Coupling MFEM with Structured Mesh Libraries

September 2025

Chris Vogl[LLNL], Ann Almgren[LBNL], Dylan Copeland[LLNL], Aaron Fisher[LLNL], Gabriel Pinochet-Soto[LLNL/PSU], Tzanio Kolev[LLNL], Jean Sexton[LBNL]

Prepared by LLNL under Contract DE-AC52-07NA27344.

Interest from structured mesh applications to couple with MFEM

- Many application codes take a structured mesh approach

pisale

GeoClaw

- Such codes are supported by structured mesh libraries

SAMRAI

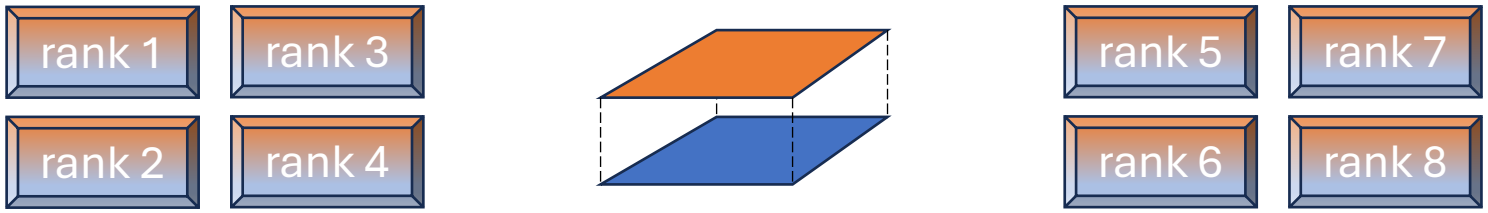
amrclaw

- Interest in coupling those libraries with MFEM for various applications
  - fluid-structure interaction (e.g., MFEM computes stresses on a solid within a fluid)
  - thermal conduction (e.g., MFEM computes thermal conduction on a solid within a fluid)

Current efforts target two paradigms for coupling

- Coupling on independent domains: “S + MFEM”
  - Each code solves a problem in different domains using different hardware

  - example: S & MFEM solve advection problem in different parts of a rectangular domain
- Coupling on one domain: “S with MFEM”
  - Each code solves a separate problem on the same domain using the same hardware

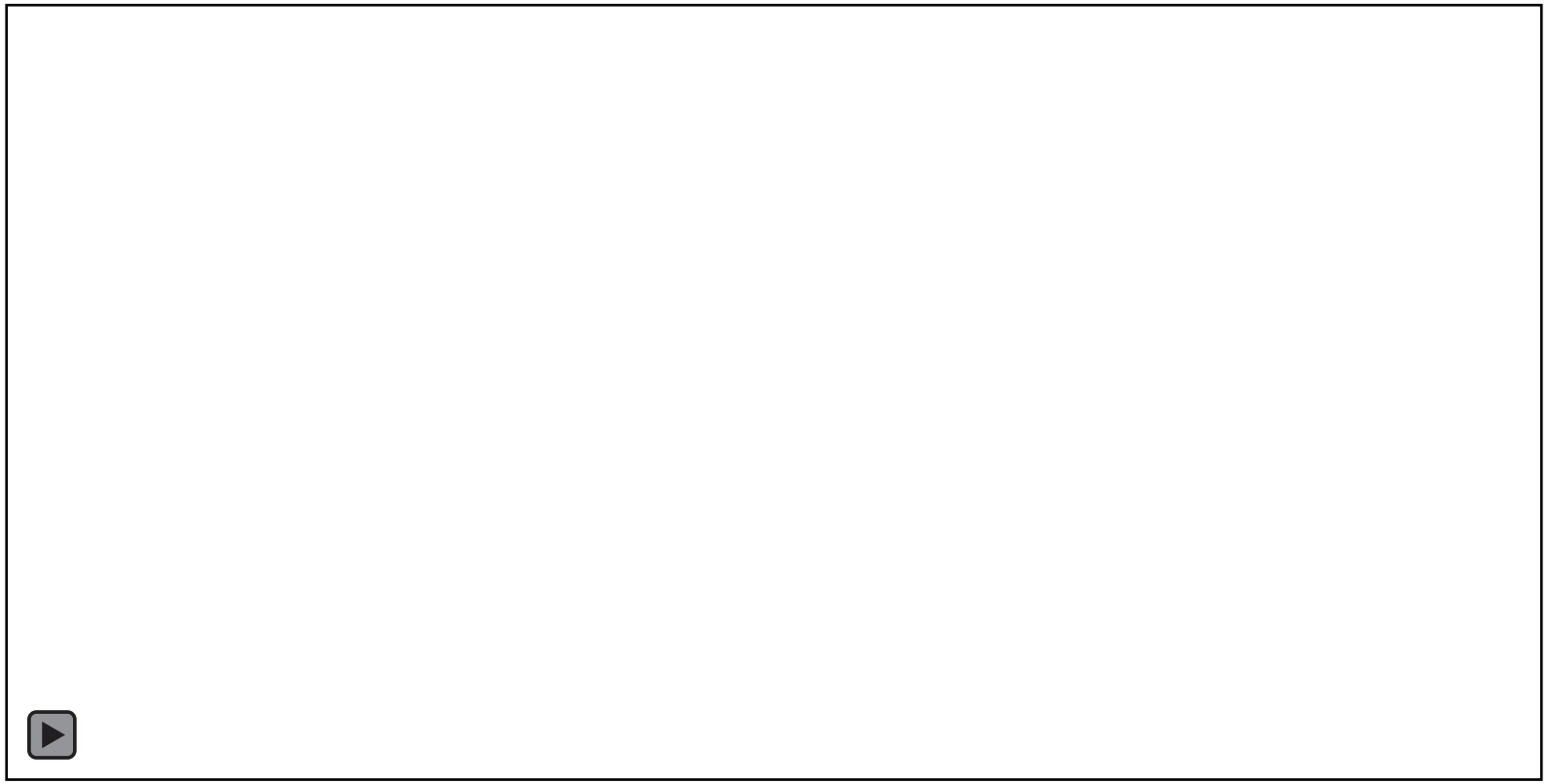


  - example: S solves Euler problem, MFEM solves thermal problem, all on the unit square

AMReX is a block-structured AMR library developed out of LBNL

Features

- C++, Fortran and Python interfaces
- Support for cell-centered, face-centered, edge-centered, and nodal data
- Support for hyperbolic, parabolic, and elliptic solves on hierarchical adaptive grid structure
- Optional subcycling in time for time-dependent PDEs
- Support for particles
- Embedded boundary description of irregular geometry
- Parallelization via flat MPI, OpenMP, CUDA, HIP, SYCL, hybrid MPI + aforementioned or MPI/MPI
- Parallel I/O
- Plotfile format supported by Amrvis, yt, VisIt, and ParaView.
- Built-in profiling tools



feature list & movie from https://amrex-codes.github.io/amrex

AMReX+MFEM: solve ODE with “parallel” operator splitting

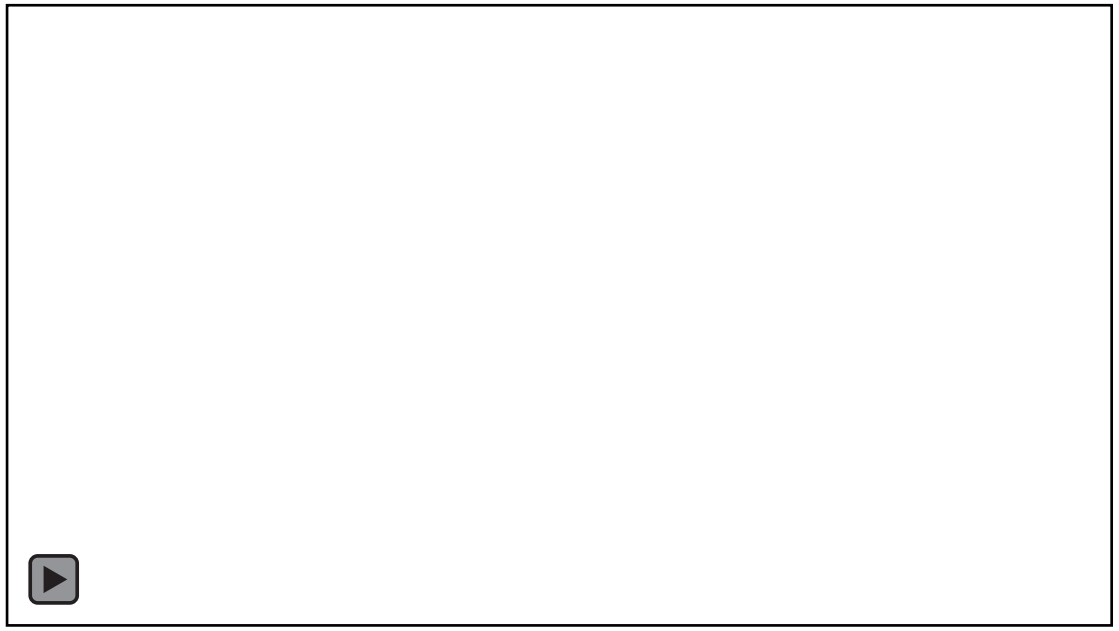
```
const bool is_amrex = ...;
MPI_Comm_split(MPI_COMM_WORLD, is_amrex, -1, &communicator);
const int exchange_world_rank = ...;

std::unique_ptr<ODE> ode;
if (is_amrex) ode = std::make_unique<amrex::AdvectionODE>(communicator, ...);
else          ode = std::make_unique<mfem::AdvectionODE>(communicator, ...);

double current_time = 0;
while (current_time < final_time)
{
  if (is_amrex) ode->ReceiveFieldValues(MPI_COMM_WORLD, exchange_world_rank);
  else          ode->SendFieldValues(MPI_COMM_WORLD, exchange_world_rank);

  if (is_amrex) ode->SendFieldValues(MPI_COMM_WORLD, exchange_world_rank);
  else          ode->ReceiveFieldValues(MPI_COMM_WORLD, exchange_world_rank);

  ode->Advance(current_time, dt);
}
```

1. Determine whether *this* rank is for the AMReX or MFEM subdomain, split global communicator, and identify global rank to exchange with
2. Initialize either an AMReX ODE object or MFEM ODE object with subdomain communicator
3. Enter time loop.
4. AMReX ranks request spatial locations from MFEM ranks that then respond with field values
5. MFEM ranks request spatial locations from AMReX ranks that then respond with field values
6. All ranks advance their subdomain solution

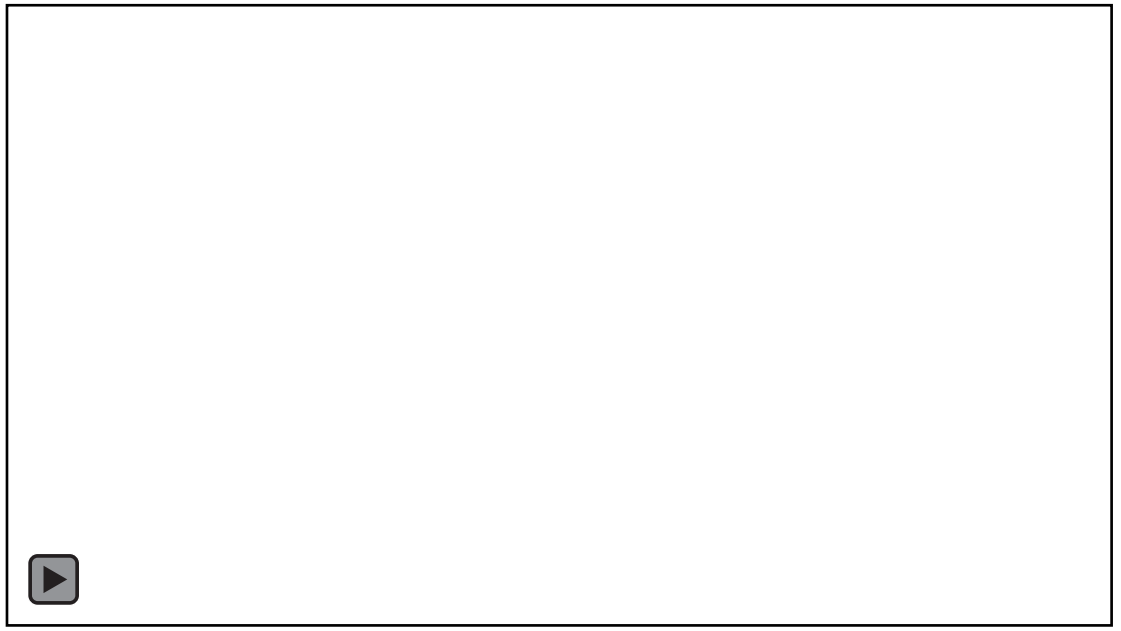
AMReX+MFEM: scalar advection solved on connected subdomains

Solve $\partial_t y + \vec{v} \cdot \nabla y = 0$ on $\Omega = \Omega_1 \bigoplus \Omega_2$ (Ω_1, Ω_2 being connected subdomains)

Ω_1



Ω_2



MFEM subdomain: Ω_1

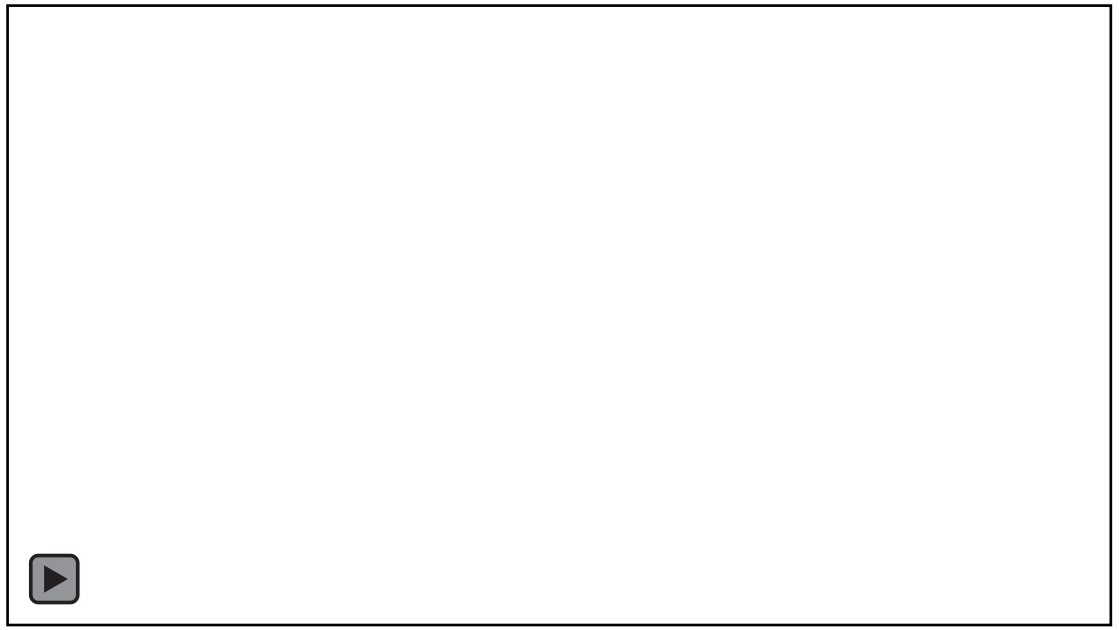
AMReX subdomain: Ω_2

Special thanks to Jean for wrangling 3D vis!

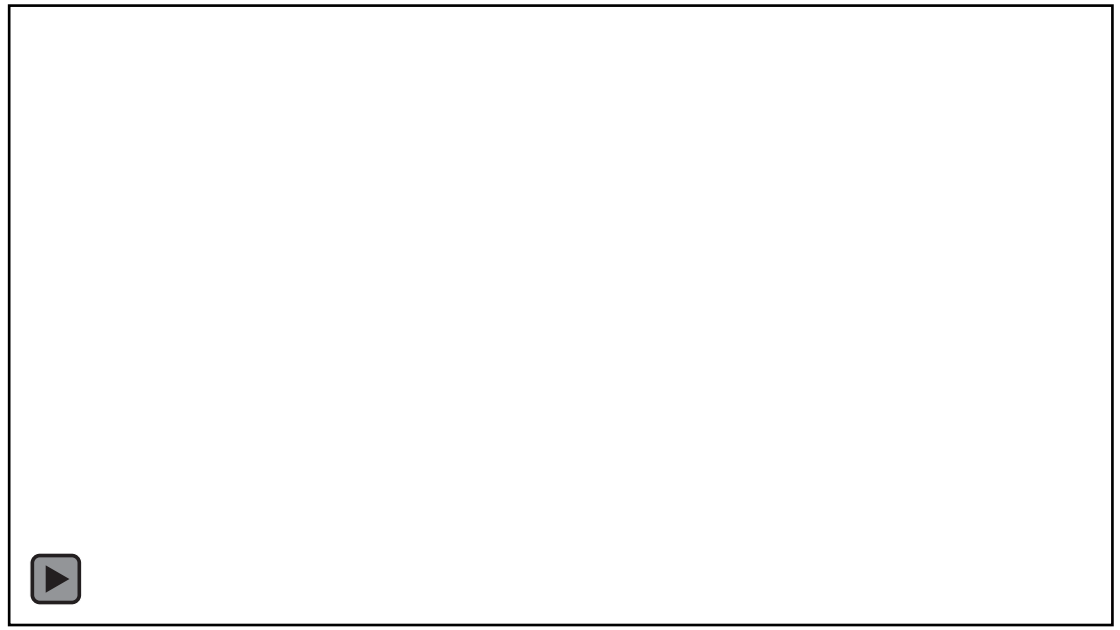
AMReX+MFEM: scalar advection solved on disconnected subdomains

Solve $\partial_t y + \vec{v} \cdot \nabla y = 0$ on $\Omega = \Omega_1 \bigoplus \Omega_2$ (Ω_1, Ω_2 **are now both disconnected**)

Ω_1



Ω_1

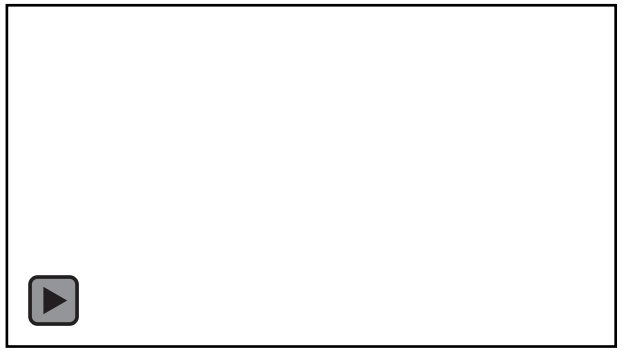


MFEM subdomain: Ω_1

AMReX subdomain: Ω_2

AMReX+MFEM: FindPoints used for MFEM values requested by AMReX

```
AdvectionODE::AdvectionODE(MPI_Comm communicator, ...)
{
  ...
  ParMesh(communicator, serialMesh);
  mesh.SetCurvature(order, false, mesh.Dimension(), ORDERING);
  auto pointsFinder = std::make_unique<FindPointsGSLIB>(communicator);
  pointsFinder->Setup(mesh);
  ...
}
```

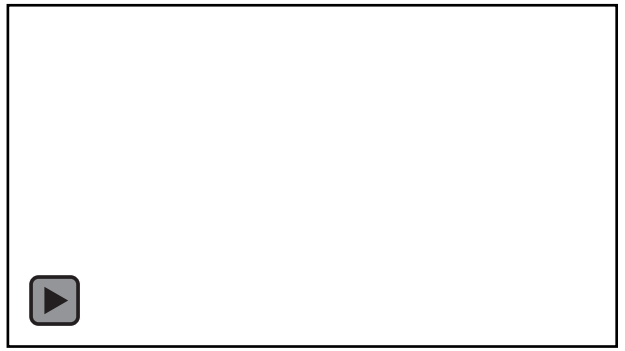


1. Initialize ParMesh and FindPoints with subdomain communicator
2. Use global communicator to receive spatial locations requested by AMReX
3. Use FindPoints (subdomain communicator) to interpolate MFEM values at those spatial points
4. Use global communicator to send values to AMReX

```
void AdvectionODE::SendFieldValues(MPI_Comm communicator, int rank)
{
    int count;
    MPI_Status status;
    MPI_Probe(rank, MPI_COORDS_TAG, communicator, &status);
    MPI_Get_count(&status, MPI_DOUBLE, &count);
    Vector points(count);
    MPI_Recv(points.GetData(), points.Size(), MPI_DOUBLE, rank,
        MPI_COORDS_TAG, communicator, MPI_STATUS_IGNORE);

    pointsFinder->FindPoints(points, ORDERING);
    Vector values;
    pointsFinder->Interpolate(*u, values);

    MPI_Send(values.GetData(), values.Size(), MPI_DOUBLE, rank,
        MPI_FIELDVALUES_TAG, communicator);
}
```

AMReX+MFEM: SubMesh used to determine MFEM spatial points

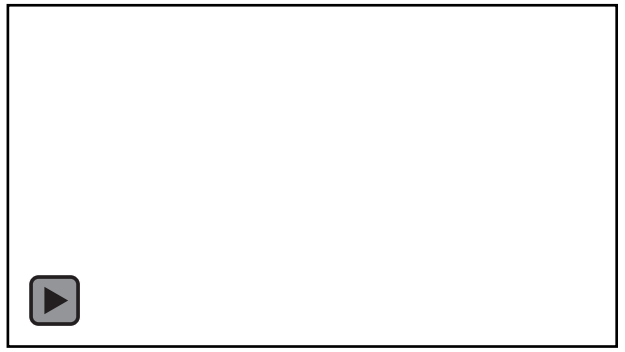
```
AdvectionODE::AdvectionODE(MPI_Comm communicator, ...)
{
  ...
  auto boundaryMesh = std::make_unique<SubMesh>(
    SubMesh::CreateFromBoundary(mesh, boundaryAttributes));
  auto boundaryFEC = std::make_unique<L2_FECollection>(order,
    boundaryMesh->Dimension(), basisType);
  auto boundaryFES = std::make_unique<FiniteElementSpace>(boundaryMesh.get(),
    boundaryFEC.get());
  auto boundaryGF = std::make_unique<GridFunction>(boundaryFES.get());

  DenseMatrix couplingPoints(mesh.SpaceDimension(), boundaryFES->GetNDofs());
  int count = 0;
  for (int el=0; el < boundaryFES->GetNE(); el++)
  {
    DenseMatrix temp;
    boundaryFES->GetElementTransformation(el)->Transform(
      boundaryFES->GetFE(el)->GetNodes(), temp);
    couplingPoints.CopyMN(temp, 0, count);
    count += temp.NumCols();
  }
  ...
}
```



1. Create a SubMesh using boundary attribute flags that indicate inflow boundaries
2. Use SubMesh elements to obtain matrix of spatial points for MFEM to request from AMReX

AMReX+MFEM: SubMesh also used set values received from AMReX

```
void AdvectionODE::ReceiveFieldValues(MPI_Comm communicator, int rank)
{
  const int size = couplingPoints.NumCols()*couplingPoints.NumRows();
  MPI_Send(couplingPoints.GetData(), size, MPI_DOUBLE, rank,
    MPI_COORDS_TAG, communicator);

  Vector values(couplingPoints.NumCols());
  MPI_Recv(values.GetData(), values.Size(), MPI_DOUBLE, rank,
    MPI_FIELDVALUES_TAG, communicator, MPI_STATUS_IGNORE);

  boundaryGF->SetFromTrueDofs(values);
  SubMesh::Transfer(*boundaryGF, inflowGF);
  auto inflowCoefficient =
      std::make_unique<GridFunctionCoefficient>(&inflowGF);
  advectionOperator->SetInflow(std::move(inflowCoefficient));
}
```



1. Use global communicator to send spatial points to AMReX
2. Use global communicator to receive AMReX values
3. Use SubMesh to set values as inflow coefficient for advection operator object

Note that mfem::AdvectionODE **implements the general "S + MFEM" paradigm**

PISALE is an ALE code leveraging the structured AMR library SAMRAI

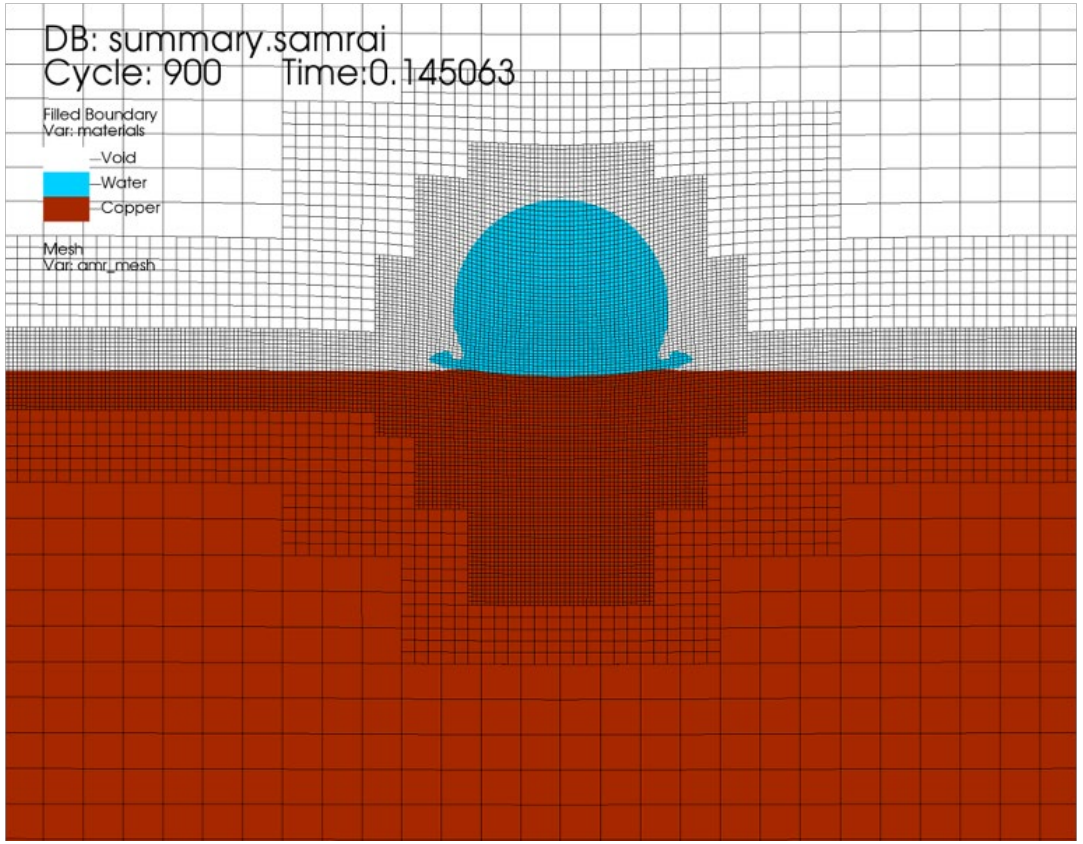
"The PISALE code uses an explicit time-marching Lagrange step to advance the flow-field through a physical time step.

The optional second phase involves a modification of the grid and a remapping of the solution to the new grid. "

"The PISALE code name comes from the acronym Pacific Island Structured-AMR with ALE."

"SAMRAI (Structured Adaptive Mesh Refinement Application Infrastructure) is an object---oriented C++ class library developed in the Center for Applied Scientific Computing (CASC) at Lawrence Livermore National Laboratory (LLNL).

It provides extensive support for parallel SAMR application development and it is designed to serve as a framework of software components that can be shared across a diverse range of SAMR applications."



PISALE blurb & image from https://pisale.bitbucket.io/

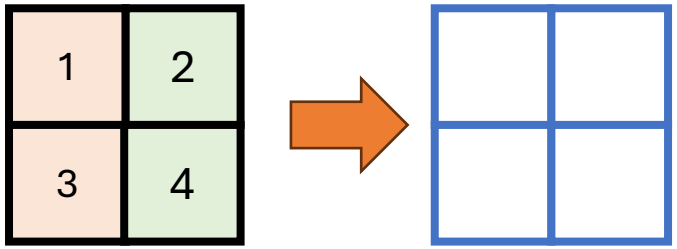
SAMRAI blurb from https://computing.llnl.gov/projects/samrai/faq

PISALE with MFEM: transfer values using custom gather / scatter.

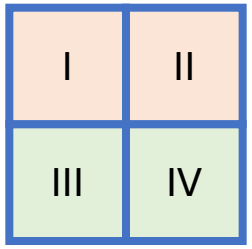
1. Create MFEM mesh that matches PISALE grid

rank 1

rank 2

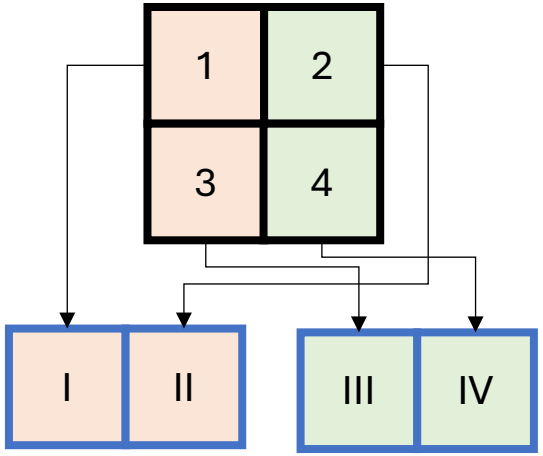


2. Distribute MFEM elements (likely different ranks than PISALE)



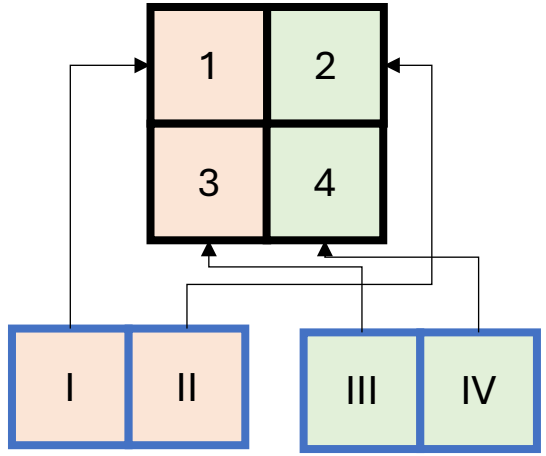

```
MeshOps(boost::shared_ptr<PatchHierarchy> hierarchy);
```

3. Use MPI gather to transfer values from PISALE to MFEM



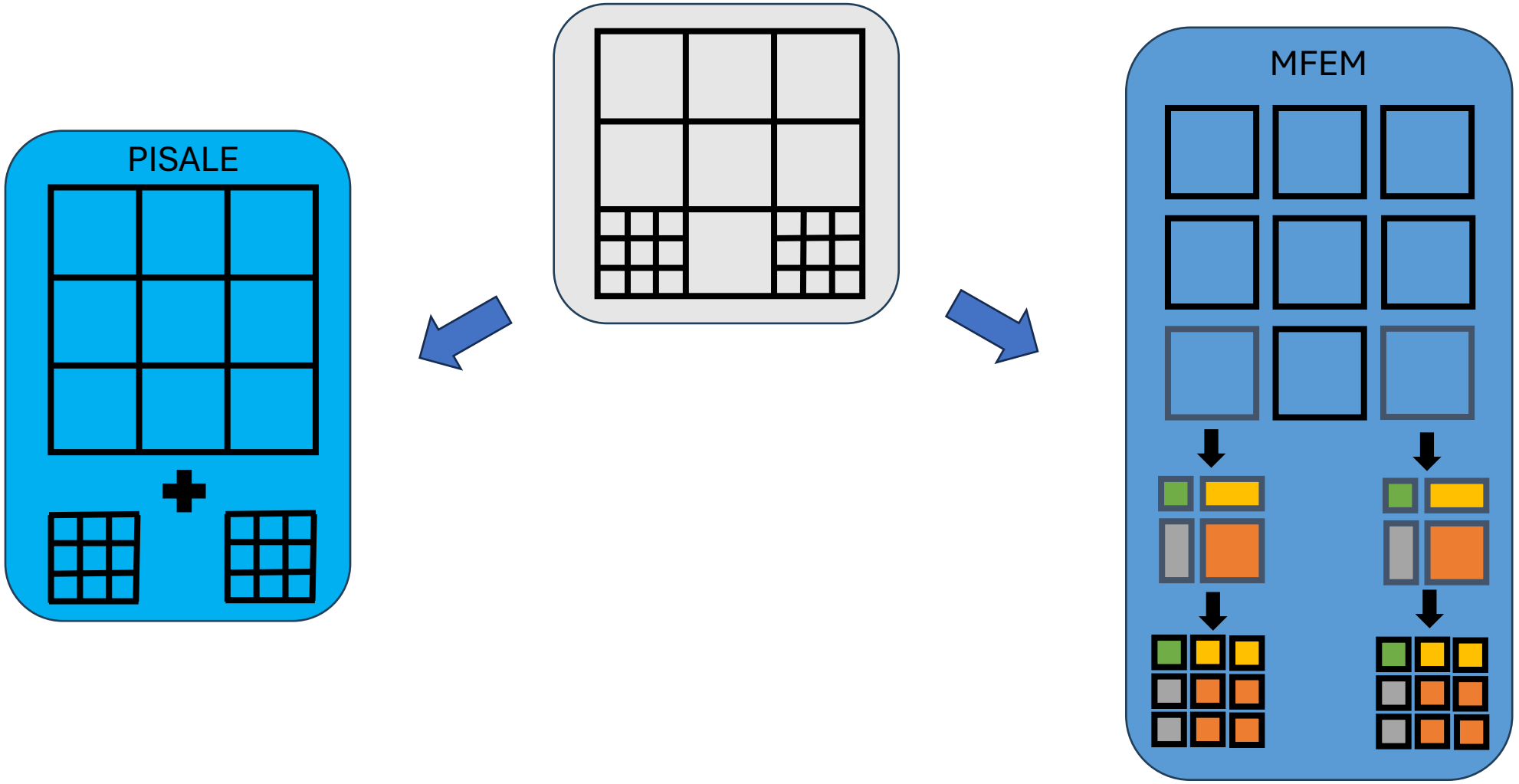

```
std::vector<std::unique_ptr<ParGridFunction>> transferToMFEM(
   const int position_id, const std::vector<int>& node_ids,
   const std::vector<int>& cell_ids);
```

4. Use MPI scatter to transfer values from MFEM to PISALE



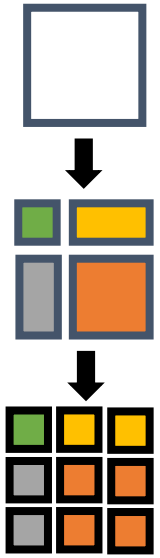

```
void transferToSAMRAI(int position_id,
   std::vector<std::pair<int, ParGridFunction&>> node_fields,
   std::vector<std::pair<int, ParGridFunction&>> cell_fields)
```

PISALE with MFEM: support 3-to-1 mesh with anisotropic refinement



PISALE with MFEM: mesh domain with new anisotropic refinement

```
const double h_target = 1.0/3.0;
Array<Refinement> refinements(refine_element_inds.Size());
for (int i=0; i < refinements.Size(); i+
   refinements[i] = Refinement(refine_el
      {{Refinement::X, h_target}, {Refin
serial_mesh.GeneralRefinement(refinement
```



1. Initial 1/3 refinement
2. Further refinement

Special thanks to Dylan for anisotropic refinement!

```
Table coarse_to_fine;
serial_mesh.ncmesh->GetRefinementTransforms().MakeCoarseToFineTable(coarse_to_fine);
refinements.DeleteAll();
for (const int& original_element_ind : refine_element_inds)
{
   Array<int> new_element_inds;
   coarse_to_fine.GetRow(original_element_ind, new_element_inds);
   for (const int& new_element_ind : new_element_inds)
   {
      Vector x0, h;
      ElementTransformation& transform = *serial_mesh.GetElementTransformation(new_element_ind);
      const IntegrationPoint ip0 = {0.0, 0.0}, ip1 = {1.0, 1.0};
      transform.Transform(ip0, x0); transform.Transform(ip1, h);
      h -= x0;
      if (std::abs(h[0] - h_target) > 1e-12 && std::abs(h[1] - h_target) > 1e-12)
         refinements.Append(Refinement(new_element_ind,
            {{Refinement::X, h_target/h[0]}, {Refinement::Y, h_target/h[1]}}));
      else if (std::abs(h[0] - h_target) > 1e-12)
         refinements.Append(Refinement(new_element_ind, Refinement::X, h_target/h[0]));
      else if (std::abs(h[1] - h_target) > 1e-12)
         refinements.Append(Refinement(new_element_ind, Refinement::Y, h_target/h[1]));
   }
}
serial_mesh.GeneralRefinement(refinements);
```

PISALE with MFEM: Approach successful on triple-point problem

pisale

PISALE & Laghos alternate advancing the s

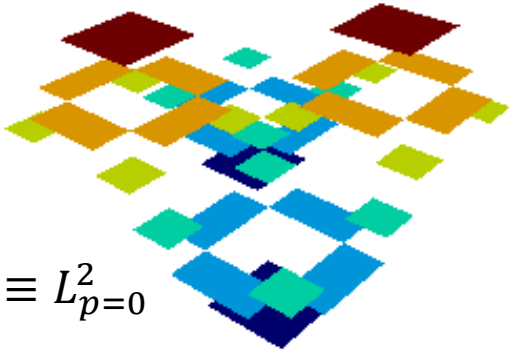
Note that MeshOps **implements th**

Need for polynomial representations constructed from cell averages

- Representing cell information as $L^2_{p=0}$ limits benefit of higher-order methods
- Developing "reconstruction" approaches as local optimization problems

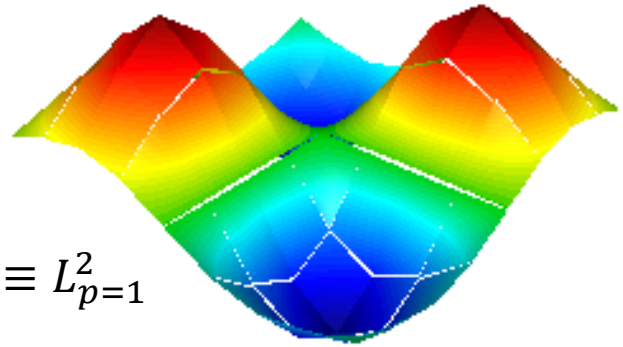
$$u_E = argmin_{u \in V_h} f_E\left(u, \{\bar{u}_{E_k}\}\right) \text{ subject to } \sigma_E(u) \equiv \frac{1}{|E|}\int_E u\, dV = \bar{u}_E$$

$$f_E = \frac{1}{2}\sum_{E_\ell \in N(E)} \left(\sigma_{E_\ell}(u) - \bar{u}_{E_\ell}\right)^2$$



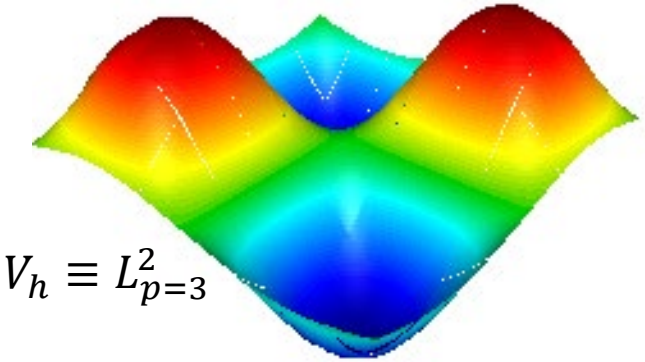
$V_h \equiv L^2_{p=0}$

$$f_E = \frac{1}{2}\sum_{E_\ell \in N(E)} \left(\int_{E \cap E_\ell} u\, dA - \frac{1}{2}(\bar{u}_E + \bar{u}_{E_\ell})\right)^2$$



$V_h \equiv L^2_{p=1}$

Special thanks to Gabriel for investigating methods



$V_h \equiv L^2_{p=3}$

“Need” for automatic fetching of third-party library dependencies

Configure MFEM with FETCH_TPLS flag

```
~/tmp/mfem$ mkdir build && cd build
~/tmp/mfem/build$ cmake -DMFEM_USE_MPI=ON -DFETCH_TPLS=ON ../
-- Found MPI_CXX: /usr/lib/x86_64-linux-gnu/libmpichcxx.so (found version "3.1")
-- Found MPI: TRUE (found version "3.1")
-- Will fetch HYPRE 2.33.0 to be built with -DCMAKE_BUILD_TYPE:STRING=Release
-- Will fetch METIS 4.0.3 to be built with default options
-- Looking for POSIXClocks ...
--    checking library: <standard c/c++>
-- Found POSIXClocks: TRUE
```

MFEM build will also fetch & build METIS

```
[  0%] Creating directories for 'metis'
[  3%] Performing download step (git clone) for 'metis'
Cloning into 'metis'...
HEAD is now at b60352f Update README.md
[  3%] No patch step for 'metis'
[  3%] Performing configure step for 'metis'
[  3%] Performing build step for 'metis'
ar: creating ../libmetis.a
a - coarsen.o
a - fm.o
```

```
[  0%] Creating directories for 'hypre'
[  0%] Performing download step (git clone) for 'hypre'
Cloning into 'hypre'...
HEAD is now at d3243638 Release 2.33.0 (#1260)
[  0%] No patch step for 'hypre'
[  0%] Performing configure step for 'hypre'
loading initial cache file /home/vogl2/tmp/mfem/build/f
-- CMake Executable: /home/vogl2/local/cmake-3.26.4_top
-- CMake Version: 3.26.4
-- Hypre Version: v2.33.0-0-gd3243638e (HEAD)
```

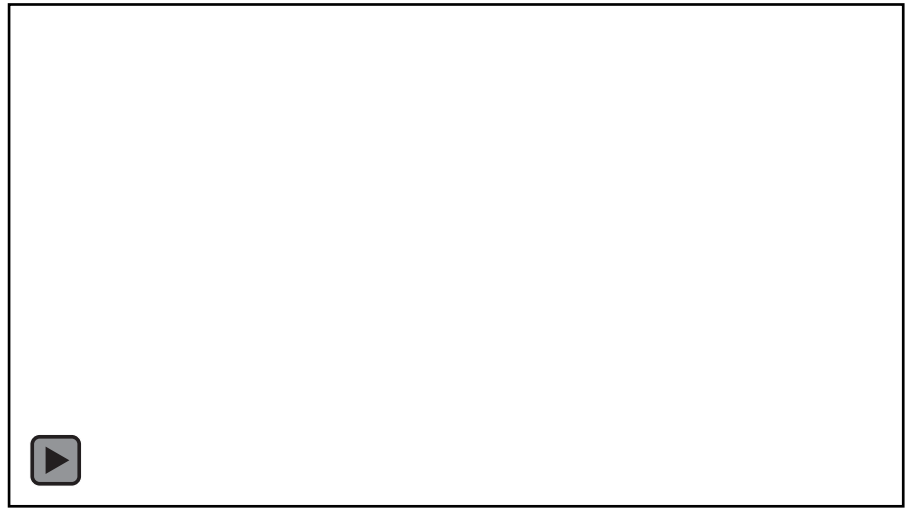
MFEM build step will fetch, configure, and build hypre

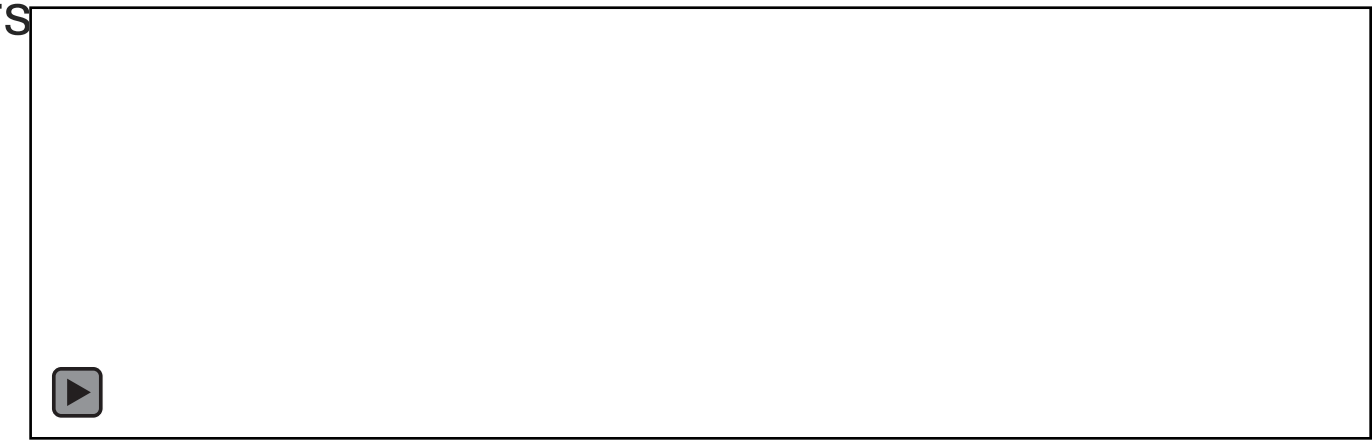
```
[  0%] Performing build step for 'hypre'
[  0%] Building C object CMakeFiles/HYPRE.dir/blas/dasum.c.o
[  0%] Building C object CMakeFiles/HYPRE.dir/blas/daxpy.c.o
[  0%] Building C object CMakeFiles/HYPRE.dir/blas/dcopy.c.o
```

- same cmake build type used for all TPLs
- GSLIB support under review
- other TPLs to be added as needed

Summary and Acknowledgements

- Code Availability
  - AMReX + MFEM: website here, contact me for demo code
  - PISALE with MFEM: planned SAMRAI transfer miniapp
  - Reconstruction: miniapp under development
  - TPL fetching: hypre and METIS in master & GSLIB in PR
- Big thank you to all the collaborators
  - AMReX + MFEM: Tzanio, Jean, and Ann
  - PISALE with MFEM: Dylan and Aaron
  - Reconstruction: Gabriel





Work funded by the US Department of Energy grant DE-SC0024728