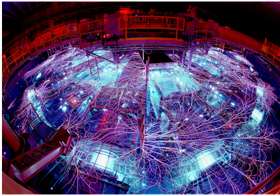
Exceptional service in the national interest

Sandia National Laboratories



Panzer

A Finite Element Assembly Engine within the Trilinos Framework

Jason M. Gates, Roger P. Pawlowksi, Eric C. Cyr

Sandia National Laboratories

March 3, 2017

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-mission 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. SAND NO. 2017-1977 C

What is Panzer?

- C++ Library
- General finite element assembly engine for multi-physics simulation
- Supports 1-, 2-, & 3-D unstructured mesh calculations
- Supplies quantities needed for advanced solution and analysis algorithms
  - residuals
  - Jacobians
  - parameter sensitivities
  - stochastic residuals/etc.

What is Panzer?

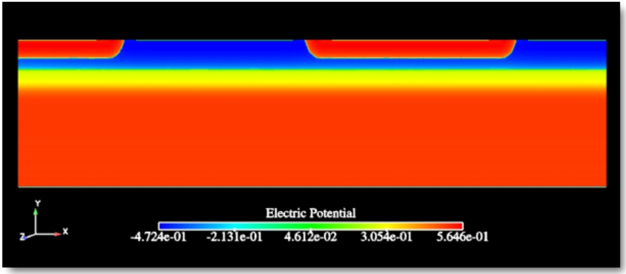
- Contains no physics-specific code—physics applications are light-weight front ends
- Massively parallel for complex physics
- Leverages template-based generic programming[5] to assemble quantities of interest
- Incorporates 35 Trilinos packages

What is Panzer not?

- Application
- Domain specific language
- Front end preprocessor/interpreter
- deal.II, FEniCS, MFEM, MOOSE, Sundance

Panzer's History

- Lessons learned from Sandia's PDE physics codes
  - Charon1, MPSalsa, etc.
- Monolithic application $\longrightarrow$ library of packages
- Capabilities explored/developed $\longrightarrow$ Phalanx, Panzer

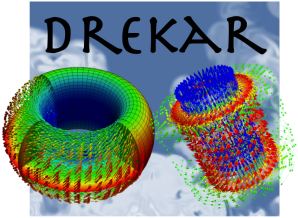


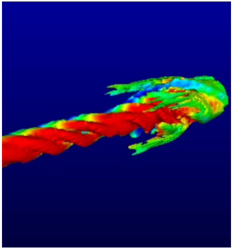
Panzer's History

- Drekar: advanced algorithm demonstration
- Applications (Drekar, Charon2, EMPIRE, etc.) drive Panzer's requirements, design goals
  - Coupled multi-physics
  - Large scale simulation (>100k cores)
  - Finite element focussed (currently)
  - Embedded analysis (AD, sensitivities)
  - Technology sharing and deployment
- Panzer provides applications with flexible infrastructure, core technologies

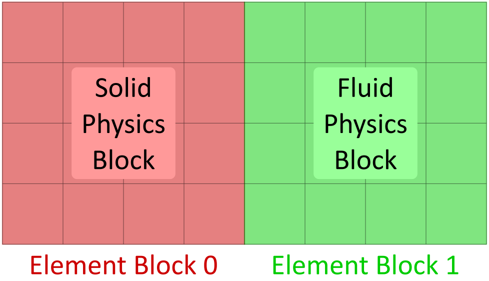
Panzer Enables

- Applications
  - Turbulent CFD
  - Magnetohydrodynamics
  - Semiconductor devices
- Supporting technologies
  - Algebraic multigrid
  - Block preconditioning
  - Uncertainty quantification
  - IMEX
  - PDE constrained optimization
  - Compatible discretizations



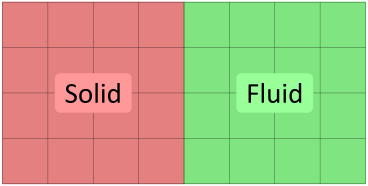


Element & Physics Blocks



- Users divide the domain into element blocks
- Each element block maps to a single physics block
- Physics blocks contain a list of equation sets

Equation Sets



- Equation sets define the form of the PDE
- Details are filled in using closure models

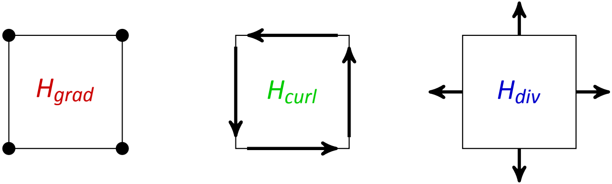
Navier-Stokes
$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \nabla p = f$$
$$\nabla \cdot u = 0$$

Energy
$$\frac{\partial T}{\partial t} + u \cdot \nabla T - \nabla \cdot (\sigma \nabla T) = 0$$

Data Mapping Utilities

Finite element discretizations have changed

- Historically used nodal equal-order finite elements
- New code embraces mixed discretizations
- Also using high-order compatible discretizations
- H_{grad} (nodal), H_{curl} (edge), H_{div} (face)
- Requires extra data management (orientations)



Data Mapping Utilities

Three primary pieces:

FieldPattern	Describes basis layout & continuity of fields
ConnManager	Mesh topology from field pattern (mesh abstraction)
DOFManager	Manages and computes unknown field numbers

- Panzer = mesh-agnostic
- panzer_stk = concrete implementation of ConnManager

FieldPattern

Linear pressure, temperature

Quadratic velocities

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \nabla p = f$$

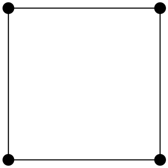
$$\nabla \cdot u = 0$$

$$\frac{\partial T}{\partial t} + u \cdot \nabla T - \nabla \cdot (\sigma \nabla T) = 0$$

FieldPattern

Linear pressure, temperature

Quadratic velocities



p, T

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \nabla p = f$$

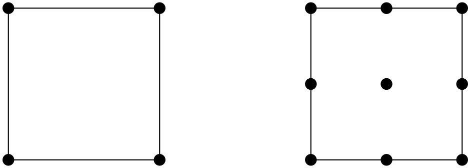
$$\nabla \cdot u = 0$$

$$\frac{\partial T}{\partial t} + u \cdot \nabla T - \nabla \cdot (\sigma \nabla T) = 0$$

FieldPattern

Linear pressure, temperature

Quadratic velocities



p, T $\qquad$ u_x, u_y

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \nabla p = f$$

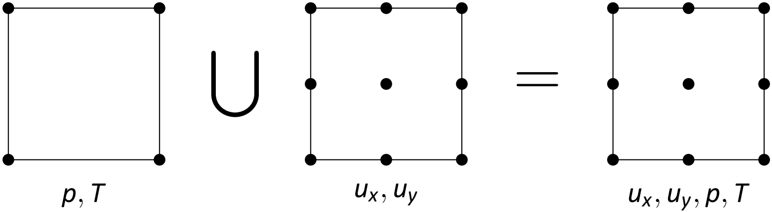
$$\nabla \cdot u = 0$$

$$\frac{\partial T}{\partial t} + u \cdot \nabla T - \nabla \cdot (\sigma \nabla T) = 0$$

FieldPattern

Linear pressure, temperature

Quadratic velocities



$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \nabla p = f$$

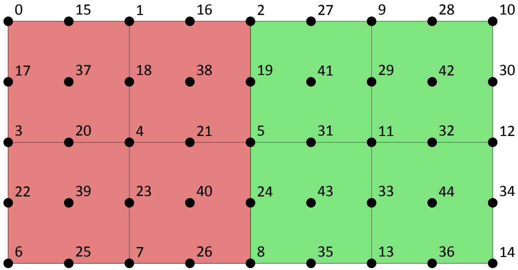
$$\nabla \cdot u = 0$$

$$\frac{\partial T}{\partial t} + u \cdot \nabla T - \nabla \cdot (\sigma \nabla T) = 0$$

ConnManager

Linear pressure, temperature

Quadratic velocities



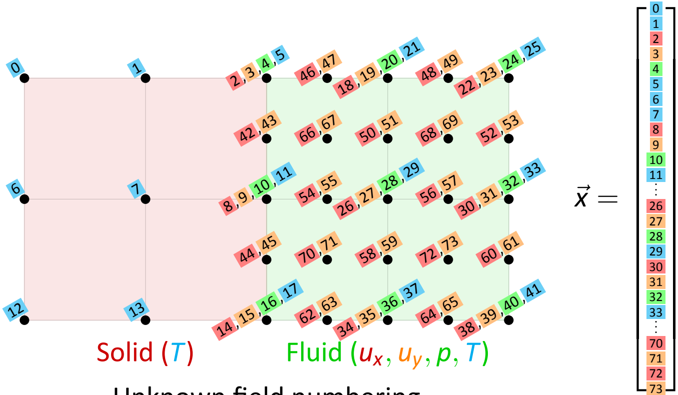
Solid (T) Fluid (u_x, u_y, p, T)

Numbering = mesh topology

DOFManager[1]

Linear pressure, temperature

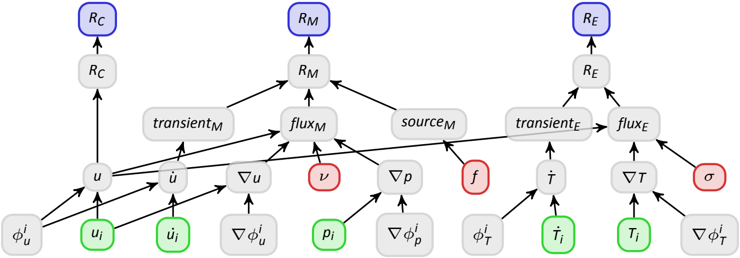
Quadratic velocities



Unknown field numbering

DAG-Based Assembly (Phalanx)[2, 3, 4]

- Decompose complex model into graph of simple kernels
- Rapid development, separation of concerns, extensibility
- Automated dependency tracking
- Topological sort to order evaluations

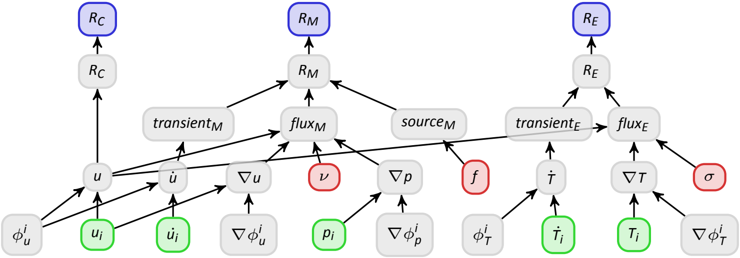


$$R_M = \int_\Omega \left(\frac{\partial u}{\partial t} + u \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \nabla p - f \right) \cdot v \, d\Omega, \quad R_C = \int_\Omega (\nabla \cdot u) \cdot v \, d\Omega, \quad R_E = \int_\Omega \left(\frac{\partial T}{\partial t} + u \cdot \nabla T - \nabla \cdot (\sigma \nabla T) \right) \cdot v \, d\Omega$$

DAG-Based Assembly (Phalanx)[2, 3, 4]

Sandia National Laboratories

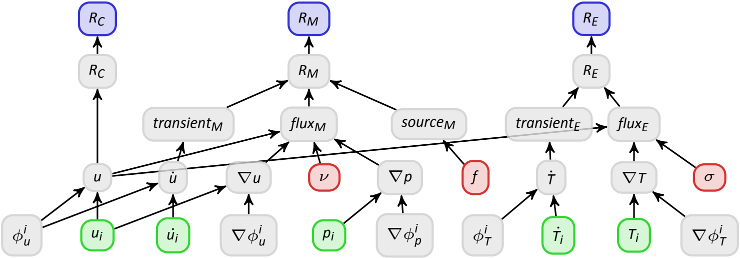
- Nodes can be swapped out
- Separation of data and kernels operating on the data
- Multi-physics complexity handled automatically
- Easy to add equations, change models, test in isolation



$$R_M = \int_\Omega \left(\frac{\partial u}{\partial t} + u \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \nabla p - f \right) \cdot v \, d\Omega, \quad R_C = \int_\Omega (\nabla \cdot u) \cdot v \, d\Omega, \quad R_E = \int_\Omega \left(\frac{\partial T}{\partial t} + u \cdot \nabla T - \nabla \cdot (\sigma \nabla T) \right) \cdot v \, d\Omega$$

Evaluators

- Declare fields to evaluate (or to contribute to)
- Declare dependent fields
- Function to perform evaluation
- Templated on evaluation type
  - Specializations for scatters & gathers
  - User code reused for residual, Jacobian, Hessian, etc.



$$R_M = \int_\Omega \left(\frac{\partial u}{\partial t} + u \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \nabla p - f \right) \cdot v \, d\Omega, \quad R_C = \int_\Omega (\nabla \cdot u) \cdot v \, d\Omega, \quad R_E = \int_\Omega \left(\frac{\partial T}{\partial t} + u \cdot \nabla T - \nabla \cdot (\sigma \nabla T) \right) \cdot v \, d\Omega$$

An Example Problem

git clone git@github.com:trilinos/Trilinos

cd Trilinos/packages/panzer/adapters-stk/tutorial/siamCse17

$$
\begin{aligned}
-\Delta u(x,y) + k^2 u(x,y) &= f(x,y), \quad && (x,y) \in \Omega = (0,1) \times (0,1) \\
u(x,y) &= 0, && (x,y) \in \partial\Omega
\end{aligned}
$$

An Example Problem

git clone git@github.com:trilinos/Trilinos

cd Trilinos/packages/panzer/adapters-stk/tutorial/siamCse17

$$\begin{aligned} -\Delta u(x,y) + k^2 u(x,y) &= \sin(2\pi x)\sin(2\pi y), \quad && (x,y) \in \Omega \\ u(x,y) &= 0, && (x,y) \in \partial\Omega \end{aligned}$$

An Example Problem

git clone git@github.com:trilinos/Trilinos

cd Trilinos/packages/panzer/adapters-stk/tutorial/siamCse17

$$
\begin{aligned}
-\Delta u(x,y) + (1 - 8\pi^2)u(x,y) &= \sin(2\pi x)\sin(2\pi y), \quad && (x,y) \in \Omega \\
u(x,y) &= 0, && (x,y) \in \partial\Omega
\end{aligned}
$$

An Example Problem

git clone git@github.com:trilinos/Trilinos

cd Trilinos/packages/panzer/adapters-stk/tutorial/siamCse17

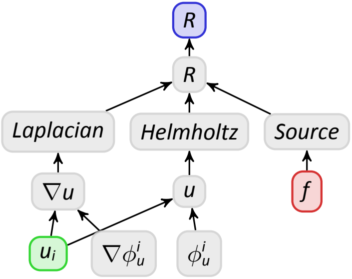
$$
\begin{aligned}
-\Delta u(x,y) + (1-8\pi^2)u(x,y) &= \sin(2\pi x)\sin(2\pi y), \quad && (x,y) \in \Omega \\
u(x,y) &= 0, && (x,y) \in \partial\Omega
\end{aligned}
$$

Weak Form

$$
\int_\Omega \nabla u \cdot \nabla v \, d\Omega + (1-8\pi^2)\int_\Omega uv \, d\Omega = \int_\Omega \sin(2\pi x)\sin(2\pi y) v \, d\Omega
$$

An Example Problem

$$R = \int_\Omega \nabla u \cdot \nabla v \, d\Omega + (1 - 8\pi^2) \int_\Omega uv \, d\Omega - \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega = 0$$



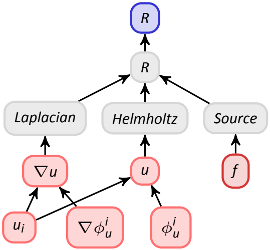
Create an EquationSet

```cpp
// myEquationSet.hpp
template<typename EvalT>
class MyEquationSet
  :
  public panzer::EquationSet_DefaultImpl<EvalT>
{
  public:
    MyEquationSet(...);
    void buildAndRegisterEquationSetEvaluators(...) const;
  private:
    std::string dofName_;
} // end of class MyEquationSet
```

Add the Degree of Freedom and its Gradient

```
// myEquationSetImpl.hpp
template<typename EvalT>
MyEquationSet<EvalT>::
MyEquationSet(...)
{
  ...
  dofName_ = "U";
  std::string basisType("HGrad");
  int basisOrder(1), integrationOrder(2);
  this->addDOF(dofName_, basisType,
    basisOrder, integrationOrder);
  this->addDOFGrad(dofName_);
  ...
  this->setupDOFs();
} // end of Constructor
```

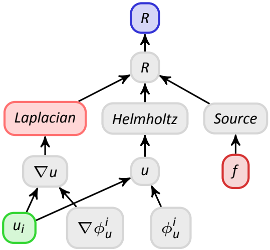
$$
\begin{aligned}
R = &\int_\Omega \nabla u \cdot \nabla v \, d\Omega \\
&+ (1 - 8\pi^2) \int_\Omega uv \, d\Omega \\
&- \int_\Omega \sin(2\pi x)\sin(2\pi y) v \, d\Omega \\
= &\; 0
\end{aligned}
$$



Add the Laplacian Term

```
// still in myEquationSetImpl.hpp
template<typename EvalT> void
  MyEquationSet::
buildAndRegisterEquationSetEvaluators(
  PHX::FieldManager<panzer::Traits>& fm,
    ...) const
{
  Teuchos::RCP<panzer::IntegrationRule>
    ir =
    this->getIntRuleForDOF(dofName_);
  Teuchos::RCP<panzer::BasisIRLayout>
    basis =
    this->getBasisIRLayoutForDOF(dofName_);
  ...
```

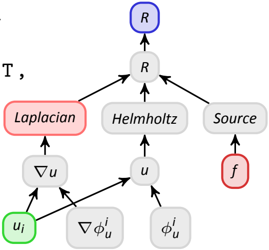
$$R = \int_\Omega \nabla u \cdot \nabla v \, d\Omega + (1 - 8\pi^2) \int_\Omega uv \, d\Omega - \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega = 0$$



Add the Laplacian Term

```
std::string laplacianName("RESIDUAL_"
  + dofName_ + "_LAPLACIAN");
Teuchos::ParameterList p;
p.set("Residual Name", laplacianName);
p.set("Flux Name", "GRAD_" + dofName_);
p.set("IR", ir);
p.set("Basis", basis);
p.set("Multiplier", 1.0);
Teuchos::RCP<PHX::Evaluator<panzer::Traits>>
  op = Teuchos::rcp(new
  panzer::Integrator_GradBasisDotVector<EvalT,
    panzer::Traits>(p));
this->template
  registerEvaluator<EvalT>(fm, op);
...
```

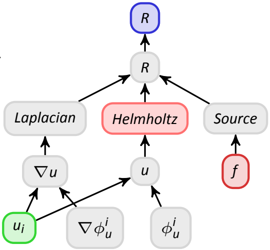
$$
\begin{aligned}
R = & \int_\Omega \nabla u \cdot \nabla v \, d\Omega \\
& + (1 - 8\pi^2) \int_\Omega uv \, d\Omega \\
& - \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega \\
= & \; 0
\end{aligned}
$$



Add the Helmholtz Term

```
// still in
  buildAndRegisterEquationSetEvaluators()
  std::string helmholtzName("RESIDUAL_"
    + dofName_ + "_HELMHOLTZ");
  Teuchos::ParameterList p;
  p.set("Residual Name", helmholtzName);
  p.set("Value Name", dofName_);
  p.set("IR", ir);
  p.set("Basis", basis);
  p.set("Multiplier",
    (1.0 - 8.0 * M_PI * M_PI));
  Teuchos::RCP<PHX::Evaluator<panzer::Traits>>
    op = Teuchos::rcp(new
    panzer::Integrator_BasisTimesScalar<EvalT,
      panzer::Traits>(p));
  this->template
    registerEvaluator<EvalT>(fm, op);
  ...
```

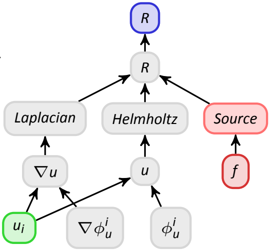
$$
\begin{aligned}
R = & \int_\Omega \nabla u \cdot \nabla v \, d\Omega \\
& + (1 - 8\pi^2) \int_\Omega uv \, d\Omega \\
& - \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega \\
= & \; 0
\end{aligned}
$$



Add the Source Term

```
// still in
  buildAndRegisterEquationSetEvaluators()
  std::string sourceName("RESIDUAL_" +
    dofName_ + "_SOURCE");
  Teuchos::ParameterList p;
  p.set("Residual Name", sourceName);
  p.set("Value Name", dofName_ +
    "_SOURCE");
  p.set("IR", ir);
  p.set("Basis", basis);
  p.set("Multiplier", -1.0);
  Teuchos::RCP<PHX::Evaluator<panzer::Traits>>
    op = Teuchos::rcp(new
    panzer::Integrator_BasisTimesScalar<EvalT,
      panzer::Traits>(p));
  this->template
    registerEvaluator<EvalT>(fm, op);
  ...
```

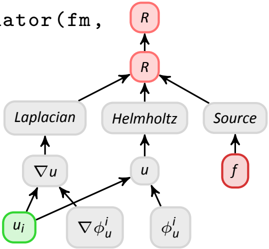
$$
\begin{aligned}
R = &\int_\Omega \nabla u \cdot \nabla v \, d\Omega \\
&+ (1 - 8\pi^2) \int_\Omega uv \, d\Omega \\
&- \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega \\
= 0
\end{aligned}
$$



Add the Residual

$$R = \int_\Omega \nabla u \cdot \nabla v \, d\Omega + (1 - 8\pi^2) \int_\Omega uv \, d\Omega - \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega = 0$$

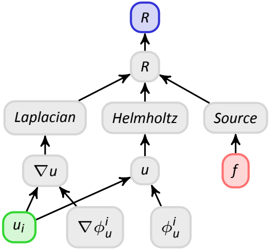
```
// still in
  buildAndRegisterEquationSetEvaluators()
  std::vector<std::string>
    residualOperatorNames{laplacianName,
    helmholtzName, sourceName};
  this->buildAndRegisterResidualSummationEvaluator(fm,
    dofName_, residualOperatorNames);
} // end of
  buildAndRegisterEquationSetEvaluators()
```



Create the Source Function

```
// sourceTerm.hpp
template<typename EvalT, typename Traits>
class MySourceTerm
  :
  public
    PHX::EvaluatorWithBaseImpl<Traits>,
  public PHX::EvaluatorDerived<EvalT,
    Traits>
{
  public:
    MySourceTerm(...);
    void postRegistrationSetup(...);
    void evaluateFields(...);
  private:
    PHX::MDField<EvalT::ScalarT,
      panzer::Cell, panzer::Point>
      result;
    int irDegree_, irIndex_;
} // end of class MySourceTerm
```

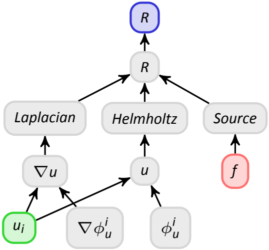
$$R = \int_\Omega \nabla u \cdot \nabla v \, d\Omega + (1 - 8\pi^2) \int_\Omega uv \, d\Omega - \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega = 0$$



Create the Source Function

```
// sourceTermImpl.hpp
...
evaluateFields(typename Traits::EvalData
  workset)
{
  const auto& coords =
    workset.int_rules[irIndex]->ip_coordinates;
  Kokkos::parallel_for(workset.num_cells,
    [=] (const panzer::index_t c)
  {
    for (int p(0);
      p < result.extent_int(1); ++p)
    {
      const double& x(coords(c, p, 0)),
        y(coords(c, p, 1));
      result(c, p) = sin(2 * M_PI * x)
        * sin(2 * M_PI * y);
    } // end loop over the IPs
  }); // end loop over the cells
} // end of evaluateFields()
```

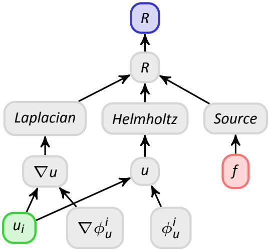
$$
\begin{aligned}
R = &\int_\Omega \nabla u \cdot \nabla v \, d\Omega \\
&+ (1 - 8\pi^2) \int_\Omega uv \, d\Omega \\
&- \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega \\
= &\ 0
\end{aligned}
$$



Create the ClosureModelFactory

```
// closureModelFactory.hpp
template<typename EvalT>
class MyClosureModelFactory
  :
  public
    panzer::ClosureModelFactory<EvalT>
{
  public:
    typedef std::vector<Teuchos::RCP<
      PHX::Evaluator<panzer::Traits>>>
        EvalVec;
    typedef Teuchos::RCP<EvalVec>
      EvalVecRCP;
    EvalVecRCP buildClosureModels(...)
      const;
} // end of class MyClosureModelFactory
```

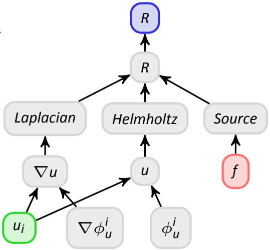
$$
\begin{aligned}
R = &\int_\Omega \nabla u \cdot \nabla v \, d\Omega \\
&+ (1 - 8\pi^2) \int_\Omega uv \, d\Omega \\
&- \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega \\
= &\ 0
\end{aligned}
$$



Create the ClosureModelFactory

```
// closureModelFactoryImpl.hpp
template<typename EvalT>
EvalVecRCP MyClosureModelFactory<EvalT>::
buildClosureModels(..., const
  Teuchos::RCP<panzer::IntegrationRule>&
  ir, ...) const
{
  EvalVecRCP evaluators =
    Teuchos::rcp(new EvalVec);
  ...
  Teuchos::RCP<PHX::Evaluator<panzer::Traits>>
    e =
    Teuchos::rcp(new MySourceTerm<EvalT,
      panzer::Traits>("U_SOURCE", *ir));
  evaluators->push_back(e);
  ...
  return evaluators;
} // end of buildClosureModels()
```

$$
\begin{aligned}
R = & \int_\Omega \nabla u \cdot \nabla v \, d\Omega \\
& + (1 - 8\pi^2) \int_\Omega uv \, d\Omega \\
& - \int_\Omega \sin(2\pi x) \sin(2\pi y) v \, d\Omega \\
= & \; 0
\end{aligned}
$$



Summary of Steps

```
git clone git@github.com:trilinos/Trilinos
cd Trilinos/packages/panzer/adapters-stk/tutorial/siamCse17
```

1. Create an EquationSet
   - 1.1 Add the degree of freedom and its gradient
   - 1.2 Add the Laplacian term
   - 1.3 Add the Helmholtz term
   - 1.4 Add the source term
   - 1.5 Add the residual
2. Create the source function
3. Create the ClosureModelFactory

Concluding remarks

- Application developers focus on complexities in physics models, boundary conditions, etc.
- Rapid prototyping with relative ease
- Advanced analysis = free
- Use Panzer $\longrightarrow$ use Trilinos
- How I use Trilinos
  - Every-day use: Panzer, Teuchos, Thyra, Phalanx, Epetra/Tpetra
  - Every once in a while: NOX, LOCA, Piro, Teko

References

[1] Eric C. Cyr, Mark Hoemmen, Roger P. Pawlowski, and Ben Seefeldt. "Parallel Unknown Numbering for the Finite-Element Method". In: (2017). Submitted to ACM TOMS.

[2] Patrick K. Notz, Roger P. Pawlowski, and James C. Sutherland. "Graph-Based Software Design for Managing Complexity and Enabling Concurrency in Multiphysics PDE Software". In: *ACM Trans. Math. Softw.* 39.1 (Nov. 2012), 1:1–1:21. ISSN: 0098-3500. DOI: 10.1145/2382585.2382586. URL: http://doi.acm.org/10.1145/2382585.2382586.

[3] Roger P. Pawlowski, Eric T. Phipps, and Andrew G. Salinger. "Automating Embedded Analysis Capabilities and Managing Software Complexity in Multiphysics Simulation, Part I: Template-based Generic Programming". In: *Sci. Program.* 20.2 (Apr. 2012), pp. 197–219. ISSN: 1058-9244. DOI: 10.1155/2012/202071. URL: http://dx.doi.org/10.1155/2012/202071.

[4] Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Steven J. Owen, Christopher M. Siefert, and Matthew L. Staten. "Automating Embedded Analysis Capabilities and Managing Software Complexity in Multiphysics Simulation, Part II: Application to Partial Differential Equations". In: *Sci. Program.* 20.3 (July 2012), pp. 327–345. ISSN: 1058-9244. DOI: 10.1155/2012/818262. URL: http://dx.doi.org/10.1155/2012/818262.

[5] Eric T. Phipps and Roger P. Pawlowski. "Efficient Expression Templates for Operator Overloading-based Automatic Differentiation". In: *CoRR* abs/1205.3506 (2012). URL: http://arxiv.org/abs/1205.3506.