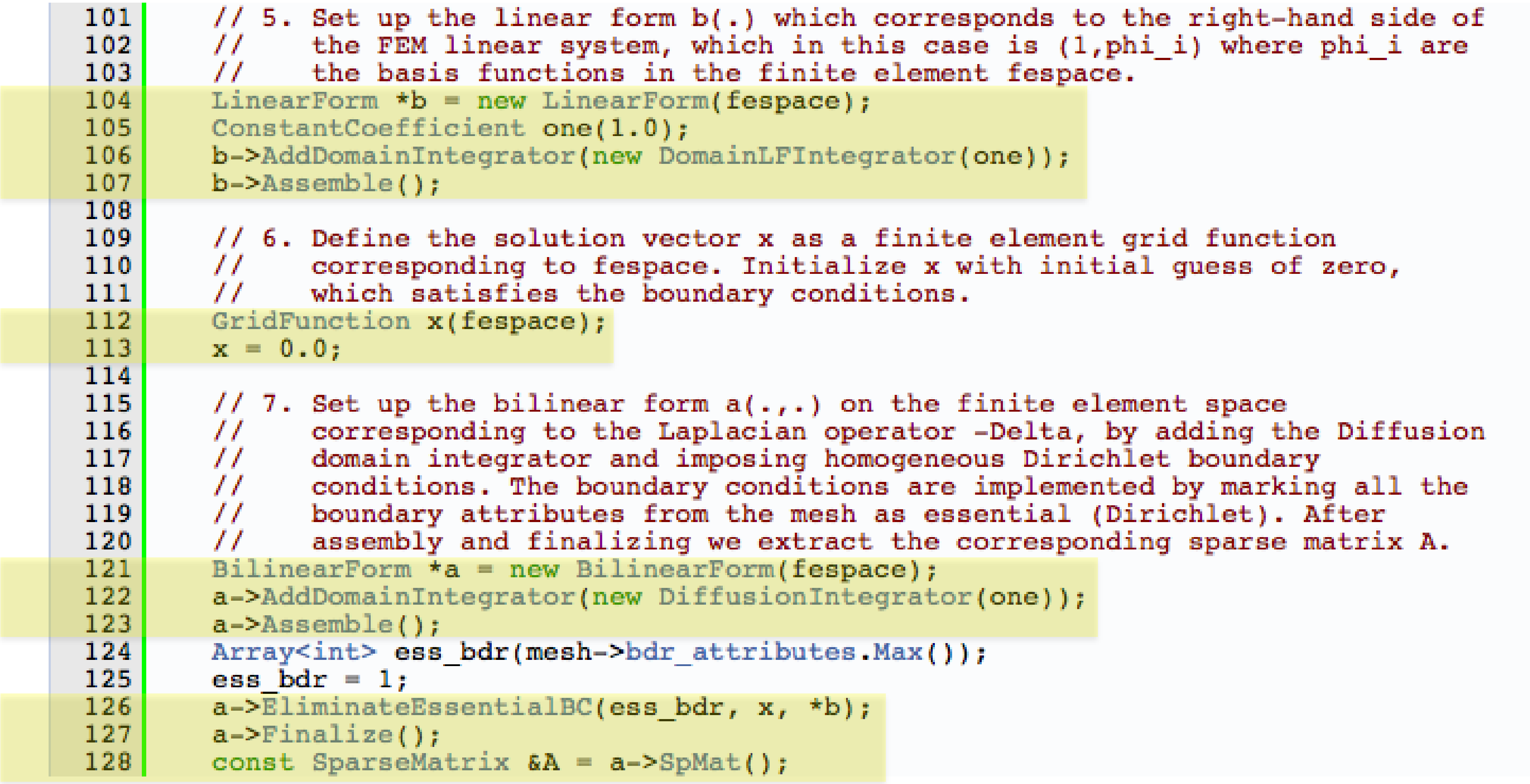
- Finite element space



```
// 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

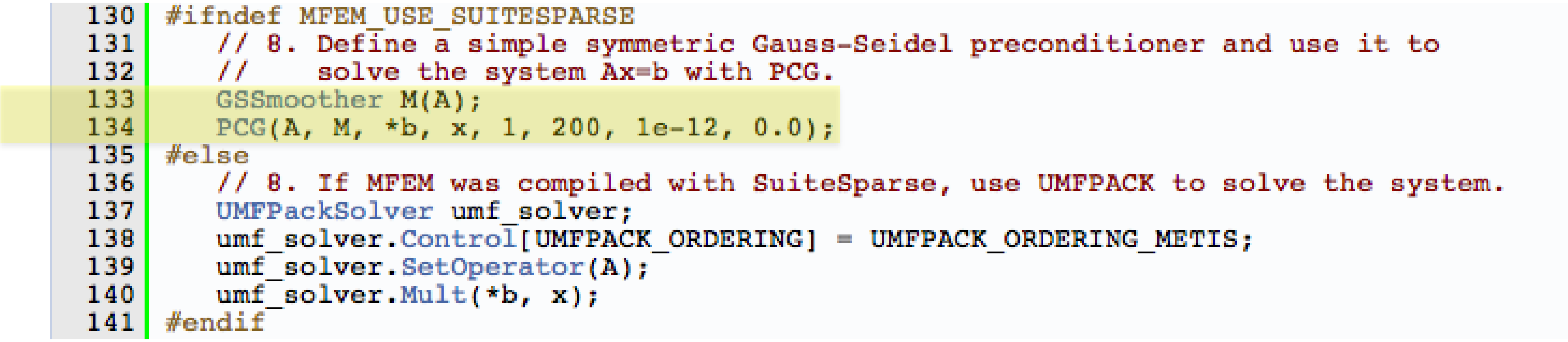
- Initial guess, linear/bilinear forms



```
   // 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();
```

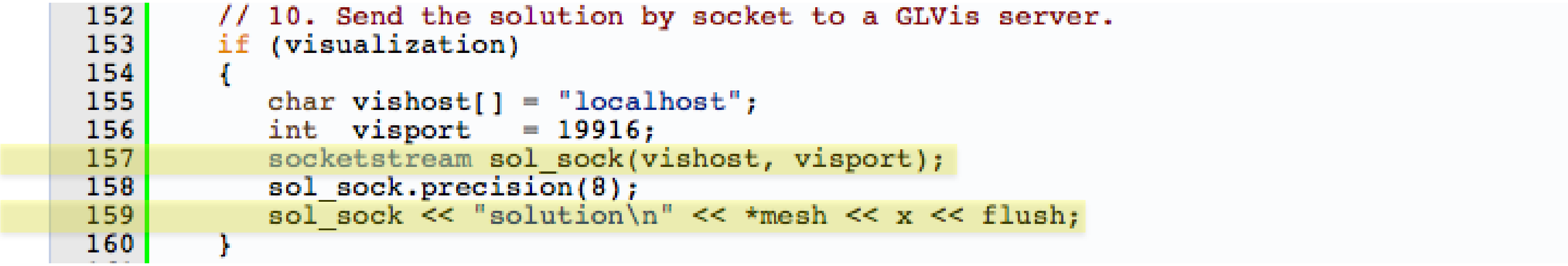
Example 1 – Laplace equation

- 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
```

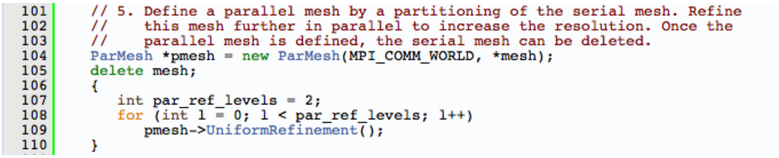
- Visualization



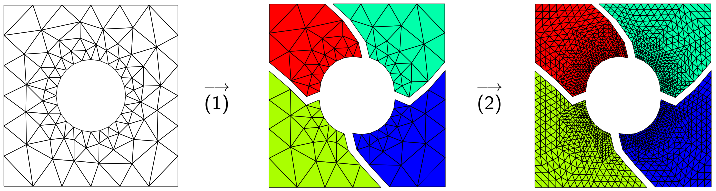
```
   // 10. Send the solution by socket to a GLVis server.
   if (visualization)
   {
      char vishost[] = "localhost";
      int  visport   = 19916;
      socketstream sol_sock(vishost, visport);
      sol_sock.precision(8);
      sol_sock << "solution\n" << *mesh << x << flush;
   }
```

Example 1 – parallel Laplace equation

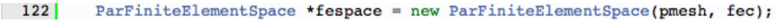
- Parallel mesh



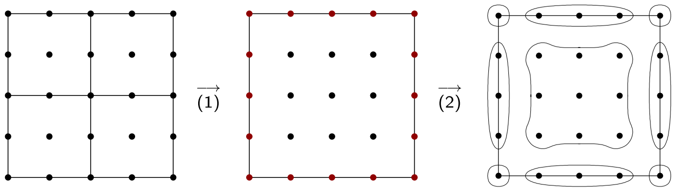
```
// 5. Define a parallel mesh by a partitioning of the serial mesh. Refine
//    this mesh further in parallel to increase the resolution. Once the
//    parallel mesh is defined, the serial mesh can be deleted.
ParMesh *pmesh = new ParMesh(MPI_COMM_WORLD, *mesh);
delete mesh;
{
   int par_ref_levels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



- Parallel finite element space

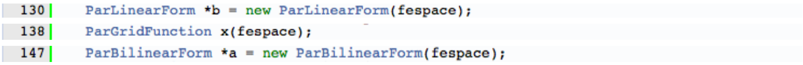


```
ParFiniteElementSpace *fespace = new ParFiniteElementSpace(pmesh, fec);
```



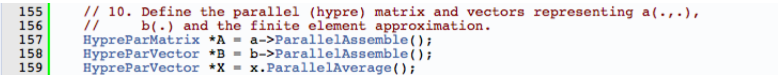
$$P : true\_dof \mapsto dof$$

- Parallel initial guess, linear/bilinear forms



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

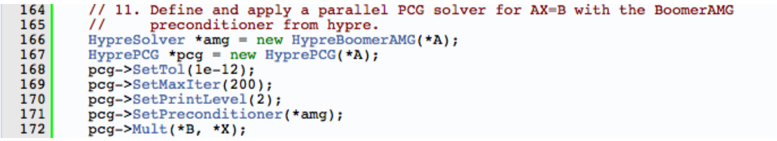
- Parallel assembly



```
// 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();
```

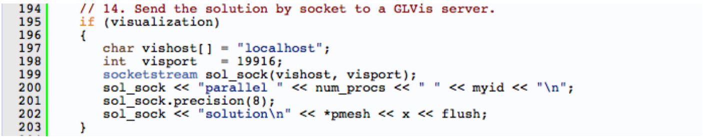
$$A = P^T a P \qquad B = P^T b \qquad x = PX$$

- Parallel linear solve with AMG

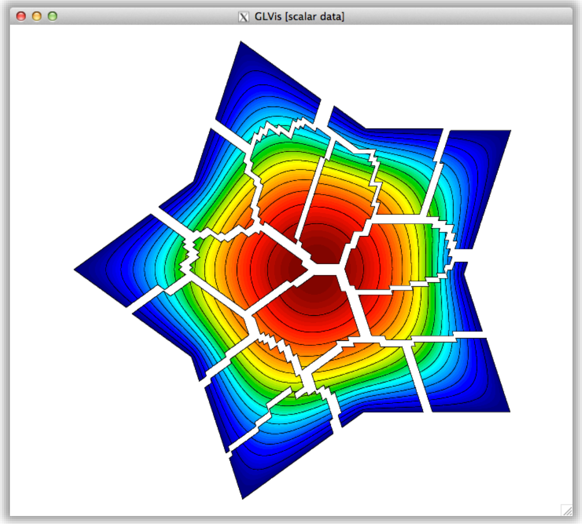


```
// 11. Define and apply a parallel PCG solver for AX=B with the BoomerAMG
//     preconditioner from hypre.
HypreSolver *amg = new HypreBoomerAMG(*A);
HyprePCG *pcg = new HyprePCG(*A);
pcg->SetTol(1e-12);
pcg->SetMaxIter(200);
pcg->SetPrintLevel(2);
pcg->SetPreconditioner(*amg);
pcg->Mult(*B, *X);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

- Visualization

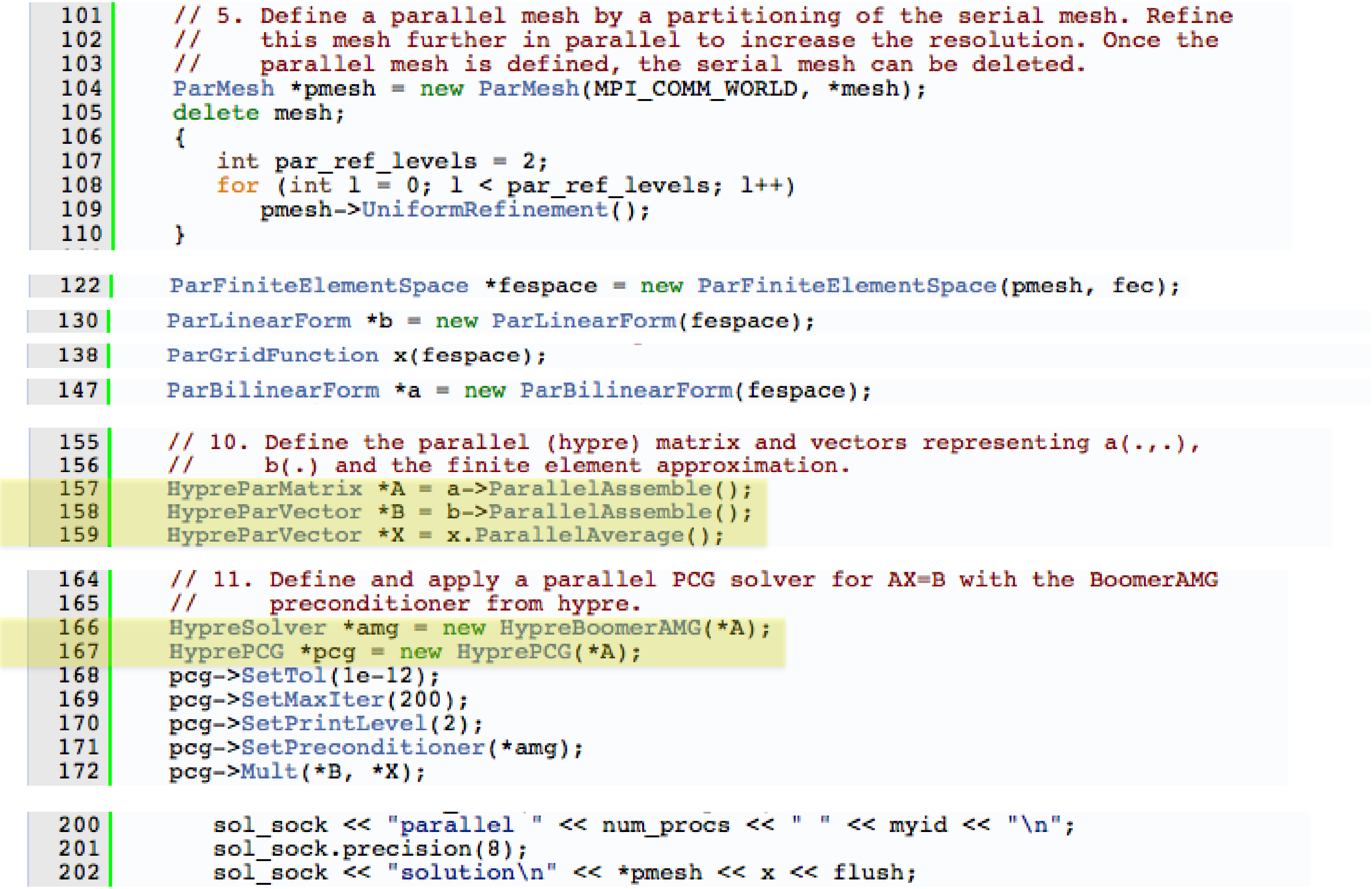


```
// 14. Send the solution by socket to a GLVis server.
if (visualization)
{
   char vishost[] = "localhost";
   int  visport   = 19916;
   socketstream sol_sock(vishost, visport);
   sol_sock << "parallel " << num_procs << " " << myid << "\n";
   sol_sock.precision(8);
   sol_sock << "solution\n"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



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

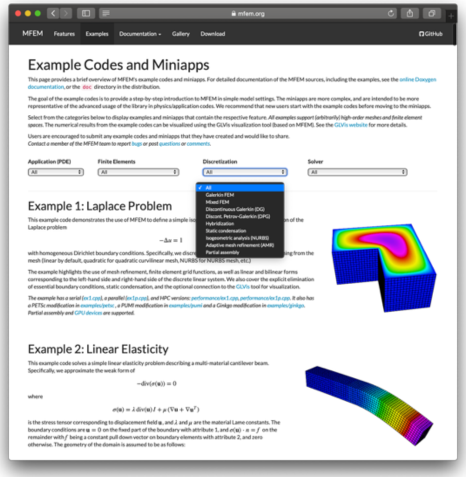
Example 1 – parallel Laplace equation



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

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

   ParLinearForm *b = new ParLinearForm(fespace);

   ParGridFunction x(fespace);

   ParBilinearForm *a = new ParBilinearForm(fespace);

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

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

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

MFEM example codes: mfem.org/examples

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



Example Codes and Miniapps

This page provides a brief overview of MFEM's example codes and miniapps. For detailed documentation of the MFEM sources, including the examples, see the online Doxygen documentation, or the doc directory in the distribution.

The goal of the example codes is to provide a step-by-step introduction to MFEM in simple model settings. The miniapps are more complex, and are intended to be more representative of the advanced usage of the library in physics/application codes. We recommend that new users start with the example codes before moving to the miniapps.

Select from the categories below to display examples and miniapps that contain the respective 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 and miniapps that they have created and would like to share.
Contact a member of the MFEM team to report bugs or post questions or comments.

Application (PDE): All
Finite Elements: All
Discretization: All
Solver: All

✓ All
Galerkin FEM
Mixed FEM
Discontinuous Galerkin (DG)
Discont. Petrov-Galerkin (DPG)
Hybridization
Static condensation
Isogeometric analysis (NURBS)
Adaptive mesh refinement (AMR)
Partial assembly

Example 1: Laplace Problem

This example code demonstrates the use of MFEM to define a simple iso n of the Laplace problem

$$-\Delta u = 1$$

with homogeneous Dirichlet boundary conditions. Specifically, we discr ing 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 essential boundary conditions, static condensation, and the optional connection to the GLVis tool for visualization.

The example has a serial (ex1.cpp), a parallel (ex1p.cpp), and HPC versions: performance/ex1.cpp, performance/ex1p.cpp. It also has a PETSc modification in examples/petsc, a PUMI modification in examples/pumi and a Ginkgo modification in examples/ginkgo. Partial assembly and GPU devices are supported.

Example 2: Linear Elasticity

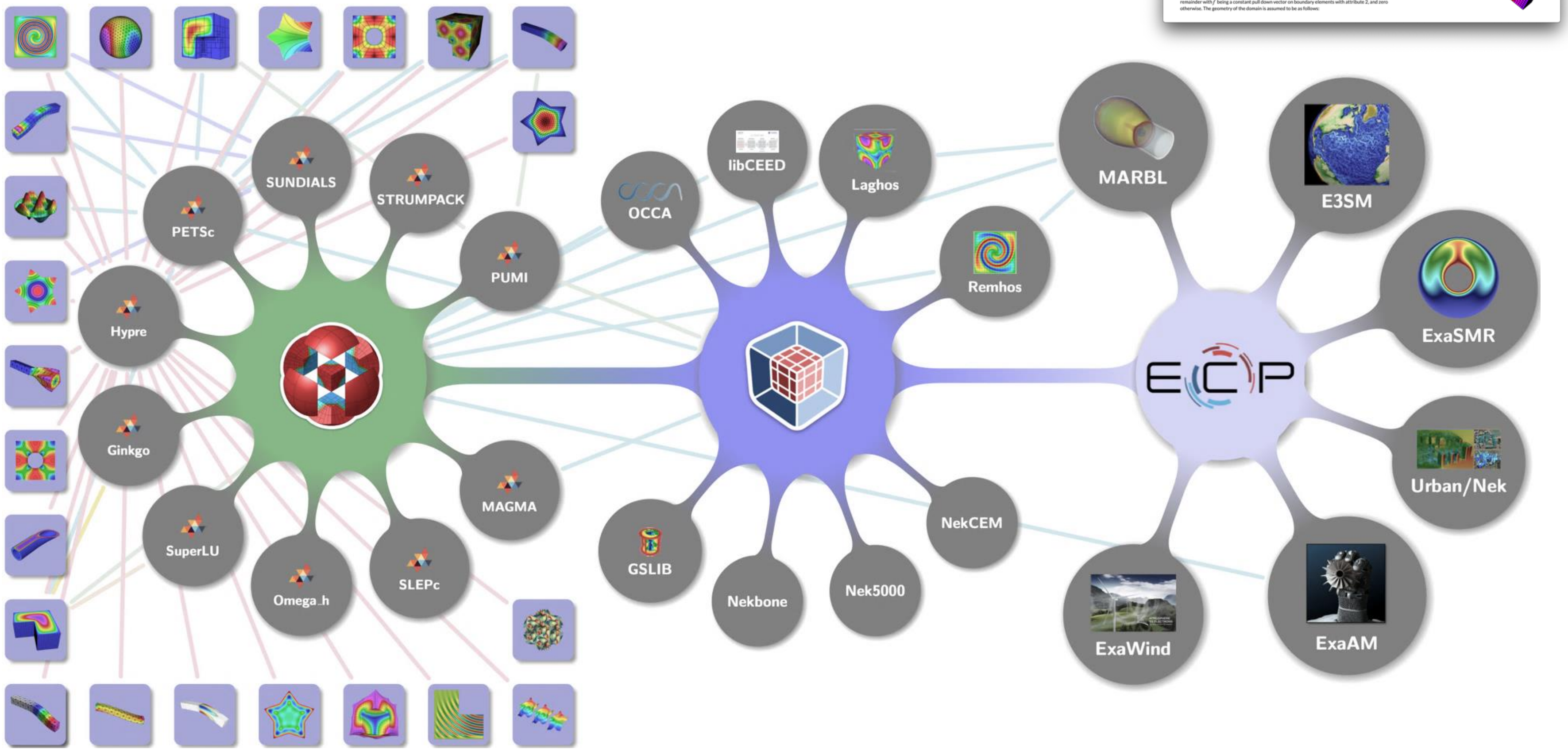
This example code solves a simple linear elasticity problem describing a multi-material cantilever beam. Specifically, we approximate the weak form of

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

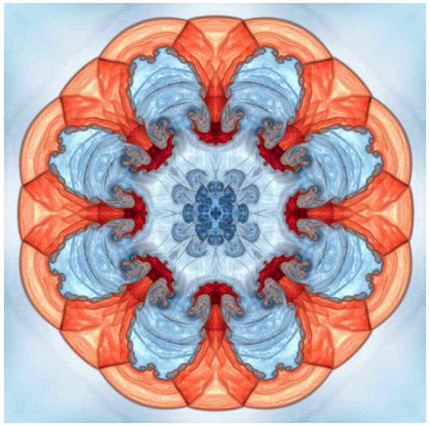
where

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

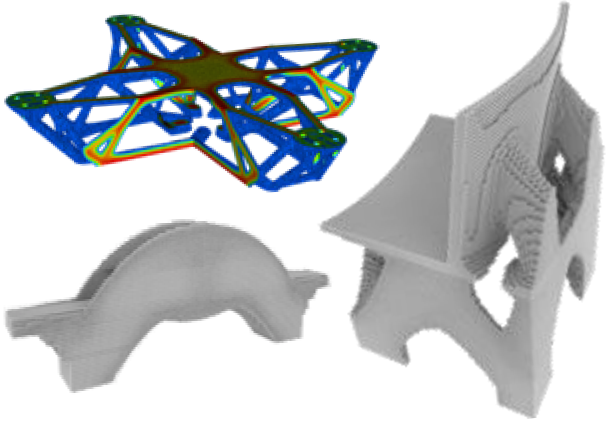
is the stress tensor corresponding to displacement field $\mathbf{u}$, and λ and μ are the material Lame constants. The boundary conditions are $\mathbf{u} = 0$ on the fixed part of the boundary with attribute 1, and $\sigma(\mathbf{u}) \cdot n = f$ on the remainder with f being a constant pull down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



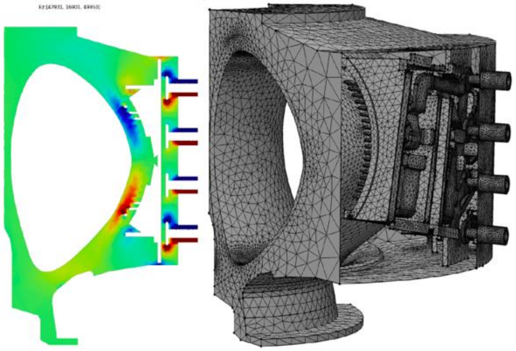
Some large-scale simulation codes powered by MFEM



Inertial confinement fusion (BLAST)

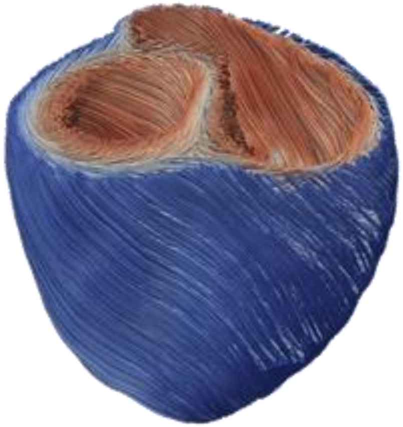


Topology optimization for additive manufacturing (LiDO)

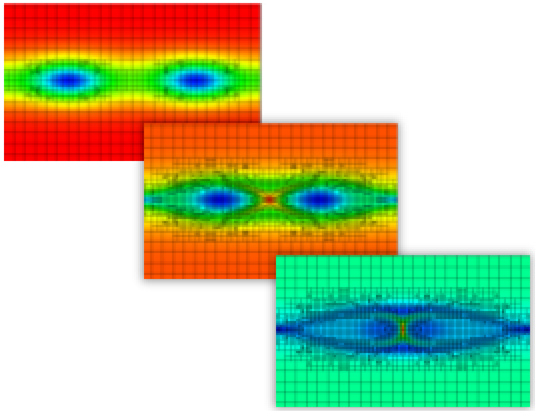


Core-edge tokamak EM wave propagation (SciDAC, RPI)

MRI modeling (Harvard Medical)



Heart modeling (Cardioid)



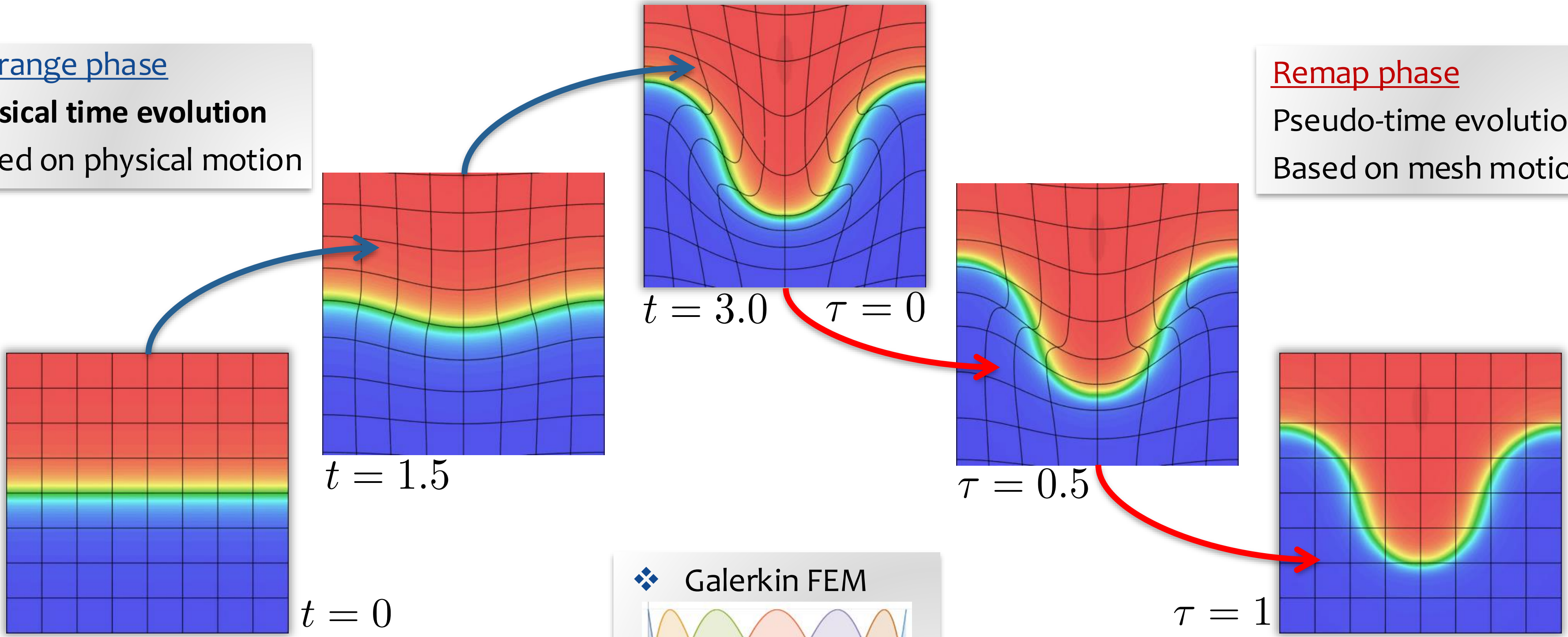
Adaptive MHD island coalescence (SciDAC, LANL)

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

Lagrange phase

Physical time evolution

Based on physical motion



Remap phase

Pseudo-time evolution

Based on mesh motion

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

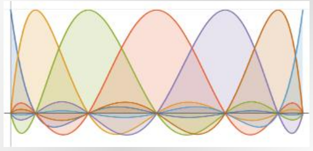
Momentum Conservation: $\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \nabla \cdot \sigma$

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

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

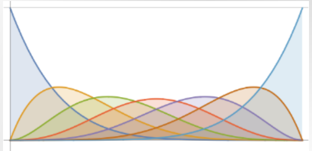
Equation of Motion: $\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{v}$

❖ Galerkin FEM



Gauss-Lobatto basis

❖ Discont. Galerkin



Bernstein basis

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

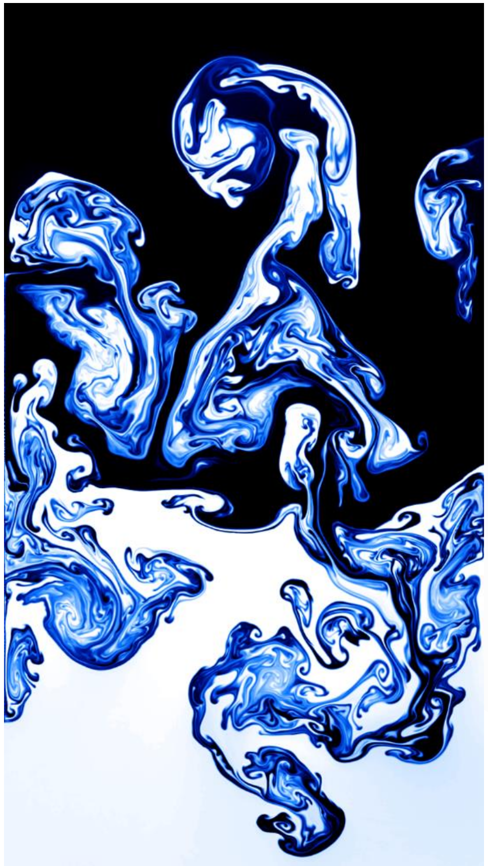
Momentum Conservation: $\frac{\mathrm{d}(\rho\vec{v})}{\mathrm{d}\tau} = \vec{v}_m \cdot \nabla(\rho\vec{v})$

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

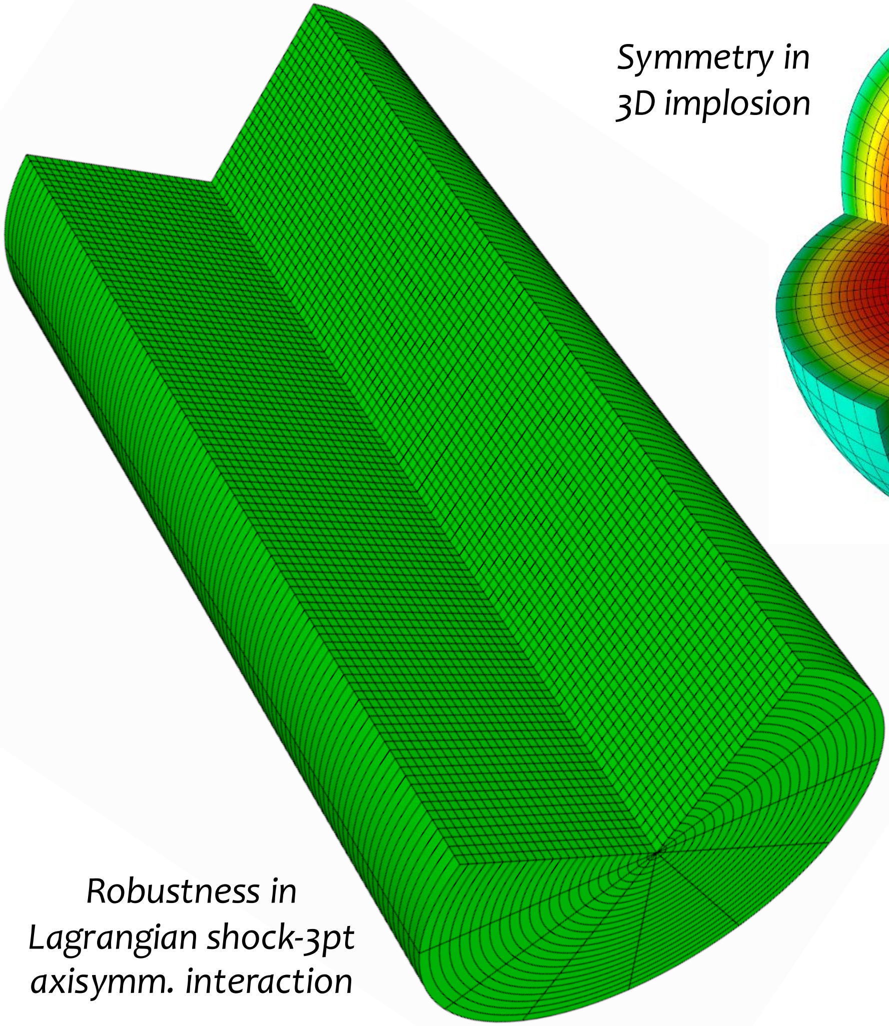
Energy Conservation: $\frac{\mathrm{d}(\rho e)}{\mathrm{d}\tau} = \vec{v}_m \cdot \nabla(\rho e)$

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

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

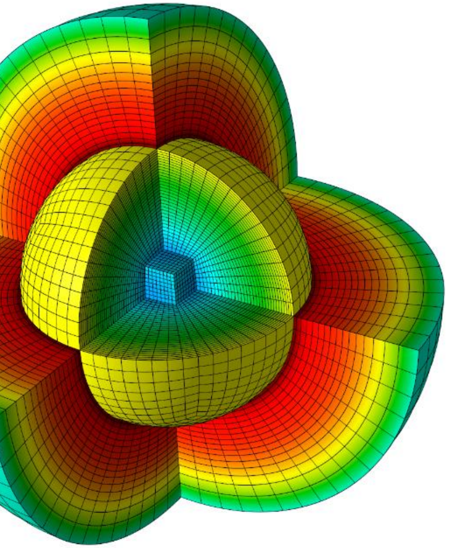


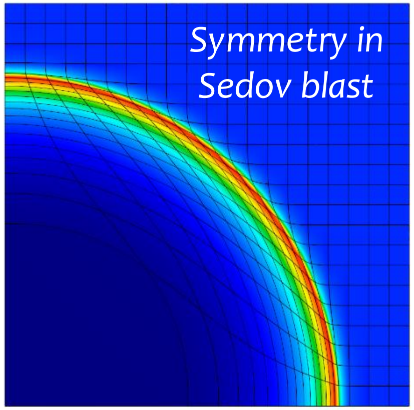
Parallel ALE for Q4 Rayleigh-Taylor instability (256 cores)



Robustness in Lagrangian shock-3pt axisymm. interaction

Symmetry in 3D implosion

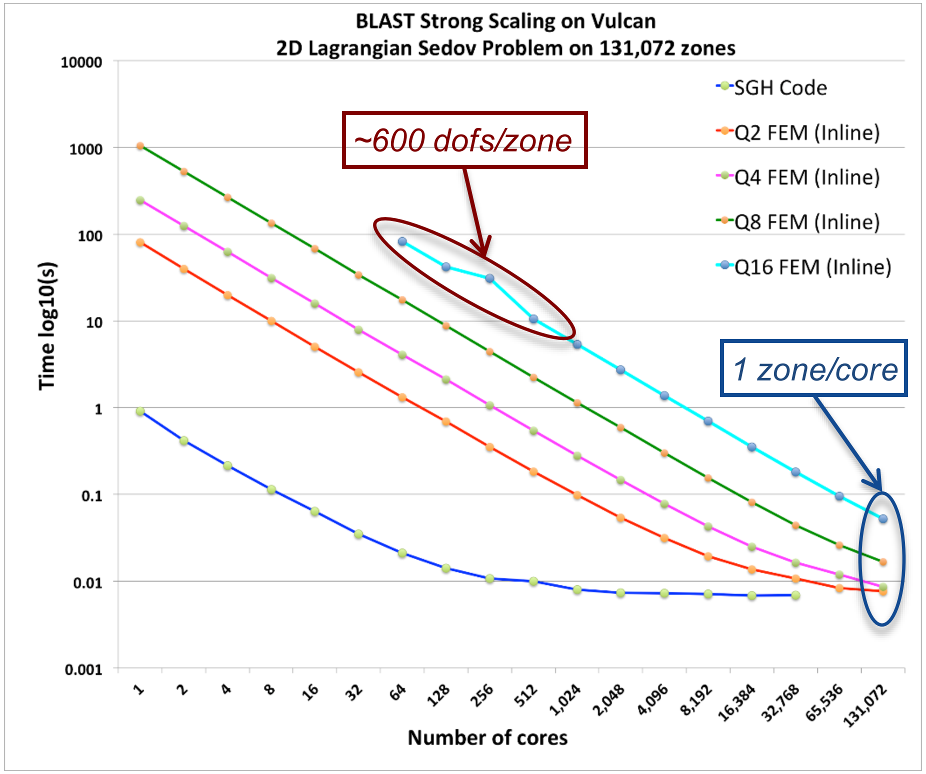




Symmetry in Sedov blast

High-order finite elements have excellent strong scalability

Strong scaling, p-refinement

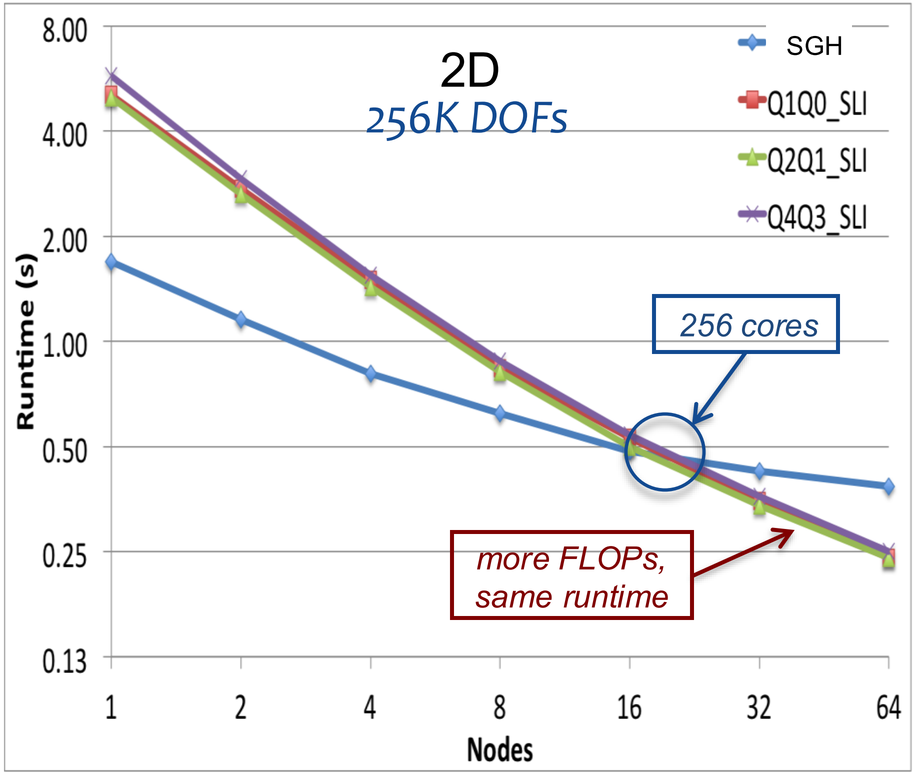


~600 dofs/zone

1 zone/core

Finite element partial assembly

Strong scaling, fixed #dofs



2D
256K DOFs

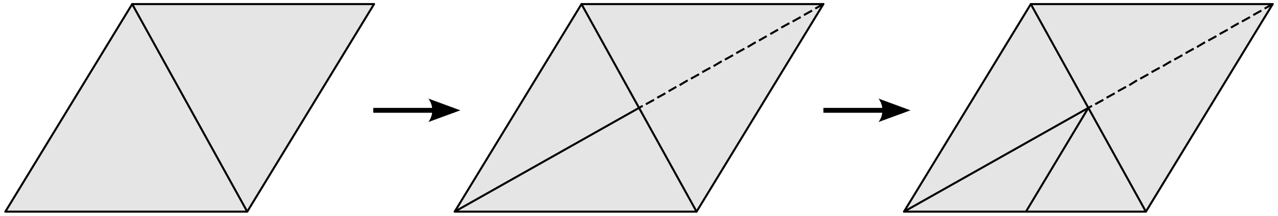
256 cores

more FLOPs, same runtime

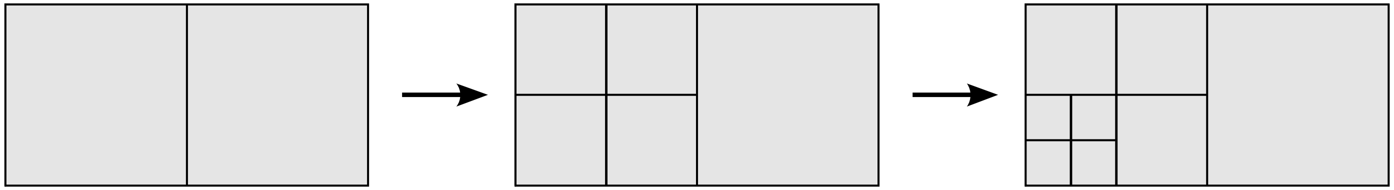
FLOPs increase faster than runtime

Conforming & Nonconforming Mesh Refinement

- Conforming refinement



- Nonconforming refinement



  - Natural for quadrilaterals and hexahedra

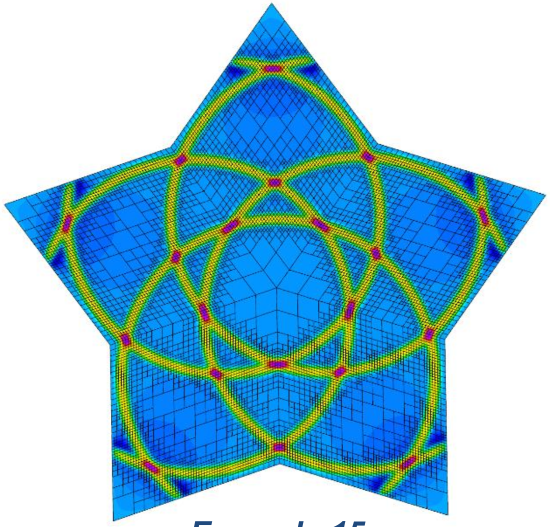
MFEM’s unstructured AMR infrastructure

Adaptive mesh refinement on library level:

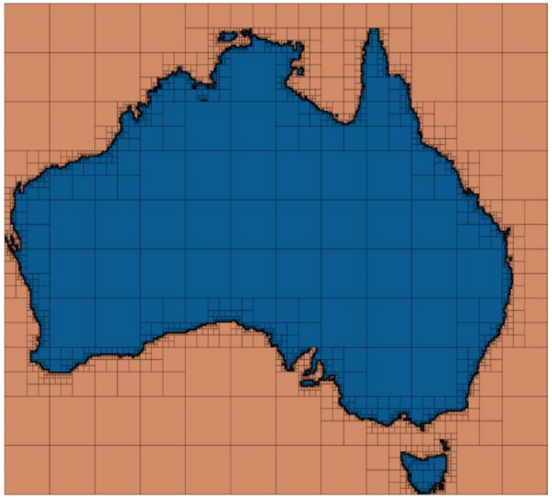
- Conforming local refinement on simplex meshes
- ***Non-conforming refinement for quad/hex meshes***
- h-refinement with fixed p

General approach:

- any high-order finite element space, H1, H(curl), H(div), …, on any high-order curved mesh
- 2D and 3D
- arbitrary order hanging nodes
- anisotropic refinement
- derefinement
- serial and parallel, including parallel load balancing
- independent of the physics (easy to incorporate in applications)



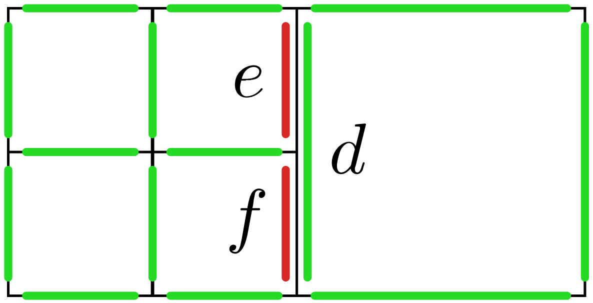
Example 15



Shaper miniapp

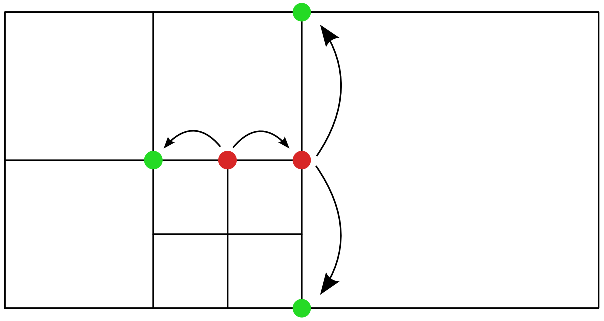
General nonconforming constraints

H(curl) elements



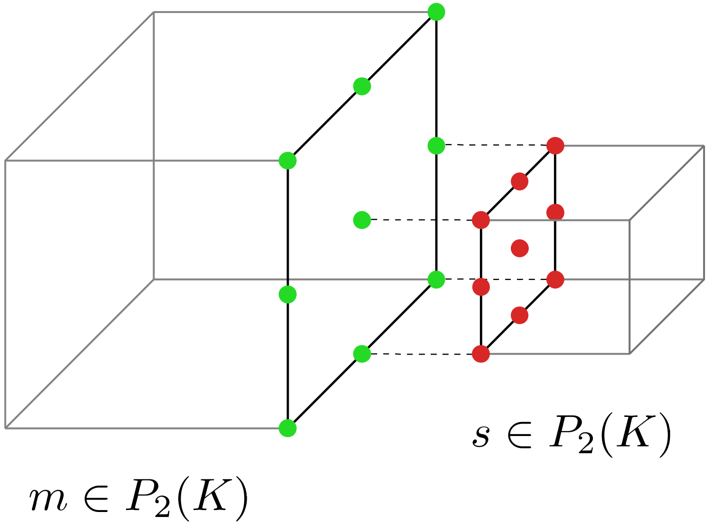
Constraint: e = f = d/2

Indirect constraints



More complicated in 3D…

High-order elements



Constraint: local interpolation matrix

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

Nonconforming variational restriction

- General constraint:

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

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

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

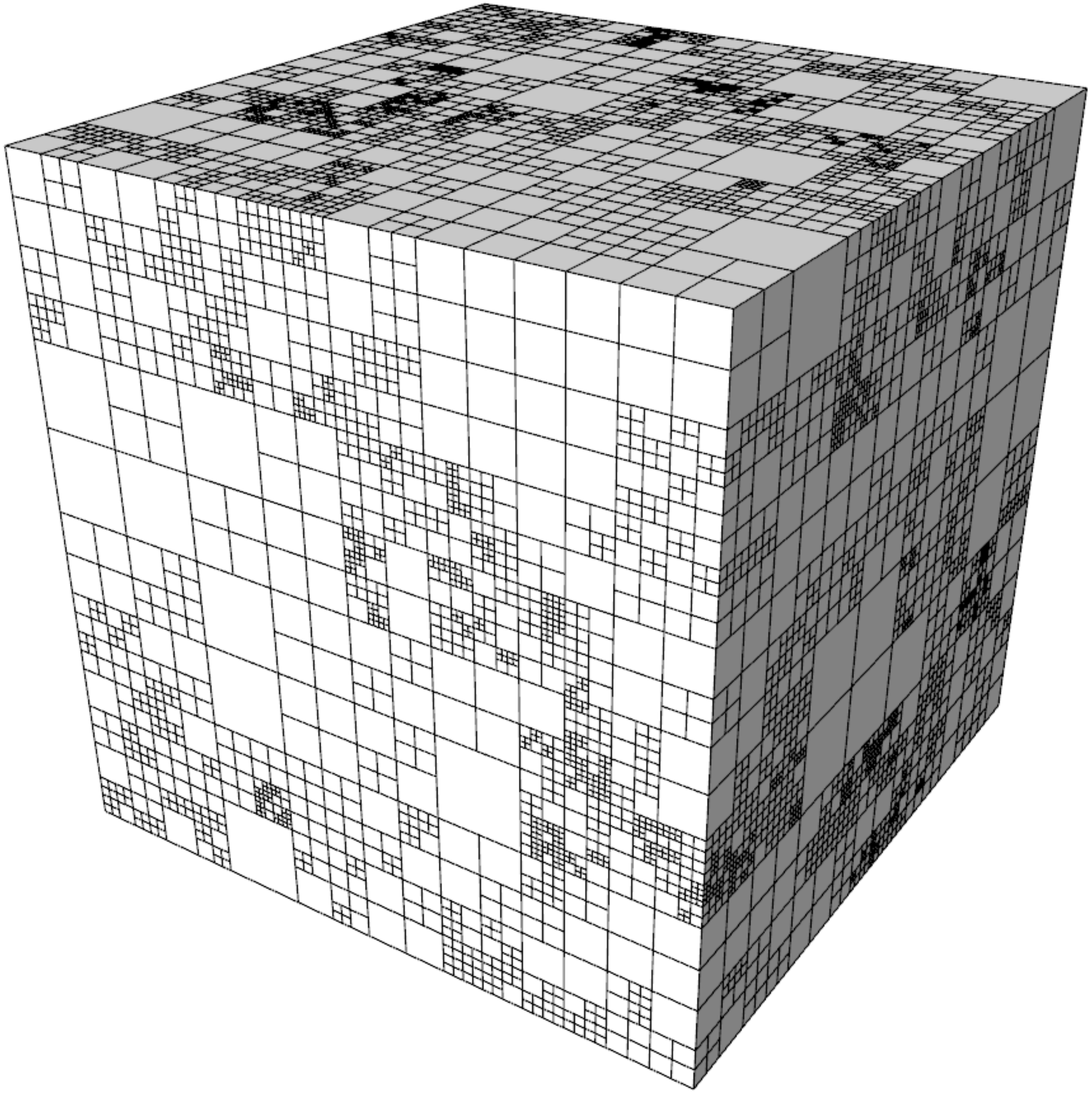
W – interpolation for slave DOFs

- Constrained problem:

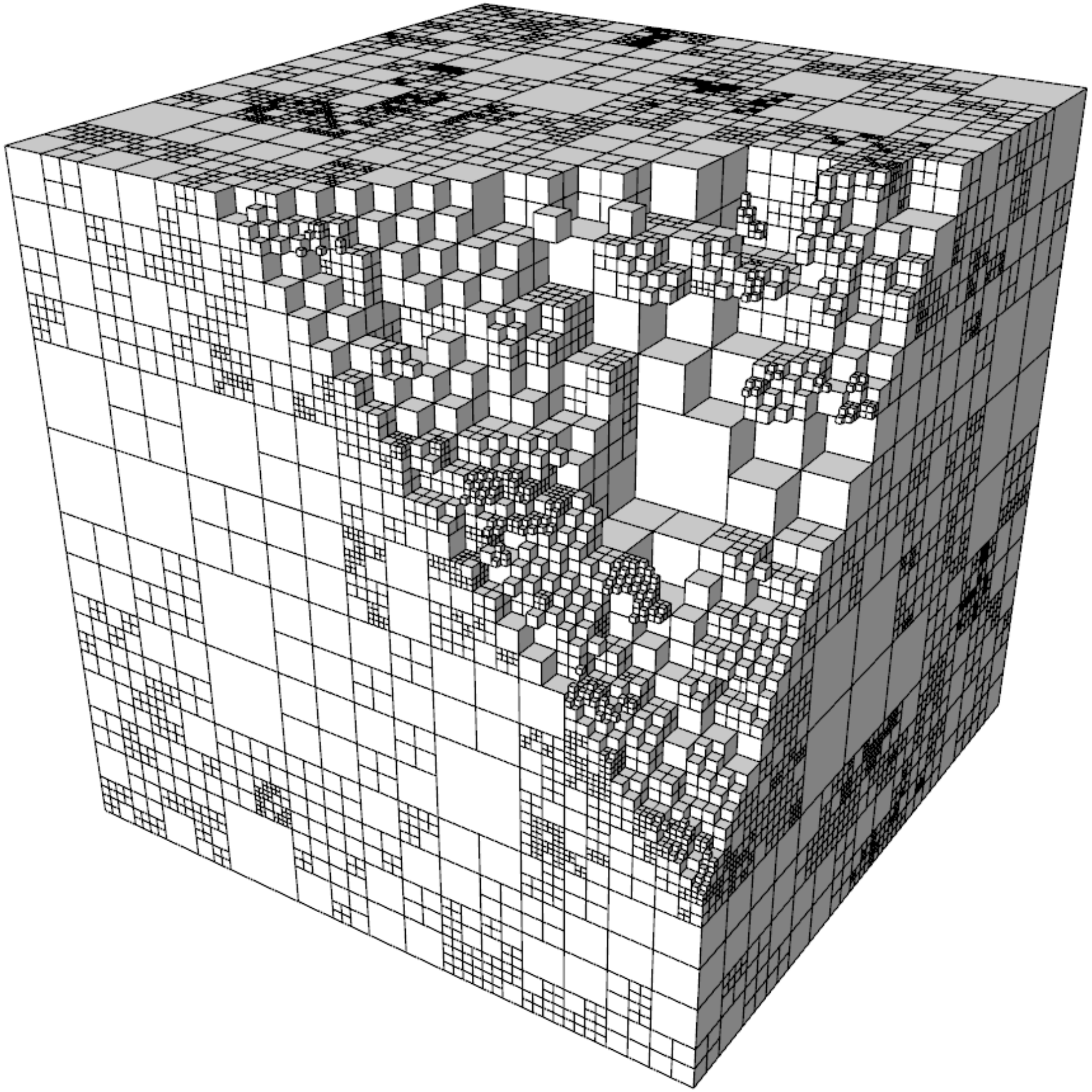
$$P^T A P x = P^T b,$$

$$y = Px.$$

Nonconforming variational restriction

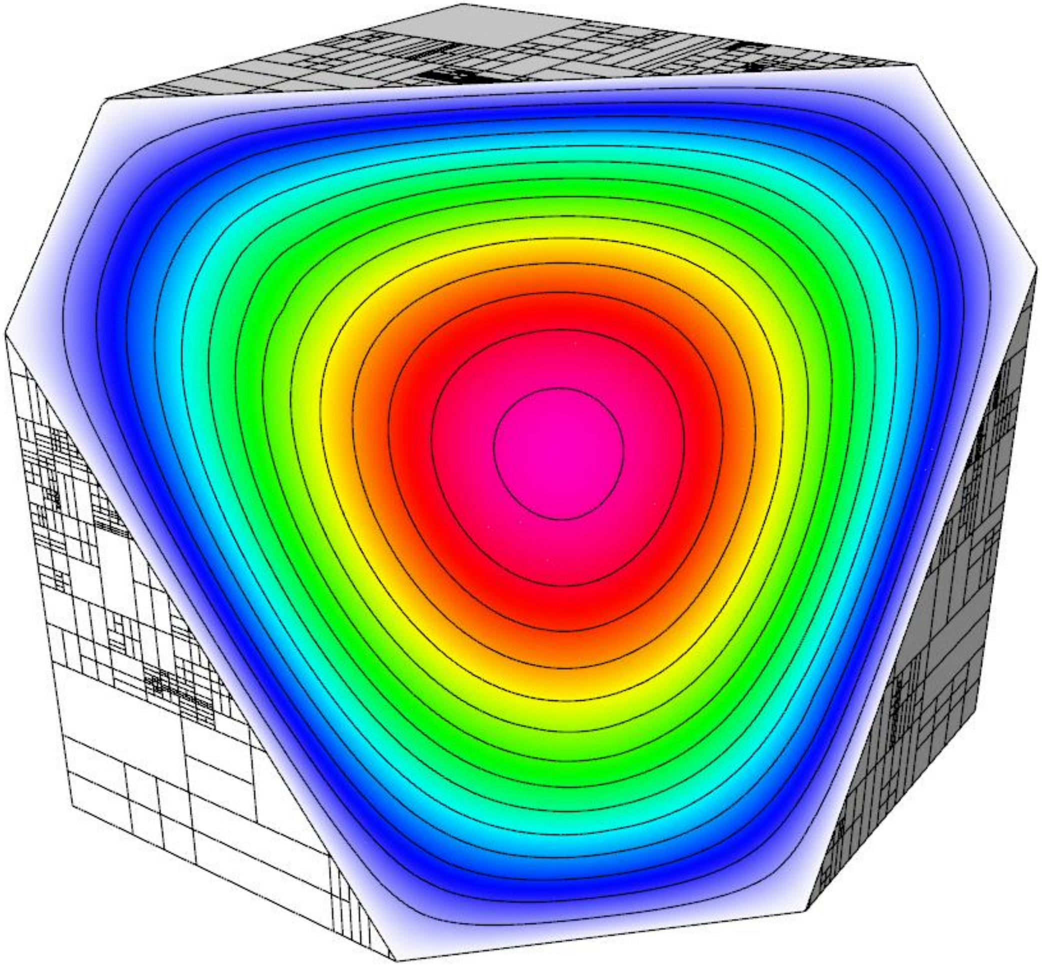


Nonconforming variational restriction



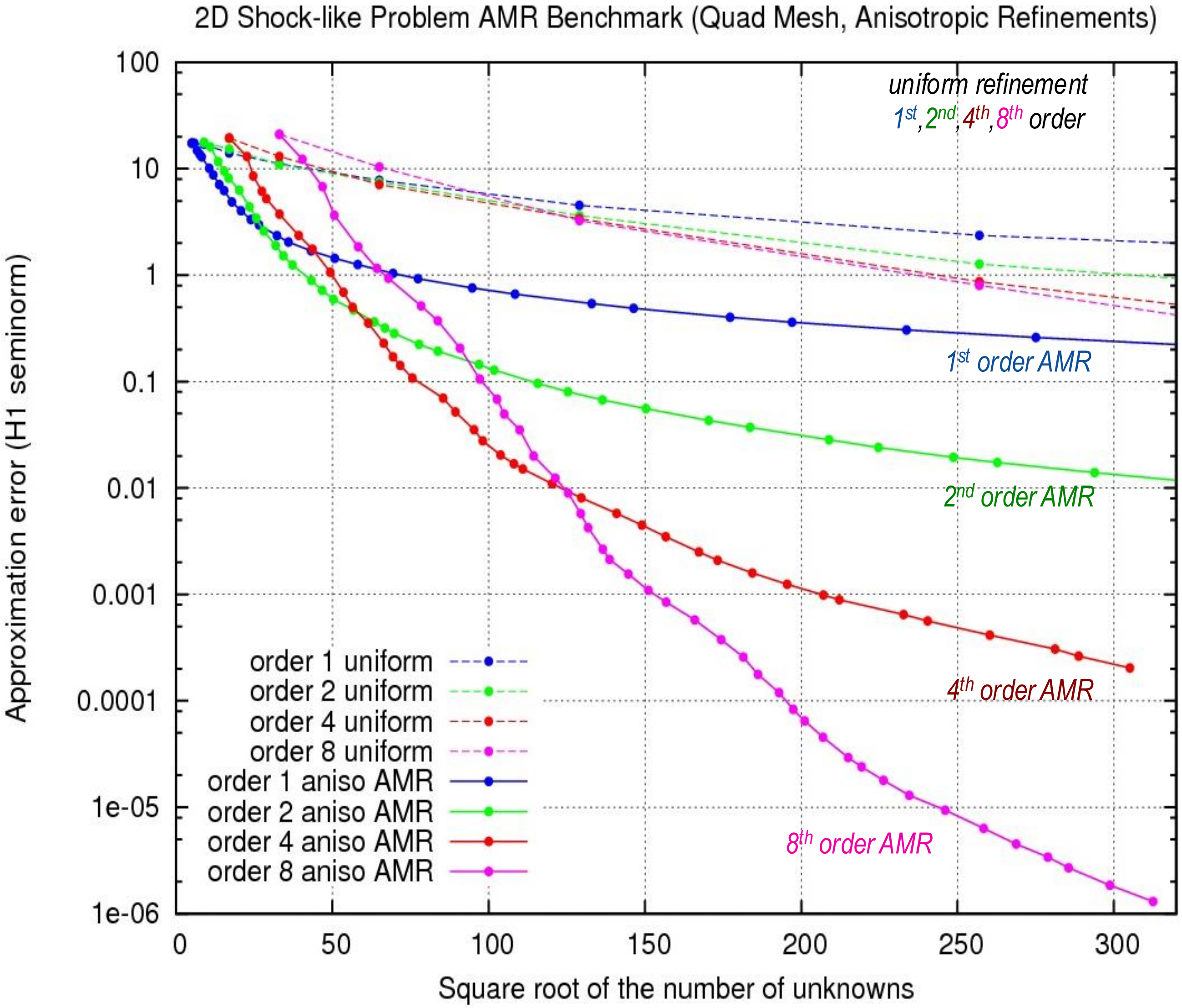
Regular assembly of A on the elements of the (cut) mesh

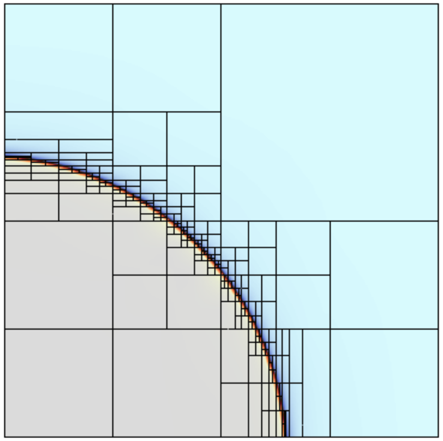
Nonconforming variational restriction

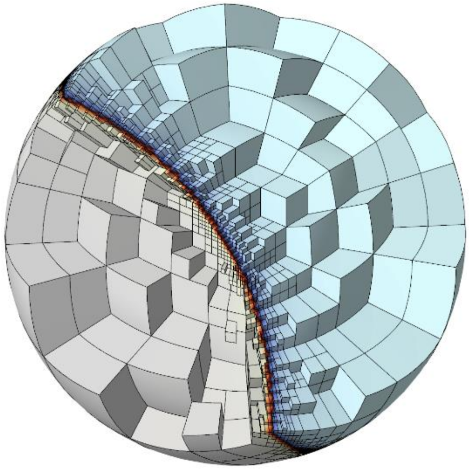


Conforming solution y = P x

AMR = smaller error for same number of unknowns

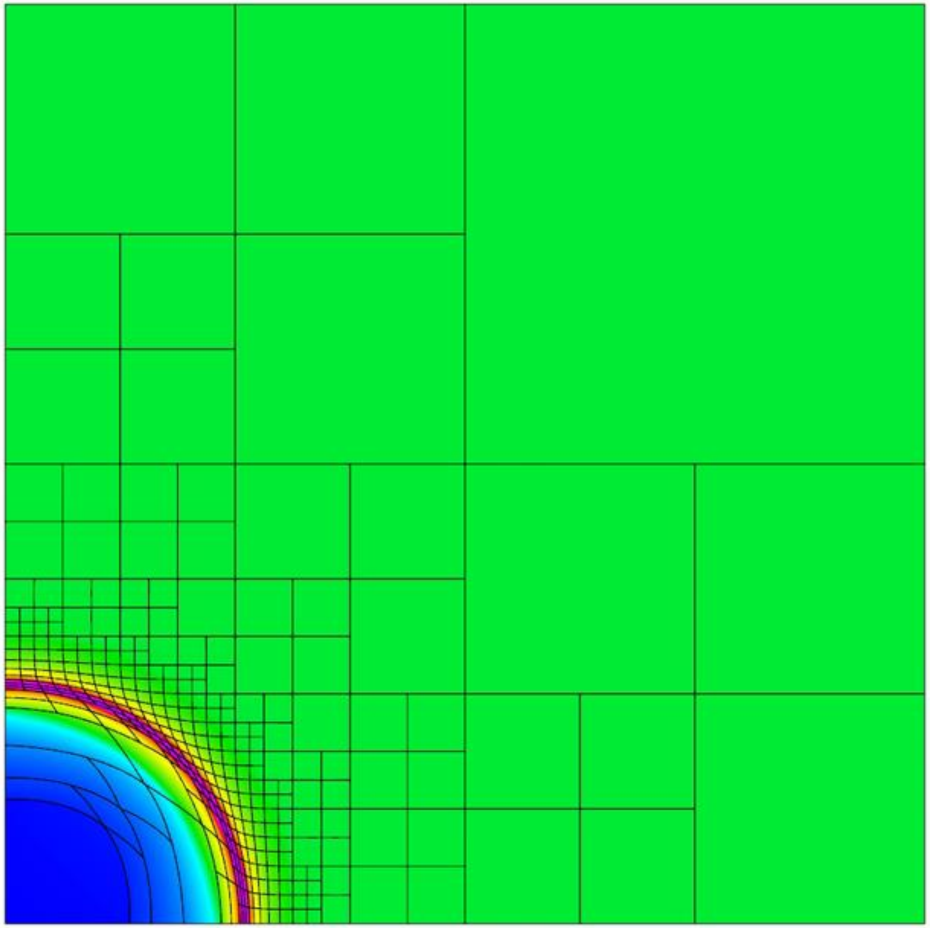




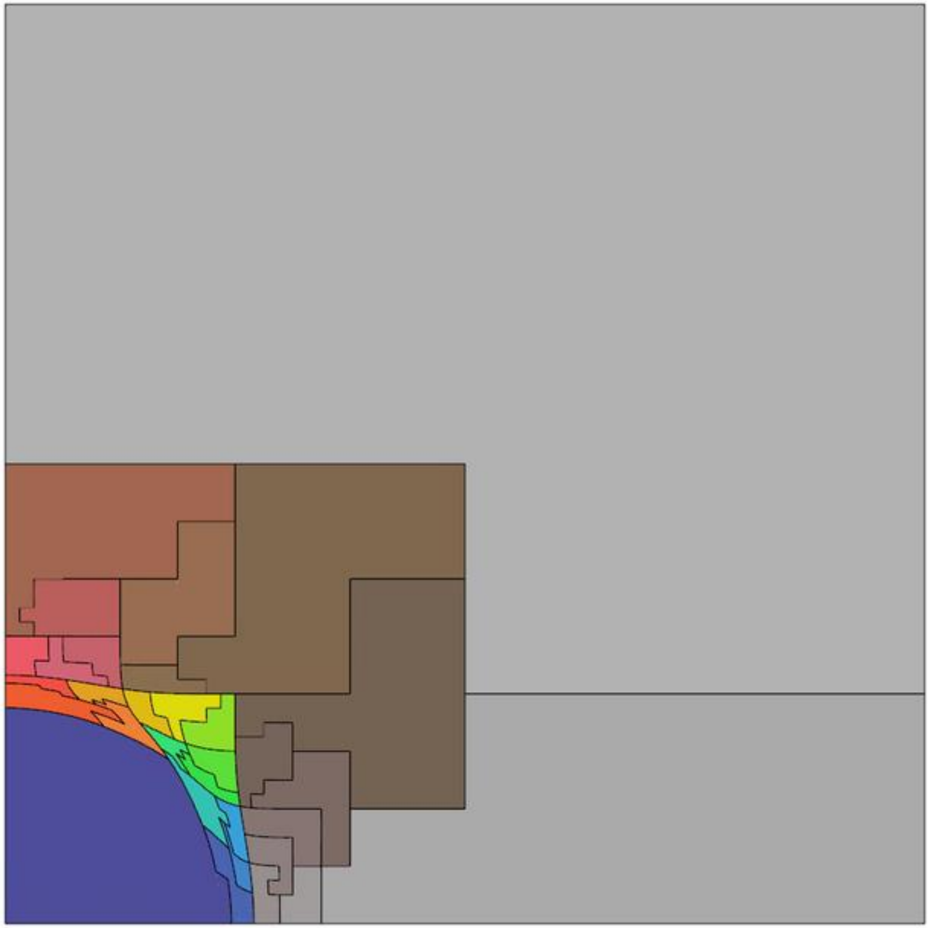


Anisotropic adaptation to shock-like fields in 2D & 3D

Parallel dynamic AMR, Lagrangian Sedov problem

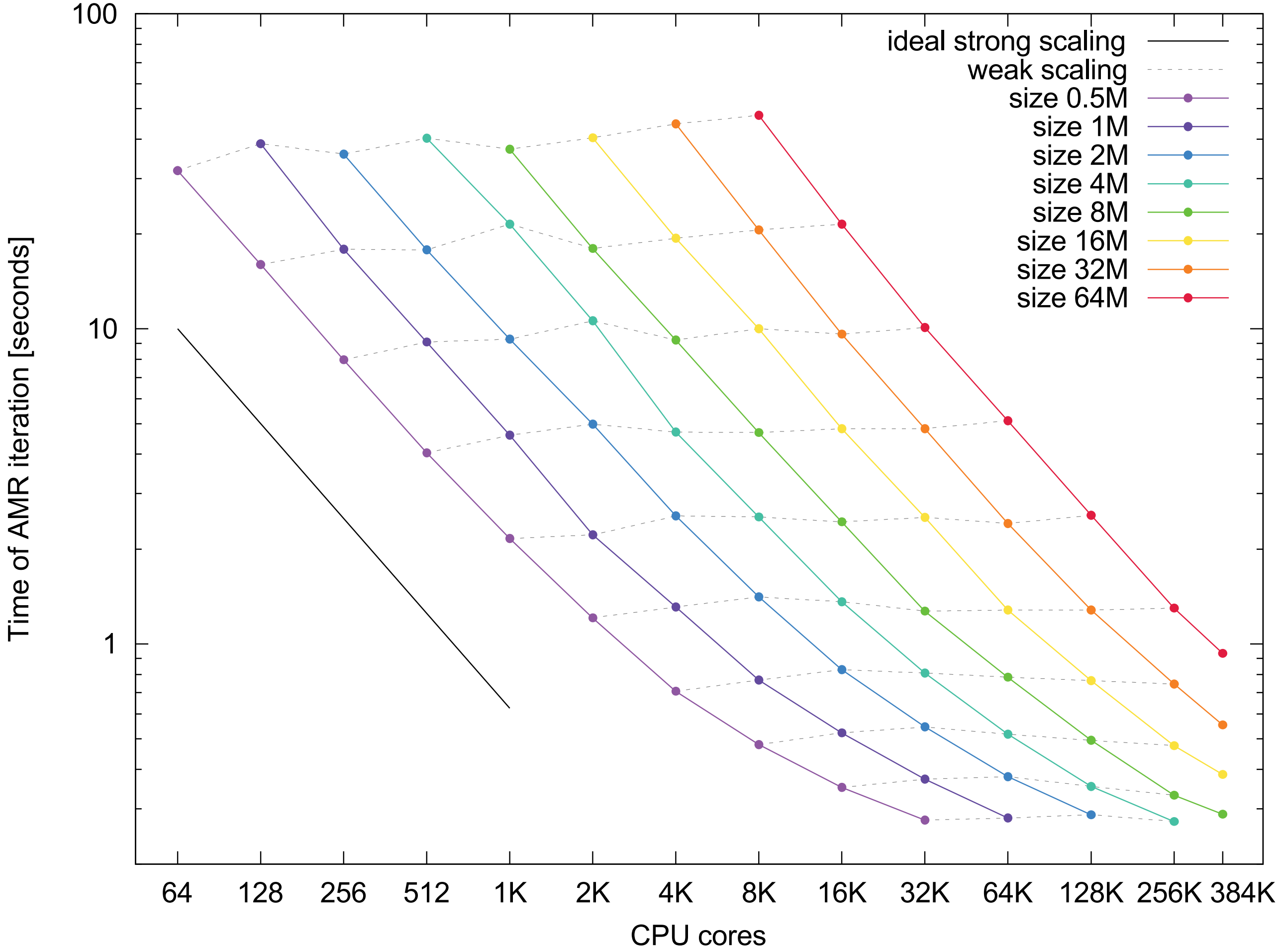


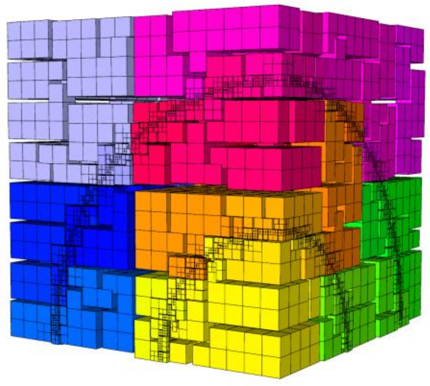
Adaptive, viscosity-based refinement and derefinement. 2^nd^ order Lagrangian Sedov



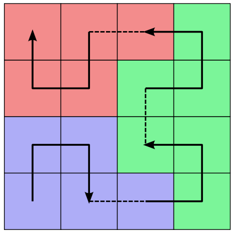
Parallel load balancing based on space-filling curve partitioning, 16 cores

Parallel AMR scaling to ~400K MPI tasks





Parallel decomposition (2048 domains shown)



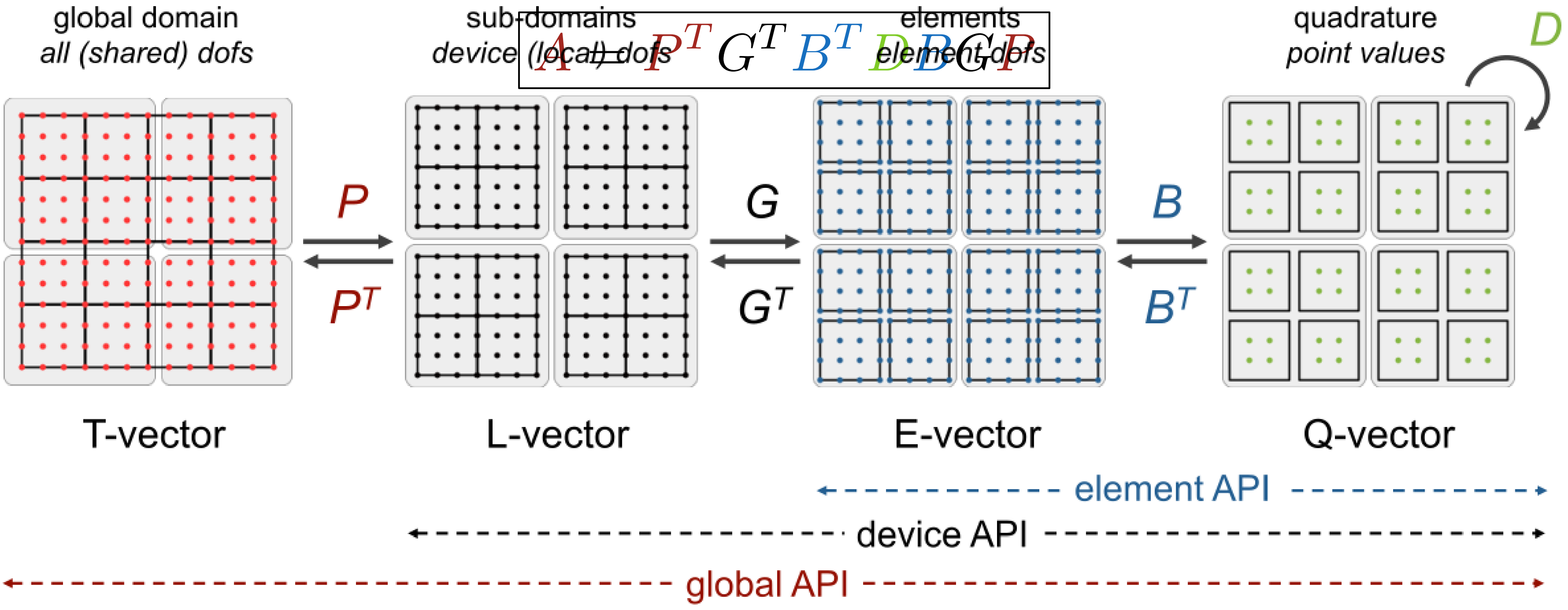
Parallel partitioning via Hilbert curve

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

Fundamental finite element operator decomposition

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

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



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

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

Example of a fast high-order operator

***Poisson problem** in variational form*

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

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

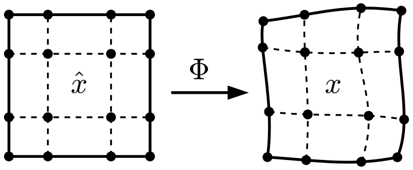
***Stiffness matrix** (unit coefficient)*

$$\underset{\overset{\uparrow}{A_{ij}}}{\int_\Omega \nabla\varphi_i \nabla\varphi_j} = \sum_E \int_E \nabla\varphi_i \nabla\varphi_j$$

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

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

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

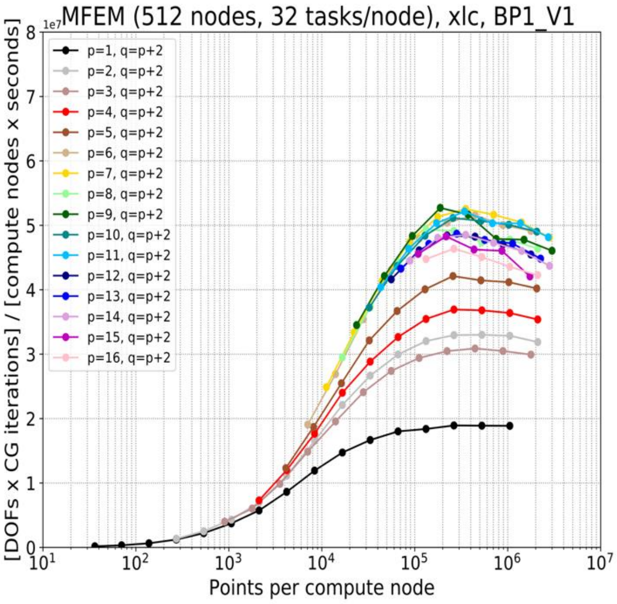


- *G* is usually Boolean (except AMR)
- Element matrices $A_E = B^T D B$, are full, account for bulk of the physics, can be applied in parallel

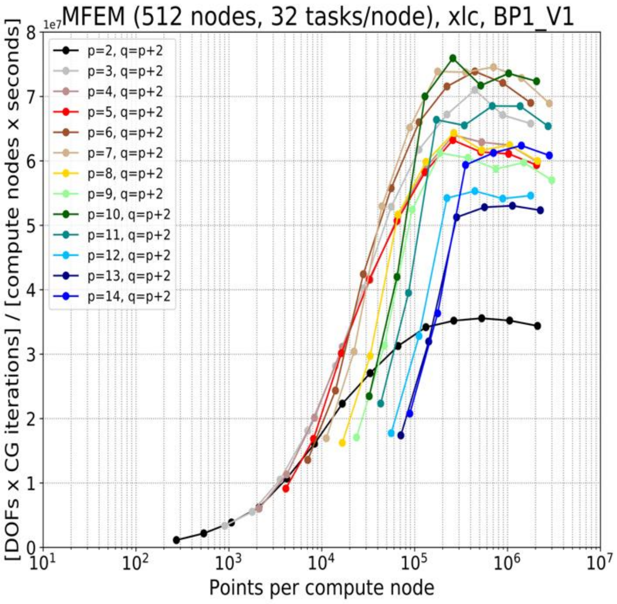
$$\begin{bmatrix} A^1 & & & \\ & A^2 & & \\ & & \ddots & \\ & & & A^4 \end{bmatrix}$$

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

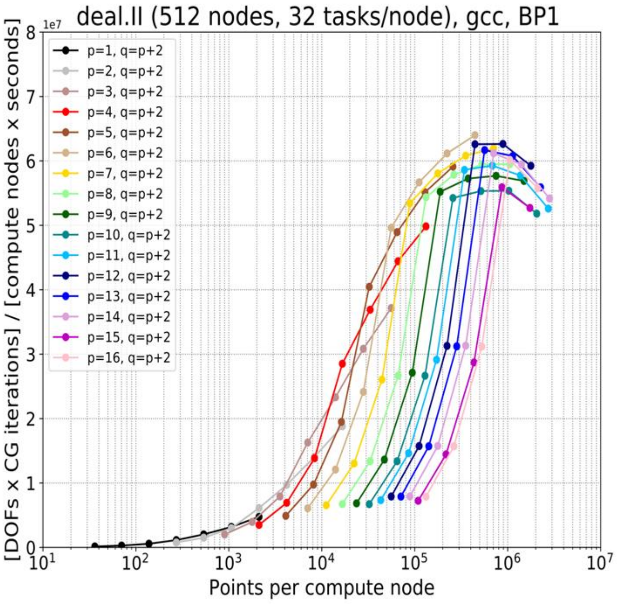
CEED BP1 bakeoff on BG/Q



Nek5000



MFEM-improved

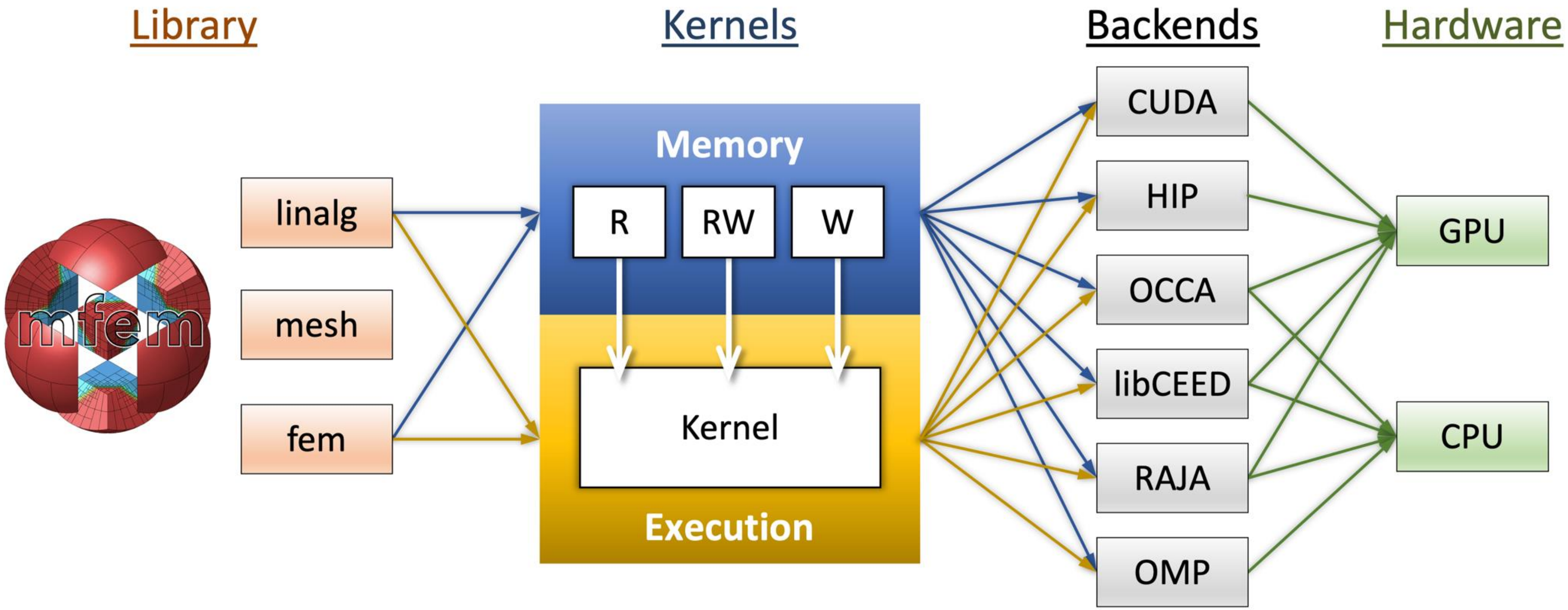


deal.ii

- ✔ All runs done on BG/Q (for repeatability), 16384 MPI ranks. Order p = 1, ...,16; quad. points q = p + 2.
- ✔ BP1 results of MFEM+xlc (left), MFEM+xlc+intrinsics (center), and deal.ii + gcc (right) on BG/Q.
- ✔ **Paper: "Scalability of High-Performance PDE Solvers", IJHPCA, 2020**
- ✔ Cooperation/collaboration is what makes the bake-offs rewarding.

Device support in MFEM

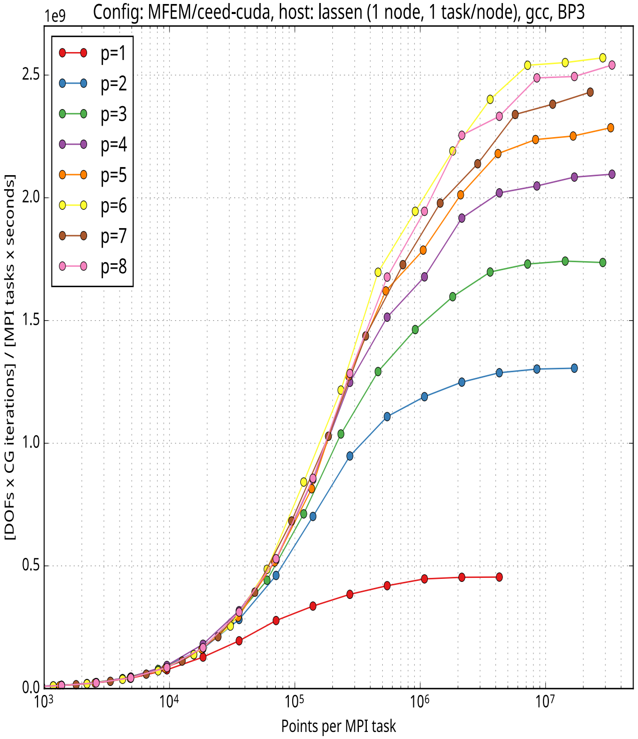
MFEM support GPU acceleration in many linear algebra and finite element operations



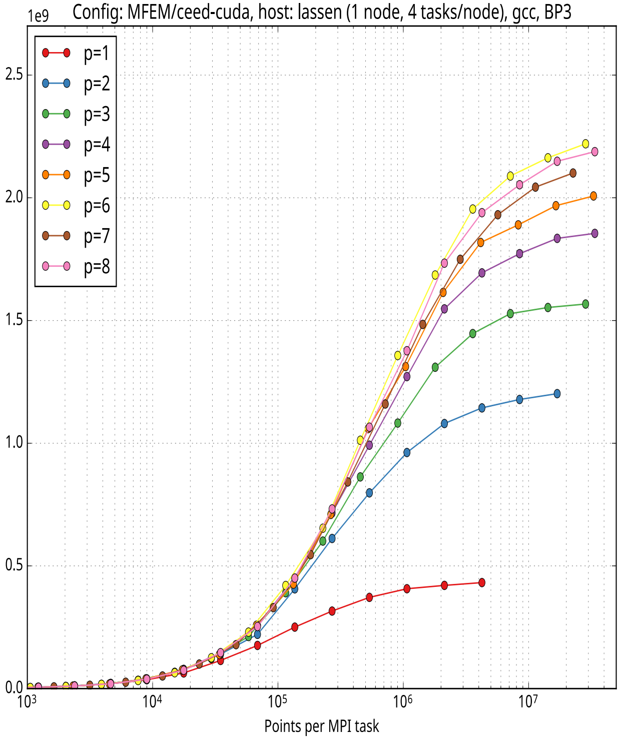
- Several MFEM examples + miniapps have been ported with small changes
- Many kernels have a single source for CUDA, RAJA and OpenMP backends
- **Backends are runtime selectable, can be mixed**
- **Recent improvements in CUDA, HIP, RAJA, SYCL, …**

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

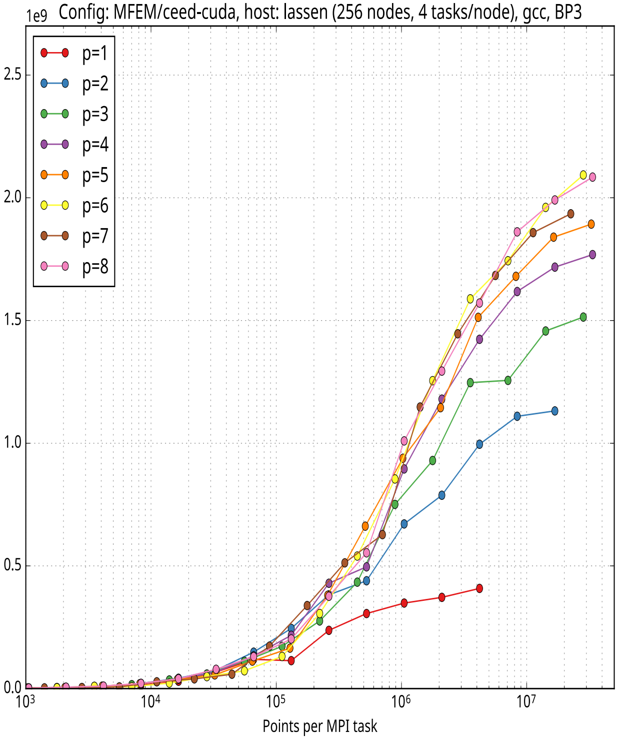
MFEM performance on multiple GPUs



1 GPU



4 GPUs



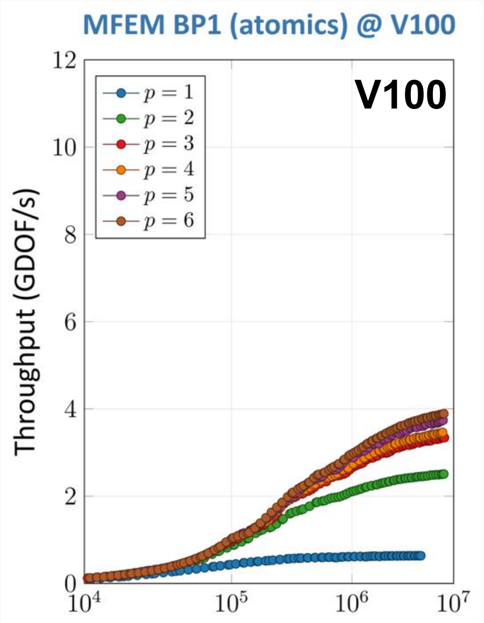
1024 GPUs

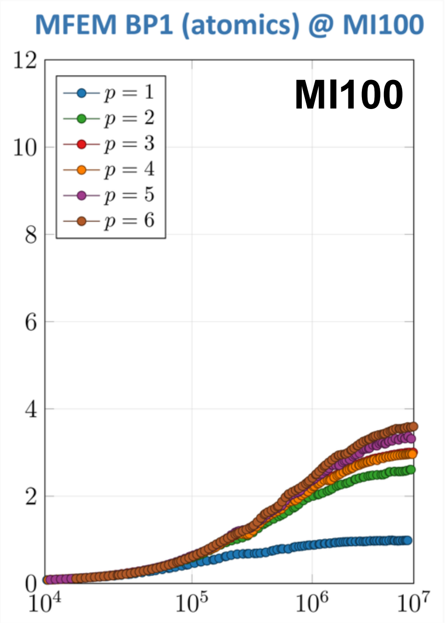
Single GPU performance: **2.6 GDOF/s**
Problem size: 10+ million

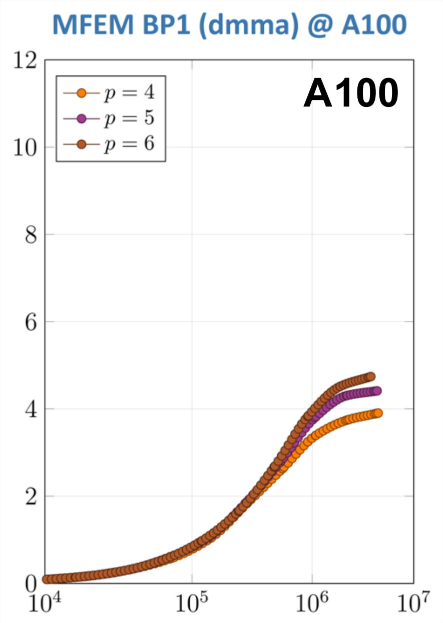
Best total performance: **2.1 TDOF/s**
Largest size: 34 billion

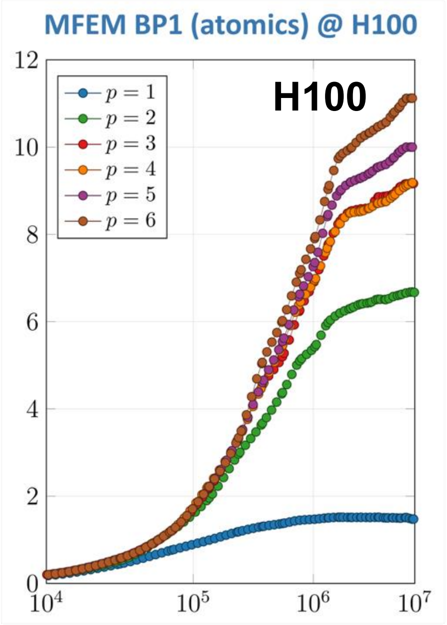
Optimized kernels for MPI buffer packing/unpacking on the GPU

Recent improvements on NVIDIA and AMD GPUs


MFEM BP1 (atomics) @ V100


MFEM BP1 (atomics) @ MI100


MFEM BP1 (dmma) @ A100


MFEM BP1 (atomics) @ H100

New MFEM GPU kernels: perform on both V100 + MI100,

have better strong scaling,

can utilize tensor cores on A100

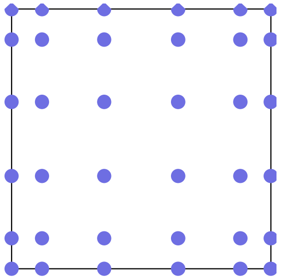
achieve 10+ GDOFs on H100

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

Matrix-free preconditioning

- ***Explicit matrix assembly impractical at high order:***
  - Polynomial degree p, spatial dimension d
  - Matrix assembly + sparse matvecs:
    - $\mathcal{O}(p^{2d})$ memory transfers
    - $\mathcal{O}(p^{3d})$ computations
    - can be reduced to $\mathcal{O}(p^{2d+1})$ computations by sum factorization
  - Matrix-free action of the operator (partial assembly):
    - $\mathcal{O}(p^{d})$ memory transfers – *optimal*
    - $\mathcal{O}(p^{d+1})$ computations – *nearly-optimal*
    - efficient iterative solvers ***if combined with effective preconditioners***

$\mathcal{O}(p^d)$ element dofs

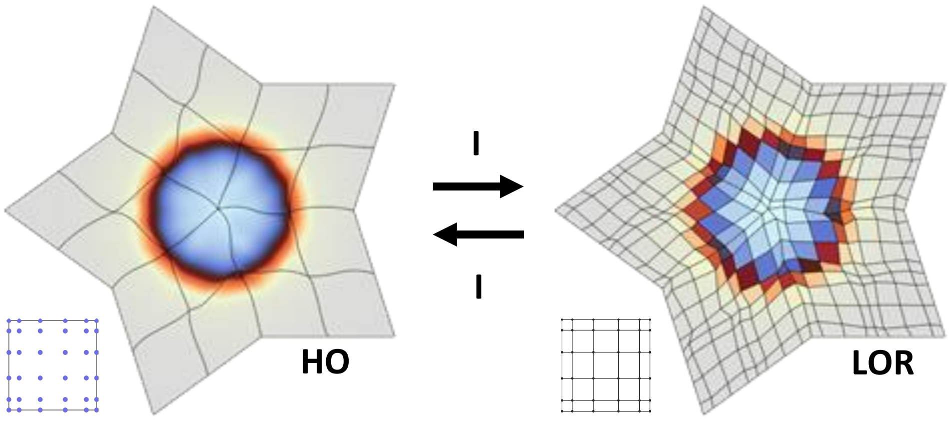


$p+1$

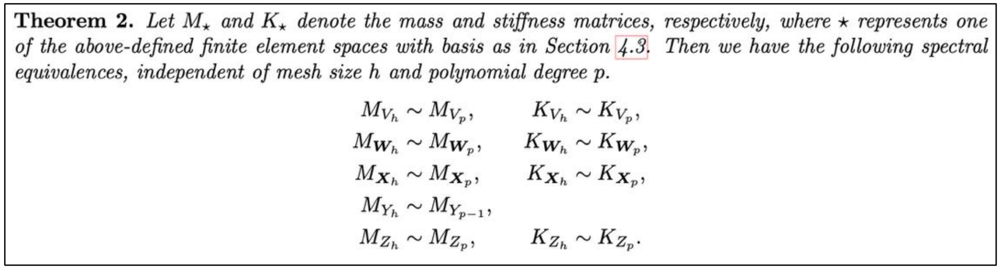
- ***Challenges:***
  - Traditional matrix-based preconditioners (e.g. AMG) not available
  - Condition number of diffusion systems grows like $\mathcal{O}(p^3/h^2)$

Low-Order-Refined (LOR) preconditioning

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



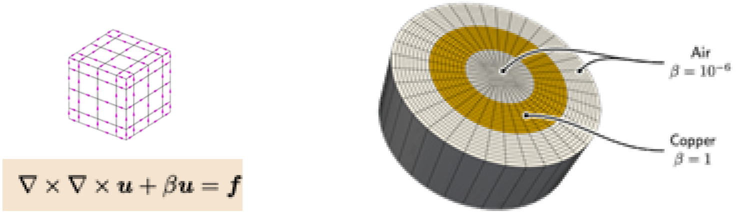
- Pick LOR space and HO basis so **P=R=I** (Gerritsma, Dohrmann)
- **A$_{LOR}$** is sparse and spectrally equivalent to **A$_{HO}$**



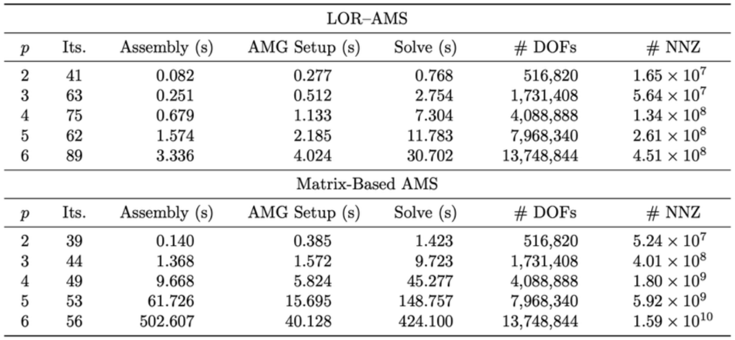
Theorem 2. *Let $M_\star$ and $K_\star$ denote the mass and stiffness matrices, respectively, where $\star$ represents one of the above-defined finite element spaces with basis as in Section 4.3. Then we have the following spectral equivalences, independent of mesh size h and polynomial degree p.*

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

- **(A$_{HO}$)$^{-1}$** ≈ **(A$_{LOR}$)$^{-1}$** ≈ **B$_{LOR}$** - can use BoomerAMG, AMS, ADS



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

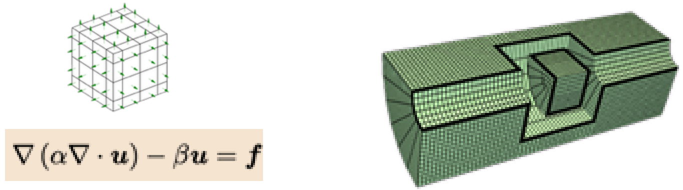


LOR–AMS

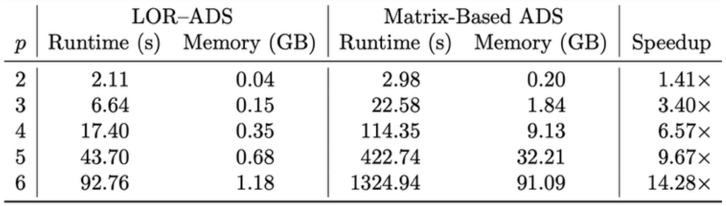
p	Its.	Assembly (s)	AMG Setup (s)	Solve (s)	# DOFs	# NNZ
2	41	0.082	0.277	0.768	516,820	1.65×10^7
3	63	0.251	0.512	2.754	1,731,408	5.64×10^7
4	75	0.679	1.133	7.304	4,088,888	1.34×10^8
5	62	1.574	2.185	11.783	7,968,340	2.61×10^8
6	89	3.336	4.024	30.702	13,748,844	4.51×10^8

Matrix-Based AMS

p	Its.	Assembly (s)	AMG Setup (s)	Solve (s)	# DOFs	# NNZ
2	39	0.140	0.385	1.423	516,820	5.24×10^7
3	44	1.368	1.572	9.723	1,731,408	4.01×10^8
4	49	9.668	5.824	45.277	4,088,888	1.80×10^9
5	53	61.726	15.695	148.757	7,968,340	5.92×10^9
6	56	502.607	40.128	424.100	13,748,844	1.59×10^{10}



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



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

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

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

- **More information and publications**
  - MFEM – **mfem.org**
  - BLAST – **computation.llnl.gov/projects/blast**
  - CEED – **ceed.exascaleproject.org**

- **Open-source software**

mfem

CEED
EXASCALE DISCRETIZATIONS

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

Q4 Rayleigh-Taylor single-material ALE on 256 processors

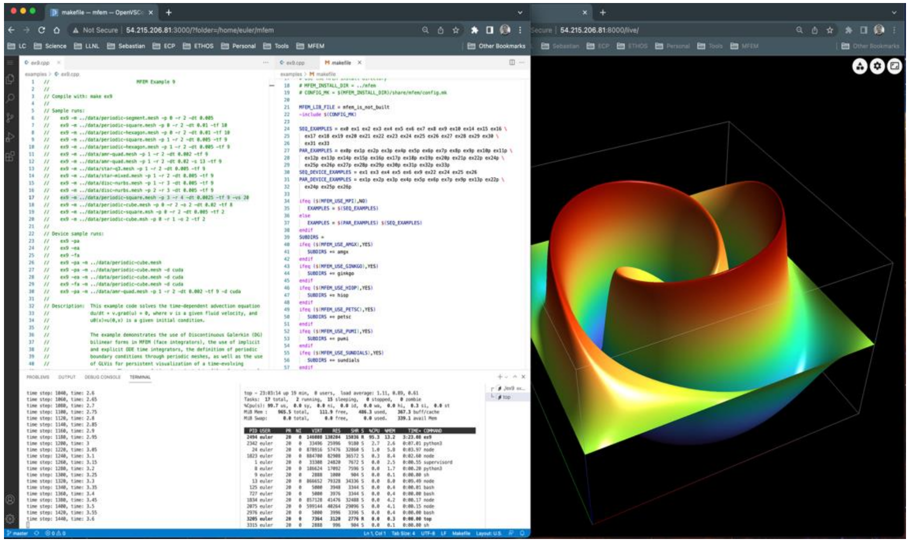
Upcoming MFEM Events

MFEM at LLNL HPC Tutorials

September 9, 2025

MFEM Community Workshop

September 10-11, 2025
Portland State University + Virtual



```
//                       MFEM Example 9
//
// Compile with: make ex9
//
// Sample runs:
//    ex9 -m ../data/periodic-segment.mesh -p 0 -r 2 -dt 0.005
//    ex9 -m ../data/periodic-square.mesh -p 0 -r 2 -dt 0.01 -tf 10
//    ex9 -m ../data/periodic-hexagon.mesh -p 0 -r 2 -dt 0.01 -tf 10
//    ex9 -m ../data/periodic-square.mesh -p 1 -r 2 -dt 0.005 -tf 9
//    ex9 -m ../data/periodic-hexagon.mesh -p 1 -r 2 -dt 0.005 -tf 9
//    ex9 -m ../data/amr-quad.mesh -p 1 -r 2 -dt 0.002 -tf 9
//    ex9 -m ../data/amr-quad.mesh -p 1 -r 2 -dt 0.02 -s 13 -tf 9
//    ex9 -m ../data/star-q3.mesh -p 1 -r 2 -dt 0.005 -tf 9
//    ex9 -m ../data/star-mixed.mesh -p 1 -r 2 -dt 0.005 -tf 9
//    ex9 -m ../data/disc-nurbs.mesh -p 1 -r 3 -dt 0.005 -tf 9
//    ex9 -m ../data/disc-nurbs.mesh -p 2 -r 3 -dt 0.005 -tf 9
//    ex9 -m ../data/periodic-square.mesh -p 3 -r 4 -dt 0.0025 -tf 9 -vs 20
//    ex9 -m ../data/periodic-cube.mesh -p 0 -r 2 -o 2 -dt 0.02 -tf 8
//    ex9 -m ../data/periodic-square.msh -p 0 -r 2 -dt 0.005 -tf 2
//    ex9 -m ../data/periodic-cube.msh -p 0 -r 1 -o 2 -tf 2
//
// Device sample runs:
//    ex9 -pa
//    ex9 -ea
//    ex9 -fa
//    ex9 -pa -m ../data/periodic-cube.mesh
//    ex9 -pa -m ../data/periodic-cube.mesh -d cuda
//    ex9 -ea -m ../data/periodic-cube.mesh -d cuda
//    ex9 -fa -m ../data/periodic-cube.mesh -d cuda
//    ex9 -pa -m ../data/amr-quad.mesh -p 1 -r 2 -dt 0.002 -tf 9 -d cuda
//
// Description:  This example code solves the time-dependent advection equation
//               du/dt + v.grad(u) = 0, where v is a given fluid velocity, and
//               u0(x)=u(0,x) is a given initial condition.
//
//               The example demonstrates the use of Discontinuous Galerkin (DG)
//               bilinear forms in MFEM (face integrators), the use of implicit
//               and explicit ODE time integrators, the definition of periodic
//               boundary conditions through periodic meshes, as well as the use
//               of GLVis for persistent visualization of a time-evolving
```

```
# MFEM_INSTALL_DIR = ../mfem
# CONFIG_MK = $(MFEM_INSTALL_DIR)/share/mfem/config.mk

MFEM_LIB_FILE = mfem_is_not_built
-include $(CONFIG_MK)

SEQ_EXAMPLES = ex0 ex1 ex2 ex3 ex4 ex5 ex6 ex7 ex8 ex9 ex10 ex14 ex15 ex16 \
 ex17 ex18 ex19 ex20 ex21 ex22 ex23 ex24 ex25 ex26 ex27 ex28 ex29 ex30 \
 ex31 ex33
PAR_EXAMPLES = ex0p ex1p ex2p ex3p ex4p ex5p ex6p ex7p ex8p ex9p ex10p ex11p \
 ex12p ex13p ex14p ex15p ex16p ex17p ex18p ex19p ex20p ex21p ex22p ex24p \
 ex25p ex26p ex27p ex28p ex29p ex30p ex31p ex32p ex33p
SEQ_DEVICE_EXAMPLES = ex1 ex3 ex4 ex5 ex6 ex9 ex22 ex24 ex25 ex26
PAR_DEVICE_EXAMPLES = ex1p ex2p ex3p ex4p ex5p ex6p ex7p ex9p ex13p ex22p \
 ex24p ex25p ex26p

ifeq ($(MFEM_USE_MPI),NO)
   EXAMPLES = $(SEQ_EXAMPLES)
else
   EXAMPLES = $(PAR_EXAMPLES) $(SEQ_EXAMPLES)
endif
SUBDIRS =
ifeq ($(MFEM_USE_AMGX),YES)
   SUBDIRS += amgx
endif
ifeq ($(MFEM_USE_GINKGO),YES)
   SUBDIRS += ginkgo
endif
ifeq ($(MFEM_USE_HIOP),YES)
   SUBDIRS += hiop
endif
ifeq ($(MFEM_USE_PETSC),YES)
   SUBDIRS += petsc
endif
ifeq ($(MFEM_USE_PUMI),YES)
   SUBDIRS += pumi
endif
ifeq ($(MFEM_USE_SUNDIALS),YES)
   SUBDIRS += sundials
endif
```

```
time step: 1040, time: 2.6
time step: 1060, time: 2.65
time step: 1080, time: 2.7
time step: 1100, time: 2.75
time step: 1120, time: 2.8
time step: 1140, time: 2.85
time step: 1160, time: 2.9
time step: 1180, time: 2.95
time step: 1200, time: 3
time step: 1220, time: 3.05
time step: 1240, time: 3.1
time step: 1260, time: 3.15
time step: 1280, time: 3.2
time step: 1300, time: 3.25
time step: 1320, time: 3.3
time step: 1340, time: 3.35
time step: 1360, time: 3.4
time step: 1380, time: 3.45
time step: 1400, time: 3.5
time step: 1420, time: 3.55
time step: 1440, time: 3.6
```

```
top - 23:03:14 up 19 min,  0 users,  load average: 1.11, 0.89, 0.61
Tasks:  17 total,   2 running,  15 sleeping,   0 stopped,   0 zombie
%Cpu(s): 99.7 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
MiB Mem :    965.5 total,    111.9 free,    486.3 used,    367.3 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.    339.1 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 2494 euler     20   0  140080 130204  15036 R  95.3  13.2   3:23.00 ex9
 2342 euler     20   0   33496  25996   9100 S   2.7   2.6   0:07.01 python3
   24 euler     20   0  878916  57476  32860 S   1.0   5.8   0:03.97 node
 1823 euler     20   0  884700  82908  36572 S   0.3   8.4   0:02.60 node
    1 euler     20   0   33308  24820   7672 S   0.0   2.5   0:00.55 supervisord
    8 euler     20   0  186624  17092   7596 S   0.0   1.7   0:00.20 python3
    9 euler     20   0    2888   1000    904 S   0.0   0.1   0:00.00 sh
   13 euler     20   0  866652  79328  34336 S   0.0   8.0   0:09.49 node
  125 euler     20   0    5000   3948   3344 S   0.0   0.4   0:00.01 bash
  727 euler     20   0    5000   3976   3344 S   0.0   0.4   0:00.00 bash
 1834 euler     20   0  857128  41476  32488 S   0.0   4.2   0:00.17 node
 2075 euler     20   0  599144  40264  29096 S   0.0   4.1   0:00.15 node
 2976 euler     20   0    5000   3996   3396 S   0.0   0.4   0:00.00 bash
 3205 euler     20   0    7364   3120   2776 R   0.0   0.3   0:00.00 top
 3315 euler     20   0    2888    996    904 S   0.0   0.1   0:00.00 sh
```

https://hpcic.llnl.gov/

https://mfem.org/workshop

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

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

Disclaimer
This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.