Unstructured Meshing Technologies

Presented to
ATPESC 2020 Participants

Aaron Fisher (LLNL) & Mark Shephard (RPI)

Date 08/10/2021
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).

![Diagram of high-order mesh refinement](image)
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 $H_1$, $H(\text{curl})$, $H(\text{div})$- and $L_2$ 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

mfem.org
(v3.4, May/2018)
Example 1 – Laplace equation

- **Mesh**
  ```cpp
  // 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;
  // if (mesh) {
  //   cerr << "Can not open mesh file " << mesh_file << ":f" << endl;
  //   return 2;
  // }
  // mesh = new Mesh(mesh_file, 1, 1);
  // mesh->close();
  // int dim = mesh->Dimension();
  // if (dim != 2) {
  //   cerr << "Length number that gives a final mesh with no more than 50,000 
  // elements.
  // }
  // return 0;
  
  // 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
  // 
  // FiniteElementSpace *fespace = new FiniteElementSpace(mesh, 1);
  // fespace->refine(ref_levels);
  // fespace->createGridFunction(x);
  // x = 0.9;
  // 
  // 4. Define the basis functions for each element on the mesh. Here we use continuous
  // Lagrange elements of the specified order. If order < 1, we 
  // instead use an isoparametric/isogeometric space.
  
  
  // 5. Set up the linear form b(...) which corresponds to the right-hand side of 
  // the FEM linear system, which in this case is \( l(\phi_i) \) where \( \phi_i \) are 
  // the basis functions in the finite element space.
  
  // LinearForm *b = new LinearForm(fespace);
  // ConstantCoefficient one(1.0);
  // b->AddDomainIntegrator(new DomainIntegrator(one));
  // b->Assemble();
  
  // 6. Define the solution vector x as a finite element grid function 
  // corresponding to the fespace. Initialize x with initial guess of zero, 
  // which satisfies the boundary conditions.
  
  // GridFunction x(fespace);
  // x = 0.9;
  
  // 7. Set up the bilinear form a(\(\phi_i, \phi_j\)) 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();
  
  // 8. Solve the system \( Ax = b \) with PCG.
  // GCRelaxer M(A);
  // M.Set(A, X, b, x, 1, 1e-12, 0.0);
  #ifndef FEM
  #endif
  // 9. If FEM was compiled with SuiteSparse, use UMFPACK to solve the system.
  #ifdef USE_SUITESPARSE
  UMFPACKSolver umf_solver;
  #endif
  umf_solver.Control(UMFPACK_ORDERING = UMFPACK_ORDERING_METIS);
  umf_solver.SetOperator(A);
  umf_solver.Mult(x, b);
  #endif
  
  
  // 10. Send the solution by socket to a CLVis server.
  
  if (visualization) {
    char wishport[] = "localhost";
    int visitport = 19916;
    socketstream sol_sock(wishport, visitport);
    sol_sock.precision(8);
    sol_sock << "solution\n" << *mesh << x << flush;
  }
  ```

- **Linear solve**

- **Visualization**

- **Finite element space**

- **Initial guess, linear/bilinear forms**

  - works for any mesh & any H1 order
  - builds without external dependencies
Example 1 – Laplace equation

- Mesh

```cpp
// 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 << "Can 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

```cpp
88 // 4. Define a finite element space on the mesh. Here we use continuous
89 // Lagrange finite elements of the specified order. If order < 1, we
90 // instead use an isoparametric/isogeometric space.
91 FiniteElementCollection *fec;
92 if (order > 0)
93     fec = new H1_FECollection(order, dim);
94 else if (mesh->GetNodes())
95     fec = mesh->GetNodes()->OwnFEC();
96 else
97     fec = new H1_FECollection(order = 1, dim);
98 FiniteElementSpace *fespace = new FiniteElementSpace(mesh, fec);
99 cout << "Number of unknowns: " << fespace->GetVSize() << endl;
```
Example 1 – Laplace equation

- Initial guess, linear/bilinear forms

```cpp
// 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 DomainLPIIntegrator(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

```cpp
130  #ifdef MFEM_USE_SUITESPARSE
131  // 8. Define a simple symmetric Gauss-Seidel preconditioner and use it to
132  // solve the system Ax=b with PCG.
133  GSmoother M(A);
134  PCG(A, M, *b, x, 1, 200, 1e-12, 0.0);
135  #else
136  // 8. If MFEM was compiled with SuiteSparse, use UMFPACK to solve the system.
137  UMFPackSolver umf_solver;
138  umf_solver-Control[UMFPACK_ORDERING] = UMFPACK_ORDERING_METIS;
139  umf_solver.SetOperator(A);
140  umf_solver.Mult(*b, x);
141  #endif
```

- Visualization

```cpp
152  // 10. Send the solution by socket to a GLVis server.
153  if (visualization)
154  {
155      char vishost[] = "localhost";
156      int visport = 19916;
157      socketstream sol_sock(vishost, visport);
158      sol_sock.precision(8);
159      sol_sock << "solution\n" << *mesh << x << flush;
160  }
```
Example 1 – parallel Laplace equation

- Parallel mesh

```
101 // 5. Define a parallel mesh by a partitioning of the serial mesh. Define
102 // this mesh further in parallel to increase the resolution. Once the
103 // parallel mesh is defined, the serial mesh can be deleted.
104 ParMesh *pmesh = new ParMesh(MPI_COMM_WORLD, *mesh);
105 delete mesh;
106 {
107   int par_ref_levels = 2;
108   for (int l = 0; l < par_ref_levels; l++)
109     pmesh->UniformRefinement();
110 }
```

- Parallel finite element space

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

- Parallel initial guess, linear/bilinear forms

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

- Parallel assembly

```
155 // 10. Define the parallel (hypre) matrix and vectors representing x, b, and the finite element approximation.
156 HypreParMatrix *A = a->ParallelAssembly();
157 HypreParVector *b = b->ParallelAssembly();
158 HypreParVector *x = x->ParallelAssembly();
```

```
A = P^T a P  
B = P^T b  
x = P X
```

- Parallel linear solve with AMG

```
164 // 11. Define and apply a parallel PCG solver for Ax=b with the BoomerAMG
165 // preconditioner from hypre.
166 HypreSolver *amp = new HypreBoomerAMG(*a);
167 hypreCG *pog = new HyprePCG(*amp);
168 pog->SetTol(1e-12);
169 pog->SetMaxIter(200);
170 pog->SetPrintLevel(2);
171 pog->SetPreconditioner(*amp);
172 pog->Multi(*b, *x);
```

- Visualization

```
184 // 14. Send the solution by socket to a GLVis server.
185 // (visualization)
186 {
187   char vishost[] = "localhost";
188   int visport = 19916;
189   socketstream sol_sock(vishost, visport);
190   sol_sock << "parallel"
191       << num_procs << " "
192       << myid << ";n"
193       << sol_tol
194       << errornorm(x);
195       << x
196       << flush;
197 }
```

- highly scalable with minimal changes
- build depends on hypre and METIS
MFEM example codes – mfem.org/examples

Example 1: Laplace Problem

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

\[ -\Delta u = f \]

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

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

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

Example 2: Linear Elasticity

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

\[ -\text{div}(\sigma(u)) = 0 \]

where

\[ \sigma(u) = \lambda \text{div}(u) I + \mu (\nabla u + \nabla u^T) \]

is the stress tensor corresponding to displacement field \( u \) and \( \lambda \) and \( \mu \) are the material Lame constants. The boundary conditions are \( u = 0 \) on the fixed part of the boundary with attribute 1, and \( \sigma(u) \cdot n = f \) 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:
Discretization Demo & Lesson
https://xsdk-project.github.io/MathPackagesTraining2021/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

- **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

www.llnl.gov/casc/hypre

mfem.org

www.llnl.gov/casc/blast
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

**Physical time evolution**
- $t = 0$
- $t = 1.5$
- $t = 3.0$

**Pseudo-time evolution**
- $\tau = 0$
- $\tau = 0.5$
- $\tau = 1$

**Galerkin FEM**
- Gauss-Lobatto basis

**Discont. Galerkin**
- Bernstein basis

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

- **Momentum Conservation:**
  \[ \frac{d\vec{v}}{dt} = \nabla \cdot \sigma \]

- **Mass Conservation:**
  \[ \frac{d\rho}{dt} = -\rho \nabla \cdot \vec{v} \]

- **Energy Conservation:**
  \[ \frac{de}{dt} = \sigma : \nabla \vec{v} \]

- **Equation of Motion:**
  \[ \frac{d\vec{x}}{dt} = \vec{v} \]

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

- Momentum Conservation:
  \[ \frac{d(\rho \vec{v})}{d\tau} = \vec{v}_m \cdot \nabla (\rho \vec{v}) \]

- Mass Conservation:
  \[ \frac{d\rho}{d\tau} = \vec{v}_m \cdot \nabla \rho \]

- Energy Conservation:
  \[ \frac{d(\rho e)}{d\tau} = \vec{v}_m \cdot \nabla (\rho e) \]

- Mesh velocity:
  \[ \vec{v}_m = \frac{d\vec{x}}{d\tau} \]
High-order finite elements lead to more accurate, robust and reliable hydrodynamic simulations
High-order finite elements have excellent strong scalability

**Strong scaling, p-refinement**

- ~600 dofs/zone
- 1 zone/core

**Strong scaling, fixed #dofs**

- 256K DOFs
- 256 cores
- more FLOPs, same runtime

**Finite element partial assembly**

**FLOPs increase faster than runtime**
Unstructured Mesh R&D: Mesh optimization and high-quality interpolation between meshes

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

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

DG advection-based interpolation (ALE remap, Example 9, radiation transport)
Unstructured Mesh R&D: Accurate and flexible finite element visualization

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

**GLVis:** native MFEM lightweight OpenGL visualization tool

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

[GLVis](http://glvis.org)  
[VisIt](http://visit.llnl.gov)

BLAST computation on 2\textsuperscript{nd} order tet mesh
MFEM’s unstructured AMR infrastructure

Adaptive mesh refinement on library level:
- Conforming local refinement on simplex meshes
- \textit{Non-conforming refinement for quad/hex meshes}
- \textit{h-refinement with fixed p}

General approach:
- any high-order finite element space, \(H1, H(\text{curl}), H(\text{div}), \ldots\), 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)
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

2D Shock-like Problem AMR Benchmark (Quad Mesh, Anisotropic Refinements)

- Uniform refinement
- 1st, 2nd, 4th, 8th order AMR

Approximation error (H1 seminorm)

Square root of the number of unknowns

Anisotropic adaptation to shock-like fields in 2D & 3D
Parallel dynamic AMR, Lagrangian Sedov problem

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

Parallel load balancing based on space-filling curve partitioning, 16 cores
Parallel AMR scaling to ~400K MPI tasks

- 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”)

![Parallel decomposition (2048 domains shown)](image1)

![Parallel partitioning via Hilbert curve](image2)
ceed.exascaleproject.org

2 Labs, 5 Universities, 30+ researchers
MFEM Scaling on Multiple GPUs

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

MFEM-BP3, 3D, Lassen 4 x V100 GPUS / node
Resources

- **More information and publications**
  - MFEM – [mfem.org](https://mfem.org)
  - MFEM Project - [github.com/mfem](https://github.com/mfem)
  - MFEM Repo - [github.com/mfem/mfem](https://github.com/mfem/mfem)
  - MFEM Issues - [github.com/mfem/mfem/issues](https://github.com/mfem/mfem/issues)
  - BLAST – [computation.llnl.gov/projects/blast](https://computation.llnl.gov/projects/blast)
  - CEED – [ceed.exascaleproject.org](https://ceed.exascaleproject.org)

- **Virtual Community Workshop (Oct 20, 2021)**
  - Website – [mfem.org/workshop](https://mfem.org/workshop)
  - Technical presentations from MFEM team
  - Contributed talks from application devs
  - Demos of MFEM through python, and Jupyter notebooks
  - Discussions of development roadmap, and application areas

Q4 Rayleigh-Taylor single-material ALE on 256 processors
Fundamental finite element operator decomposition

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

\[
A = P^T G^T B^T D B G P
\]

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

[GitHub link](github.com/ceed/benchmarks)
Tensorized partial assembly

$$B_{ki} = \varphi_i(q_k) = \varphi_{i_1}^{1d}(q_{k_1}) \varphi_{i_2}^{1d}(q_{k_2}) = B_{k_{1i_1}}^{1d} B_{k_{2i_2}}^{1d}$$

$$U_{k_1k_2} = B_{k_{1i_1}}^{1d} B_{k_{2i_2}}^{1d} V_{i_1i_2} \rightarrow U = B^{1d} V (B^{1d})^T$$

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Memory</th>
<th>Assembly</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Matrix Assembly</td>
<td>(O(p^{2d}))</td>
<td>(O(p^{3d}))</td>
<td>(O(p^{2d}))</td>
</tr>
<tr>
<td>Partial Assembly</td>
<td>(O(p^d))</td>
<td>(O(p^d))</td>
<td>(O(p^{d+1}))</td>
</tr>
</tbody>
</table>

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!
FASTMath Unstructured Mesh Technologies

K.D. Devine¹, V. Dobrev², D.A. Ibanez¹, T. Kolev², K.E. Jansen³, O. Sahni⁴, M.S. Shephard⁴, G.M. Slota⁴, C.W. Smith⁴

¹Sandia National Laboratories
²Lawrence Livermore National Laboratory
³University of Colorado
⁴Rensselaer Polytechnic Institute
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
▪ 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
FASTMath Unstructured Mesh Developments

Technology 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 PIC – Tools to support PIC on unstructured meshes
- Unstructured Mesh ML and UQ – ML for data reduction, adaptive mesh UQ
- In Situ Vis and Data Analytics – Tools to gain insight as simulations execute
FASTMath Unstructured Mesh Tools and Components

• FE Analysis codes
  – MFEM (https://mfem.org/)
  – LGR (https://github.com/SNLComputation/lgrtk)
  – PHASTA (https://github.com/phasta/phasta)

• Unstructured Mesh Infrastructure
  – Omega_h (https://github.com/SNLComputation/omega_h)
  – PUMI/MeshAdapt (https://github.com/SCOREC/core)
  – PUMIpic (https://github.com/SCOREC/pumi-pic)

• Load balancing, task placement
  – Zoltan (https://github.com/sandialabs/Zoltan)
  – Zoltan2 (https://github.com/trilinos/Trilinos/tree/master/packages/zoltan2)
  – Xtra-PULP (https://github.com/HPCGraphAnalysis/PuLP)
  – EnGPar (http://scorec.github.io/EnGPar/)

• Unstructured Mesh PIC applications
  – XGCm (https://github.com/SCOREC/xgcm) – private repo
  – GITRm (https://github.com/SCOREC/gitrm) – private repo
Parallel Unstructured Mesh Infrastructure

Support unstructured mesh interactions on exascale systems
– Mesh hierarchy to support interrogation and modification
– Maintains linkage to original geometry
– Conforming mesh adaptation
– Inter-process communication
– Supports field operations

Tools
– Omega_h – full CPU/GPU support
– PUMI – CPU based curved mesh adapt.
– PUMIPic – Unstructured mesh with particles for GPU implementations
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 \textit{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 “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
Mesh Adaptation Status

• Applied to very large scale models – 92B elements on 3.1M processes on ¾ million cores

• Local solution transfer supported through callback

• Effective storage of solution fields on meshes

• Supports adaptation with boundary layer meshes
Mesh Adaptation 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.

Images showing mesh adaptation process.
Mesh Adaptation

• Supports adaptation of curved elements

• Adaptation based on multiple criteria, examples
  – Level sets at interfaces
  – Tracking particles
  – Discretization errors
  – Controlling element shape in evolving geometry

(a) [Image]
(b) [Image]
Load Balancing, Dynamic Load balancing

• Purpose: Balance, rebalance computational load while controlling communications
  – 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 multi-constraint multi-objective label propagation-based graph partitioner
Reduce application communication time at extreme scale

- **Partitioning and load balancing**: assign work to processes in ways that avoid process idle time and minimize communication
- **Task mapping**: assign processes to cores in ways that reduce messages distances and network congestion
- **Important in extreme-scale systems**:
  - Small load imbalances can waste many resources
  - Large-scale networks can cause messages to travel long routes and induce congestion
- **Challenge** to develop algorithms that…
  - account for underlying architectures & hierarchies
  - run effectively side-by-side with application across many platforms (multicore, GPU)
Zoltan/Zoltan2 Toolkits: Partitioners

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

**Geometric**

- Recursive Coordinate Bisection
- Recursive Inertial Bisection
- Multi-Jagged Multi-section
- Space Filling Curves

**Topology-based**

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

- PHG Hypergraph Partitioning
- Interface to PaToH (Ohio St.)
EnGPar Dynamic Load Balancing

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

- Multi-(hyper)graph supports multiple dependencies (edges) between application work/data items (vertices)
- Application defined vertex and edges
- Diffusion sends from heavily loaded parts to lighter parts
- On a 1.3B element mesh, in 8 seconds EnGPar reduces a 53% vtx imbalance to 6%, elm imbalance of 5%, edge cut increase by 1%
- Applied to PIC calculations to support particle balance – flexibility of vertex and edge definition critical to attaining 20% reduction in total run time
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 of active flow control to aircraft tails
Blood flow on the arterial system
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
- Omega\_h parallel mesh adaptation is integrated with Albany to do:
  - Estimate error
  - Adapt the mesh
- Ice sheet mesh is modified to minimize degrees of freedom
- Field of interest is the ice sheet velocity
Application interactions – RF Fusion

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

CAD model of antenna array
Supporting Unstructured Mesh for Particle-in-Cell Calculations

PUMIPic data structures are mesh centric
- Mesh is distributed as needed by the application in terms of PICparts
- Mesh can be graded and anisotropic
- Particle data associated with elements
- Operations take advantage of distributed mesh topology
- Mesh 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
  - GPU version defines buffer as set of neighboring parts – dramatic reduction in memory and communication costs
Mesh Data Structure for Heterogeneous Systems

• Mesh topology/adaptation tool - Omega
  – Conforming mesh adaptation (coarsening past initial mesh, refinement, swap)
  – Manycore and GPU parallelism using Kokkos, CUDA, or HIP
  – Distributed mesh via mesh partitions with MPI communications
  – Support for mesh-based fields

• Recent RPI developments:
  – Mixed mesh adjacency storage and query
  – Two-way mesh matching for periodic BC
  – Ported and tested on AMD GPUs using HIP

Adaptation following rotating flow field.

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

<table>
<thead>
<tr>
<th>triangle</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>adj vertex</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

| adj triangle | 0 | 0 | 1 | 1 | 0 | 1 |
| offset vertex | 0 | 1 | 3 | 4 | 6 |

Mesh entity adjacency arrays.
PUMIPic Particle Data Structures

- Layout of particles in memory is critical to performance. Requirements:
  - Optimizes push (sort/rebuild), scatter, and gather operations
  - Associates particles with mesh elements
  - Changes in the number of particles per element
  - Evenly distributes work with a range of particle distributions (e.g. uniform, Gaussian, exponential, etc.)
  - Stores a lot of particles per GPU – low overhead

- Particle data structure interface and implementation
  - API abstracts implementation for PIC code developers
  - CSR, Sell-C-σ, CabanaM
  - Performance is a function of particle distribution
  - Cabana AoSoA w/a CSR index of elements-to-particles are promising

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 PUMIPic

- Particle push
- Adjacency based search
  - Faster than grid based search
- Element-to-particle association update
- Particle Migration
- Particle path visualization
- Mesh partitioning w/buffer regions
- Mesh field association
- Poisson field solve using PETSc DMPlex on GPUs
- Checkpoint/restart of particle and mesh data – supports customization for each application

PUMIPic based XGCm Edge Plasma Code

XGCm is a version of XGC being built on PUMIPic
- targeting execution of all operations on GPUs
Testing of PUMIPic for use in XGC like push
- 2M elements, 1M vertices, 2 to 128 poloidal planes
- Pseudo push and particle-to-mesh gyro scatter
- Tested on up to 24,576 GPUs of Summit with 1.1 trillion particles, for 100 iterations: push, adjacency
- Weak scaling up to 24576 GPUs (4096 nodes) with 48 million particles per GPU
XGCm status: All operations on GPU
- Ion and electron charge scatter and push
- Electrostatic potential calculation
- Gyro-kinetic electric field calculation and gather
- Poisson solve
PUMIPic based GITRm Impurity Transport Code

• Incorporates impurity transport capabilities of GITR
• 3D mesh for cases such including divertor cassettes, tiles, limiters, specific diagnostics/probes etc.

• Status
  – Physics equivalent to GITR
  – Particle initialization directly on 3D mesh
  – 3D mesh design/control including anisotropy to properly represent the background fields
  – Field transfer from SOLPS to 3D mesh
  – Non-uniform particle distribution
    – evolves quickly in time
  – Load balancing particles via EnGPar
  – Distance to boundary for sheath E field
  – Post-processing on 3D unstructured mesh

Statistical comparison for 1 million particles, 100k timesteps (Pitts et al. 2017)
Hands-on Exercise: Workflow Introduction

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