MOOSE-based Thermal-Hydraulics Tools

Description, Status, and Needs

Mauricio E. Tano, Ph.D.

Moose-Based Thermal-Hydraulics tools

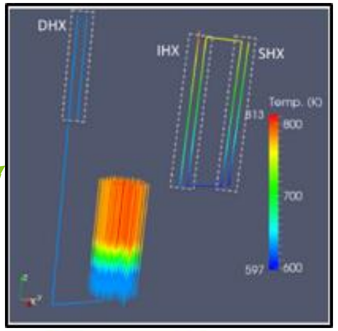
- MOOSE provides versatile, general-purpose thermal-hydraulics applications.
- These applications solve for:
  - mass, momentum, and energy conservation
  - in multicomponent, multiphase flows
  - using incompressible, weakly-compressible, or fully compressible formulations
  - for steady-state or transients
  - in lumped parameters and/or multidimensional (1, 2, or full 3D) geometries.

Fidelity + Computational Cost

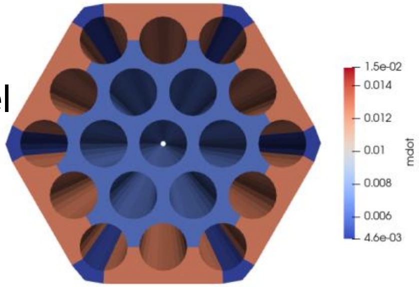
- 0D ODE Models
- Lumped Parameters Simulations
- Subchannel Simulations
- Coarse-Mesh CFD
- Reynolds-Average Navier Stokes Simulations
- Large Eddy Simulations
- Direct Numerical Simulations

Supported by MOOSE modules

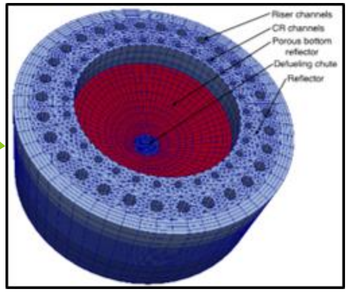
Thermal Hydraulics Module



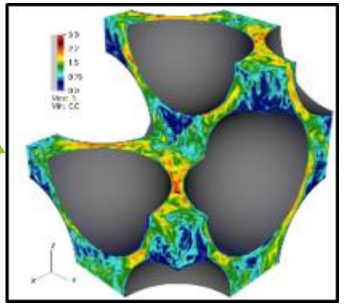
Subchannel Module



MOOSE Navier Stokes Module



Nek5000

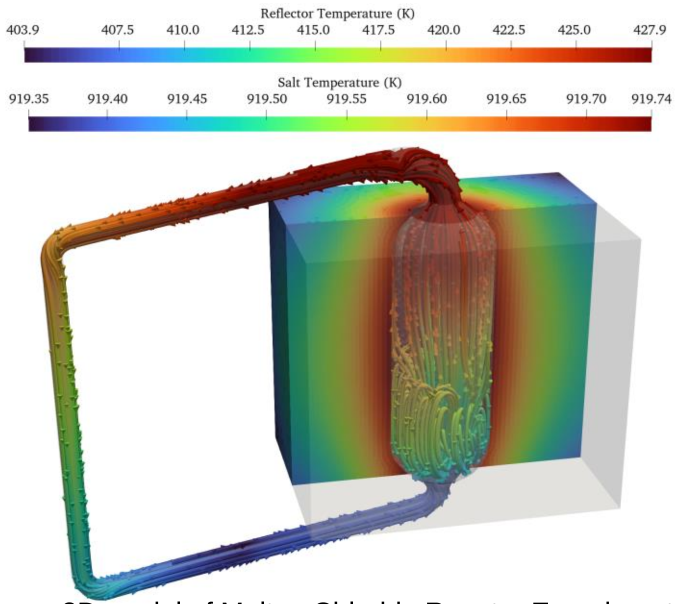


Key MOOSE-based Thermal-Hydraulics Tools

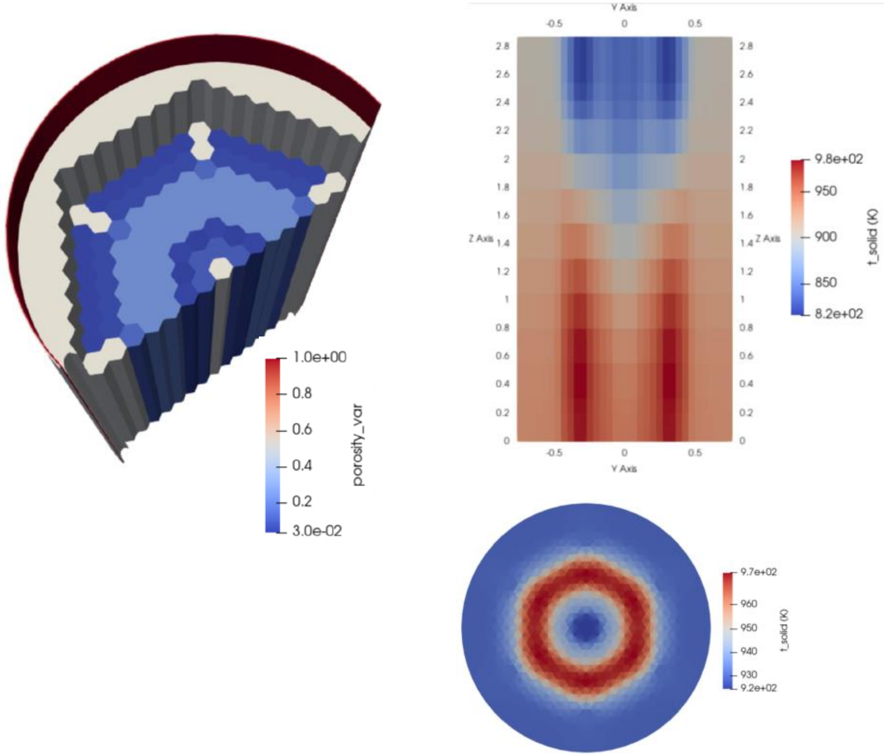
Module	Scale	Flow-Formulation	Dimension	Typical Element Count	Typical Runtime	Typical Simulations
Navier-Stokes Module / Pronghorn	Coarse-Mesh CFD Reynolds-Average Navier Stokes (RANS) Simulations	• Incompressible, Weakly-Compressible, or Fully-Compressible • Single- or multi-phase • Single- or multi-component flow	• Typically, 2D, 2D axisymmetric, or 3D • Can also be used in 1D	10,000	1 minute	• Flow through nuclear reactor core or plena • 3D multi-phase flow in pipes • Natural convection flow in open cavities
Subchannel Module	Subchannel Scale	• Incompressible or Weakly-Compressible • Single-phase • Single- or multi-component flow	• Typically, 3D • Can be used in 1D and 2D	100,000	10 seconds	• Flow development through nuclear reactor fuel assembly • Thermal-hydraulics analysis of nuclear reactor assembly blockage • Natural convection cooling in nuclear reactors low-flow assemblies
Thermal-Hydraulics Module	Lumped-Parameters Simulations	• Compressible • Single-phase	1D, 0D	100	10 seconds	• Heat extraction unit from nuclear reactor core • Thermal loops with significant compressibility effects

Pronghorn + MOOSE NS Module

- Numerical solvers:
  - Stabilized Finite Elements: 2017 –
  - Finite Volumes: 2021 –
- Fluid types:
  - Incompressible
  - Weakly-compressible
  - Compressible
- Flow regimes:
  - Laminar
  - Turbulent
- Flow types:
  - Free-flow
  - Porous media flow
  - Two-phase flow
- Validation:
  - Hundreds of regression test on canonical flows
  - 17 completed ERCOFTAC cases for diverse flow conditions



3D model of Molten Chloride Reactor Experiment
P. German; M. Tano
Model: FV, WCNS, Turbulent, Free-flow

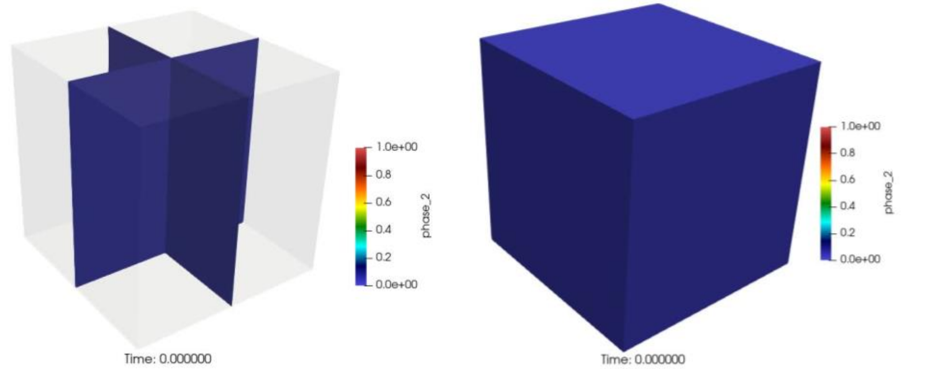


3D Model of High-Temperature Test Facility
V. Kyriakopoulos, M. Tano, P. Balestra, S. Schunert
Model: FV, WCNS, Laminar, Porous

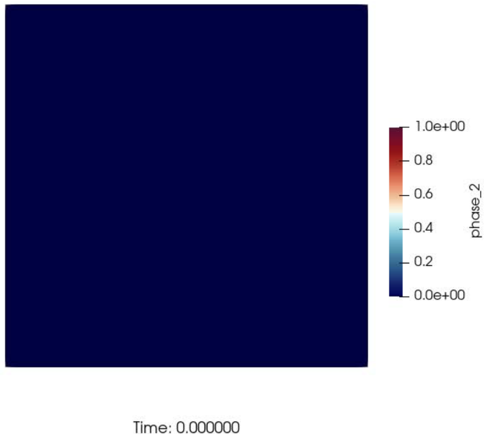
Melt Pool Simulation
A. Lindsay
Model: ALE, INS, Laminar, Free-flow

Contact Angle Equilibration in Molten Salt – Gas permeation
M. Tano; V. Prithivirajan
Model: DG-FE, INS, Laminar, Two-phase

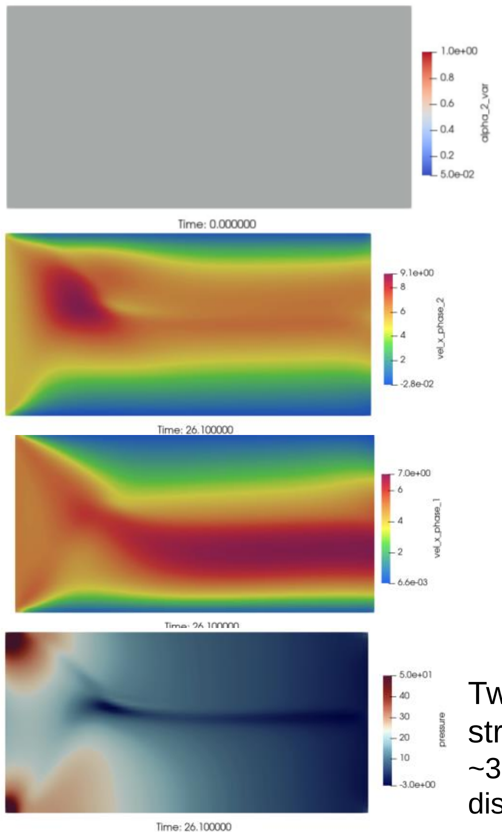
Examples MOOSE Navier-Stokes Validation Cases



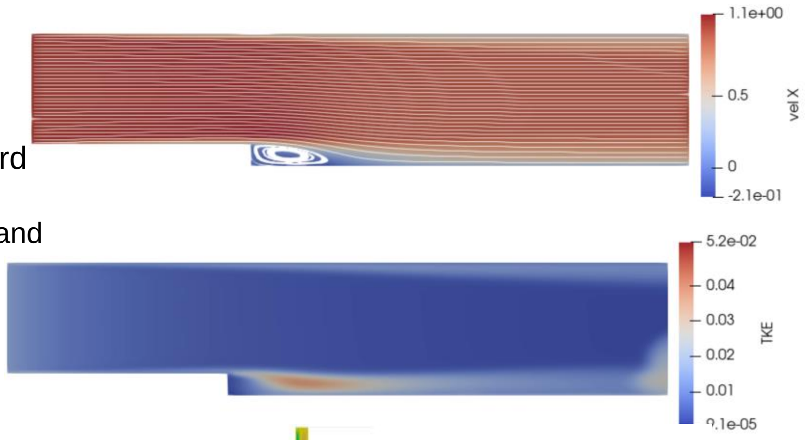
Two-phase Rayleigh-Benard Convection
~1.4% error in void fraction distribution vs experiments



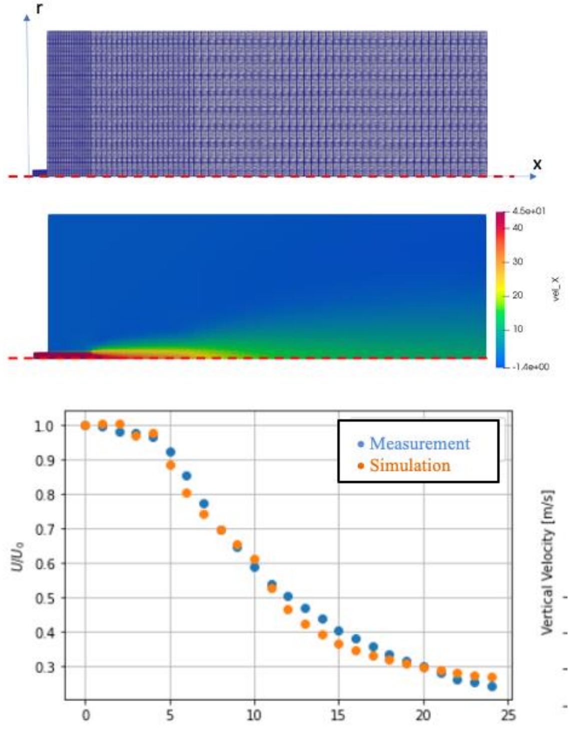
Two-phase Kelvin-Helmholtz Convection
~2.3% error in void fraction distribution vs experiments



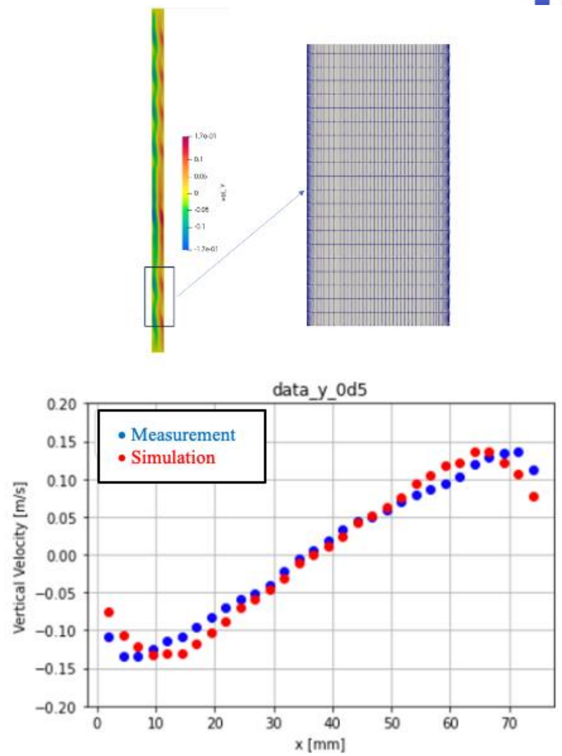
Two-phase flow stratification in channel
~3.2% error in velocity distribution vs experiments



Single-phase backward facing step test
~4.1% error in velocity and pressure distribution vs experiments



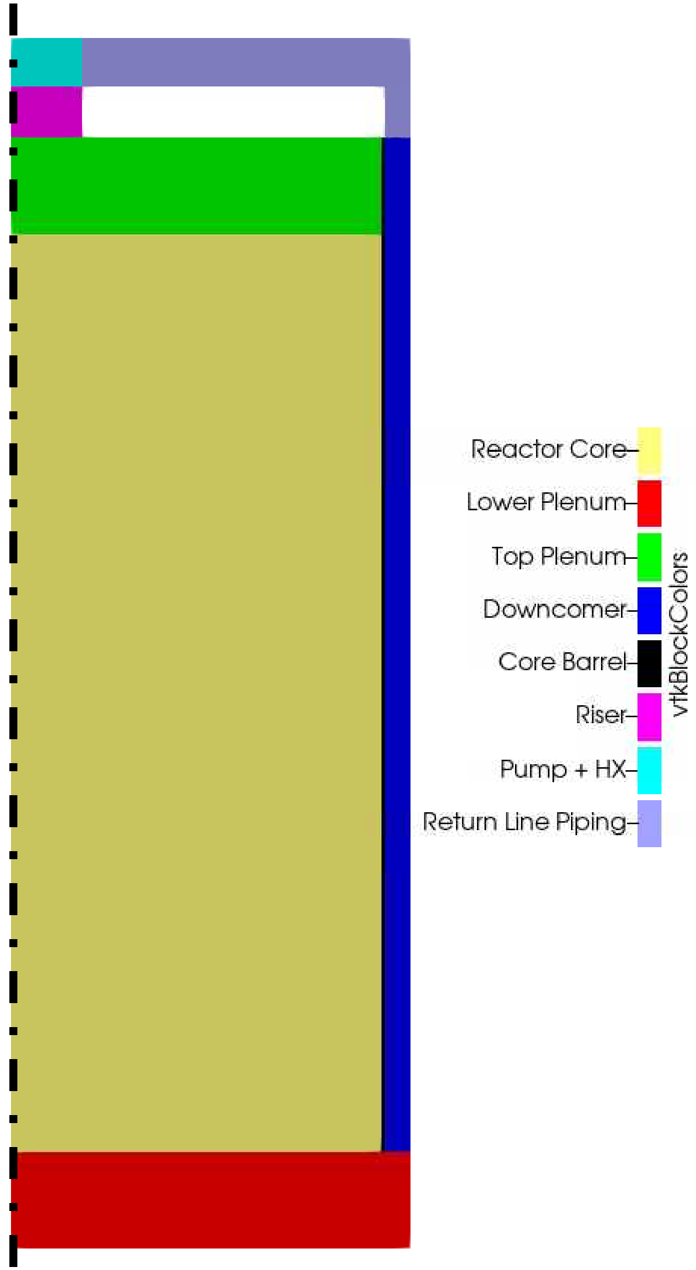
Air jet expansion in a a channel
~4.1% error in velocity distribution vs experiments



High-Rayleigh natural convection in tall cavity
~5.2% error in temperature velocity distribution vs experiments

Key Model Features

- Axisymmetric model
- Following the circulation of the molten salt, the model includes:
    1. Reactor Core
    2. Top Plenum
    3. Riser
    4. Pump + HX
    5. Return Line Piping
    6. Downcomer
    7. Bottom plenum
- The solid core barrel is included in the model.
    - Conjugated heat transfer is implemented between the reactor core and downcomer with the core barrel



Thermal-Hydraulics Formulation in Pronghorn

- General porous media model in Pronghorn:

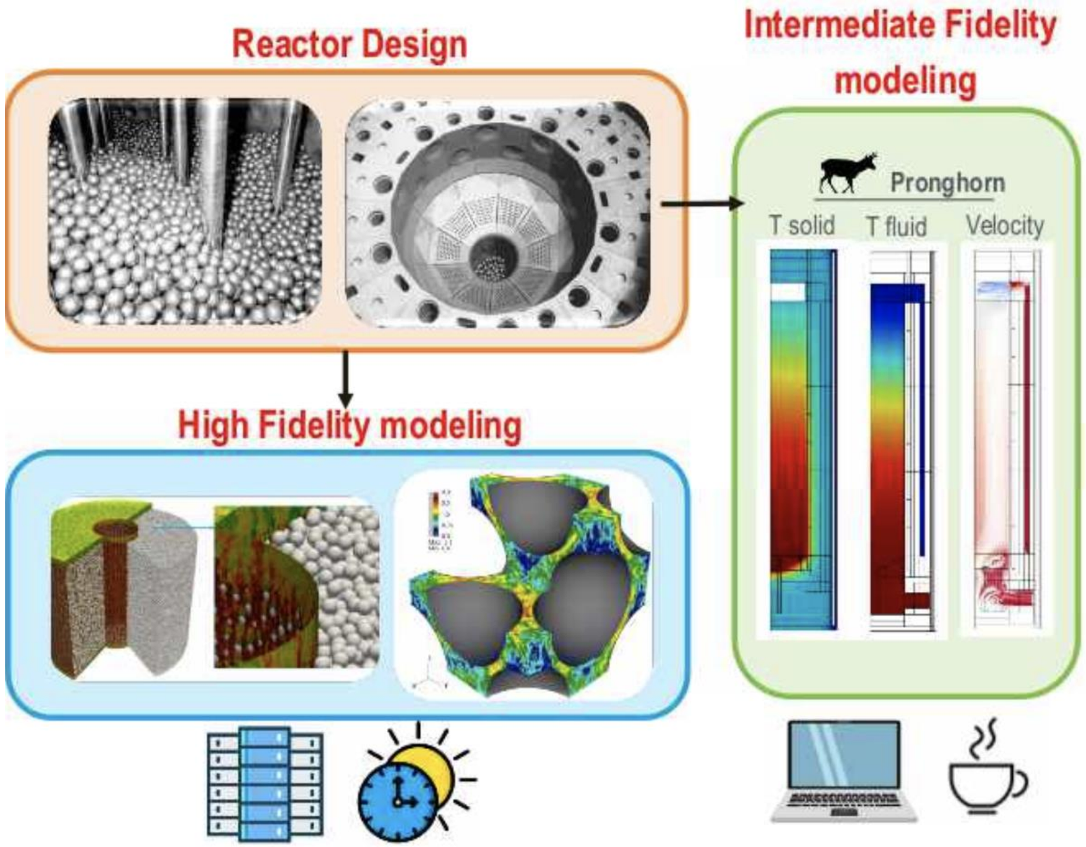
$$\gamma \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0$$

$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\gamma^{-1} \rho \boldsymbol{u}\boldsymbol{u}) =$$
$$= -\gamma \nabla p + \gamma \rho \boldsymbol{g} + \nabla \cdot \left(\mu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)\right) - \rho W$$

$$\gamma \frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho h \boldsymbol{u}) = \nabla \cdot \left(\kappa_{eff} \nabla T\right) - \alpha_{ls}(T - Ts) + q_f'''$$

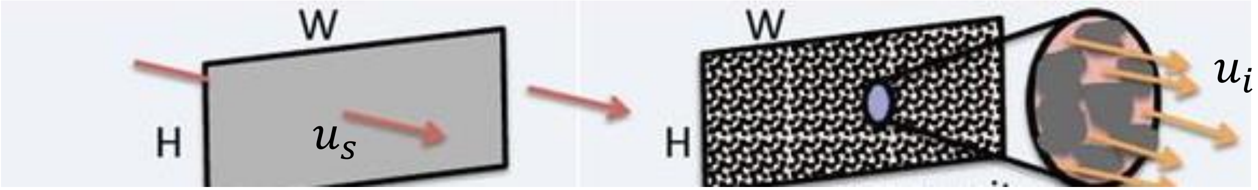
- Standard notation; γ is the porosity and W is the porous friction term, e.g., the Darcy-Forchheimer coefficient given by:

$$W \leftarrow \frac{150\nu}{D^2} \frac{(1-\gamma)^2}{\gamma^2} \boldsymbol{u} + \frac{1.75}{D} \frac{1-\gamma}{\gamma} |\boldsymbol{u}|\boldsymbol{u}$$



Solution Variables

- Solving for:
  - Superficial Velocity (linear momentum conservation)
  - Pressure (mass conservation)
  - Fluid Temperature (energy conservation in the fluid)
  - Solid Temperature (energy conservation in the solid)
- Note that we solve for superficial vs. interstitial velocity, i.e., $u_s = u_i\gamma$




```
[Variables]
  [superficial_vel_x]
    type              = PINSFVSuperficialVelocityVariable
    initial_condition = 1e-8
    block             = ${fluid_blocks}
  []
  [superficial_vel_y]
    type              = PINSFVSuperficialVelocityVariable
    initial_condition = 1e-8
    block             = ${fluid_blocks}
  []
  [pressure]
    type              = INSFVPressureVariable
    initial_condition = ${p_outlet}
    block             = ${fluid_blocks}
  []
  [T_fluid]
    type              = INSFVEnergyVariable
    initial_condition = ${T_Salt_initial}
    block             = ${fluid_blocks}
  []
  [T_solid]
    type              = INSFVEnergyVariable
    initial_condition = ${T_Salt_initial}
    block             = ${solid_blocks}
  []
[]
```

Fluid Properties

- Using coded thermophysical properties for MSRE salt
- A significant number of common nuclear coolants are supported in Pronghorn and the open-source MOOSE-NS module

```
[FluidProperties]
  [fluid_properties_obj]
    type                   = SimpleFluidProperties
    density0               = 2705.8554   # kg/m^3
    thermal_expansion      = 0.000177319 # K^{-1}
    cp                     = 1868.0      # J/kg·K
    viscosity              = 0.008268    # Pa-s11
    thermal_conductivity   = 1.4         # W/m·K
  []
[]
```

FluidProperties

Fluid Properties App	
AddFluidPropertiesAction	Add a UserObject object to the simulation.
BrineFluidProperties	Fluid properties for brine
CO2FluidProperties	Fluid properties for carbon dioxide (CO2) using the Span & Wagner EOS
CaloricallyImperfectGas	Fluid properties for an ideal gas with imperfect caloric behavior.
FlibeFluidProperties	Fluid properties for flibe
FlinakFluidProperties	Fluid properties for flinak
HeliumFluidProperties	Fluid properties for helium
HydrogenFluidProperties	Fluid properties for Hydrogen (H2)
IdealGasFluidProperties	Fluid properties for an ideal gas
IdealRealGasMixtureFluidProperties	Class for fluid properties of an arbitrary vapor mixture
LeadBismuthFluidProperties	Fluid properties for Lead Bismuth eutectic 2LiF-BeF2
LeadFluidProperties	Fluid properties for Lead
MethaneFluidProperties	Fluid properties for methane (CH4)
NaClFluidProperties	Fluid properties for NaCl
NaKFluidProperties	Fluid properties for NaK
NitrogenFluidProperties	Fluid properties for Nitrogen (N2)
SalineMoltenSaltFluidProperties	Molten salt fluid properties using Saline
SimpleFluidProperties	Fluid properties for a simple fluid with a constant bulk density

From: https://mooseframework.inl.gov/syntax/index.html

Fuel Salt Thermal-Hydraulics Solution 1/2

```
[NavierStokesFV]
  # Basic settings - weakly-compressible, turbulent flow with buoyancy
  block                             = ${fluid_blocks}
  compressibility                   = 'weakly-compressible'
  porous_medium_treatment           = true
  add_energy_equation               = true
  gravity                           = '0.0 -9.81 0.0'

  # Variable naming
  velocity_variable                 = 'superficial_vel_x superficial_vel_y'
  pressure_variable                 = 'pressure'
  fluid_temperature_variable        = 'T_fluid'

  # Numerical schemes
  pressure_face_interpolation       = average
  momentum_advection_interpolation = upwind
  mass_advection_interpolation      = upwind
  energy_advection_interpolation    = upwind
  velocity_interpolation            = rc

  # Porous & Friction treatement
  use_friction_correction           = true
  friction_types                    = 'darcy forchheimer'
  friction_coeffs                   = 'Darcy_coefficient Forchheimer_coefficient'
  consistent_scaling                = 100.0
  porosity_smoothing_layers         = 2
  turbulence_handling               = 'mixing-length'
```

Flow formulation

Solve Variables

Numerical Discretization

Porous Media Treatment for Reactor Core
The friction coefficients are defined as materials

Fuel Salt Thermal-Hydraulics Solution 2/2

```
# fluid properties
density                     = 'rho'
dynamic_viscosity           = 'mu'
thermal_conductivity        = 'kappa'
specific_heat               = 'cp'

# Energy source-sink
external_heat_source        = 'power_density_fuel'

# Boundary Conditions
wall_boundaries             = 'left      top       bottom    right     loop_boundary '
momentum_wall_types         = 'symmetry  slip      noslip    noslip    noslip'
energy_wall_types           = 'heatflux  heatflux  heatflux  heatflux  heatflux'
energy_wall_function        = '0         0         0         0         0'

# Constrain Pressure
pin_pressure                = true
pinned_pressure_value       = ${p_outlet}
pinned_pressure_point       = '0.0 2.13859 0.0'
pinned_pressure_type        = point-value-uo

# Passive Scalar -- solved separetely to integrate porosity jumps
add_scalar_equation         = false

#Scaling -- used mainly for nonlinear solves
momentum_scaling            = 1e-3
mass_scaling                = 10
```

Thermophysical properties
Defined by materials using the fluid_properties_obj

External heat source
Defined by auxiliary variables

Boundary conditions

Pressure pin
Needed to uniquely define the pressure in a closed-loop system

Scaling parameters
Needed to improve conditioning of the Jacobian in nonlinear solves

Extra Kernels for the solid domain

```
[energy_storage]
  type                    = PINSFVEnergyTimeDerivative
  variable                = T_solid
  rho                     = rho_s
  cp                      = cp_s
  is_solid                = true
[]
[solid_energy_diffusion_core]
  type                    = PINSFVEnergyAnisotropicDiffusion
  variable                = T_solid
  kappa                   = 'effective_thermal_conductivity'
  effective_diffusivity = true
  porosity                = 1
[]
[heat_source]
  type                    = FVCoupledForce
  variable                = T_solid
  v                       = power_density_graph
  block                   = 'core'
[]
```

Time Derivative of specific Solid Enthalpy (defining $h = \rho c_p T$)

Specific Solid Enthalpy Diffusion

External heart source
Source is defined as Auxiliary Variables

Extra kernels for pump + HX modeling

```
[pump_x]
  type                    = INSFVBodyForce
  variable                = superficial_vel_x
  functor                 = ${pump_force}
  block                   = 'pump'
  momentum_component      = 'x'
  rhie_chow_user_object = 'pins_rhie_chow_interpolator'
[]
[pump_y]
  type                    = INSFVBodyForce
  variable                = superficial_vel_y
  functor                 = ${pump_force}
  block                   = 'pump'
  momentum_component      = 'y'
  rhie_chow_user_object = 'pins_rhie_chow_interpolator'
[]
[convection_fluid_hx]
  type                    = NSFVEnergyAmbientConvection
  variable                = T_fluid
  T_ambient               = ${T_inlet_hx}
  alpha                   = ${vol_hx}
  block                   = 'pump'
[]
```

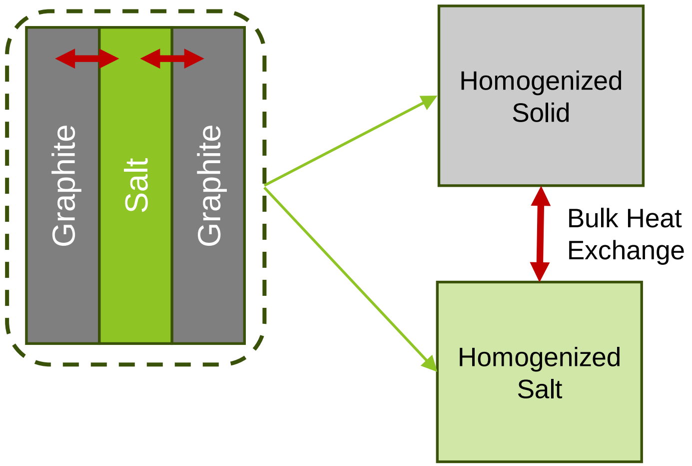
Volumetric force ($\vec{F} = [F_x, F_y]$) for the pump in the 'x' and 'y' direction

Heat exchange with external temperature ($q''' = \alpha(T - T_{ambient})$)

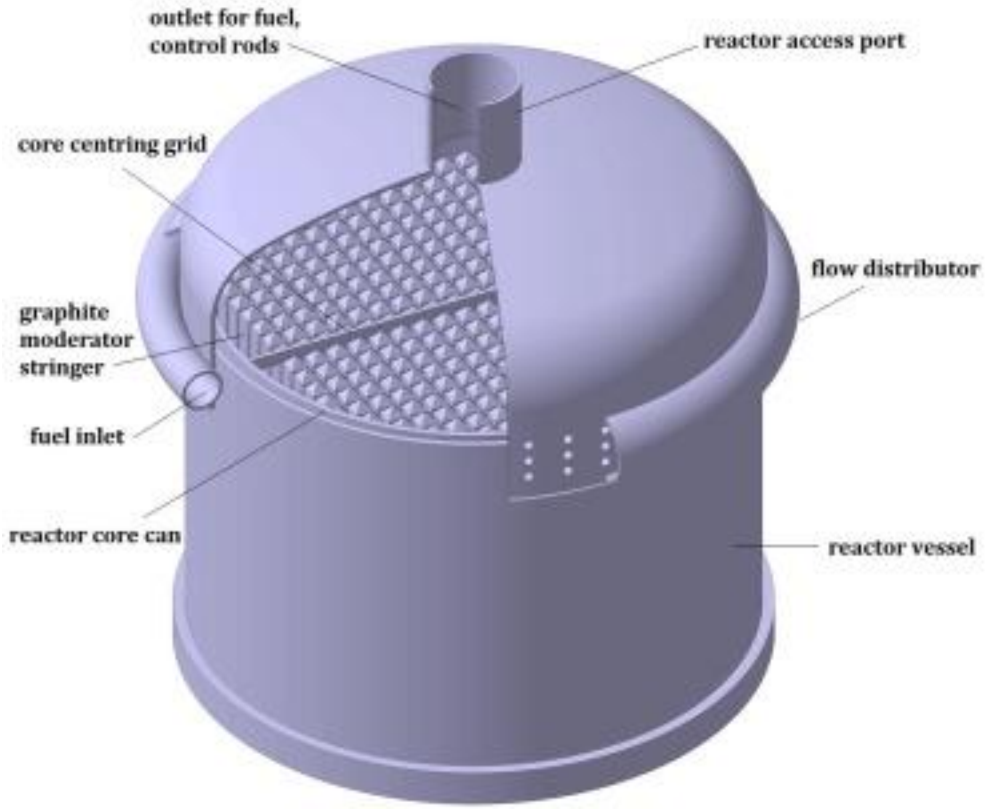
```
T_inlet_hx              = 904.55              # Salt inlet temperature (K)
```

Liquid/Solid Heat exchanges in the MSRE

- The MSRE core is modeled as a porous media, where the liquid and solid domains are homogenized
  - Bulk heat exchange occurs between the liquid and solid phases



- Conjugated heat transfer occurs between the downcomer, after the flow distributor, and the reactor core through the reactor core can or core barrel



From: Tadepalli, S. C., Gupta, A., & Umasankari, K. (2017). Neutronic analysis of MSRE and its study for validation of ARCH code. *Nuclear Engineering and Design*, *320*, 1-8.

Modeling bulk heat transfer at reactor core

```
[convection_core]
  type                      = PINSFVEnergyAmbientConvection
  variable                  = T_solid
  T_fluid                   = T_fluid
  T_solid                   = T_solid
  is_solid                  = true
  h_solid_fluid             = ${bulk_htc}
  block                     = 'core'
[]
```

Bulk heat transfer **from liquid** salt **to solid** core

```
[convection_core_completmeent]
  type                      = PINSFVEnergyAmbientConvection
  variable                  = T_fluid
  T_fluid                   = T_fluid
  T_solid                   = T_solid
  is_solid                  = false
  h_solid_fluid             = ${bulk_htc}
  block                     = 'core'
[]
```

Bulk heat transfer **from solid** core **to liquid** salt

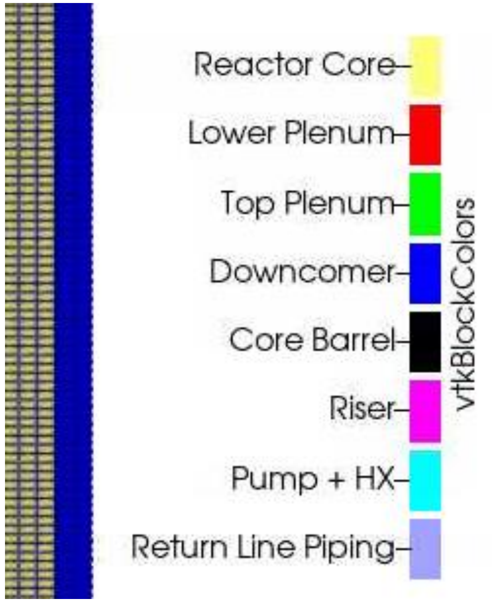
```
bulk_htc               = 20000.0          # (W/(m3.K)) core bulk volumetric heat exchange coefficient (already callibrated)
```

Modeling conjugated heat transfer trough core barrel

```
[FVInterfaceKernels]
  # Conjugated heat transfer with core barrel
  [convection]
    type                  = FVConvectionCorrelationInterface
    variable1             = T_fluid
    variable2             = T_solid
    boundary              = 'core_barrel'
    h                     = ${bulk_htc}
    T_solid               = T_solid
    T_fluid               = T_fluid
    subdomain1            = 'core down_comer lower_plenum upper_plenum'
    subdomain2            = 'core_barrel'
    wall_cell_is_bulk     = true
  []
[]
```

Conjugated heat transfer through reactor core barrel

Exchange occurs between the liquid salt temperature, defined on the the left between the core + lower_plenum + upper_plenum and on the right in the down_comer, and the solid temperature defined on the core barrel

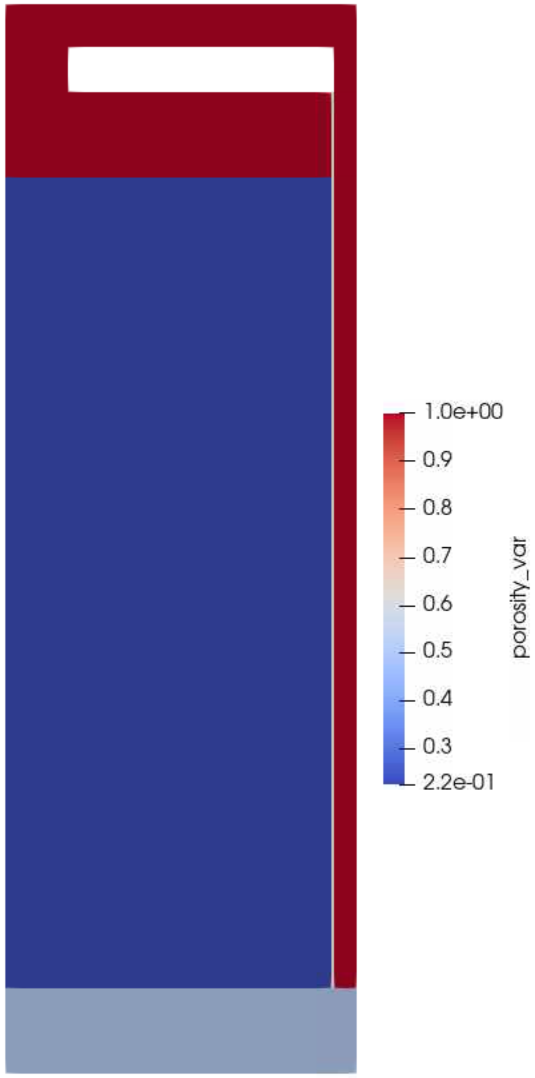


Defining materials

```
# -----------------------------------------------------------------
# Setting up material porosities at fluid blocks
# -----------------------------------------------------------------
[porosity]
  type                     = ADPiecewiseByBlockFunctorMaterial
  prop_name                = 'porosity'
  subdomain_to_prop_value = 'core            ${core_porosity}
                             lower_plenum    ${lower_plenum_porosity}
                             upper_plenum    ${upper_plenum_porosity}
                             down_comer      ${down_comer_porosity}
                             riser           ${riser_porosity}
                             pump            ${pump_porosity}
                             elbow           ${elbow_porosity}
                             core_barrel     0'
[]
# -----------------------------------------------------------------
# Setting up hydraulic diameters at fluid blocks
# -----------------------------------------------------------------
[hydraulic_diameter]
  type                     = PiecewiseByBlockFunctorMaterial
  prop_name                = 'characteristic_length'
  subdomain_to_prop_value = 'core            ${D_H_fuel_channel}
                             lower_plenum    ${D_H_plena}
                             upper_plenum    ${D_H_plena}
                             down_comer      ${D_H_downcomer}
                             riser           ${D_H_pipe}
                             pump            ${D_H_pipe}
                             elbow           ${D_H_pipe}'
  block                    = ${fluid_blocks}
[]
```

Setting up **porosity** for each block in the model

Setting up the **hydraulic diameter** for each block in the model



Setting up materials for fluid and solid properties

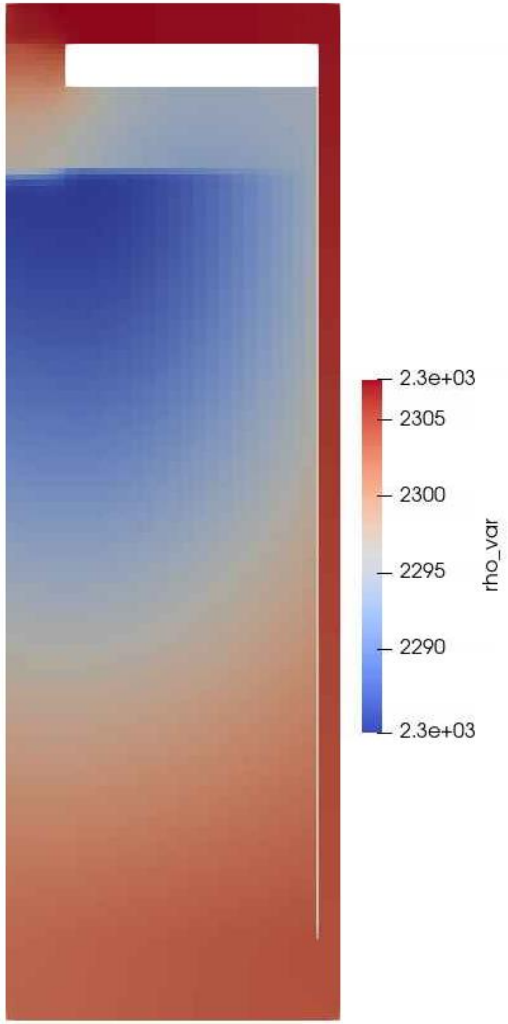
```
# ---------------------------------------------------------------------
# Setting up Fluid & Solid properties
# ---------------------------------------------------------------------
[fluid_props_to_mat_props]
  type                      = GeneralFunctorFluidProps
  pressure                  = 'pressure'
  T_fluid                   = 'T_fluid'
  speed                     = 'speed'
  characteristic_length     = characteristic_length
  block                     = ${fluid_blocks}
[]
[core_moderator]
  type                      = ADGenericFunctorMaterial
  prop_names                = 'rho_s    cp_s    k_s'
  prop_values               = '${rho_graph} ${cp_graph} ${k_graph}'
  block                     = 'core'
[]
[core_barrel_steel]
  type                      = ADGenericFunctorMaterial
  prop_names                = 'rho_s    cp_s    k_s'
  prop_values               = '${rho_steel} ${cp_steel} ${k_steel}'
  block                     = 'core_barrel'
[]
[effective_fluid_thermal_conductivity]
  type                      = ADGenericVectorFunctorMaterial
  prop_names                = 'kappa'
  prop_values               = 'k k k'
  block                     = ${fluid_blocks}
[]
[effective_solid_thermal_conductivity]
  type                      = ADGenericVectorFunctorMaterial
  prop_names                = 'effective_thermal_conductivity'
  prop_values               = 'k_s k_s k_s'
  block                     =  ${solid_blocks}
[]
```

Setting up properties for the fuel salt
Note: this properties are taken from fluid_properties_obj and define temperature-dependent thermophysical properties

Setting up properties for the graphite moderator

Setting up properties for the steel core barrel

Defining effective thermal conductivity for liquid and solid phases in porous media
Note: these properties could be anisotropic



Defining friction coefficients

```
# Drag correlations per block
[isotropic_drag_core]
  type                    = FunctorChurchillDragCoefficients
  multipliers             = '100000 100 100000'
  block                   = 'core'
[]
[drag_lower_plenum]
  type                    = FunctorChurchillDragCoefficients
  multipliers             = '10 1 10'
  block                   = 'upper_plenum'
[]
[drag_upper_plenum]
  type                    = FunctorChurchillDragCoefficients
  multipliers             = '1 1 1'
  block                   = 'lower_plenum'
[]
[drag_downcomer]
  type                    = FunctorChurchillDragCoefficients
  multipliers             = '1 1 1'
  block                   = 'down_comer'
[]
[drag_piping]
  type                    = FunctorChurchillDragCoefficients
  multipliers             = '0 0 0'
  block                   = 'riser pump elbow'
[]
```

Drag coefficients for the core
Note: blocking non-physical flow in the radial direction

Drag coefficients of lower plenum
Note: accounts for lower plenum internal structures

Drag coefficients of upper plenum
Note: single channel

Drag coefficients of downcomer
Note: single channel

Drag coefficients of piping
Note: pressure drop not modeled in detail due to return loop approximation

Churchill Darcy friction factor

$$f/8 = \left[\left(\frac{8}{\mathrm{Re}}\right)^{12} + \frac{1}{(\Theta_1 + \Theta_2)^{1.5}}\right]^{\frac{1}{12}}$$

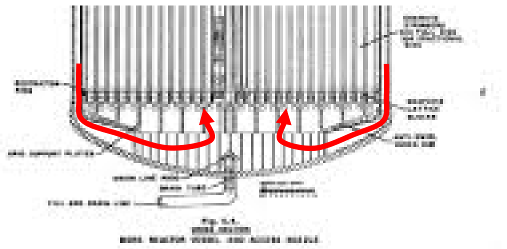
where

$$\Theta_1 = \left[-2.457 \ln\left(\left(\frac{7}{\mathrm{Re}}\right)^{0.9} + 0.27\frac{\varepsilon}{D}\right)\right]^{16}$$

$$\Theta_2 = \left(\frac{37530}{\mathrm{Re}}\right)^{16}$$

From: https://en.wikipedia.org/wiki/Darcy_friction_factor_formulae

MSRE lower plenum



Postprocessors

Multiple postprocessors to analyze MSRE performance

```
[pressure_outlet]
  type                    = SideAverageValue
  variable                = pressure
  boundary                = 'pump_inlet'
[]
[pressure_inlet]
  type                    = SideAverageValue
  variable                = 'pressure'
  boundary                = 'downcomer_outlet'
[]
[pressure_core_delta]
  type                   = ParsedPostprocessor
  function               = 'pressure_inlet - pressure_outlet'
  pp_names               = 'pressure_inlet pressure_outlet'
  execute_on             = 'initial timestep_end'
[]
[T_inlet]
  type                    = SideAverageValue
  variable                = 'T_fluid'
  boundary                = 'downcomer_outlet'
[]
[T_outlet]
  type                    = SideAverageValue
  variable                = 'T_fluid'
  boundary                = 'riser_inlet'
[]
[T_core_inlet]
  type                    = SideAverageValue
  variable                = 'T_fluid'
  boundary                = 'core_in'
[]
[T_core_outlet]
  type                    = SideAverageValue
  variable                = 'T_fluid'
  boundary                = 'core_out'
[]
```

```
[v_core_inlet]
  type                        = SideAverageValue
  variable                    = 'superficial_vel_y'
  boundary                    = 'core_in'
[]
[v_core_outlet]
  type                        = SideAverageValue
  variable                    = 'superficial_vel_y'
  boundary                    = 'core_out'
[]
[T_core_delta]
  type                       = ParsedPostprocessor
  function                   = 'T_core_outlet - T_core_inlet'
  pp_names                   = 'T_core_outlet T_core_inlet'
  execute_on                 = 'initial timestep_end'
[]
[area_pp_downcomer_inlet]
  type                        = AreaPostprocessor
  boundary                    = 'downcomer_inlet'
  execute_on                  = 'INITIAL'
[]
[vfr_downcomer]
  type                        = VolumetricFlowRate
  vel_x                       = superficial_vel_x
  vel_y                       = superficial_vel_y
  advected_quantity           = 1.0
  boundary                    = 'downcomer_inlet'
[]
[vfr_pump]
  type                        = VolumetricFlowRate
  vel_x                       = superficial_vel_x
  vel_y                       = superficial_vel_y
  advected_quantity           = 1.0
  boundary                    = 'pump_outlet'
[]
[mfr_core_inlet]
  type                        = VolumetricFlowRate
  vel_x                       = superficial_vel_x
  vel_y                       = superficial_vel_y
  advected_quantity           = rho
  boundary                    = 'downcomer_outlet'
[]
```

```
[mfr_core_outlet]
  type                        = VolumetricFlowRate
  vel_x                       = superficial_vel_x
  vel_y                       = superficial_vel_y
  advected_quantity           = rho
  boundary                    = 'pump_inlet'
[]
[core_vol]
  type                        = VolumePostprocessor
  block                       = 'core'
  execute_on                  = 'initial timestep_end'
[]
[loop_vol]
  type                        = VolumePostprocessor
  block                       = ${non_solid_blocks}
  execute_on                  = 'initial timestep_end'
[]
[Tmax_fuel]
  type                        = ElementExtremeValue
  value_type                  = max
  variable                    = T_fluid
  block                       = ${fluid_blocks}
  execute_on                  = 'initial timestep_end'
[]
[Tavg_fuel]
  type                        = ElementAverageValue
  variable                    = T_fluid
  block                       = ${fluid_blocks}
  execute_on                  = 'initial timestep_end'
[]
[Tmax_core_fuel]
  type                        = ElementExtremeValue
  value_type                  = max
  variable                    = T_fluid
  block                       = 'core'
  execute_on                  = 'initial timestep_end'
[]
[Tavg_core_fuel]
  type                        = ElementAverageValue
  variable                    = T_fluid
  block                       = 'core'
  execute_on                  = 'initial timestep_end'
[]
```

```
[Tavg_core_fuel]
  type                        = ElementAverageValue
  variable                    = T_fluid
  block                       = 'core'
  execute_on                  = 'initial timestep_end'
[]
[Tmax_mod]
  type                        = ElementExtremeValue
  value_type                  = max
  variable                    = T_solid
  block                       = 'core'
  execute_on                  = 'initial timestep_end'
[]
[Tavg_mod]
  type                        = ElementAverageValue
  variable                    = T_solid
  block                       = 'core'
  execute_on                  = 'initial timestep_end'
[]
[power_total]
  type                        = ElementIntegralVariablePostprocessor
  variable                    = power_density
  execute_on                  = 'initial timestep_end'
[]
[power_avg]
  type                        = ElementAverageValue
  variable                    = power_density
  execute_on                  = 'initial timestep_end'
[]
[power_fuel_total]
  type                        = ElementIntegralVariablePostprocessor
  variable                    = power_density_fuel
  execute_on                  = 'initial timestep_end'
[]
[power_ghrap_total]
  type                        = ElementIntegralVariablePostprocessor
  variable                    = power_density_graph
  execute_on                  = 'initial timestep_end'
[]
[power_total_2]
  type                       = ParsedPostprocessor
  function                   = 'power_ghrap_total + power_fuel_total'
  pp_names                   = 'power_ghrap_total power_fuel_total'
  execute_on                 = 'initial timestep_end'
[]
```

Next Steps

- Understand auxiliary variables and auxiliary kernels
- Running the simulation: `mpirun -n 12 blue_crab-opt -i msre_ph_ss.i`

- Analyze the simulation integrating neutron precursors
  - The equation solved for the delayed neutron precursor concentration of type i is:

$$\underbrace{\frac{\partial c_i}{\partial t}}_{\text{Time derivative}} + \underbrace{\nabla \cdot \left(\frac{\boldsymbol{u}}{\gamma} c_i\right)}_{\text{Convection in Porous Media}} - \underbrace{\nabla \cdot \left[\left(D_{c_i} + \frac{\nu_t}{Sc_t}\right) \nabla c_i\right]}_{\text{Molecular and Turbulent Diffusion}} = \underbrace{y_i f}_{\text{Fission Yield Source}} - \underbrace{\lambda c_i}_{\text{Natural Decay Sink}}$$

  - Note how this implementation is performed in `msre_ph_ss_dnp.i`
  - Run the simulation with neutron precursors: `mpirun -n 12 blue_crab-opt -i msre_ph_ss_dnp.i`

Idaho National Laboratory

Battelle Energy Alliance manages INL for the U.S. Department of Energy's Office of Nuclear Energy. INL is the nation's center for nuclear energy research and development, and also performs research in each of DOE's strategic goal areas: energy, national security, science and the environment.