SAND# 2012-9210 C

Finite Element Global Unknown Numbering for Fully-Coupled "Beyond-Nodal" Discretizations

Trilinos User Group: October 30, 2012

Eric C. Cyr
Sandia National Laboratories

Roger P. Pawlowski, Denis Ridzal, John N. Shadid
Sandia National Laboratories

Ben Seefeldt
St. John's University

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Motivation: This is where I started

Finite elements are fun!

- I don't want to do Poisson anymore
- Would like to do multi-domain multi-physics
- Evaluate performance of stable discretizations vs. stabilized
- Use compatible discretizations

My barrier to entry is unknown numbering!

- Want parallel unknown numbering
- Want mixed discretizations (numbering nodes and edges)
- Want high-order* and compatible
- Want multi-physics (fluids interacting with solids)
- Still want to support stabilized discretizations

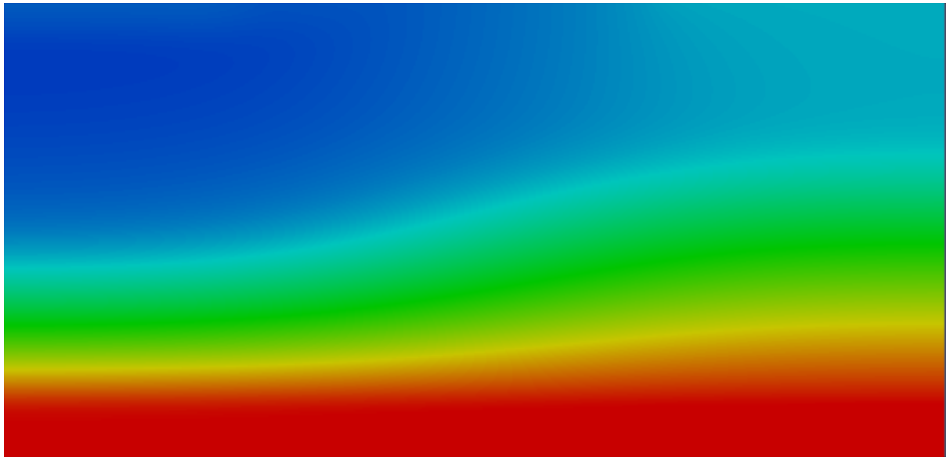
Our solution: Panzer a Trilinos package

- Technology demonstrated/matured in Drekar
- DOFManager is one component of Panzer that I will discuss today

*Don't yet have high-order (only 2nd order), will complete soon

Fully-Coupled Nodal Finite Element Simulation Technology

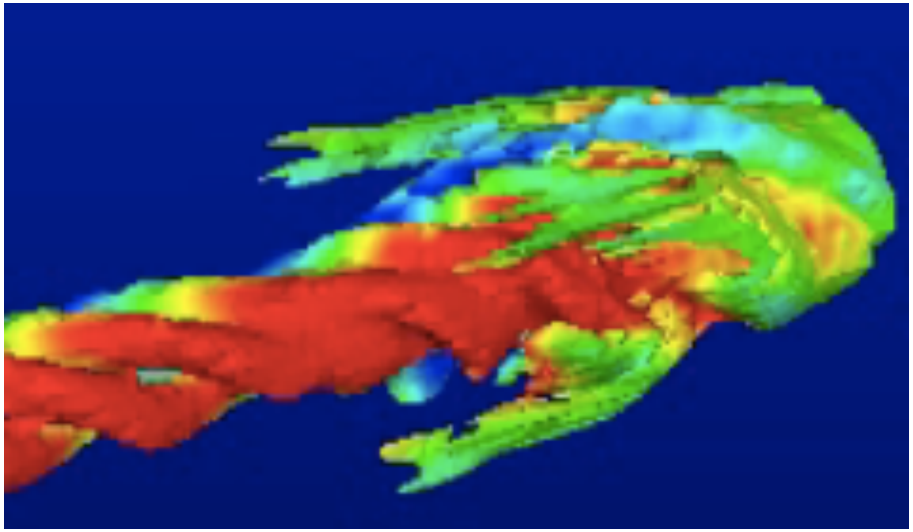
All Simulations use equal-order nodal discretizations!



BJT: Drift Diffusion (Potential)
Hydromagnetic Kelvin-Helmholtz: MHD

Fully-Coupled Nodal Finite Element Simulation Technology

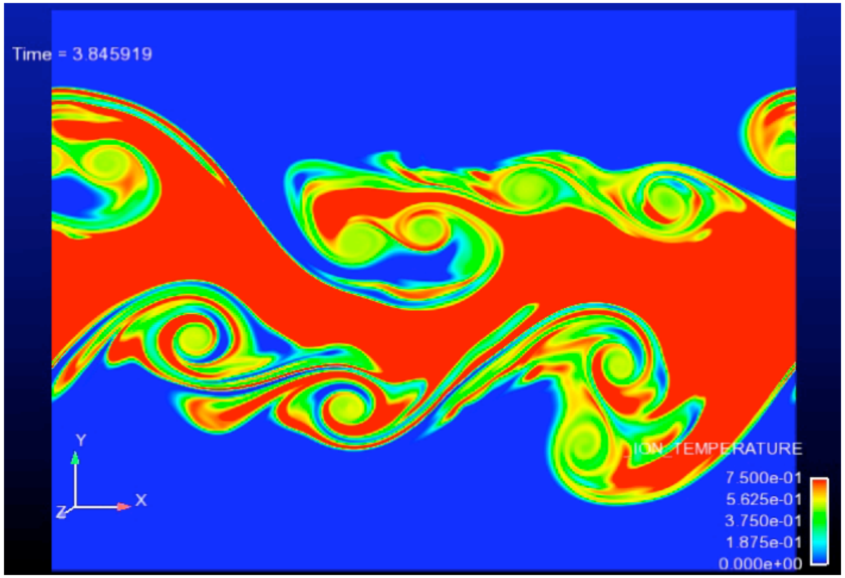
All Simulations use equal-order nodal discretizations!



Swirling Jet: LES

Fully-Coupled Nodal Finite Element Simulation Technology

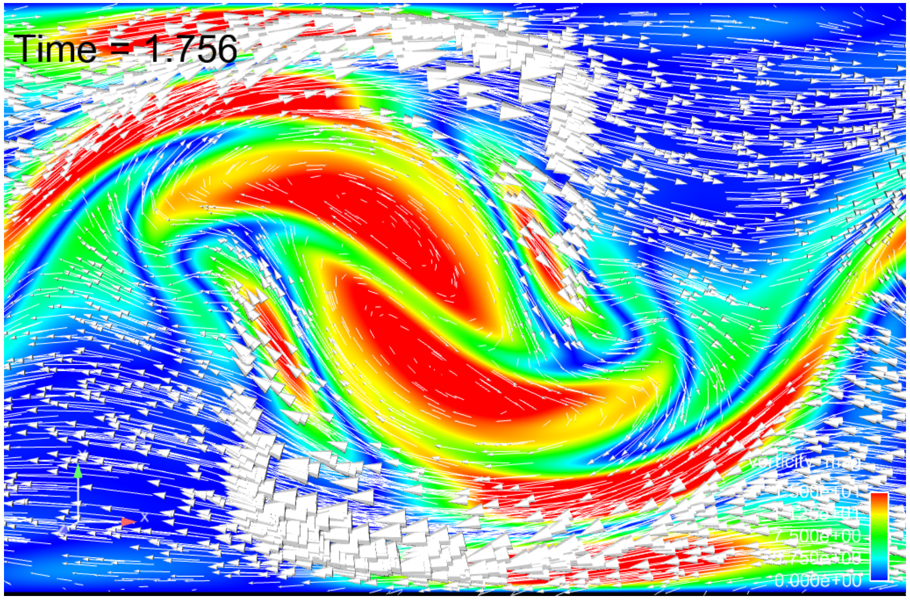
All Simulations use equal-order nodal discretizations!



Kelvin-Helmholtz Instability: Navier-Stokes

Fully-Coupled Nodal Finite Element Simulation Technology

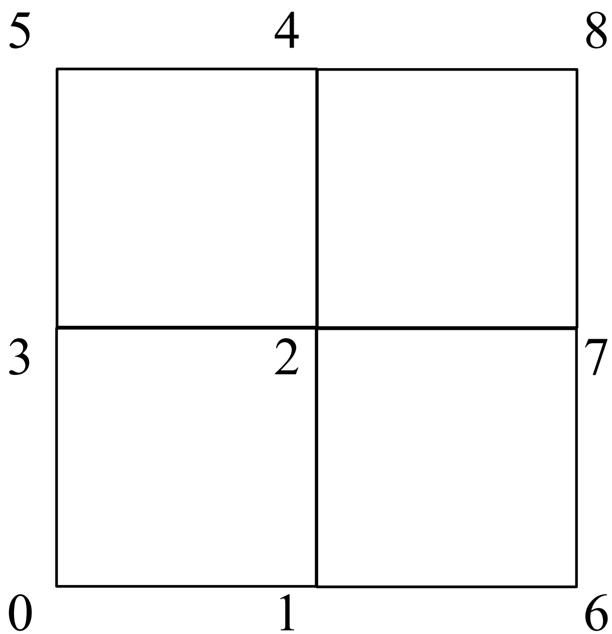
All Simulations use equal-order nodal discretizations!



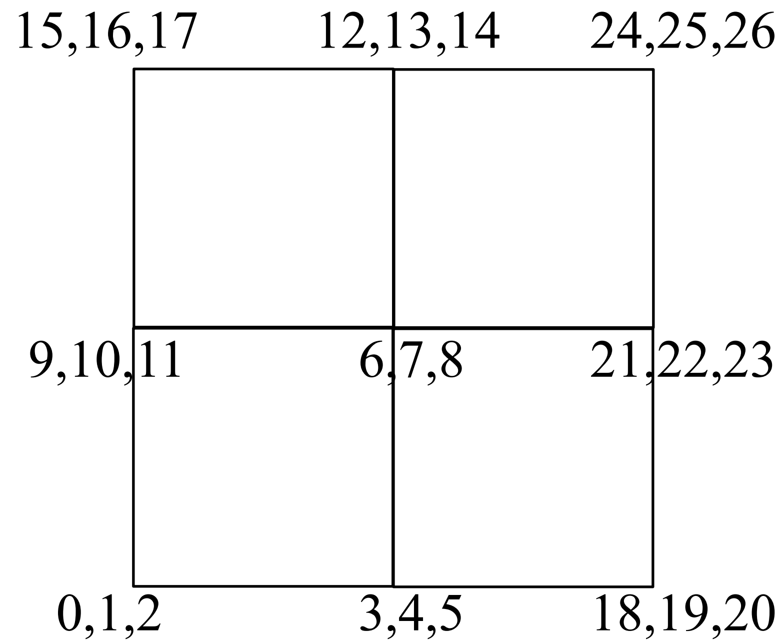
Hydromagnetic Kelvin-Helmholtz: MHD

Fully-Coupled Nodal Finite Element Simulation Technology

Mesh Node IDs



Nodal DOFs: U, V, P



Global Unknown Numbering takes Mesh IDs to Unknown IDs

- Necessary to specify ordering of linear system
- More challenging in Parallel

Parallel Nodal FEM

Pros of Parallel Nodal FEM

✓ Successful for stabilized single physics

- Fluids Dynamics, Mechanics, Semiconductors, MHD, etc…

✓ Only have to handle single-physics

✓ Simple to program, no complex unknown numbering

- Numbering easily derived from parallel mesh DB

Cons of Parallel Nodal FEM

x Code is limited to stabilized discretizations

- Mixed Navier-Stokes (Q2-Q1) requires inf-sup condition

x Multi-physics are not possible (or only through a hack)

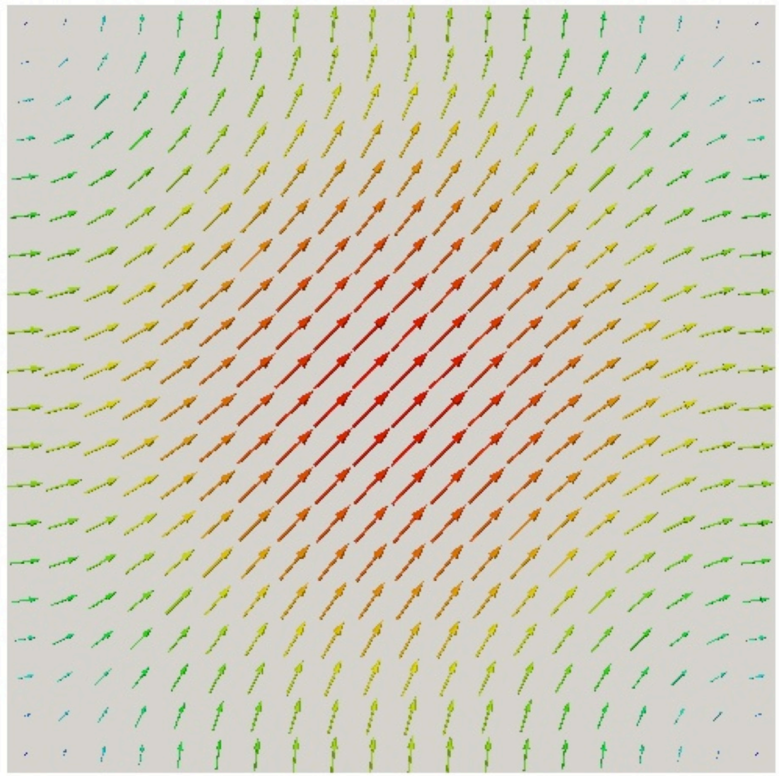
- Solid in one element block interacting with a fluid in another

x Compatible discretizations not possible

- Edge/Face basis functions can make Maxwell easier

Fully-Coupled Beyond-Nodal Discretizations

Simulations enabled by "Beyond-Nodal" Discretizations

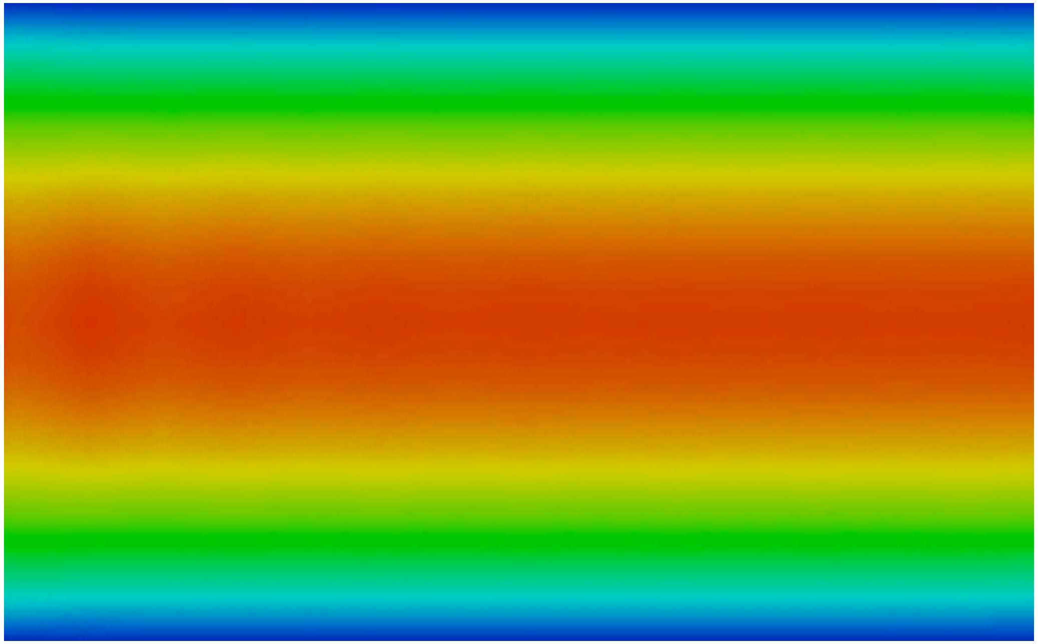


Edge/Face basis functions

- numbering edges/faces
- orientations

Fully-Coupled Beyond-Nodal Discretizations

Simulations enabled by “Beyond-Nodal” Discretizations

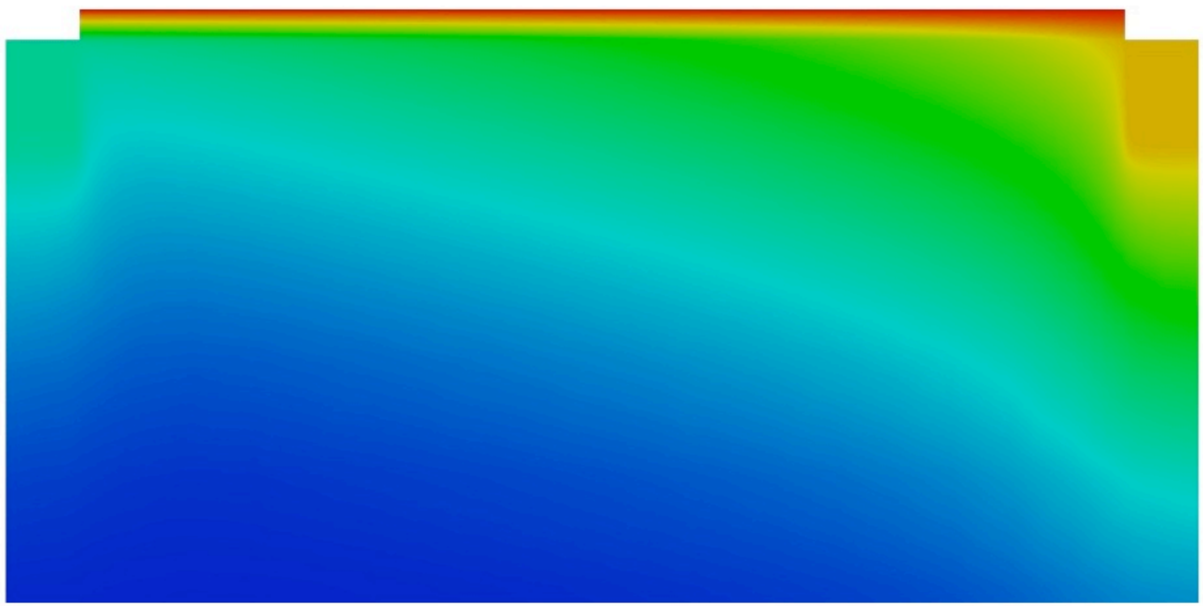


Mixed Inf-Sup stable Navier-Stokes

- Taylor-hood pairs
- No stabilization (parameters) needed

Fully-Coupled Beyond-Nodal Discretizations

Simulations enabled by "Beyond-Nodal" Discretizations

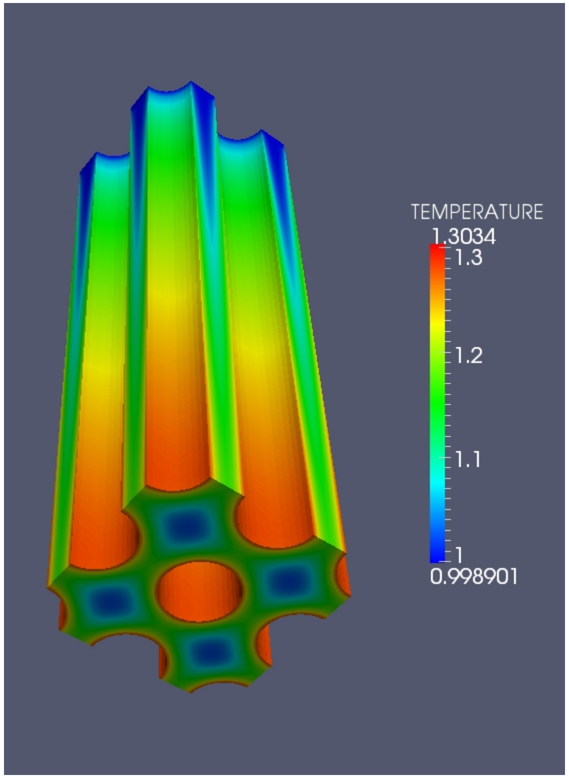


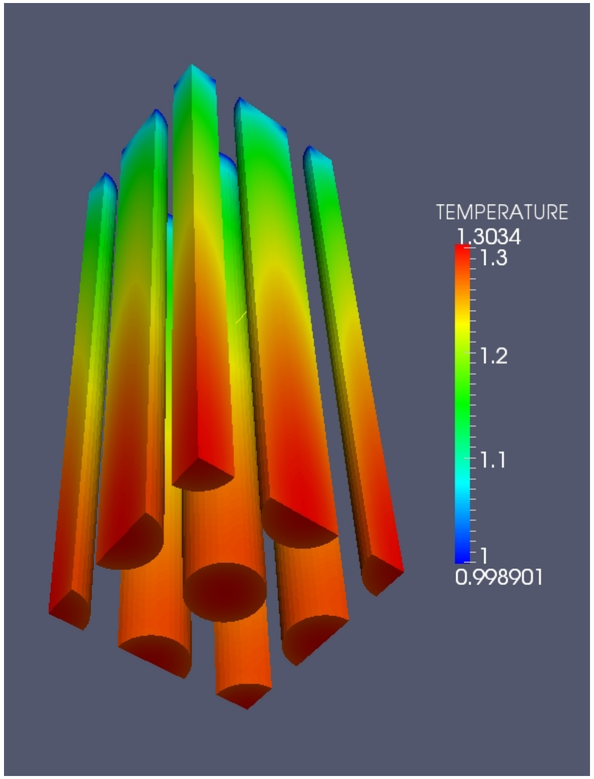
MOSFET Device (Drift-Diffusion Equations)

- Only potential eq solved in "Insulator" domain

Fully-Coupled Beyond-Nodal Discretizations

Simulations enabled by “Beyond-Nodal” Discretizations





Fully-coupled conjugate heat transfer

- Fluid unknowns and solid unknowns

Panzer: a Trilinos Package

Panzer is Trilinos package for Finite Element Assembly

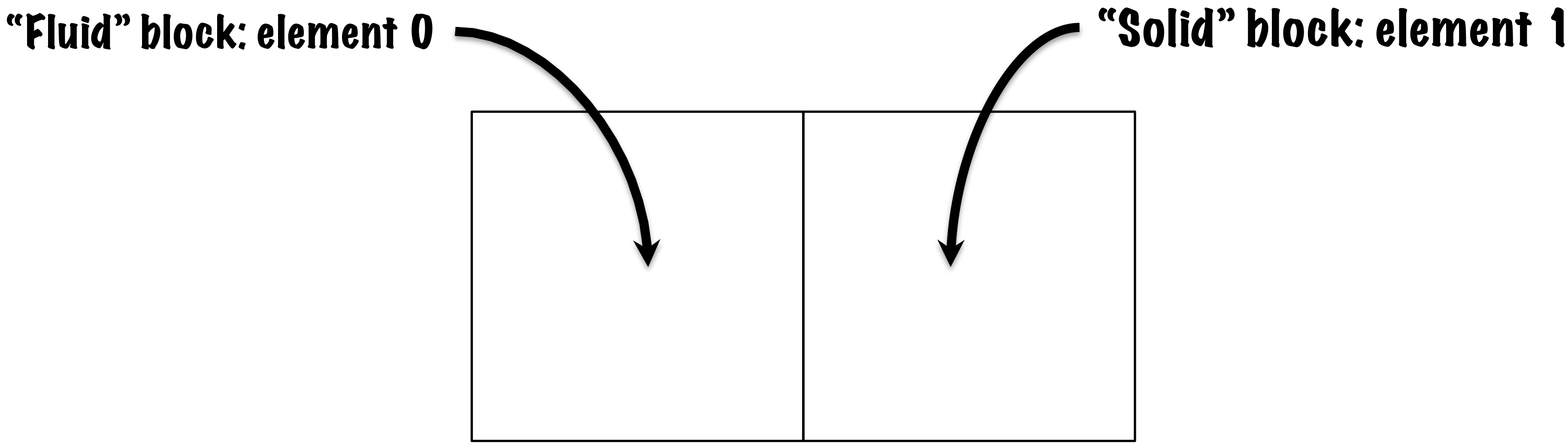
- Experimental capability
- Applications built on Panzer: Drekar (CFD, MHD), Charon2 (Drift-Diffusion)
- Expression graph based assembly (built on Phalanx)
- Automatic differentiation and embedded UQ (Sacado and Stokhos)
- Basis functions from Intrepid
- Default mesh DB from STK (or you can use your own)
- Standalone DOF manager supports “Beyond-Nodal” discretizations
    - Images from previous slide generated with Drekar and Charon2

Panzer DOFManager

Talk focuses on using Panzer DOFManager standalone

- DOFManager is a C++ class for unknown numbering
- Enables parallel “Beyond-Nodal” discretizations
- Optional use of STK mesh DB
- Several examples of assembly provided
- Flexibly implemented on Tpetra: see me or B. Seefeldt for algorithm details

Two Element Example



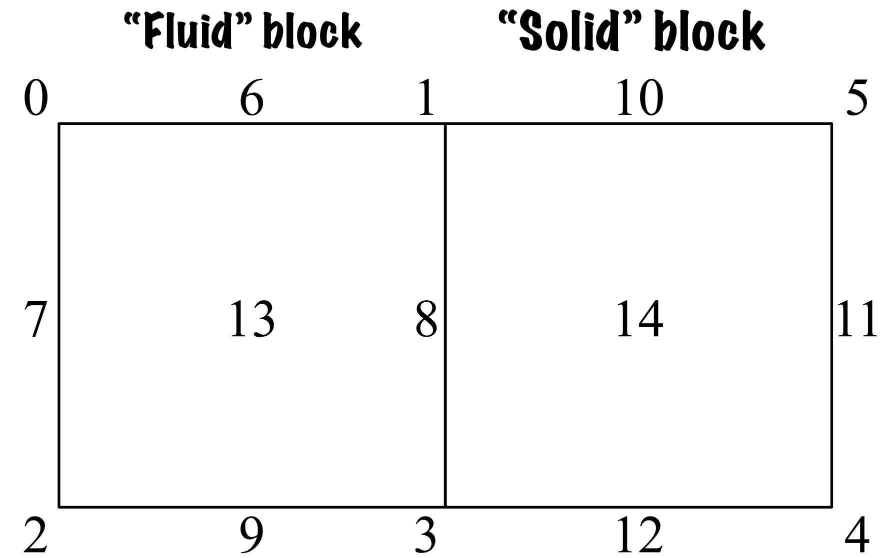
- Two element blocks (each with one element)
  1. Fluid block has fields: U, V, Pres, Temp
  2. Solid block has fields: Temp
- Stable Q2-Q1 pair for fluid unknowns U, V, Pres
- Temp is continuous Q1 across interface
  - ➢ Normal flux continuity is enforced by finite element method

Two Element Example

Possible geometry numbering

- Nodes
- Edges
- Cells

Numbering = Mesh topology

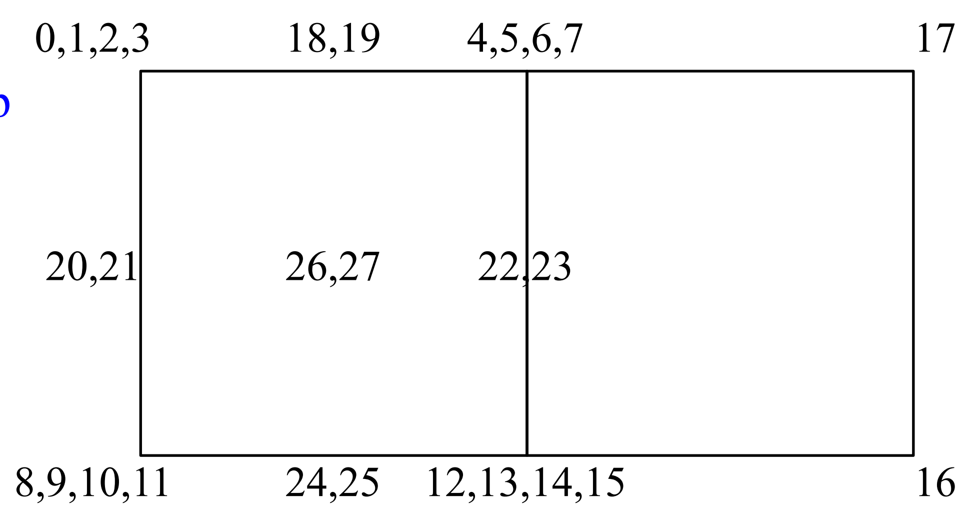


“Fluid” block

- Node numbering: U, V, Pres, Temp
- Edge numbering: U, V
- Cell numbering: U, V

“Solid” block

- Node numbering: Temp
- Edge numbering: n/a
- Cell numbering: na/a



Two Element Example: build DOFManager

```
panzer::DOFManager<int,int> dofManager = ...; // Build DOF manager from mesh topology

// Build Intrepid basis objects for Q2 (velocity) and Q1 (pressure and temperature)
RCP<Intrepid::Basis<double,FieldContainer> > q1_basis = rcp(new Intrepid::Basis_HGRAD_QUAD_C1_FEM);
RCP<Intrepid::Basis<double,FieldContainer> > q2_basis = rcp(new Intrepid::Basis_HGRAD_QUAD_C2_FEM);

// Build field patterns, these define the continuity requirements of the basis
RCP<const panzer::IntrepidFieldPattern> q1_pattern = rcp(new panzer::IntrepidFieldPattern(q1_basis));
RCP<const panzer::IntrepidFieldPattern> q2_pattern = rcp(new panzer::IntrepidFieldPattern(q2_basis));

// register fluid block fields
dofManager.addField("Fluid", "U", q2_pattern);      // Velocity fields use Q2 basis
dofManager.addField("Fluid", "V", q2_pattern);
dofManager.addField("Fluid", "Pres", q1_pattern);   // Pressure and Temperature use Q1 basis
dofManager.addField("Fluid", "Temp", q1_pattern);

// register solid block fields
dofManager.addField("Solid", "Temp", q1_pattern);

// Build global unknowns from specified fields (and patterns)
dofManager.buildGlobalUnknowns();
```

- Parallelism is handled automatically
- Global IDs can be looked up by local element ID
- Orientations can be computed as well

Two Element Example: solution gather

Solution Gather: Get element field coefficients from `Epetra_Vector`
Residual/Jacobian Scatter: fill RHS vector and Jacobian matrix

`panzer::DOFManager` has two functions supporting gather/scatter

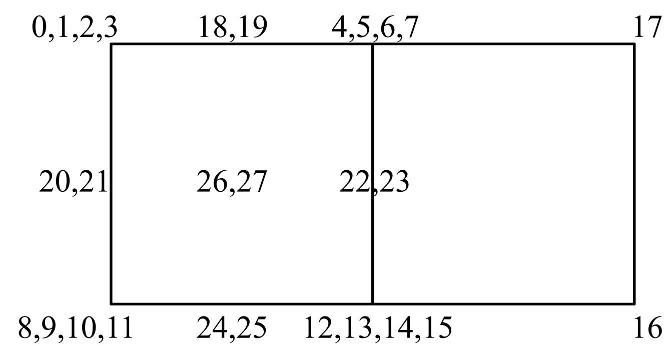
- `dofManager.getElementGIDs`: Global GIDs for an element
- `dofManager.getGIDFieldOffsets`: Index into GID array for field on an element block

For "Solid" element 1

`dofManager.getElementGIDs=`[15, 16, 17, 7]

Temp Temp Temp Temp

`dofManager.getGIDFieldOffsets("Temp")=`[0,1,2,3]



Two Element Example: solution gather

For “Fluid” element 0

`dofManager.getElementGIDs=`[8,9,10,11, 12,13,14,15, 4,5,6,7, 0,1,2,3, **nodes**
U V Pres Temp U V Pres Temp U V Pres Temp U V Pres Temp

24,25, 22,23, 18,19, 20,21 **edges**
U V U V U V U V

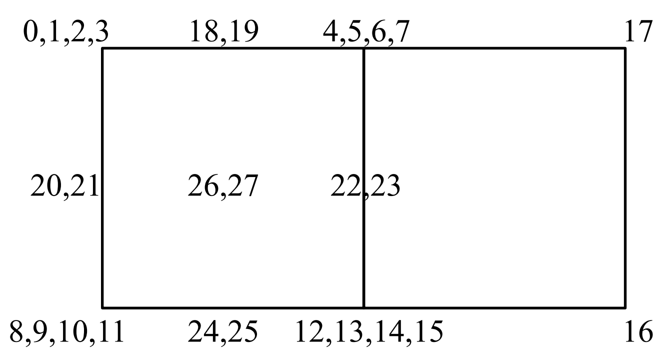
26,27] **cell**
U V

`dofManager.getGIDFieldOffsets(“U”)=`[0,4,8,12,16,18,20,22,24]

`dofManager.getGIDFieldOffsets(“V”)=`[1,5,9,13,17,19,21,23,25]

`dofManager.getGIDFieldOffsets(“Pres”)=`[2,6,10,14]

`dofManager.getGIDFieldOffsets(“Temp”)=`[3,7,11,15]



Two Element Example: solution gather

Code template for solution gather operation using DOFManager

- Code examples written for clarity of DOFManager usage
- Warning: These are not at all optimized!
- Scatter operations follow much the same pattern

```
// Extract the basis function coefficients for a field from an Epetra_Vector
//
// gids: Global element IDs constructed by a call to dofManager.getElementGIDs
// field: Index into GID array, from a call to dofManager.getGIDFieldOffsets
// x: Epetra_Vector to gather from
std::vector<double> fillField(std::vector<int> gids,std::vector<int> field,const Epetra_Vector & x)
{
  std::vector<double> coeffs;
  coeffs.resize(field.size());  // allocate memory for the coefficients

  for(std::size_t i=0;field.size();i++) {
    // get local id for looking up coefficient
    int lid = x.Map().LID(gids[field[i]]);

    coeffs[i] = x[lid]
  }

  return coeffs;
}
```

Two Element Example: solution gather

```
void gatherFluidElementUnknowns(int elementId,const Epetra_Vector & x)
{
  std::vector<int> fl_u_ind = dofManager.getGIDFieldOffsets("Fluid",dofManager.getFieldNum("U"));
  std::vector<int> fl_v_ind = dofManager.getGIDFieldOffsets("Fluid",dofManager.getFieldNum("V"));
  std::vector<int> fl_p_ind = dofManager.getGIDFieldOffsets("Fluid",dofManager.getFieldNum("Pres"));
  std::vector<int> fl_t_ind = dofManager.getGIDFieldOffsets("Fluid",dofManager.getFieldNum("Temp"));

  std::vector<int> gids;
  dofManager.getElementGIDs(elementId,gids);

  // get basis coefficients appropriate for use by intrepid (ordered correctly)
  std::vector<double> u_coeffs = fillField(gids,fl_u_ind,x);
  std::vector<double> v_coeffs = fillField(gids,fl_v_ind,x);
  std::vector<double> p_coeffs = fillField(gids,fl_p_ind,x);
  std::vector<double> t_coeffs = fillField(gids,fl_t_ind,x);
}

void gatherSolidElementUnknowns(int elementId,const Epetra_Vector & x)
{
  std::vector<int> sd_t_ind = dofManager.getGIDFieldOffsets("Solid",dofManager.getFieldNum("Temp"));

  std::vector<int> gids;
  dofManager.getElementGIDs(elementId,gids);

  // get basis coefficients appropriate for use by intrepid (ordered correctly)
  std::vector<double> t_coeffs = fillField(gids,sd_t_ind,x);
}
```

Constructing Epetra_Map objects

How do you construct Epetra (Tpetra) objects?

- DOFManager provides global ID arrays for ghosted and unique IDs
- `DOFManager::getOwnedIndices` – Unique IDs
- `DOFManager::getOwnedAndSharedIndices` – Ghosted IDs

```
std::vector<int> unique, ghosted;  // build index set for maps
dofManager.getOwnedIndices(unique);
dofManager.getOwnedAndSharedIndices(ghosted);

Epetra_Map uniqueMap(-1,unique.size(),&unique[0],Comm);     // map for solving
Epetra_Map ghostedMap(-1,ghosted.size(),&ghosted[0],Comm);  // map for assembly
```

```
Epetra_Import importer(ghostedMap,uniqueMap);
Epetra_Export exporter(ghostedMap,uniqueMap);
Epetra_Vector uniqueVec(uniqueMap), ghostedVec(ghostedMap);
// ... do lots of stuff
ghostedVec.Import(uniqueVec,importer,Insert); // Prepare ghosted vector for assembly
// ... do lots of stuff
uniqueVec.Export(ghostedVec,exporter,Add);    // Sum into global vector for solving
```

Associating Mesh Topology

`DOFManager` needs only mesh topology

- No coordinates needed
- Abstract `ConnManager` interface defines topology
- Default `ConnManager` wrapping STK in Panzer
- Use favorite mesh DB with `DOFManager` by inheriting from `ConnManager`

```
// Pure Virtual base class that interacts with DOFManager
// This class provides the DOFManager the mesh topology
class panzer::ConnManager {
  void buildConnectivity(const panzer::FieldPattern &fp);
  const GlobalOrdinal * getConnectivity(LocalOrdinal localElmtId);
  LocalOrdinal getConnectivitySize(LocalOrdinal localElmtId);
  std::string getBlockId(LocalOrdinal localElmtId);
  std::size_t numElementBlocks();
  void getElementBlocks(std::vector<std::string> &elementBlockIds);
  const std::vector<LocalOrdinal> & getElementBlock(const std::string &blockID);
};
```

Building DOFManager using STK mesh

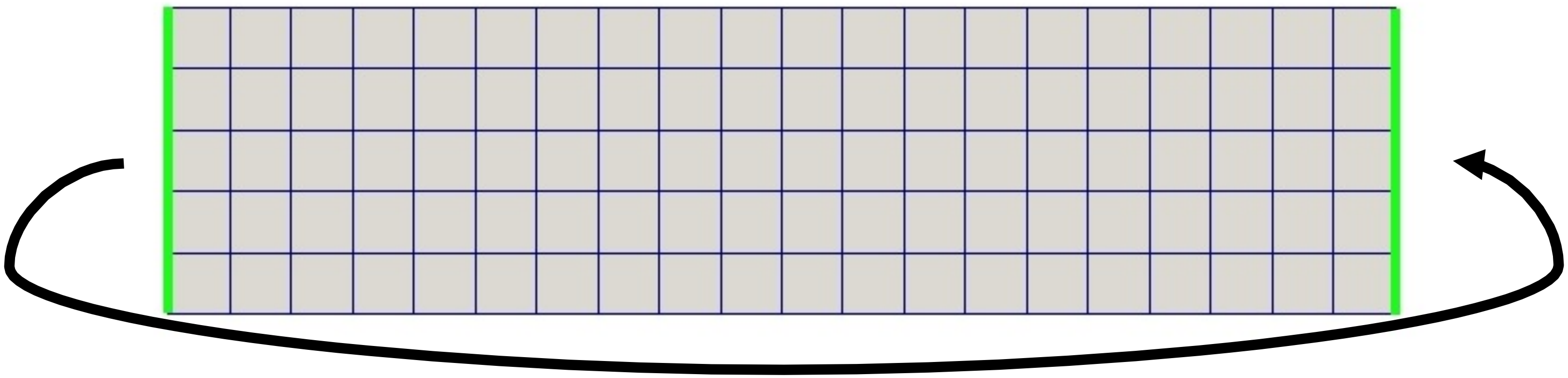
```
// Use panzer's build in STK wrapper
//   panzer will generate square and hex meshes
//   and read from exodus
RCP<panzer_stk::STK_Interface> mesh = ...

// Build a connection manager from the STK wrapper
RCP<panzer::ConnManager<int,int> > connManager
    = Teuchos::rcp(new panzer_stk::STKConnManager(mesh));

// Set the connection manager for the DOF manager
panzer::DOFManager<int,int> dofManager;
dofManager.setConnManager(connManager,MPI_COMM_WORLD);
```

- Panzer includes mesh I/O and simple meshing tools
- Provides a good path for using `DOFManager`
- Separation between "mesh➔ConnManager➔DOFManager" is powerful abstraction

Periodic boundary conditions



To make simulation periodic: Associate nodes, edges on left with the right

- Reassignment guarantees C^0 continuity
- Flux normal continuity from FEM

Periodic Modifications

↓

mesh → ConnManager → DOFManager

Unmodified

Periodic for free! Assembly doesn't change!

Conclusions

- Introduced Panzer DOFManager for global unknown numbering
  - Handles mixed discretizations
  - Handles compatible discretizations
  - Handles multi-domain multi-physics
  - Handles equal-order discretizations
  - Plans to handle high-order elements
- Showed minimal interface
  - `DOFManager::getElementGIDs` – Mapping form element to Vector
  - `DOFManager::getGIDFieldOffsets` – Mapping from fields to Vector
  - `DOFManager::getOwnedIndices` – Unique IDs
  - `DOFManager::getOwnedAndSharedIndices` – Ghosted IDs
- Can use STK mesh or your own
- Templated on global and local IDs
- panzer::DOFManager is ready to use!