## C3 Summer Boot Camp

**Sponsored by Center For Advanced Energy Studies**

Computational Modeling and Data Science Workshops

# *Introduction to MOOSE Framework – Part II*


Min Long

Department of Computer Science, Boise State


2020/08/11

Online Helpers: Andy Lau (BSU)

https://nanohub.org/groups/caesinlc3compute/workshop_page_or_topic_page

CAES Center for Advanced Energy Studies

A COLLABORATION BETWEEN

INL Idaho National Laboratory

BOISE STATE UNIVERSITY

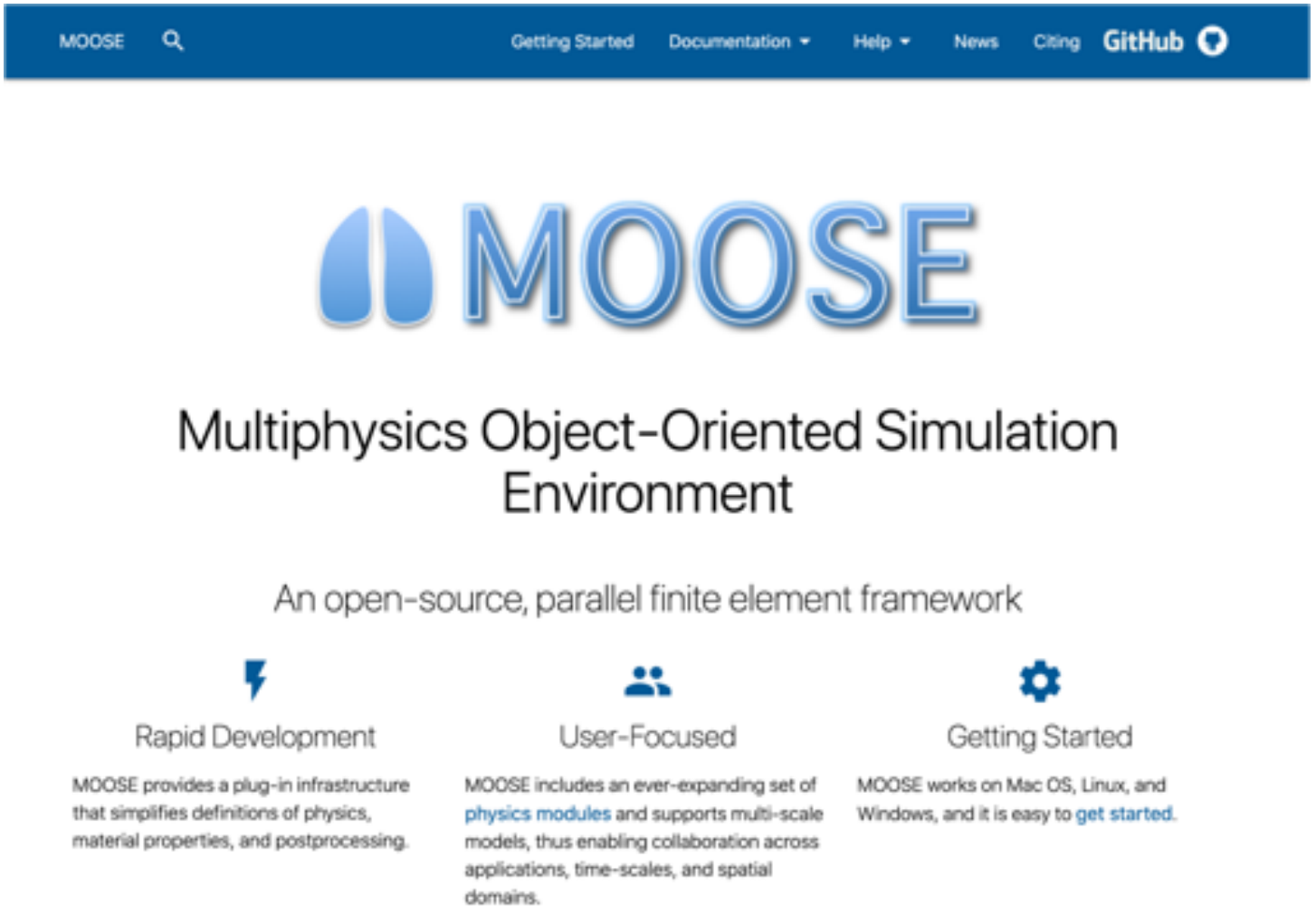
Idaho State UNIVERSITY

University of Idaho

UNIVERSITY OF WYOMING

# Reference

- http://mooseframework.org/



# Outline

- Review: Installation
- An Advanced Diffusion Problem: Darcy Flow
- Tutorial Steps

# Installation

# System Requirements

- GNU/Linux/MacOS Requirements:
  - GCC/Clang C++11 compiler (GCC > 4.8.4, or Clang > 3.5.1)
    - Intel Compilers are not supported
  - Memory (> 8 GB)
  - Processor (64bit x86)
  - Disk (> 30GB)

- Using MOOSE on Windows 10 is experimental and not fully supported.

# 1. Install GNU Compiler Collection (optional)

- Mac: Xcode command line tools

```
$ xcode-select --install
```

- Ubuntu: build-essential

```
mlong@ENG401524:~$ sudo apt-get install build-essential
[sudo] password for mlong:
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
The following packages were automatically installed and are no longer required:
  app-install-data apt-xapian-index cdrdao gcc-6-base gconf-service gconf-service-backend
  gconf2-common libappindicator1 libaprutil1-dbd-mysql libaprutil1-ldap libdbusmenu-gtk4
  libgconf-2-4 libgtkmm-3.0-1v5 libindicator7 libmime-charset-perl libmysqlclient20
  libnih-dbus1 libperl4-corelibs-perl libqt4-designer libqt4-help libqt4-scripttools
  libqt4-svg libqt4-test libqtassistantclient4 librsync1 libsombok3
  libunicode-linebreak-perl mysql-common python-apt python-aptdaemon
  python-aptdaemon.gtk3widgets python-asn1crypto python-attr python-automat python-blinker
  python-cairo python-cffi-backend python-click python-colorama python-constantly
  python-cryptography python-cups python-dbus python-debian python-debtagshw python-defer
  python-dirspec python-enum34 python-gi python-gi-cairo python-httplib2 python-hyperlink
  python-idna python-imaging python-incremental python-ipaddress python-jwt python-ldb
  python-lockfile python-oauthlib python-olefile python-openssl python-pam python-pil
  python-piston-mini-client python-pyasn1 python-qt4 python-qt4-dbus python-serial
  python-sip python-tdb python-twisted-bin python-xapian python-xdg python-zope.interface
  python3-piston-mini-client python3-xapian ruby-minitest ruby-power-assert
  software-center-aptdaemon-plugins texlive-font-utils texlive-pictures
  texlive-pictures-doc
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 238 not upgraded.
```

# 2. Install Conda Environment

- Conda is management system of open source packages and environment.
- MOOSE prefers to obtain support from Conda's libraries.
- Install Miniconda (or Anaconda)
  - Linux

```
curl -L -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh -b -p ~/miniconda3
```

  - Mac

```
curl -L -O https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
bash Miniconda3-latest-MacOSX-x86_64.sh -b -p ~/miniconda3
```

  - Or download the installer from
    - https://docs.conda.io/projects/conda/en/latest/user-guide/install/macos.html

# 3. Create a Conda MOOSE Environment

- Export PATH

```
$ export PATH=$HOME/miniconda3/bin:$PATH
```

  - Or edit your .bashrc or .profile and add the above line and run command

```
$ . ~/profile
```

- Configure Conda to work with conda-forge, and our mooseframework.org channel:

```
$ conda config --add channels conda-forge
$ conda config --add channels https://mooseframework.org/conda/moose
```

- Install the moose-libmesh and moose-tools package from mooseframework.org to the moose environment

```
$ conda create --name moose moose-libmesh moose-tools
```

- Activate MOOSE environment

```
[long@Helix:runs]$ conda activate moose
(moose) [long@Helix:runs]$
```

# 3b. Manage Conda MOOSE Environment

- Update Conda MOOSE environment

```
$ conda update --all
```

- Deactivate Conda MOOSE environment

```
$ conda deactivate
```

- Uninstall Conda MOOSE environment

```
$ conda remove --name moose -all
```

# 4. Install MOOSE Framework

- Prepare a working folder

```
$ mkdir ~/projects
$ cd ~/projects
```

- MOOSE is hosted on GitHub and should be cloned directly from there using git

```
(moose) [long@Helix:moose]$ git clone https://github.com/idaholab/moose.git
Cloning into 'moose'...
remote: Enumerating objects: 85, done.
remote: Counting objects: 100% (85/85), done.
remote: Compressing objects: 100% (78/78), done.
remote: Total 432535 (delta 32), reused 16 (delta 5), pack-reused 432450
Receiving objects: 100% (432535/432535), 317.69 MiB | 1.56 MiB/s, done.
Resolving deltas: 100% (329145/329145), done.
Updating files: 100% (25089/25089), done.
(moose) [long@Helix:moose]$
```

- Switch to a master branch

```
$ cd moose
$ git checkout master
```

# 5. Compile and Run a Test

- Compile

```
$ cd ~/projects/moose/test
$ make -j 4
```

- Run

```
$ ./run_tests -j 4
```

- A successful installation will present

```
time_integrators/convergence.implicit_astabledirk4_bootstrap/level0 ..... OK
time_integrators/convergence.implicit_astabledirk4_bootstrap/level1 ..... OK
time_integrators/convergence.implicit_astabledirk4_bootstrap/level2 ..... OK
mesh/nemesis.nemesis_repartitioning_test ..... [min_cpus=4] OK
outputs/nemesis.nemesis_elemental_replicated ..... [min_cpus=4] OK
outputs/nemesis.nemesis_scalar_replicated ..... [min_cpus=4] OK
partitioners/petsc_partitioner.ptscotch_weight_elment ..... [min_cpus=4] OK
partitioners/petsc_partitioner.ptscotch_weight_side ..... [min_cpus=4] OK
partitioners/petsc_partitioner.ptscotch_weight_both ..... [min_cpus=4] OK
partitioners/petsc_partitioner.parmetis ..... [min_cpus=4] OK
partitioners/petsc_partitioner.parmetis_weight_element ..... [min_cpus=4] OK
partitioners/petsc_partitioner.parmetis_weight_side ..... [min_cpus=4] OK
partitioners/petsc_partitioner.parmetis_weight_both ..... [min_cpus=4] OK
partitioners/petsc_partitioner.parmetis_presplit_mesh ..... [min_cpus=2] OK
--------------------------------------------------------------------------------
Ran 2333 tests in 462.2 seconds.
2333 passed, 32 skipped, 0 pending, 0 failed
```

# Create your own MOOSE application

# Create an Application

- MOOSE is designed for building custom applications
- Go out of the MOOSE repository (~/projects/moose)

```
$ cd ~/projects
```

- Run stock.sh to

```
$ ./moose/scripts/stork.sh panda
```

  - A folder of the application (panda) will be created.
  - The application will automatically link against MOOSE.

- Compile and Test Your Application

```
$ cd panda
$ make -j 4
$ ./run_tests
```

- Update MOOSE (weekly)

```
$ cd ~/projects/moose
$ git fetch orgin
$ git rebase orgin/master
```

# Application directory structure

```
(moose) [long@Helix:panda]$ ls -l
total 120
-rw-r--r--   1 long   staff   26530 Jun   9 04:28 LICENSE
-rw-r--r--   1 long   staff    2092 Jun   9 04:28 Makefile
-rw-r--r--   1 long   staff     179 Jun   9 04:28 README.md
drwxr-xr-x   4 long   staff     128 Aug 10 17:34 build
drwxr-xr-x   5 long   staff     160 Jun   9 04:28 doc
drwxr-xr-x   4 long   staff     128 Aug 10 17:22 include
drwxr-xr-x   6 long   staff     192 Aug 10 18:11 lib
-rwxr-xr-x   1 long   staff   14812 Aug 10 18:11 panda-opt
drwxr-xr-x   4 long   staff     128 Aug 10 18:22 problems
-rwxr-xr-x   1 long   staff     468 Jun   9 04:28 run_tests
drwxr-xr-x   3 long   staff      96 Jun   9 04:28 scripts
drwxr-xr-x   8 long   staff     256 Aug 10 17:21 src
drwxr-xr-x   6 long   staff     192 Jun   9 04:29 test
-rw-r--r--   1 long   staff      84 Jun   9 04:28 testroot
drwxr-xr-x   6 long   staff     192 Jun   9 04:28 unit
```

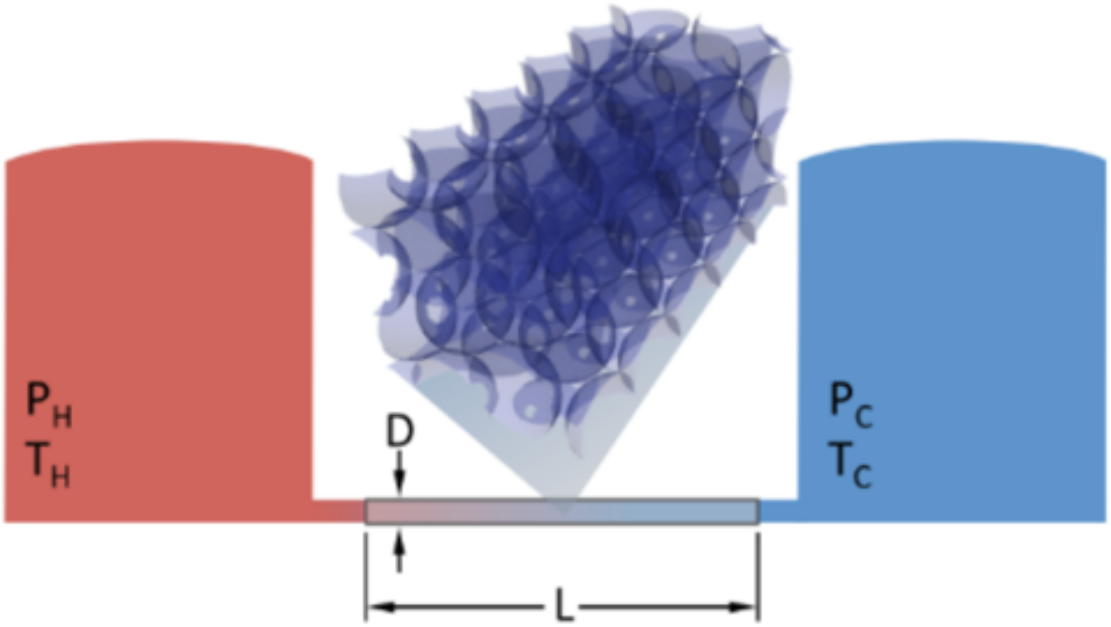
```
application/
    LICENSE
    Makefile
    run_tests
    doc/
    lib/
    src/
        main.C
        base/
        actions/
        auxkernels/
        bcs/
        dampers/
        dirackernels/
        executioners/
        functions/
        ics/
        kernels/
        materials/
        postprocessors/
        utils/
    tests/
        ... (same stuff as src)
```

# An Advanced Example: Darcy Flow

# Problem Statement

- Consider a system containing two pressure vessels at differing temperatures. The vessels are connected via a pipe that contains a filter consisting of close-packed steel spheres. Predict the velocity and temperature of the fluid inside the filter.

D=0.0514 m
L = 0.304 m




Temp = ?
Velocity = ?

Mimics an experimental setup of Pamuk and Ozdemir, "Friction factor, permeability, and inertial coefficient of oscillating flow through porous media of packed balls", Experimental Thermal and Fluid Science, v. 38, pp. 134-139, 2012.

# Model: Governing Equations

- Conservation of Mass

$$\nabla \cdot \boldsymbol{u} = 0$$

- Conservation of Energy

$$C\left(\frac{\partial T}{\partial t} + \epsilon \boldsymbol{u} \cdot \nabla T\right) - \nabla \cdot (k \nabla T) = 0$$

- Darcy's Law

$$\boldsymbol{u} = -\frac{\boldsymbol{K}}{\mu}(\nabla p - \rho \boldsymbol{g})$$

- Variables and parameters
  - Fluid velocity: $\boldsymbol{u}$ heat capacity: $C$ heat conductivity: $k$
  - Temperature: $T$ porosity: $\epsilon$ permeability tensor: $\boldsymbol{K}$
  - Pressure: $p$ density: $\rho$ gravity: $\boldsymbol{g}$

# Simplified Governing Equations when g=0

- Neglect gravity, $\boldsymbol{g} = 0$

$$-\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla p = 0$$

$$C\left[\frac{\partial T}{\partial t} - \epsilon\left(\frac{\boldsymbol{K}}{\mu}\nabla p\right) \cdot \nabla T\right] - \nabla \cdot (k \nabla T) = 0$$

They are still basically diffusion equations!

$$-\nabla \cdot \nabla u = 0 \in \Omega,$$

- Two unknowns: $p$ $T$

- Velocity is also unknown but:

$$\boldsymbol{u} = -\frac{\boldsymbol{K}}{\mu} \nabla p$$

# Porosity-dependent parameters

- The parameters $\rho$ , $C$ , and $k$ are the porosity-dependent density, heat capacity, and thermal conductivity of the combined fluid/solid medium, defined by

$$\rho \equiv \epsilon \rho_f + (1 - \epsilon)\rho_s$$

$$C \equiv \epsilon \rho_f c_{p_f} + (1 - \epsilon)\rho_s c_{p_s}$$

$$k \equiv \epsilon k_f + (1 - \epsilon)k_s$$

- where $\epsilon$ is the porosity, $c_p$ is the specific heat, and the subscripts f and s refer to fluid and solid, respectively.

# Material Properties

- Will be used in the simulation

| Property | Value | Units |
|---|---|---|
| Viscosity of water, $\mu_f$ | $7.98 \times 10^{-4}$ | $\mathrm{P \cdot s}$ |
| Density of water, $\rho_f$ | 995.7 | $\mathrm{kg/m^3}$ |
| Density of steel, $\rho_s$ | 8000 | $\mathrm{kg/m^3}$ |
| Thermal conductivity of water, $k_f$ | 0.6 | $\mathrm{W/m\,K}$ |
| Thermal conductivity of steel, $k_s$ | 18 | $\mathrm{W/m\,K}$ |
| Specific heat capacity of water, $c_{pf}$ | 4181.3 | $\mathrm{J/(kg\,K)}$ |
| Specific heat capacity of steel, $c_{ps}$ | 466 | $\mathrm{J/(kg\,K)}$ |

# What's next?

- Everything is ready except for a solver to PDEs
- MOOSE!
- How to use MOOSE?
- /moose/tutorials/darcy_thermo_mech

# Tutorial Steps

- Step 1: Geometry and Diffusion
- Step 2: Pressure Kernel
- Step 3: Pressure Kernel with Material
- Step 4: Velocity Auxiliary Variable
- (Step 5: Heat Conduction)
- (Step 6: Equation Coupling)
- Step 7: Mesh Adaptivity
- Step 8: Postprocessors
- Step 9: Mechanics
- Step 10: Multiscale Simulation
- Step 11: Custom Syntax

# Step 1: Geometry and Diffusion

- Solve a simple but standard diffusion problem, which requires no manual coding but has been implemented by MOOSE

$$-\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla p = 0$$

$$-\nabla \cdot \nabla p = 0$$

# Step 2: Pressure Kernel

- Implement real Darcy Pressure equation, but not simple diffusion equation
- Need to implement a MOOSE **`Kernel`** object

$$-\nabla \cdot \nabla p = 0$$

$$-\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla p = 0$$

# Step 3: Pressure Kernel with Material

- What if parameters are not constant to the pressure **`Kernel`** diffusion object?
  - Equation is still the same

$$-\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla p = 0$$

- A **`Material`** system can be used to supply the properties parameters that can vary in space and time, as well as be coupled to variables in the simulation

$$\boldsymbol{K}(r)$$

# Step 4: Velocity Auxiliary Variable

- The velocity doesn't present explicitly in the governing equations, so is not directly computed during the simulation

$$-\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla p = 0$$

- It can be computed by using computed pressure

$$\boldsymbol{u} = -\frac{\boldsymbol{K}}{\mu} \nabla p$$

- **`Auxiliary`** System in MOOSE can do this work.

# Step 5: Heat Conduction

- Solve the transient heat equation using the “heat conduction” module

$$C\left[\frac{\partial T}{\partial t}-\epsilon(\frac{\boldsymbol{K}}{\mu}\nabla p)\cdot\nabla T\right]-\nabla\cdot(k\nabla T)=0$$

$$C\frac{\partial T}{\partial t}-\nabla\cdot(k\nabla T)=0$$

# Step 6: Equation Coupling

- Couple the diffusion term and advection term to the heat equation in the same system

$$-\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla p = 0$$

$$C\left[\frac{\partial T}{\partial t} - \epsilon\left(\frac{\boldsymbol{K}}{\mu}\nabla p\right) \cdot \nabla T\right] - \nabla \cdot (k \nabla T) = 0$$

# Step 1: Geometry and Diffusion

# Step 1: Geometry and Diffusion

- Solve a simple diffusion problem, which requires no manual coding but has been implemented by MOOSE

$$-\nabla \cdot \nabla p = 0$$

  - In PDEs, we can use $u$ to represent the unknown variable on the simulation domain $\Omega$
- The steady-state diffusion equation:

$$-\nabla \cdot \nabla u = 0 \quad \Rightarrow \quad (\nabla \psi_i, \nabla u_h) = 0 \qquad \forall \psi_i$$

  - $\psi_i$ are the test functions and $u_h$ is the finite element solution
- BCs (2D):
  - On the left: $u = 0$ on the right: $u = 4000$
  - On the remaining boundaries (natural BC):

$$\nabla u \cdot \hat{\boldsymbol{n}} = 0 \quad \Rightarrow \quad \langle \psi_i, \nabla u_h \cdot \hat{\boldsymbol{n}} \rangle = 0 \qquad \forall \psi_i$$

# Input files

- All MOOSE capabilities and customized application are compiled into a single executable

- An input file is used to select necessary capabilities to perform a simulation
  - Which equation? Heat conduction? Tensor Mechanics?...

- "hierarchical input text" format is used in MOOSE

- Example
  - /moose/tutorials/darcy_thermo_mech/step01_diffusion/problems/step1.i

# At least six blocks are needed

- [Mesh]: define the geometry of the domain

```
[Mesh]
  type = GeneratedMesh # Can generate simple lines, rectangles and rectangular prisms
  dim = 2              # Dimension of the mesh
  nx = 100             # Number of elements in the x direction
  ny = 10              # Number of elements in the y direction
  xmax = 0.304         # Length of test chamber
  ymax = 0.0257        # Test chamber radius
[]
```

- [Variables]: Define the unknown(s) of the problem

```
[Variables]
  [pressure]
    # Adds a Linear Lagrange variable by default
  []
[]
```

- [Kernels]: Define the equation(s) to solve
- [BCs]: Define the boundary condition(s) to solve
- [Executioner]: Define how the problem will be solved
- [Outputs]: Define how the solution will be output

```
[Outputs]
  exodus = true # Output Exodus format
[]
```

# Input files

$$-\nabla \cdot \nabla u = 0$$

$$(\nabla \psi_i, \nabla u_h) = 0 \qquad \forall \psi_i$$

- [Kernels]:

```
[Kernels]
  [diffusion]
    type = ADDiffusion  # Laplacian operator using automatic differentiation
    variable = pressure # Operate on the "pressure" variable from above
  []
[]
```

- 1-sentence FEM principle (practical)
  - FEM solves PDEs by forming the Jacobian Matrix of the (PDE's) Residual vector $\bar{R}(\bar{u}_n)$
- MOOSE's FEM
  - Kernel method computeQpResidual is called to compute $\bar{R}(\bar{u}_n)$
  - Automatic Differentiation (AD) from **`MetaPhyicL`** package is introduced to do Automatic Jacobian Calculation – recommended for beginners.
    - We can develop entire apps without writing a single J statement.

# ADDiffusion.h

- moose/framework/include/kernels

```
#pragma once

#include "ADKernelGrad.h"

class ADDiffusion : public ADKernelGrad
{
public:
  static InputParameters validParams();

  ADDiffusion(const InputParameters & parameters);

protected:
  virtual ADRealVectorValue precomputeQpResidual() override;
};
```

# ADDiffusion.C

- moose/framework/src/kernels

```
#include "ADDiffusion.h"

registerMooseObject("MooseApp", ADDiffusion);

InputParameters
ADDiffusion::validParams()
{
  auto params = ADKernelGrad::validParams();
  params.addClassDescription("Same as `Diffusion` in terms of physics/residual, but the Jacobian "
                             "is computed using forward automatic differentiation");
  return params;
}

ADDiffusion::ADDiffusion(const InputParameters & parameters) : ADKernelGrad(parameters) {}

ADRealVectorValue
ADDiffusion::precomputeQpResidual()
{
  return _grad_u[_qp];
}
```

$$(\nabla\psi_i, \nabla u_h) = 0 \qquad \forall\psi_i$$

# Input files

- [Executioner]

```
[Executioner]
  type = Steady         # Steady state problem
  solve_type = NEWTON # Perform a Newton solve, uses AD to compute Jacobian terms
  petsc_options_iname = '-pc_type -pc_hypre_type' # PETSc option pairs with values below
  petsc_options_value = 'hypre boomeramg'
[]
```

- Available options include:
  - **PJFNK**: Preconditioned Jacobian Free Newton Krylov (default)
    - improves convergence
  - **JFNK**: Jacobian Free Newton Krylov
    - approximates a Jacobian vector product
  - **NEWTON**: Performs solve using exact Jacobian for preconditioning
  - **FD**: PETSc computes terms using a Finite Difference method (debug)

# Run it

- Go to moose/tutorials/darcy_thermo_mech/step01_diffusion

```
[long@Helix:step01_diffusion]$ conda activate moose
(moose) [long@Helix:step01_diffusion]$ make
(moose) [long@Helix:step01_diffusion]$ cd problems
(moose) [long@Helix:problems]$ lt
total 608
-rw-r--r--  1 long  staff   1586 Jun  9 04:15 step1.i
-rwxr-xr-x  1 long  staff   1152 Jun  9 04:15 step1.py
-rw-r--r--  1 long  staff    248 Jun  9 04:15 tests
(moose) [long@Helix:problems]$
```

- Method 1: Peacock

```
$ ~/projects/moose/python/peacock/peacock -i step1.i
```

- Method 2:

```
$ ../darcy_thermo_mech-opt -i step1.i
```

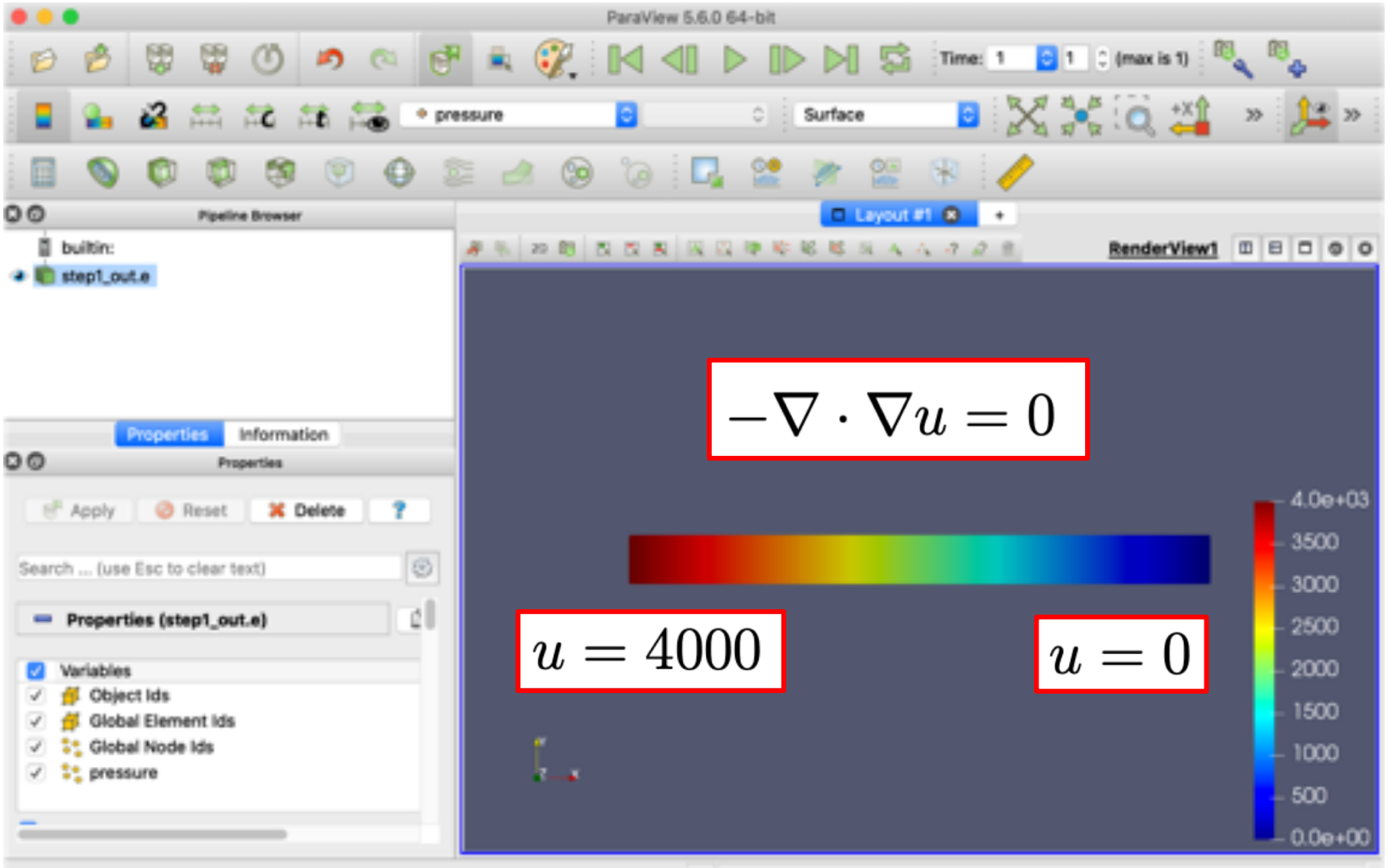
- Method 3: MPI

```
$ mpiexec -n 4 ../darcy_thermo_mech-opt -i step1.i
```

# Visualize it

- Peacock or Paraview

```
$ ~/projects/moose/python/peacock/peacock -r step1_out.e
```



# Step 2: (Customized) Pressure Kernel

# Step 2: (Customized) Pressure Kernel

- Implement real Darcy Pressure equation, but not simple one

$$-\nabla \cdot \nabla p = 0 \Rightarrow -\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla p = 0 \Rightarrow \left(\nabla \psi_i, \frac{\boldsymbol{K}}{\mu} \nabla p\right) - \langle \psi_i, \frac{\boldsymbol{K}}{\mu} \nabla p \cdot \hat{\boldsymbol{n}} \rangle = 0$$

- Need to implement a new customized **`Kernel`** object
- **`Kernel`** is a C++ class, which inherits from **`MooseObject`**
- **`MooseObject`**
  - is used for computing volume integrals of PDEs.
  - allows common structure for all applications
  - basis for the modular design in MOOSE
- A **`Kernel`** is a piece of physics, or an operator in a PDE
  - must override computeQpResidual() to change physics

# Write a customized Pressure Kernel

- **`Kernel`** class can be derived into header (.h) and source (.c)
- Header:
  - Declaration
  - step02_darcy_pressure/include/kernels/DarcyPressure.h
- Source:
  - Definition
  - step02_darcy_pressure/src/kernels/DarcyPressure.C

# Header in Pressure Kernel

- DarcyPressure.h

```
#pragma once

// Including the "ADKernel" Kernel here so we can extend it
#include "ADKernel.h"

/**
 * Computes the residual contribution: K / mu * grad_u * grad_phi.
 */
class DarcyPressure : public ADKernel
{
public:
  static InputParameters validParams();

  DarcyPressure(const InputParameters & parameters);

protected:
  /// ADKernel objects must override precomputeQpResidual
  virtual ADReal computeQpResidual() override;

  /// References to be set from input file
  const Real & _permeability;
  const Real & _viscosity;
};
```

$$(\nabla\psi_i, \frac{\boldsymbol{K}}{\mu}\nabla p)$$

DarcyPresure Object is to be created using input parameters

InputParameters Object is created using validParams method

computeQpResidual is called to compute $\bar{R}(\bar{u}_n)$

# Source

- .C

```cpp
#include "DarcyPressure.h"

registerMooseObject("DarcyThermoMechApp", DarcyPressure);

InputParameters
DarcyPressure::validParams()
{
  InputParameters params = ADKernel::validParams();
  params.addClassDescription("Compute the diffusion term for Darcy pressure ($p$) equation: "
                             "$-\\nabla \\cdot \\frac{\\mathbf{K}}{\\mu} \\nabla p = 0$");

  // Add a required parameter.  If this isn't provided in the input file MOOSE will error.
  params.addRequiredParam<Real>("permeability", "The permeability ($\\mathrm{K}$) of the fluid.");

  // Add a parameter with a default value; this value can be overridden in the input file.
  params.addParam<Real>(
      "viscosity",
      7.98e-4,
      "The viscosity ($\\mu$) of the fluid in Pa, the default is for water at 30 degrees C.");
  return params;
}

DarcyPressure::DarcyPressure(const InputParameters & parameters)
  : ADKernel(parameters),

    // Get the parameters from the input file
    _permeability(getParam<Real>("permeability")),
    _viscosity(getParam<Real>("viscosity"))
{
}

ADReal
DarcyPressure::computeQpResidual()
{
  return (_permeability / _viscosity) * _grad_test[_i][_qp] * _grad_u[_qp];
}
```

Kernel should be registered

Supported parameters: Real, int, RealVectorValue, RealTensorValue…

```
[Kernels]
  [darcy_pressure]
    type = DarcyPressure
    variable = pressure
    permeability = 0.8451e-9 # (m^2) 1mm spheres
  []
[]
```

$$\left(\nabla \psi_i, \frac{\boldsymbol{K}}{\mu} \nabla p\right)$$

# Input file for the new Kernel

- See darcy_thermo_mech/step02_darcy_pressure/problems
- step2.i

```
[Kernels]
  [darcy_pressure]
    type = DarcyPressure
    variable = pressure
    permeability = 0.8451e-9 # (m^2) 1mm spheres.
  []
[]
```

# Kernel Object Members

- _u, _grad_u:
  - Value and its gradient this Kernel is operating on.
- _phi, _grad_phi:
  - Value ( $\phi$ ) and gradient ( $\nabla\phi$ ) of the trial functions at the quadrature points.
- _test, _grad_test:
  - Value ( $\psi$ ) and gradient ( $\nabla\psi$ ) of the test functions at the q-points.
- _q_point:
  - coordinates of the current quadrature point.
- _i, _j:
  - Current index for test and trial functions, respectively.
- _qp:
  - Current quadrature point index.
- _current_elem:
  - A pointer to the current element that is being operated on.

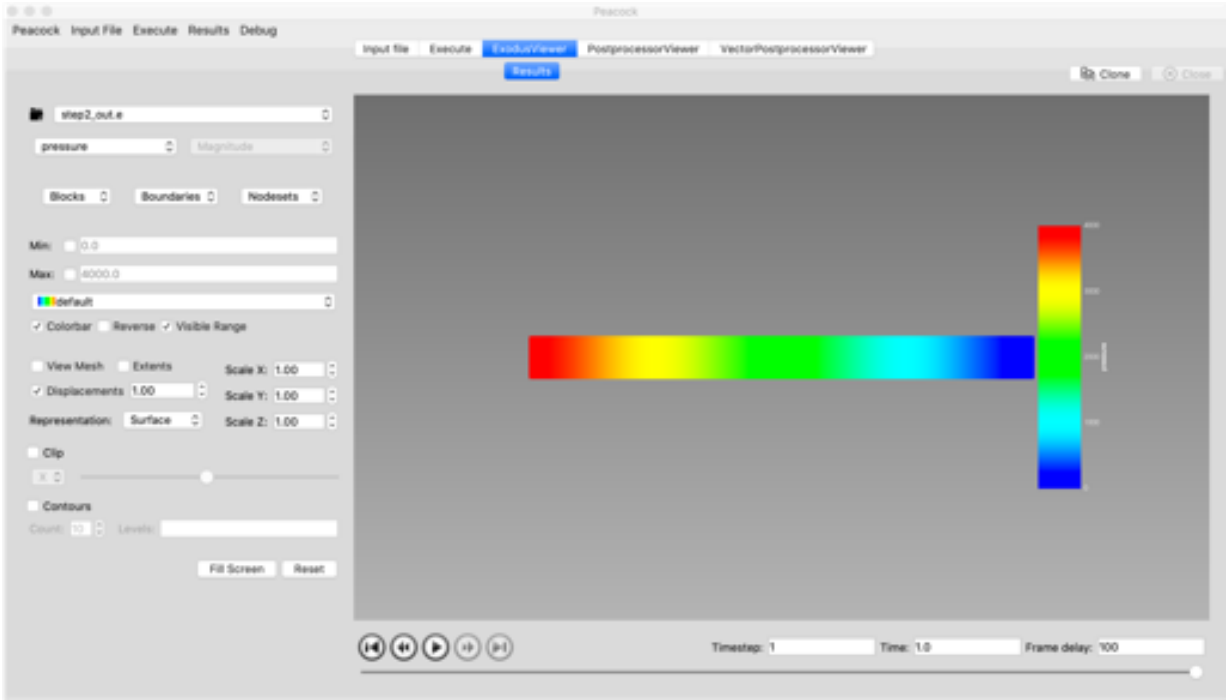
# Run it and Visualize it

- Run

```
$ ../darcy_thermo_mech-opt -i step2.i
```

- Visualize

```
$ ~/projects/moose/python/peacock/peacock -r step2_out.e
```



# Exercise: Output System

- Output system is modular and expandable
  - Create multiple output objects
    - at specific time
    - to custom subsets of variables
    - to various file types
- Short-cut syntax

```
[Outputs]
  exodus = true
[]
```

```
[Outputs]
  [out]
    type = exodus
  []
[]
```

# Exercise: Output System

- Common Parameters

```
[Outputs]
  interval = 10
  exodus = true
  [all]
    type = exodus
    interval = 1  # overrides interval from top-level
  []
[]
```

# Exercise: Output System

- Output Names
  - Default naming scheme: “input_file_name_out”
  - Sub-blocks can be used to retrieve information at different time
  - “file_base” variable can override all default naming

```
[Outputs]
  exodus = true    # create step2_out.e in this case
  [other]          # create step2_other.e
    type = exodus
    interval = 2
  []
  [base]
    type = exodus
    file_base = out # simply creates out.e
  []
[]
```

# Step 3: Pressure Kernel with Material

# Step 3: Pressure Kernel with Material

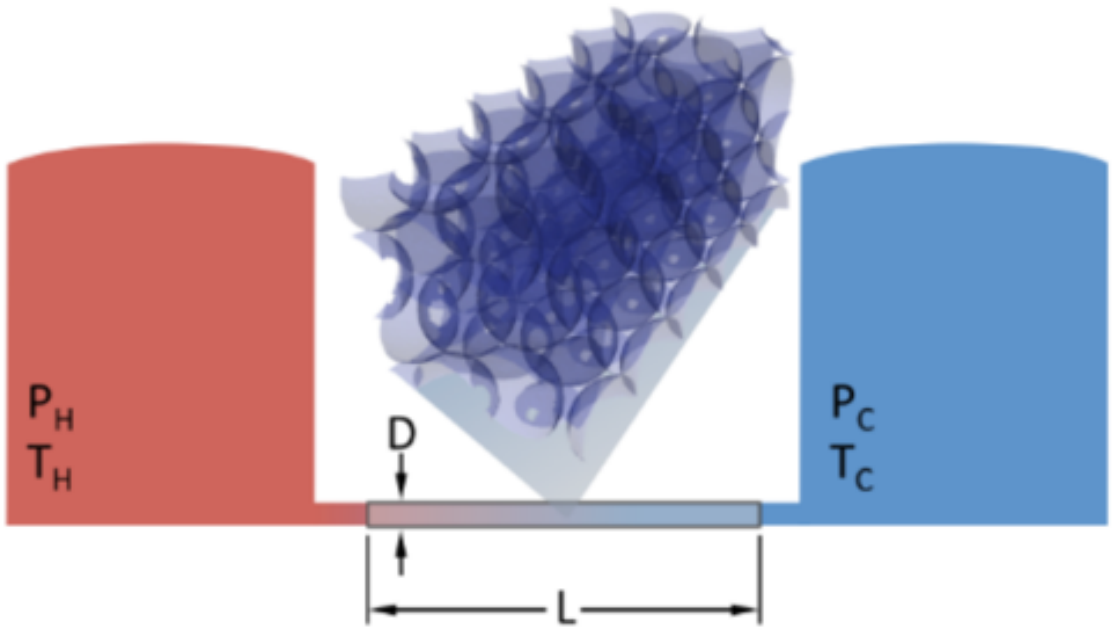
- What if parameters are not constant to the pressure **`Kernel`** diffusion object?
  - Example, if permeability is not constant, but a function of sphere size?

$$-\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla p = 0$$

- A **`Material`** system can be used to supply the properties parameters that can vary in space and time, as well as be coupled to variables in the simulation
  - **`Material`** Object produce properties
  - Other MOOSE objects use properties

# Step 3: Pressure Kernel with Material

- Example: a “PackedColumn” Material
  - Permeability (varies with sphere size, need interpolation)
  - Viscosity
  - Computed with a **`Material`** Object: PackedColumn




```
[Materials]
  [column]
    type = PackedColumn
  []
[]
```

```
[Materials]
  [column]
    type = PackedColumn
    radius = '1 + 2/3.04*x'
    outputs = exodus
  []
[]
```

# Produce and Consume properties

- Each property to be produced must be declared
  - declareProperty<TYPE>()
  - declareADProperty<TYPE>()
    - For properties associated with AD system
    - Use within AD Objects to create AD material properties.
  - Override computeQpProperties() to computer the declared properties at quadrature points.
- Get produced material properties
  - getMaterialProperty<TYPE>()
  - getADMaterialProperty<TYPE>()
- (stateful) material properties
  - getMaterialPropertyOld<TYPE>(): at 1 time step earlier
  - getMaterialPropertyOlder<TYPE>(): at 2 time steps earlier

# Supported Output Types

- Material properties can be of arbitrary (C++) type, but not all types can be output.
- The following is a list of the Material property types that support automatic output.

| Type | AuxKernel | Variable Name(s) |
|---|---|---|
| Real | MaterialRealAux | prop |
| RealVectorValue | MaterialRealVectorValueAux | prop_1, prop_2, and prop_3 |
| RealTensorValue | MaterialRealTensorValueAux | prop_11, prop_12, prop_13, prop_21, etc. |

# Function System

- A system for defining analytical expressions based on the spatial location (x, y, z) and time t.
- A specific **`FunctionName`** object is created by inheriting from **`Function`** class and override virtual value() function
- Functions can be accessed in most MOOSE objects by calling getFunction("name").
- Many MOOSE objects utilize functions:
  - **`FunctionDirichletBC`**
  - **`FunctionNeumannBC`**
  - **`FunctionIC`**
  - Object has a function parameter set in the input file

# PackedColumn.h

- Go to step03_darcy_material/include/materials

```
class PackedColumn : public Material
{
public:
  static InputParameters validParams();

  PackedColumn(const InputParameters & parameters);

protected:
  /// Necessary override. This is where the values of the properties are computed.
  virtual void computeQpProperties() override;

  /// The radius of the spheres in the column
  const Function & _radius;

  /// Value of viscosity from the input file
  const Real & _input_viscosity;

  /// Compute permeability based on the radius (mm)
  LinearInterpolation _permeability_interpolation;

  /// The permeability (K)
  ADMaterialProperty<Real> & _permeability;

  /// The viscosity of the fluid (mu)
  ADMaterialProperty<Real> & _viscosity;
};
```

# PackedColumn.C

```
#include "PackedColumn.h"
#include "Function.h"

registerMooseObject("DarcyThermoMechApp", PackedColumn);

InputParameters
PackedColumn::validParams()
{
  InputParameters params = Material::validParams();

  // Parameter for radius of the spheres used to interpolate permeability.
  params.addParam<FunctionName>("radius",
                                "1.0",
                                "The radius of the steel spheres (mm) that are packed in the "
                                "column for computing permeability.");
  params.addParam<Real>(
      "viscosity",
      7.98e-4,
      "The viscosity ($\\mu$) of the fluid in Pa, the default is for water at 30 degrees C.");

  return params;
}
```

# PackedColumn.C

```
[Materials]
  [column]
    type = PackedColumn
    radius = '1 + 2/3.04*x'
    outputs = exodus
  []
[]
```

```
PackedColumn::PackedColumn(const InputParameters & parameters)
  : Material(parameters),

    // Get the parameters from the input file
    _radius(getFunction("radius")),
    _input_viscosity(getParam<Real>("viscosity")),

    // Declare two material properties by getting a reference from the MOOSE Material system
    _permeability(declareADProperty<Real>("permeability")),
    _viscosity(declareADProperty<Real>("viscosity"))
{
  // From the paper: Table 1
  std::vector<Real> sphere_sizes = {1, 3};
  std::vector<Real> permeability = {0.8451e-9, 8.968e-9};

  // Set the x,y data on the LinearInterpolation object.
  _permeability_interpolation.setData(sphere_sizes, permeability);
}

void
PackedColumn::computeQpProperties()
{
  Real value = _radius.value(_t, _q_point[_qp]);
  mooseAssert(value >= 1 && value <= 3,
              "The radius range must be in the range [1, 3], but " << value << " provided.");

  _viscosity[_qp] = _input_viscosity;
  _permeability[_qp] = _permeability_interpolation.sample(value);
}
```

# DarcyPressure Kernel must be updated

- New DarcyPressure.h (include/kernels/DarcyPressure.h)

```
protected:
  /// ADKernel objects must override precomputeQpResidual
  virtual ADReal computeQpResidual() override;

  // References to be set from Material system

  /// The permeability. Note that this is declared as a \p MaterialProperty. This means that if
  /// calculation of this property in the producing \p Material depends on non-linear variables, the
  /// derivative information will be lost here in the consumer and the non-linear solve will suffer
  const ADMaterialProperty<Real> & _permeability;

  /// The viscosity. This is declared as an \p ADMaterialProperty, meaning any derivative
  /// information coming from the producing \p Material will be preserved and the integrity of the
  /// non-linear solve will be likewise preserved
  const ADMaterialProperty<Real> & _viscosity;
```

- Old DarcyPressure.h (in step 02)

```
  /// References to be set from input file
  const Real & _permeability;
  const Real & _viscosity;
```

# DarcyPressure Kernel must be updated

- New DarcyPressure.C (src/kernels/DarcyPressure.C)

```
    _permeability(getADMaterialProperty<Real>("permeability")),
    _viscosity(getADMaterialProperty<Real>("viscosity"))
{
}

ADReal
DarcyPressure::computeQpResidual()
{
  return (_permeability[_qp] / _viscosity[_qp]) * _grad_test[_i][_qp] * _grad_u[_qp];
}
```

- Old DarcyPressure.C (in step 02)

```
    // Get the parameters from the input file
    _permeability(getParam<Real>("permeability")),
    _viscosity(getParam<Real>("viscosity"))
{
}

ADReal
DarcyPressure::computeQpResidual()
{
  return (_permeability / _viscosity) * _grad_test[_i][_qp] * _grad_u[_qp];
}
```

# Input file

- Add a material block
- step03_darcy_material/problems/step3.i

```
[Materials]
  [column]
    type = PackedColumn
  []
[]
```

- Variable sphere size: step03_darcy_material/problems/step3b.i

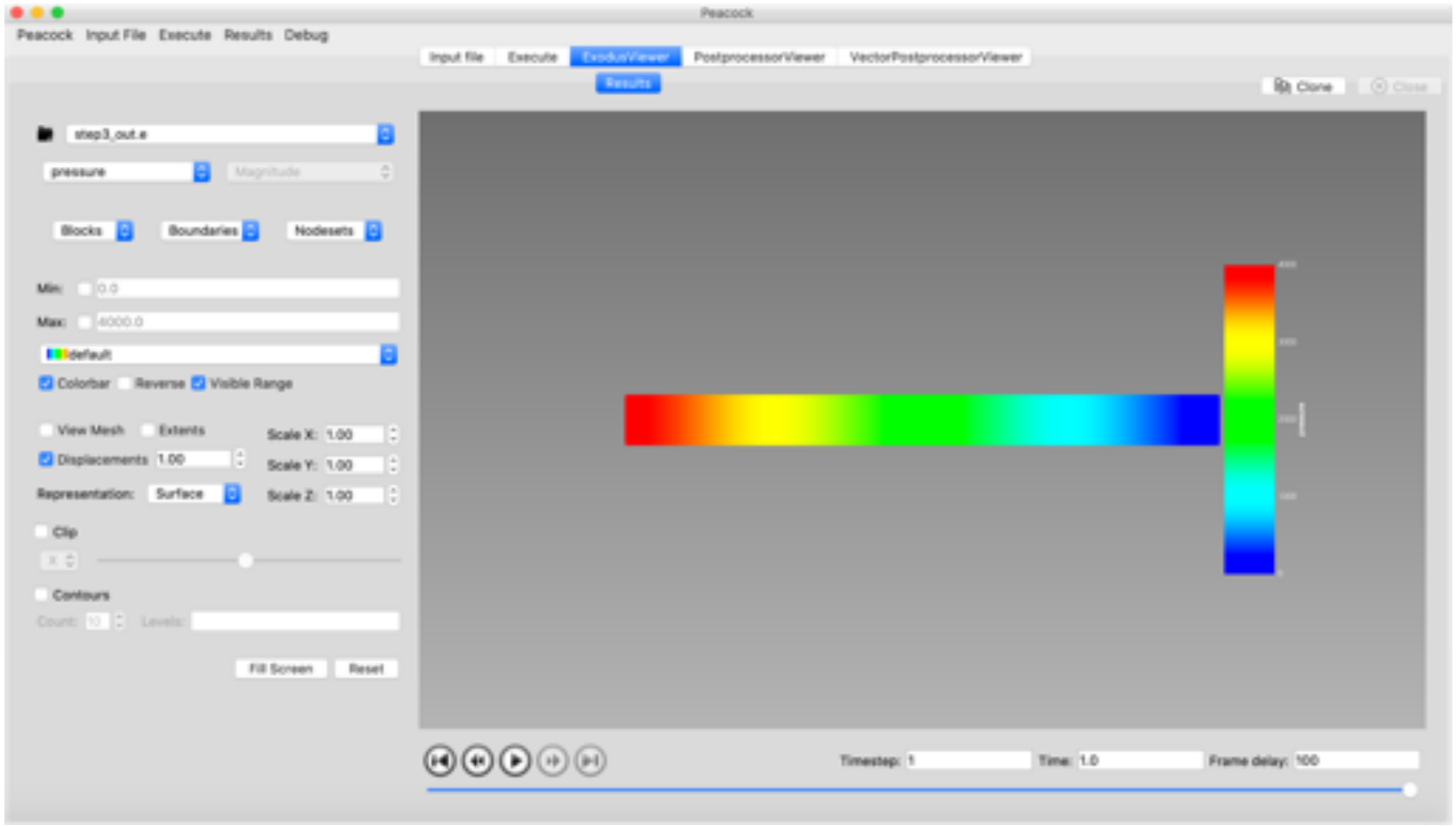
```
[Materials]
  [column]
    type = PackedColumn
    radius = '1 + 2/3.04*x'
    outputs = exodus
  []
[]
```

# Run it and Visualize it

- Commands

```
$ ../darcy_thermo_mech-opt -i step3.i
```

```
$ ~/projects/moose/python/peacock/peacock -r step3_out.e
```



# Step 4: Velocity Auxiliary Variable

# Step 4: Velocity Auxiliary Variable

- The velocity doesn't present explicitly in the governing equations, so is not directly computed during the simulation

$$-\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla p = 0$$

- If the velocity is the primary variable of interest, it can be computed by using computed pressure

$$\boldsymbol{u} = -\frac{\boldsymbol{K}}{\mu} \nabla p$$

- **`Auxiliary`** System in MOOSE can do this work.

# Auxiliary System

- For direct calculation of field variables ("AuxVaiables") that is designed for postprocessing, coupling and proxy calculations.
- Nonlinear Variable
  - A variable that is being solved in nonlinear system of PDEs using **`Kernel`** and **`BoundaryCondition`** Objects
- Auxiliary variable
  - A variable that is directly calculated using an **`AuxKernel`** object
  - Declared in [**`AuxVariables`**] input file block
  - Can serve as a proxy for nonlinear variables

$$C\left(\frac{\partial T}{\partial t}+\epsilon \boldsymbol{u}\cdot\nabla T\right)-\nabla\cdot(k\nabla T)=0$$

# Auxiliary Variables

- Element auxiliary variables
  - Compute average values per element (so, constant)

```
[AuxVariables]
  [velocity_x]
    order = CONSTANT
    family = MONOMIAL
  []
```

  - **AuxKernel** object computing element auxiliary variables can couple to both element and nodal **AuxVariables** and nonlinear variables
- Nodal auxiliary variables
  - Computed at each node and are stored as linear Lagrange variables

```
[AuxVariables]
  [aux]
    order = LAGRANGE
    family = FIRST
  []
```

  - **AuxKernel** object computing element auxiliary variables can only couple to nodal **AuxVariables** and nonlinear variables

# AuxKernel and VectorAuxKernel Objects

- Directly compute **`AuxVariable`** values by overriding computeValue()
- When operating on a nodal variable, computeValue() operates on each node
- When operating on a element variable, computeValue() operates on each element
- Difference
  - Return Real value or a RealVectorValue

```
[AuxVariables]
  [aux]
    order = LAGRANGE
    family = FIRST
  []
```

```
[AuxVariables]
  [aux]
    order = LAGRANGE_VEC
    family = FIRST
  []
```

# AuxKernel Object Members

- _u, _grad_u:
  - Value and its gradient this Kernel is operating on.
- _q_point:
  - coordinates of the current quadrature point that is only valid for elemental AuxKernels
- _qp:
  - Current quadrature point index.
- _current_elem:
  - Pointer to the current element that is being operated on.
- _current_node:
  - Pointer to the current node that is being operated on.

# New DarcyVelocity.h

- step04_velocity_aux/include/auxkernels

```
#pragma once

#include "AuxKernel.h"

/**
 * Auxiliary kernel responsible for computing the Darcy velocity given
 * several fluid properties and the pressure gradient.
 */
class DarcyVelocity : public VectorAuxKernel
{
public:
  static InputParameters validParams();

  DarcyVelocity(const InputParameters & parameters);

protected:
  /**
   * AuxKernels MUST override computeValue.  computeValue() is called on
   * every quadrature point.  For Nodal Auxiliary variables those quadrature
   * points coincide with the nodes.
   */
  virtual RealVectorValue computeValue() override;

  /// The gradient of a coupled variable
  const VariableGradient & _pressure_gradient;

  /// Holds the permeability and viscosity from the materi
  const ADMaterialProperty<Real> & _permeability;
  const ADMaterialProperty<Real> & _viscosity;
};
```

Override computeValue()

$$\boldsymbol{u} = -\frac{\boldsymbol{K}}{\mu}\nabla p$$

# New DarcyVelocity.C

- step04_velocity_aux/src/auxkernels

```
#include "DarcyVelocity.h"

#include "metaphysicl/raw_type.h"

registerMooseObject("DarcyThermoMechApp", DarcyVelocity);

InputParameters
DarcyVelocity::validParams()
{
  InputParameters params = VectorAuxKernel::validParams();

  // Add a "coupling paramater" to get a variable from the input file.
  params.addRequiredCoupledVar("pressure", "The pressure field.");

  return params;
}
```

# New DarcyVelocity.C

- step04_velocity_aux/src/auxkernels

```
DarcyVelocity::DarcyVelocity(const InputParameters & parameters)
  : VectorAuxKernel(parameters),

    // Get the gradient of the variable
    _pressure_gradient(coupledGradient("pressure")),

    // Set reference to the permeability MaterialProperty.
    // Only AuxKernels operating on Elemental Auxiliary Variables can do this
    _permeability(getADMaterialProperty<Real>("permeability")),

    // Set reference to the viscosity MaterialProperty.
    // Only AuxKernels operating on Elemental Auxiliary Variables can do this
    _viscosity(getADMaterialProperty<Real>("viscosity"))
{
}

RealVectorValue
DarcyVelocity::computeValue()
{
  // Access the gradient of the pressure at this quadrature point,
  // it requested (x, y or z). Note, that getting a particular component of a gradient is done using
  // the parenthesis operator.
  return MetaPhysicL::raw_value(_permeability[_qp] / _viscosity[_qp]) * _pressure_gradient[_qp];
}
```

Construct velocity object

raw_value: type casting from ADReal Value

# AuxKernels in step4.i

- step04_velocity_aux/problems

```
[AuxVariables]
  [velocity_x]
    order = CONSTANT
    family = MONOMIAL
  []
  [velocity_y]
    order = CONSTANT
    family = MONOMIAL
  []
  [velocity_z]
    order = CONSTANT
    family = MONOMIAL
  []
  [velocity]
    order = CONSTANT
    family = MONOMIAL_VEC
  []
[]
```

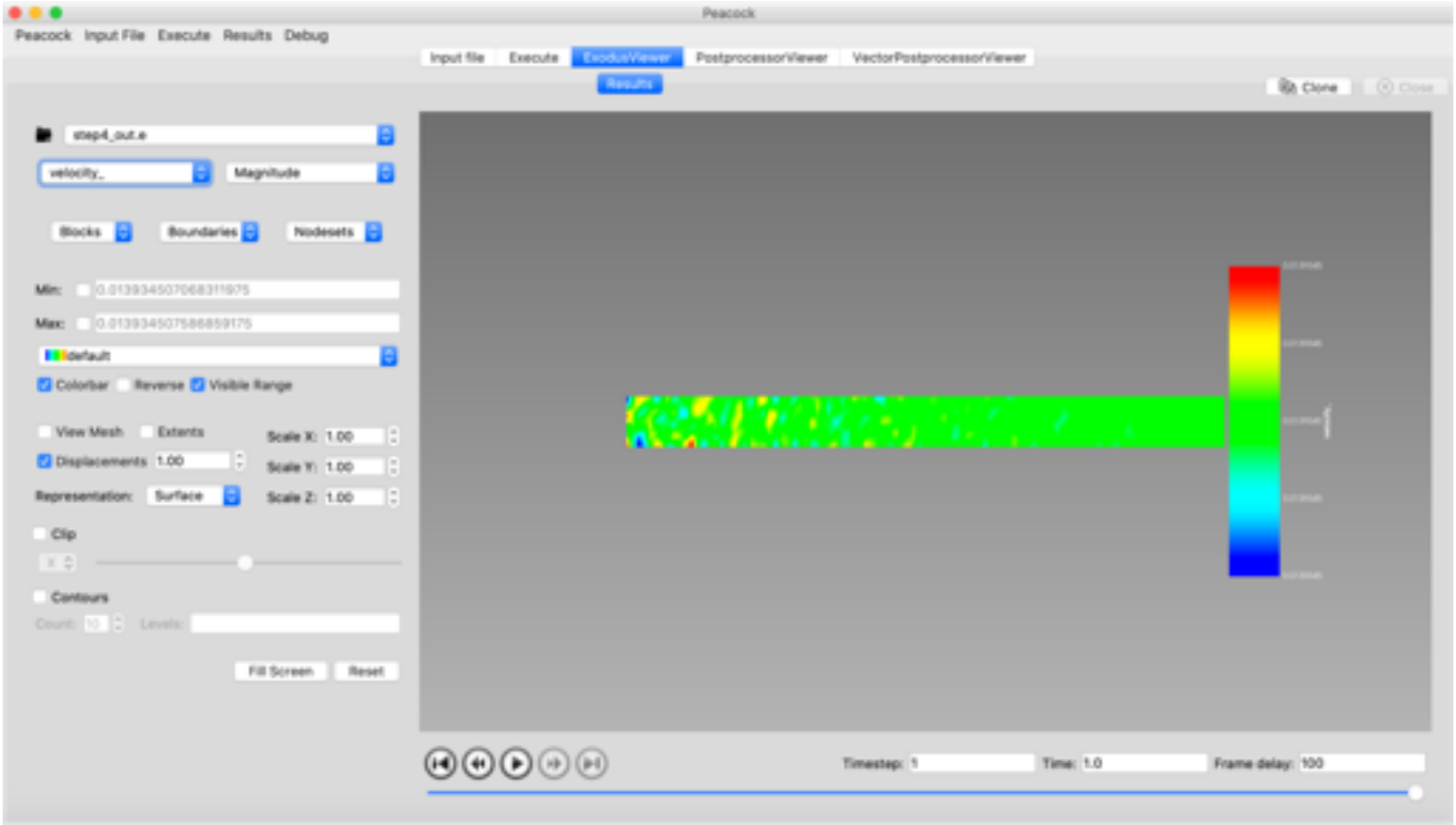
```
[AuxKernels]
  [velocity]
    type = DarcyVelocity
    variable = velocity
    execute_on = timestep_end
    pressure = pressure
  []
  [velocity_x]
    type = VectorVariableComponentAux
    variable = velocity_x
    component = x
    execute_on = timestep_end
    vector_variable = velocity
  []
  [velocity_y]
    type = VectorVariableComponentAux
    variable = velocity_y
    component = y
    execute_on = timestep_end
    vector_variable = velocity
  []
  [velocity_z]
    type = VectorVariableComponentAux
    variable = velocity_z
    component = z
    execute_on = timestep_end
    vector_variable = velocity
  []
[]
```

# Run it and Visualize it

- Commands

```
$ ../darcy_thermo_mech-opt -i step4.i
```

```
$ ~/projects/moose/python/peacock/peacock -r step4_out.e
```




Velocity is actually constant