INL/MIS-21-62652-Revision-0

Evolution of MOOSE and MOOSE-Based Tools to Address Analysis Challenges

May 2021

Mark D DeHart, Vincent M Laboure, Sebastian Schunert

Idaho National Laboratory

INL is a U.S. Department of Energy National Laboratory operated by Battelle Energy Alliance, LLC

DISCLAIMER

This information was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness, of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the U.S. Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the U.S. Government or any agency thereof.

INL/MIS-21-62652-Revision-0

Evolution of MOOSE and MOOSE-Based Tools to Address Analysis Challenges

Mark D DeHart, Vincent M Laboure, Sebastian Schunert

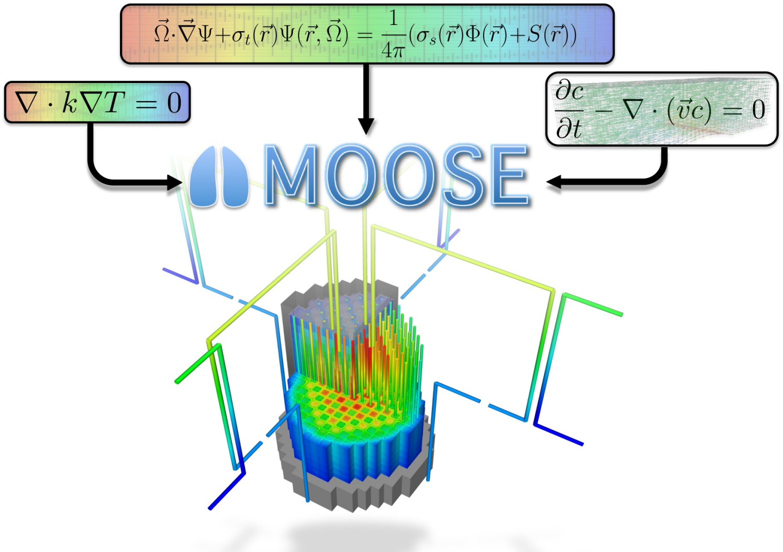
May 2021

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

INL/MIS-21-62652


$\vec{\Omega}\cdot\vec{\nabla}\Psi+\sigma_t(\vec{r})\Psi(\vec{r},\vec{\Omega}) = \frac{1}{4\pi}(\sigma_s(\vec{r})\Phi(\vec{r})+S(\vec{r}))$
$\nabla \cdot k\nabla T = 0$
$\frac{\partial c}{\partial t} - \nabla\cdot(\vec{v}c) = 0$
MOOSE

May 11, 2021

Evolution of MOOSE and MOOSE-Based Tools to Address Analysis Challenges

Presented to the NASA Digital Transformation Working Group

Presented by Vincent Labouré, Sebastian Schunert and Mark DeHart

Idaho National Laboratory

MOOSE History and Purpose

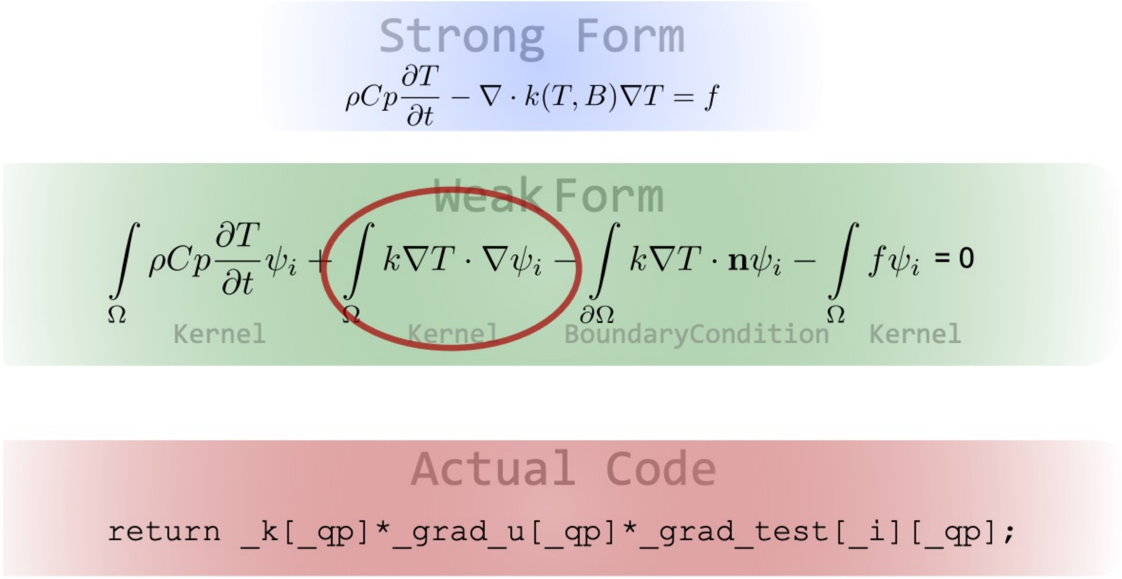
- Development started in 2008
- Open-sourced in 2014
- Designed to solve computational engineering problems and reduce the expense and time required to develop new applications by:
  - being easily extended and maintained
  - working efficiently on a few and many processors
  - providing an object-oriented, pluggable system for creating all aspects of a simulation tool
  - Not written specifically for any specific type of physics
  - In theory, applicable to any form of physics expressed as a set of PDEs

What is MOOSE?

- MOOSE is the **M**ultiphysics **O**bject-**O**riented **S**imulation **E**nvironment
- Written in C++, is an objected-oriented framework that exists as a set of linkable libraries.
- MOOSE uses the finite element method and takes physics equations (specifies in an equation-like format) and internally develops the FEM equivalent
- To solve a specific physics equation, that equation is written in its *weak form* (a FEM requirement) and the equation (or set of equations) are written in the form of MOOSE *Kernels.* A kernel is a “piece” of physics.
- It’s convenient to think of a kernel as a mathematical operator, such as a Laplacian or a convection term in a PDE.
- A kernel is typically composed of single lines of C++ code for the mathematical operator, the exact or approximate Jacobian, and one for each boundary condition.
- In MOOSE, kernels may be swapped or coupled together to achieve different application goals.

What is MOOSE?

***“To solve a specific physics equation, that equation is written in its weak form and the equation(s) are written in the form of MOOSE* Kernels”**



Strong Form

$$\rho Cp\frac{\partial T}{\partial t} - \nabla \cdot k(T,B)\nabla T = f$$

Weak Form

$$\underset{\text{Kernel}}{\int_\Omega \rho Cp\frac{\partial T}{\partial t}\psi_i} + \underset{\text{Kernel}}{\int_\Omega k\nabla T \cdot \nabla\psi_i} - \underset{\text{BoundaryCondition}}{\int_{\partial\Omega} k\nabla T \cdot \mathbf{n}\psi_i} - \underset{\text{Kernel}}{\int_\Omega f\psi_i} = 0$$

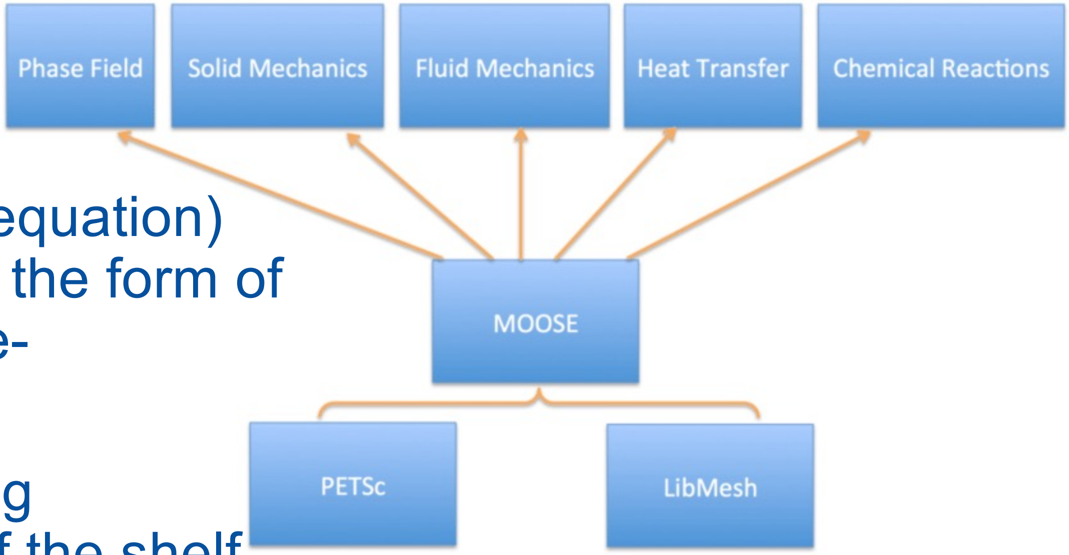
Actual Code

```
return _k[_qp]*_grad_u[_qp]*_grad_test[_i][_qp];
```

- where:
  - *_k* is the conductivity
  - *_grad_u* is the gradient of the temperature
  - *_grad_test* is the FEM test function

- A kernel is generated for each term in the equation
- Given properties (k, C_p, ρ), BC and a mesh, this C++ expression can essentially be linked to MOOSE and run (1, 2 or 3-D)
- MOOSE is only a framework including a solver – MOOSE can’t do anything by itself. Each new “animal” is compiled with and contains MOOSE

What is MOOSE (continued)



- Many kernels (e.g., conduction equation) are already built into MOOSE in the form of *modules* that don't need to be re-developed
- MOOSE is really good for solving uncommon physics where no off the shelf algorithms exist – a complete application can be developed literally in hours rather than weeks/months.
- MOOSE leverages scientific matrix solvers in PETSc and meshing toolset in libMesh
  - PETSc really helps for code parallelization as users typically only have to specify the desired number of procs/threads – the high-performance solution is inherited largely from PETSc. Almost all of the mathematical solution is done in PETSc and is very high-performance
  - libMesh provides mesh-related tools that we use for generating meshes and data translation

MOOSE Multiphysics

- MOOSE was originally developed to solve all equations at the same time for multiple physical phenomena (“physics”)
- This approach is extremely powerful as all coupled variables are solved all at the same time in a single large matrix
  - This is known as an implicit or strongly coupled solution
  - Griffin solutions with thermal feedback are often solved implicitly when flow is not included (e.g., TREAT)
- This also allows different forms of a given equation to coexist on one solution mesh
  - Different equations applied for different mesh regions
  - It is possible for a low order and a high order approximation to be used simultaneously to focus high fidelity only where needed.
- MOOSE employs the preconditioned Jacobian-Free Newton Krylov (JFNK) approach to allow solutions for very large matrices
  - JFNK doesn’t require inversion of the very large matrix
  - Allows solution of matrices that are VERY costly and impractical to solve otherwise.

MOOSE Considerations

- MOOSE is not as strong in head-to-head comparison to applications where specialized codes exist and use custom solution algorithms.
  - You would not write a neutron transport solver with the current version of MOOSE if all you wanted to do was to solve for neutron transport.
- MOOSE's focus is on flexible coupling.
  - If you want to couple said transport with T/H, conduction, radiative heat transfer and structural mechanics (for example) MOOSE will significantly accelerate the process.
  - In MOOSE, even when solved independently, the variables are common and shared (if on the same mesh)
  - MOOSE can handle conservative transfer between different meshes.
- Fuel/materials analysts use MKS units; neutronics analysts use CGS.
  - MOOSE is unit agnostic – it uses whatever units are provided
  - This can cause issues if user is not aware of differences in units
  - Allows user to use units consistent with form of data (neutronic data is always provided in CGS)
  - Codes need to be written to make unit conversions when coupling to other codes using different units.

Early Challenges in Using MOOSE

- By design, the original MOOSE approach solved all variables at one time then advanced in time, all on same time scale
  - Not all physics evolve on the same time scale
- For strongly coupled systems all equations are on the same mesh
  - Not all physics require or should use the same mesh
- Mesh generation is simple for simple configurations, and more difficult for complex configurations
  - INL and others primarily use Cubit (scripted using Python)
  - INL is now partnering with Coreform (a commercial Cubit spinoff) and plans to improve this process
- The generality of the framework induces an overhead (both memory and speed) that can be non-negligible, especially for physics that are not typically solved with FEM (e.g., thermal-fluids) or not solved strongly (e.g., neutronics).

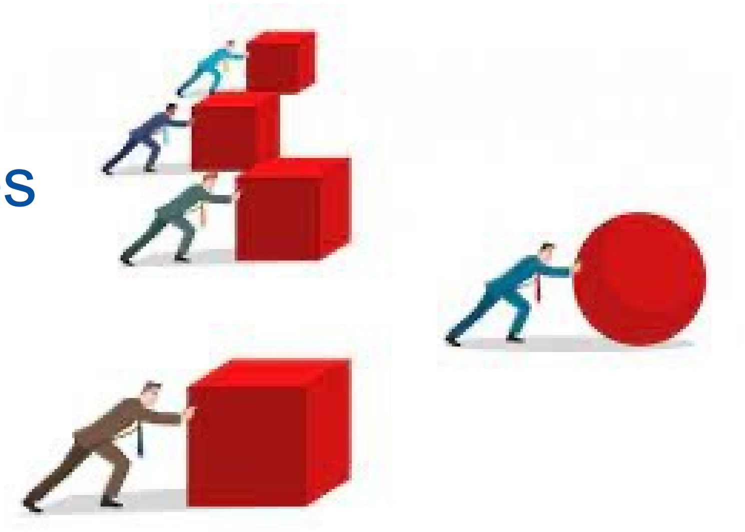
Preconditioning when using JFNK in MOOSE

- A JFNK solver requires an appropriate preconditioner to accelerate convergence
  - MOOSE provides a number of preconditioning options; however, the appropriate preconditioner can be difficult to find at times
- The ideal preconditioning operator is problem dependent
- Elliptic/parabolic (aka diffusion-like) equations typically perform well with AMG preconditioner (standard in MOOSE)
- Harder problems in MOOSE often rely on brute force (e.g., LU solve) but does not scale well as the problem gets bigger
- MOOSE has introduced Automatic Differentiation (AD) to help mathematically build an appropriate Jacobian preconditioner.
- AD does have an associated time and memory cost.
- For neutron transport (hyperbolic), building the full Jacobian is too expensive for large problems so physics-based preconditioning is often needed.
- For other physics (e.g., mechanics with contact), preconditioning remains an issue.
- More on preconditioning later.

How to Make a Multiphysics Model Efficient?

- Key #1: efficiency of each physics (speed, robustness, memory, …)
  - ➢ Example A: neutronics
  - ➢ Example B: thermal-hydraulics

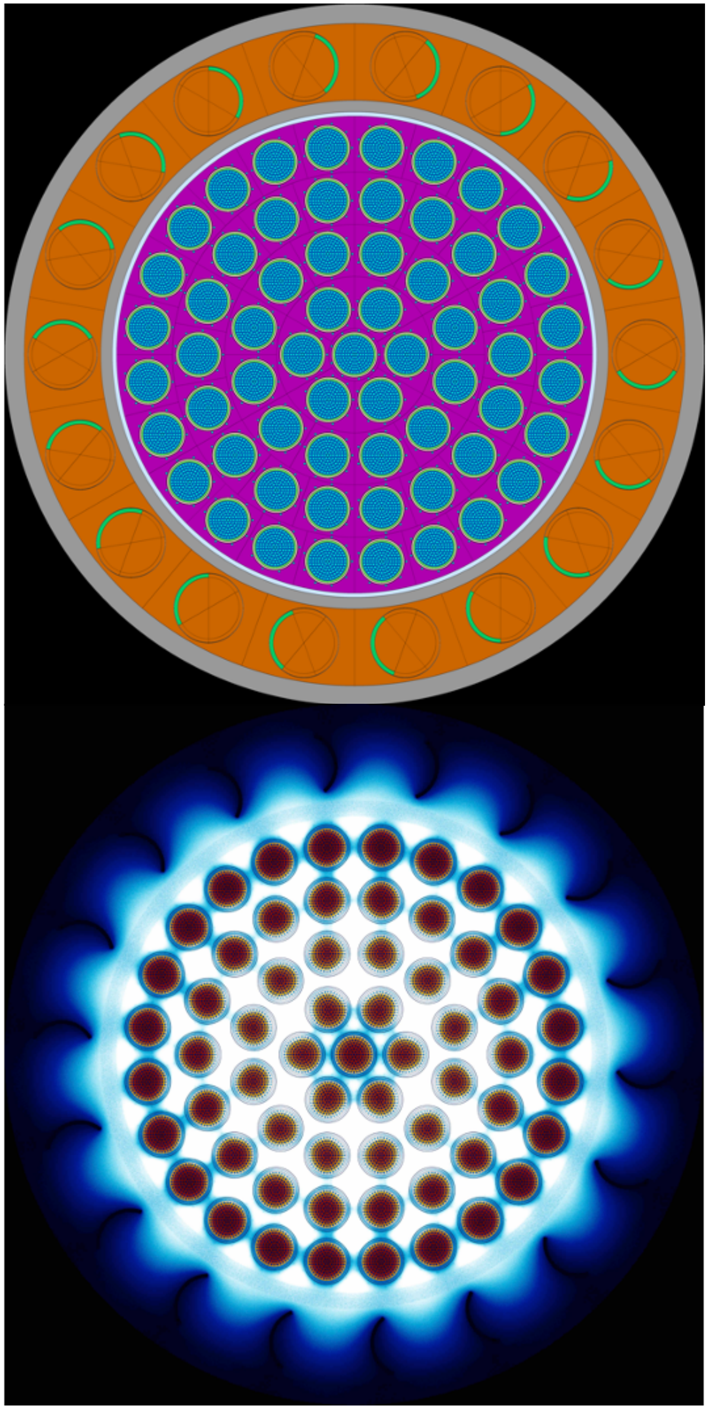
- Key #2: efficient convergence of coupling terms between physics
  - ➢ Strong vs tight coupling
  - ➢ Dealing with different time scales



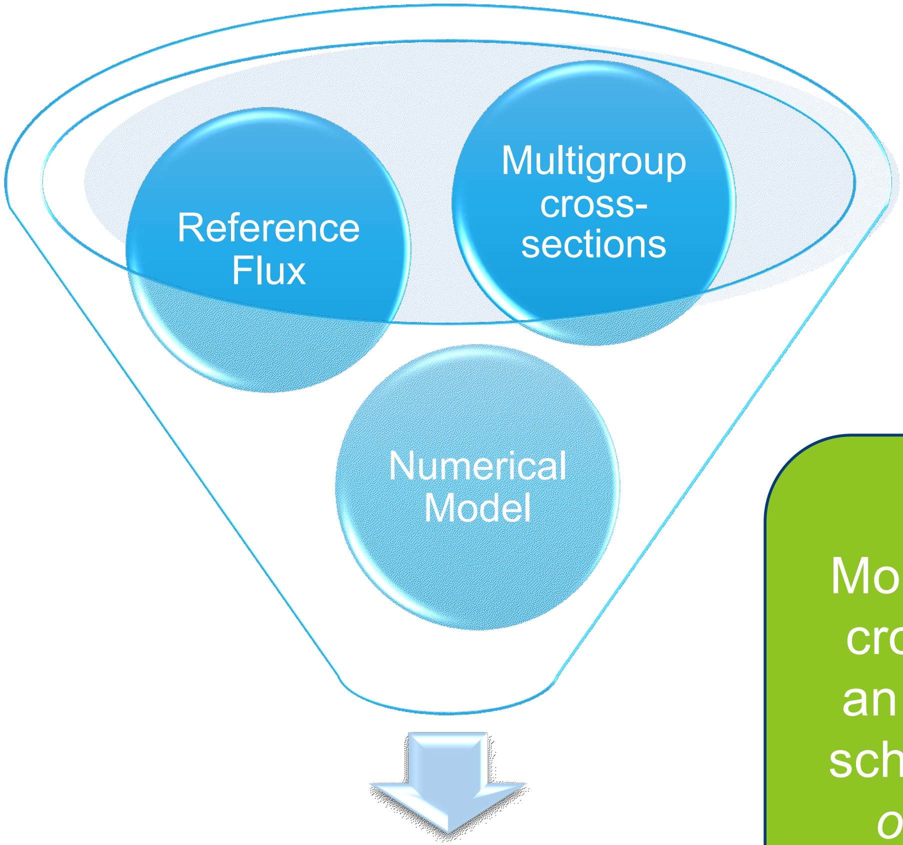
Example A: Neutronics

- 3 dimensions in space (often >10^6 elements/cells)
- 2 in neutron direction (100 or more directions),
- 1 in neutron energy (10s to 100s of groups)
- Time scales (from μs to years)
- Highly nonlinear cross sections
- Overhead with Griffin due to solving all the groups simultaneously

High-fidelity solutions are bound to take time

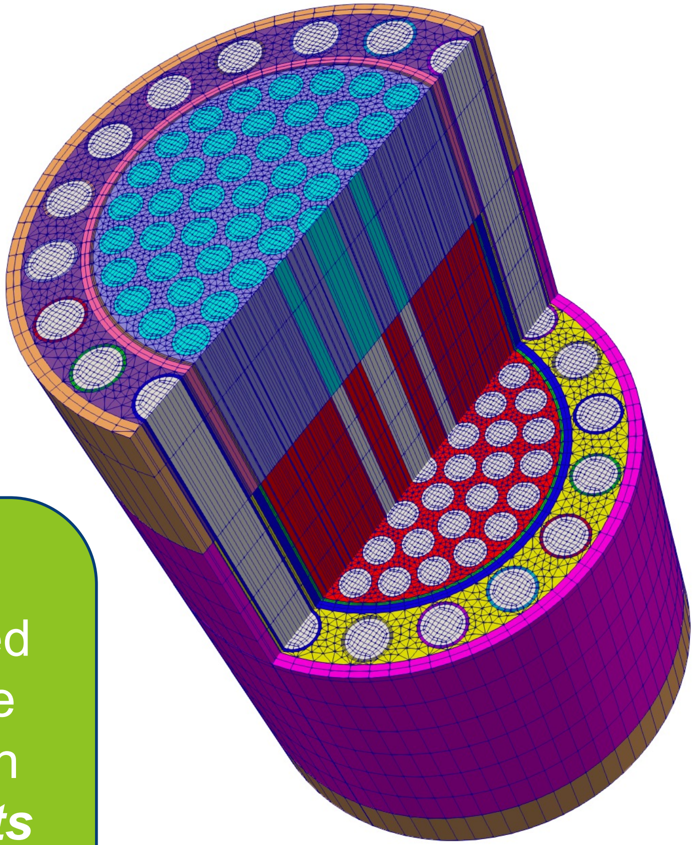


Physics-based Reduced Order Model: Super-Homogenization (SPH) Diffusion Approach



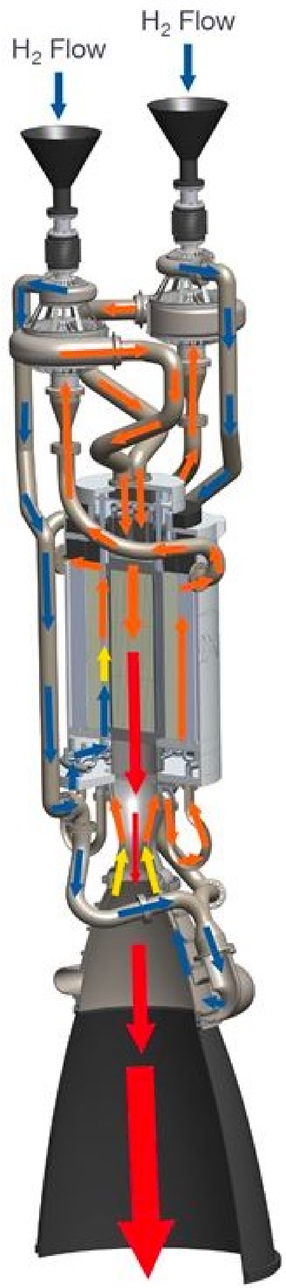
SPH-corrected library

Monte-Carlo corrected cross sections to use an ***accurate*** diffusion scheme for ***transients*** *on a* ***coarse*** *mesh*



Example B: Thermal-Hydraulics

- Originally implemented continuous Finite Element Method (FEM) in RELAP-7
- Recently replaced with discontinuous FEM with Finite Volume option
- Single phase performance is decent but suffers from solving all the equations (mass, momentum, energy) simultaneously and on the same mesh (MOOSE overhead)
- Alternative could be to “MOOSE-Wrap” a non-MOOSE tool



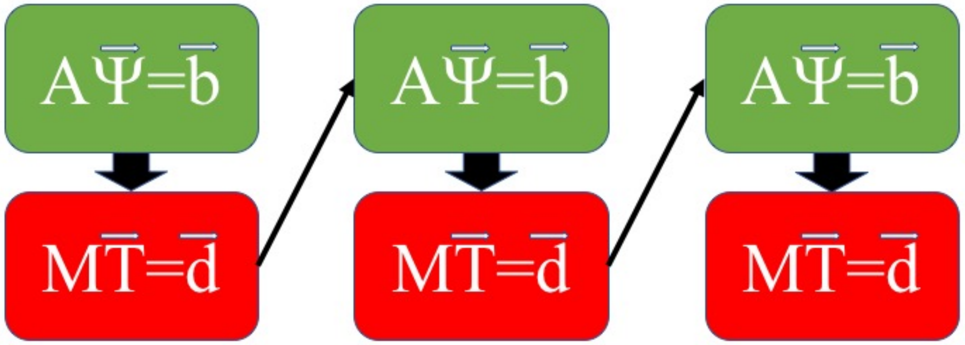
Coupling Approaches

Strongly coupled

$$\begin{pmatrix} A & \alpha \\ \beta & M \end{pmatrix} \begin{pmatrix} \vec{\Psi} \\ \vec{T} \end{pmatrix} = \begin{pmatrix} \vec{b} \\ \vec{d} \end{pmatrix}$$

- Ideal to efficiently converge (off-diagonal terms easily included in Jacobian)
- Assumes comparable scales

Tightly coupled

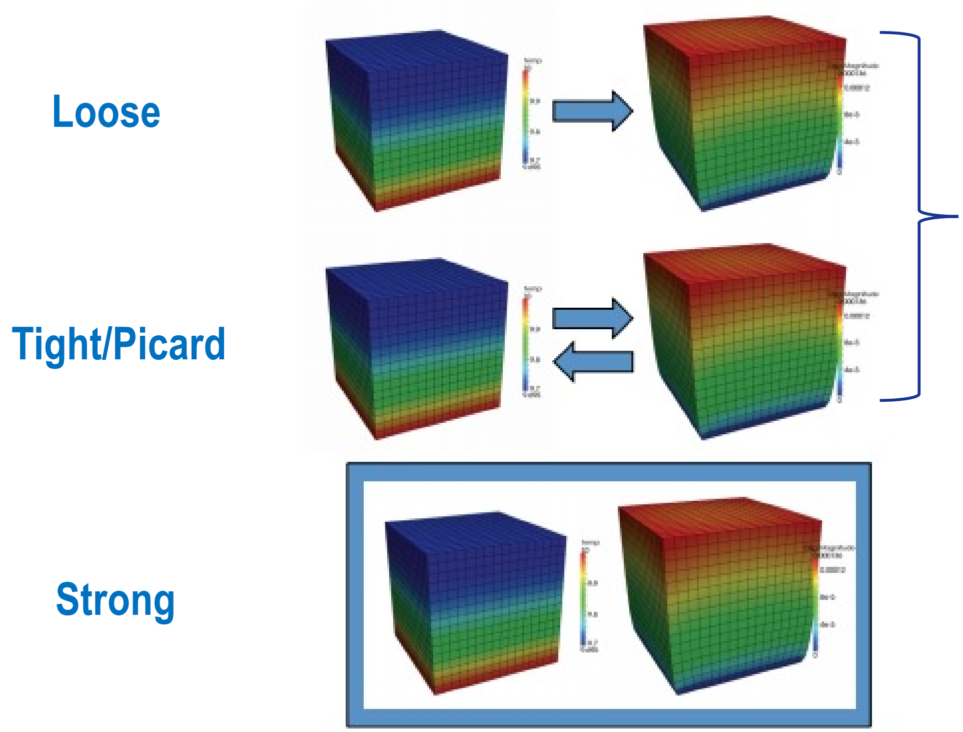


- Picard iterations may converge slowly
- Much greater flexibility when using intrinsically different tools (discretization, time scales, etc…)

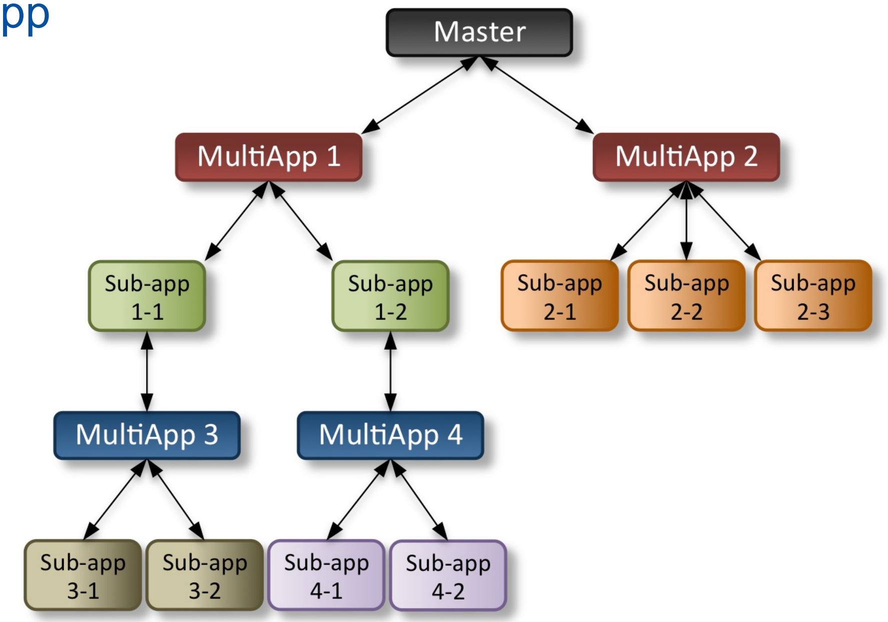
Which one is more efficient?
It all depends…

Flexibility by MultiApps

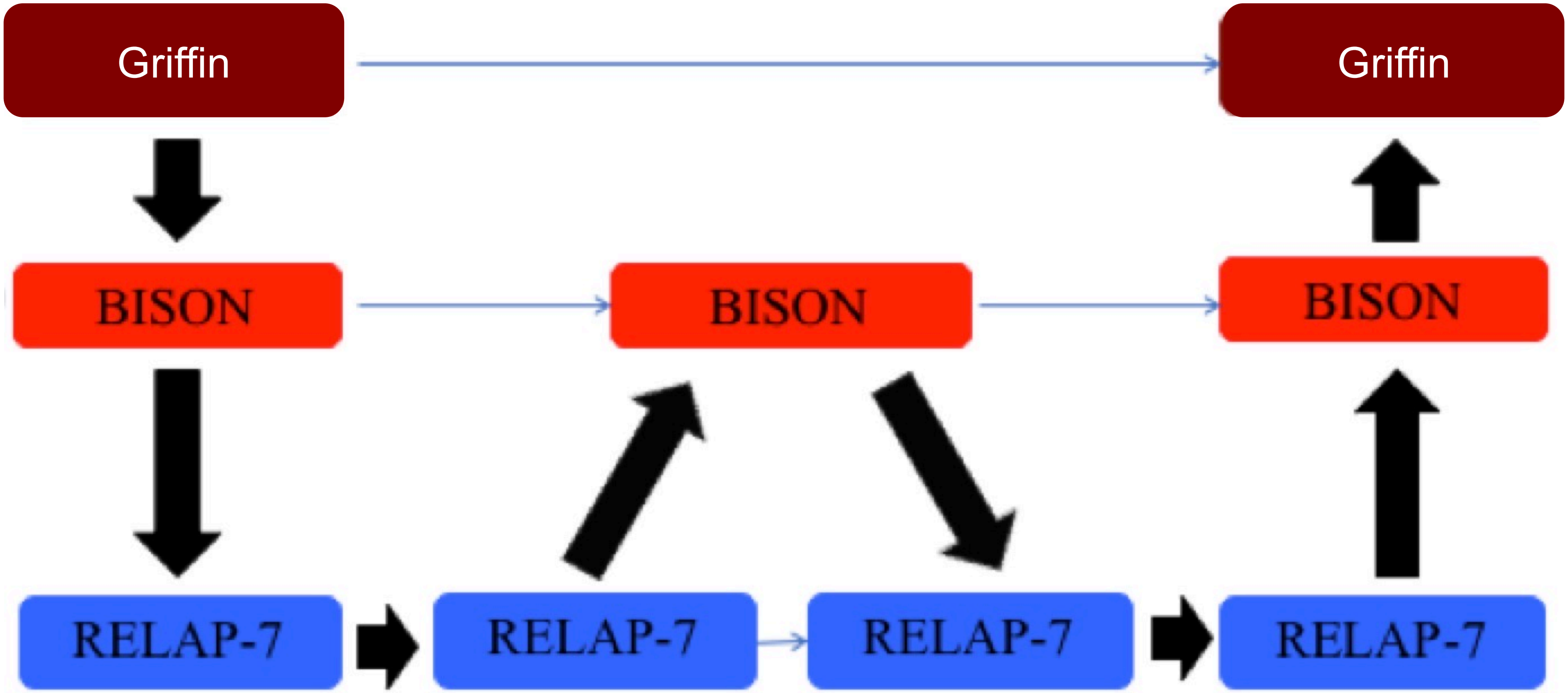
- MOOSE supports loose coupling (operator split) & Picard via MultiApps
- MOOSE supports strongly coupled solves



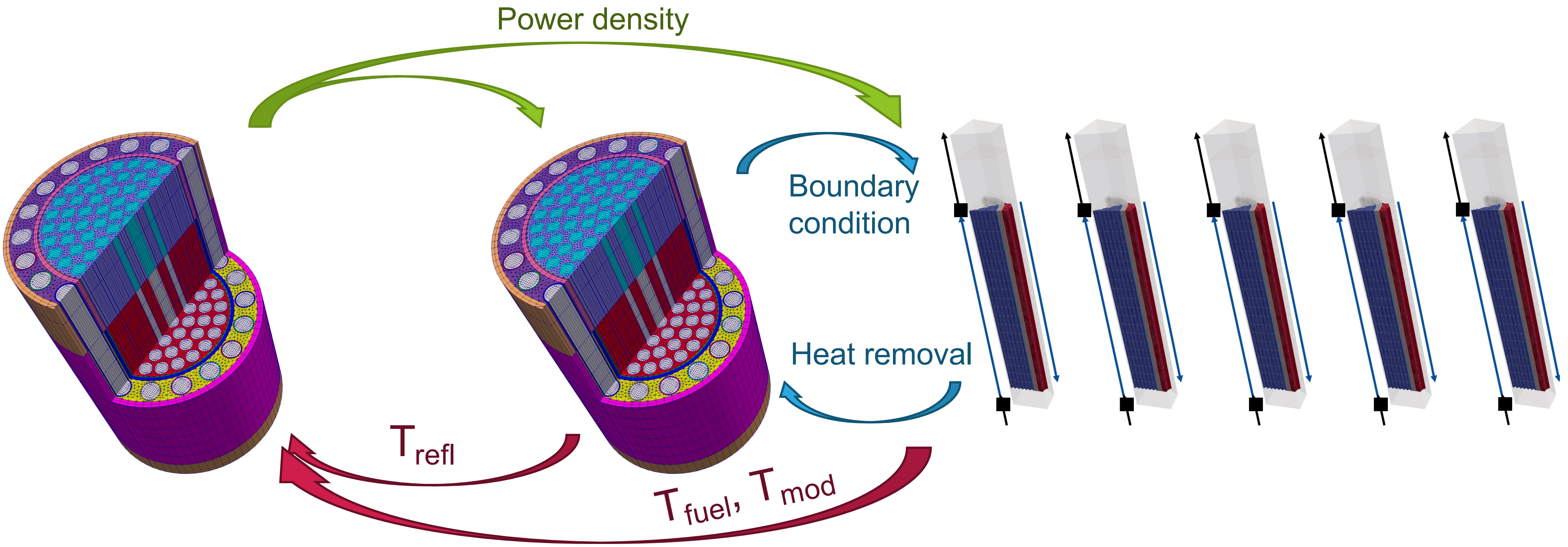
- Picard Iterations via MultiApp
- Master app owns a sub-app
- Recursive: sub-app can own its own sub-app tree
- Information transfer via flexible MOOSE transfers
- Different meshes & dimensionality
- Sub-cycling
- Mixing eigenvalue & transients



Dealing with Different Time Scales: Sub-Cycling



Example – Coupled NTP Full-Core Model



Power density

Boundary condition

Heat removal

T_{refl}

T_{fuel}, T_{mod}

Full-core Neutronics:
primary app

Full-core Heat Conduction:
sub-app 1

5 Fuel Element Heat Conduction: sub-app 2
5 TH channels: sub-app 3

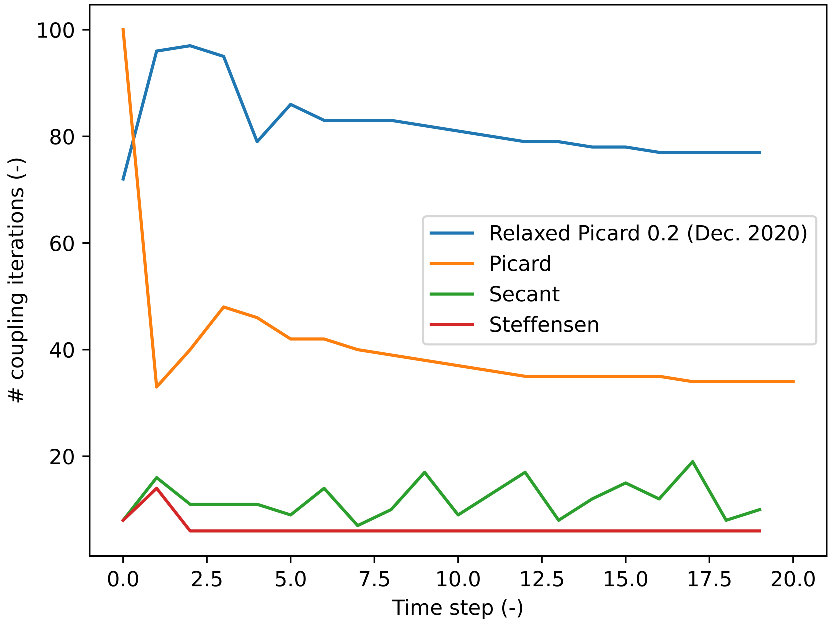
Picard iterations give a lot of flexibility but… can take time to converge coupling terms

Improvement of Picard Iteration Convergence

- Strong coupling (single nonlinear problem) is often impractical (memory consumption, time/length scales, different meshes)
- Picard iterations convergence slowly:
  - Neutronics, heat conduction, thermal-hydraulics (~200k DoFs)
  - One evaluation is 1 minute, 600 times steps, 10 Picard
  - Runtime is **10 hours**
- Acceleration:Secant's, Steffensen's methods (Anderson's method TBD)
- Promising work done by Guillaume Giudicelli @ INL
- Previous work in the DOE CASL program by Alexander Toth

"In general, we noted that Anderson provides a significant improvement in robustness."
- from A. Toth's dissertation

# Iterations for 2D/1D fluid flow coupling (courtesy G. Giudicelli)

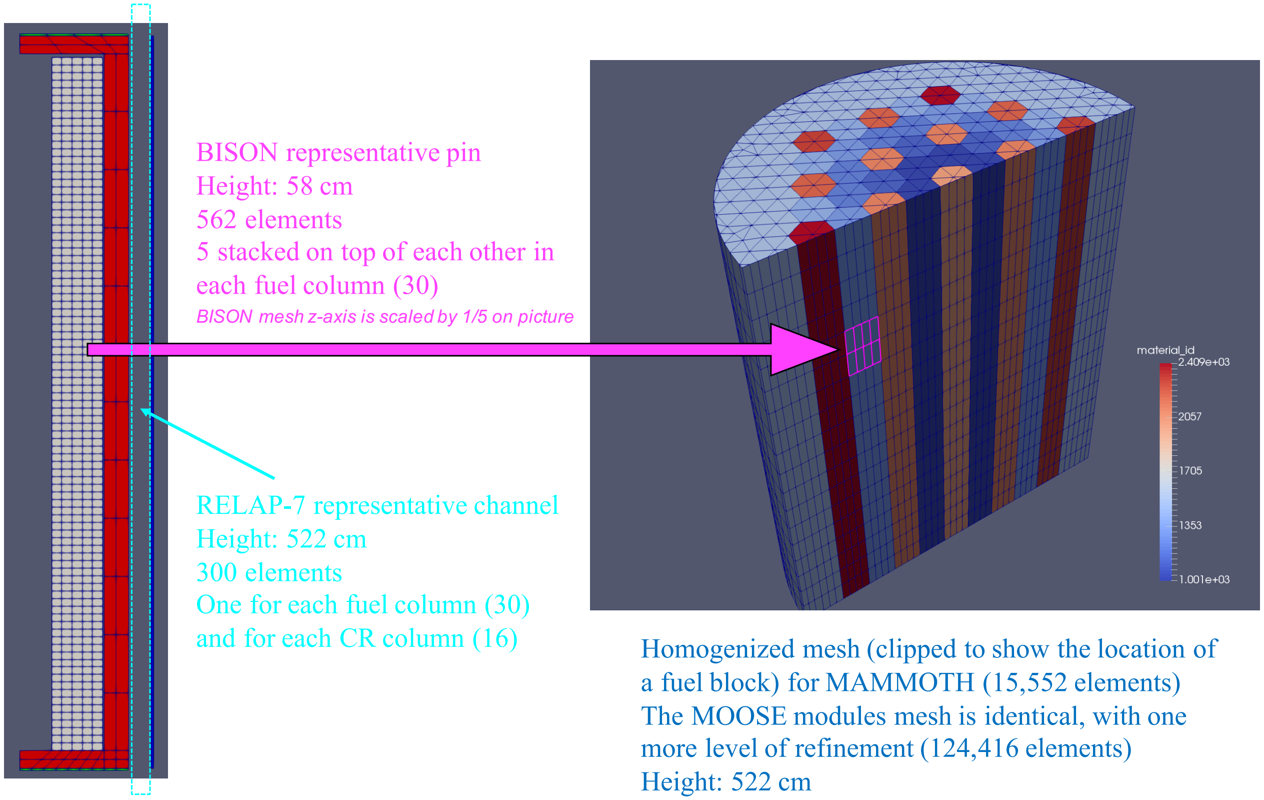


Data Transfer Between Apps

- When meshes are identical, transfer of quantities is straightforward

- Otherwise, it can become complicated to preserve energy

- Options exist but usually geometry specific (hexagonal, cylindrical, etc.)

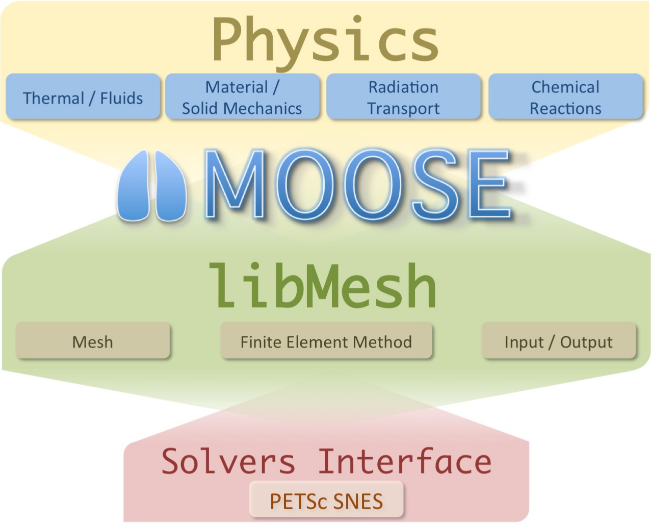


BISON representative pin
Height: 58 cm
562 elements
5 stacked on top of each other in each fuel column (30)
BISON mesh z-axis is scaled by 1/5 on picture

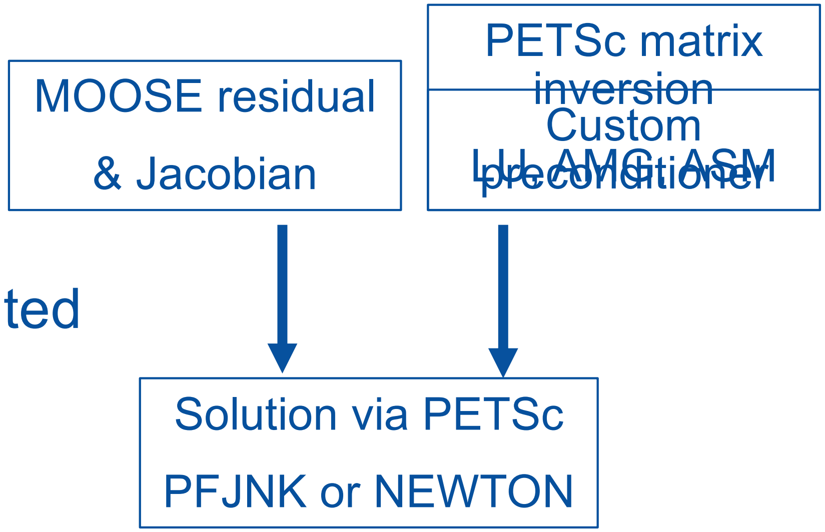
RELAP-7 representative channel
Height: 522 cm
300 elements
One for each fuel column (30) and for each CR column (16)

Homogenized mesh (clipped to show the location of a fuel block) for MAMMOTH (15,552 elements)
The MOOSE modules mesh is identical, with one more level of refinement (124,416 elements)
Height: 522 cm

MOOSE Solver Strategy (1)



- MOOSE is pluggable for physics, kernels, etc **but not** for the solver machinery
- This is **significant shortcoming** that needs to be improved
- MOOSE solution algorithm:
  - PETSc Newton/PJFNK
  - MOOSE computes Jacobian/residuals
  - PETSc matrix inversion (AMG)
  - Custom preconditioner can be implemented
- MOOSE was developed for implicit time-stepping
- Explicit time-stepping exists but framework overhead makes it inefficient!
- The MOOSE “External problem” is a path forward to make MOOSE solves more flexible! (*more later*)



MOOSE residual & Jacobian

PETSc matrix inversion

Custom preconditioner

LU, AMG, ASM

Solution via PETSc PFJNK or NEWTON

MOOSE Solver Strategy (2)

Advantage:

- Existing solvers remove burden from users
- Some problems have efficient off-the-shelf solvers (parabolic problems, AMG)
- With an intermediate level of effort, custom preconditioners can be implemented (if you can cast a custom solver as preconditioner)

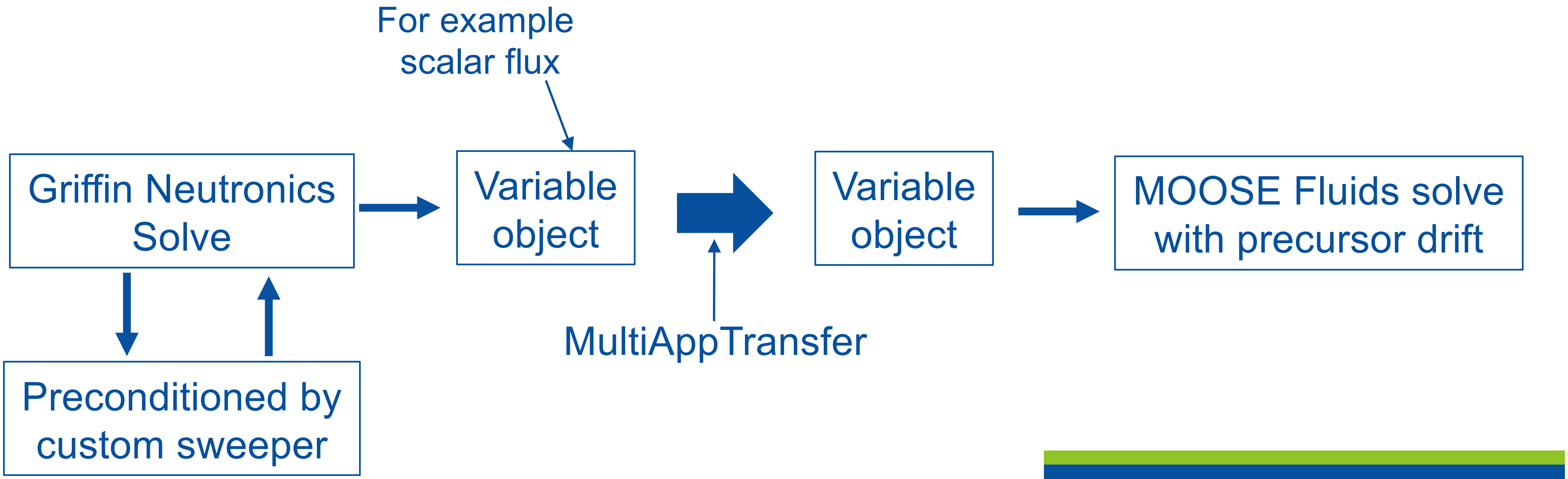
Disadvantage:

- PJFNK/Newton requires a good Jacobian (memory + compute heavy) & decent estimate of Jacobian inverse to converge fast
- Jacobians for many problems are either large or hard to invert efficiently without resorting to LU (e.g., fluid dynamics)
- Custom solution algorithms (e.g., SIMPLE in fluid dynamics) are difficult to implement when compared to other frameworks (e.g., openFOAM)
- Semi-implicit or efficient explicit schemes hard to implement and make efficient

Native Communication among MOOSE Apps(1)

- MOOSE Apps share a lot of infrastructure
  - Common mesh object
  - Common pluggable system such as: Variables, Kernels, Materials
  - Inheritance and polymorphism are used to make the objects usable across

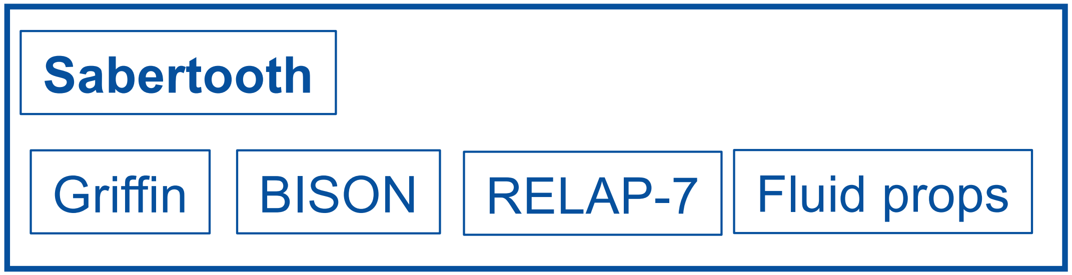
MSR application (Example)



Native Communication among MOOSE Apps(2)

- Concept of code reuse leveraging submodules
- MOOSE apps are built on the same system & inherit from the same basic objects
- You can generally combine two MOOSE apps by “loading” the objects residing in the two apps into a third app

Sabertooth (Example)



Sabertooth

Griffin

BISON

RELAP-7

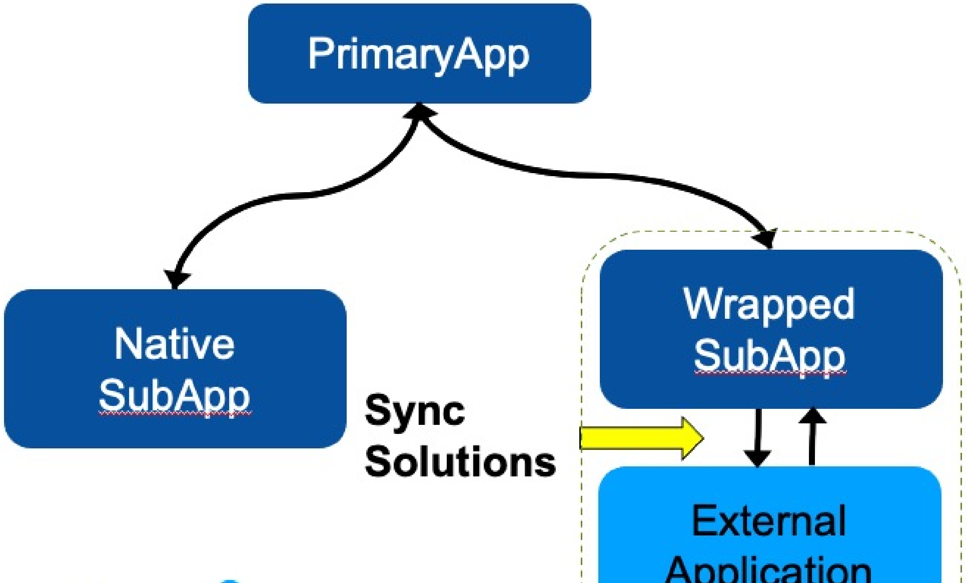
Fluid props

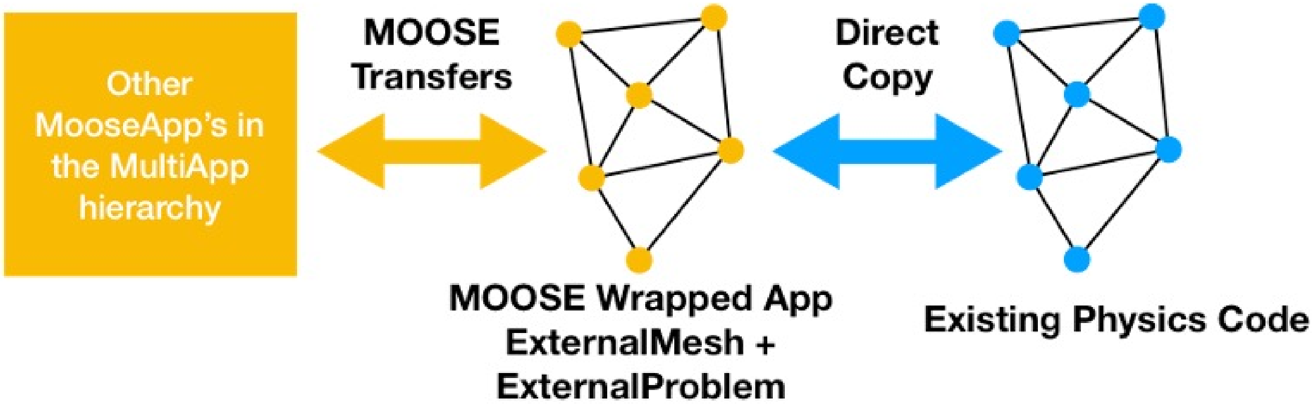
Sabertooth = neutronics + fuel performance + TH

Sabertooth is a superset of the submodule capabilities!

MOOSE-Wrapped Apps

- **MOOSE-Wrapped Apps** are **MOOSE-based executable applications that** are designed to couple external or legacy applications
- Key system 1: external mesh
  - MOOSE external mesh replicates the external code mesh
  - External mesh hooks into MOOSE functionality
- Key system 2: ExternalProblem
- Key system 3: MultiAppTransfers




Other MooseApp's in the MultiApp hierarchy
MOOSE Transfers
Direct Copy
MOOSE Wrapped App ExternalMesh + ExternalProblem
Existing Physics Code

ExternalProblem

- ExternalProblem offers a lot of flexibility to couple existing codes or develop new custom codes that talk to MOOSE

```
syncSolutions(Direction::TO_EXTERNAL_APP);
externalSolve();
syncSolutions(Direction::FROM_EXTERNAL_APP);
```

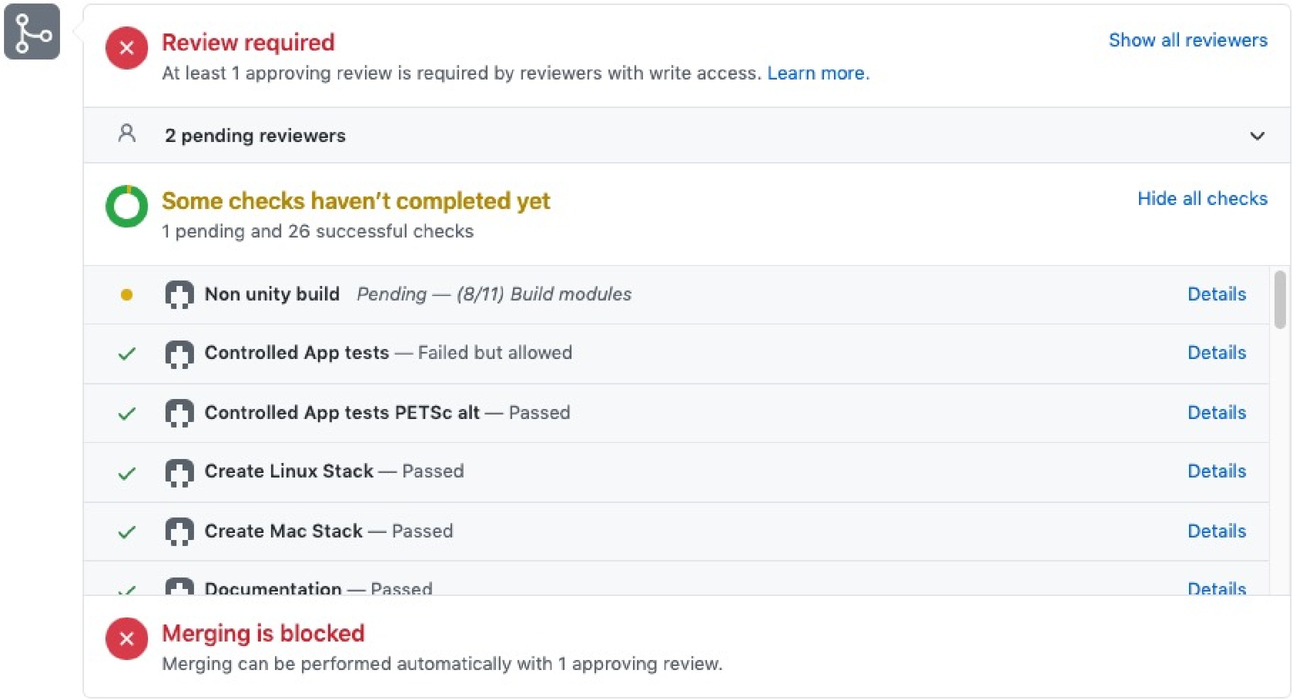
- *`syncSolutions`* ensures that information from an external code is transferred to the external mesh (usually writing information to MOOSE variables)
- *`externalSolve`* should either contain a call to the wrapped app solution algorithm or a custom algorithm can be implemented
- ExternalProblem is used to implement subchannel algorithms into MOOSE (Note: no existing code is wrapped, but a new code is written and hooked up with MOOSE)

MOOSE Development & Testing (1)

- MOOSE and most native apps follows the **continuous integration** workflow
- MOOSE relies heavily on a robust automated testing tool called CIVET
- Development first merges into *next* branch, followed by additional automatic testing and updates to *devel* and *master* branches
- MOOSE is hosted on github; export-controlled applications are hosted in INL internal gitlab servers
- Tests are developed as part of the code and reside in the repository
- Most tests are executed 60 times with varying compilers, parallel execution settings, operating systems

MOOSE Development & Testing (2)

CIVET test dialogue on github pull-request side



Review required

At least 1 approving review is required by reviewers with write access. Learn more.

Show all reviewers

2 pending reviewers

Some checks haven't completed yet

1 pending and 26 successful checks

Hide all checks

- ● Non unity build *Pending — (8/11) Build modules* Details
- ✓ Controlled App tests — Failed but allowed Details
- ✓ Controlled App tests PETSc alt — Passed Details
- ✓ Create Linux Stack — Passed Details
- ✓ Create Mac Stack — Passed Details
- ✓ Documentation — Passed Details

Merging is blocked

Merging can be performed automatically with 1 approving review.

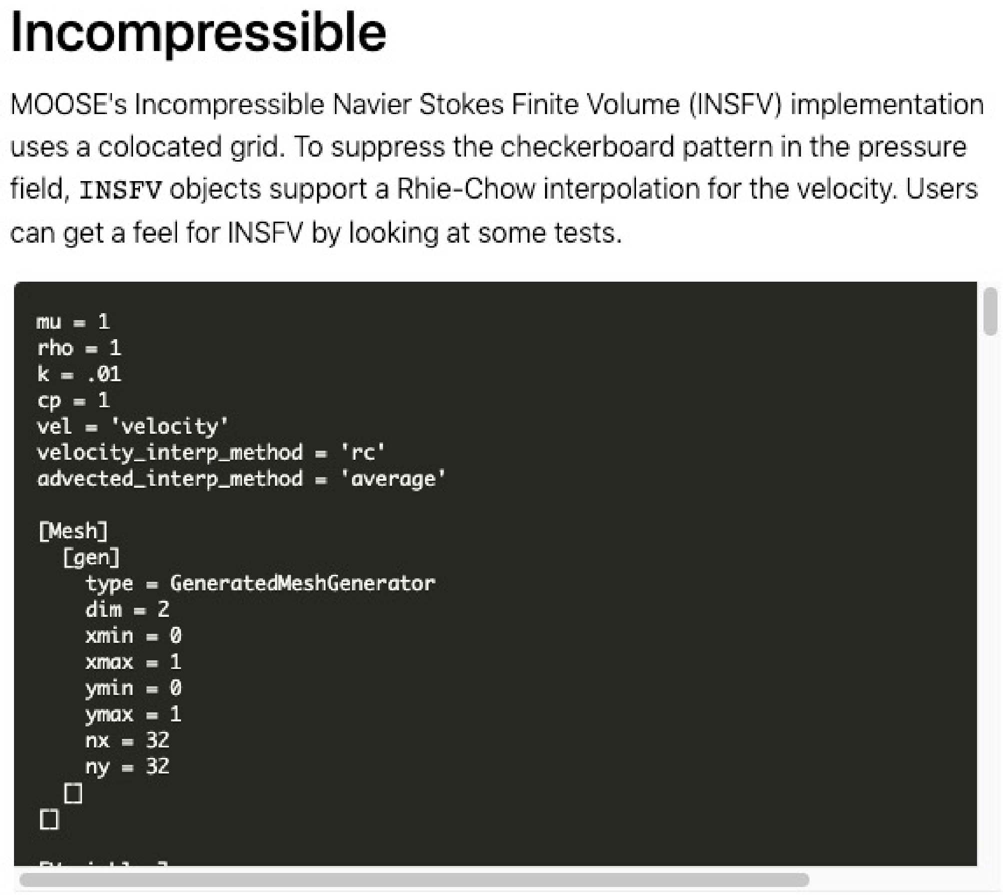
Tests for:

NEW !

Code formatting, Linux, Mac, Windows 10, Documentation, dependent internal and external applications, minimum clang & gcc (compiler) version, parallel execution, multi-threaded execution, recover & restart, debug, different PETSc & solver configs

MOOSE Documentation with MooseDocs

- MOOSE documentation is part of the source code and is version controlled
- Documentation is written in markdown syntax & pulls in source code & test input syntax
- Documentation must pass tests before code changes are accepted



Incompressible

MOOSE's Incompressible Navier Stokes Finite Volume (INSFV) implementation uses a colocated grid. To suppress the checkerboard pattern in the pressure field, `INSFV` objects support a Rhie-Chow interpolation for the velocity. Users can get a feel for INSFV by looking at some tests.

```
mu = 1
rho = 1
k = .01
cp = 1
vel = 'velocity'
velocity_interp_method = 'rc'
advected_interp_method = 'average'

[Mesh]
  [gen]
    type = GeneratedMeshGenerator
    dim = 2
    xmin = 0
    xmax = 1
    ymin = 0
    ymax = 1
    nx = 32
    ny = 32
  □
□
```

(modules/navier_stokes/test/tests/finite_volume/ins/lid-driven/lid-driven-with-energy.i)

```
# Incompressible

MOOSE's Incompressible Navier Stokes Finite Volume
(INSFV) implementation uses a
colocated grid. To suppress the checkerboard pattern in the
pressure field,
`INSFV` objects support a Rhie-Chow interpolation for the
velocity. Users can get
a feel for INSFV by looking at some tests.

:
:

!listing modules/navier_stokes/test/tests/finite_volume/ins/lid-driven/lid-driven-with-energy.i
```

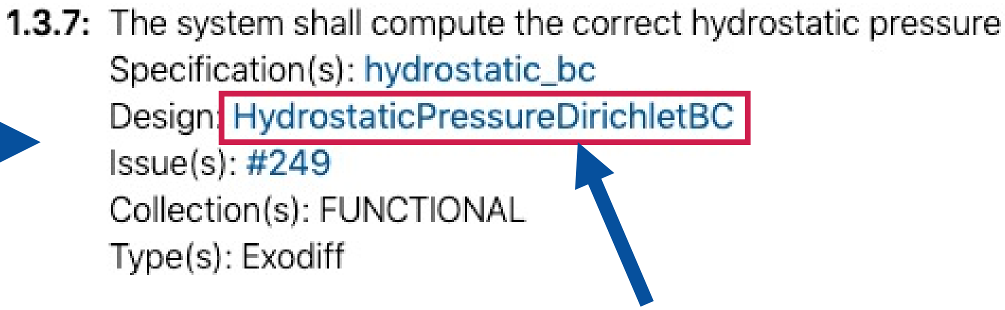
Software Quality Assurance

- INL lays out an NQA-1 compliant process in plan 4005
- MOOSE, Griffin, and many other applications are working towards being compliant with plan 4005
- MOOSE documentation provides tools to automatically generate and stay up-to-date on NQA-1 documentation

Input to test harness (single regression test)

```
[Tests]
  design = 'Hydrostatic[..]BC.md'
  issues = '#249'
  [hydrostatic_bc]
    type = Exodiff
    input = hydrostatic_bc.i
    requirement = ”The code shall …"
  []
[]
```

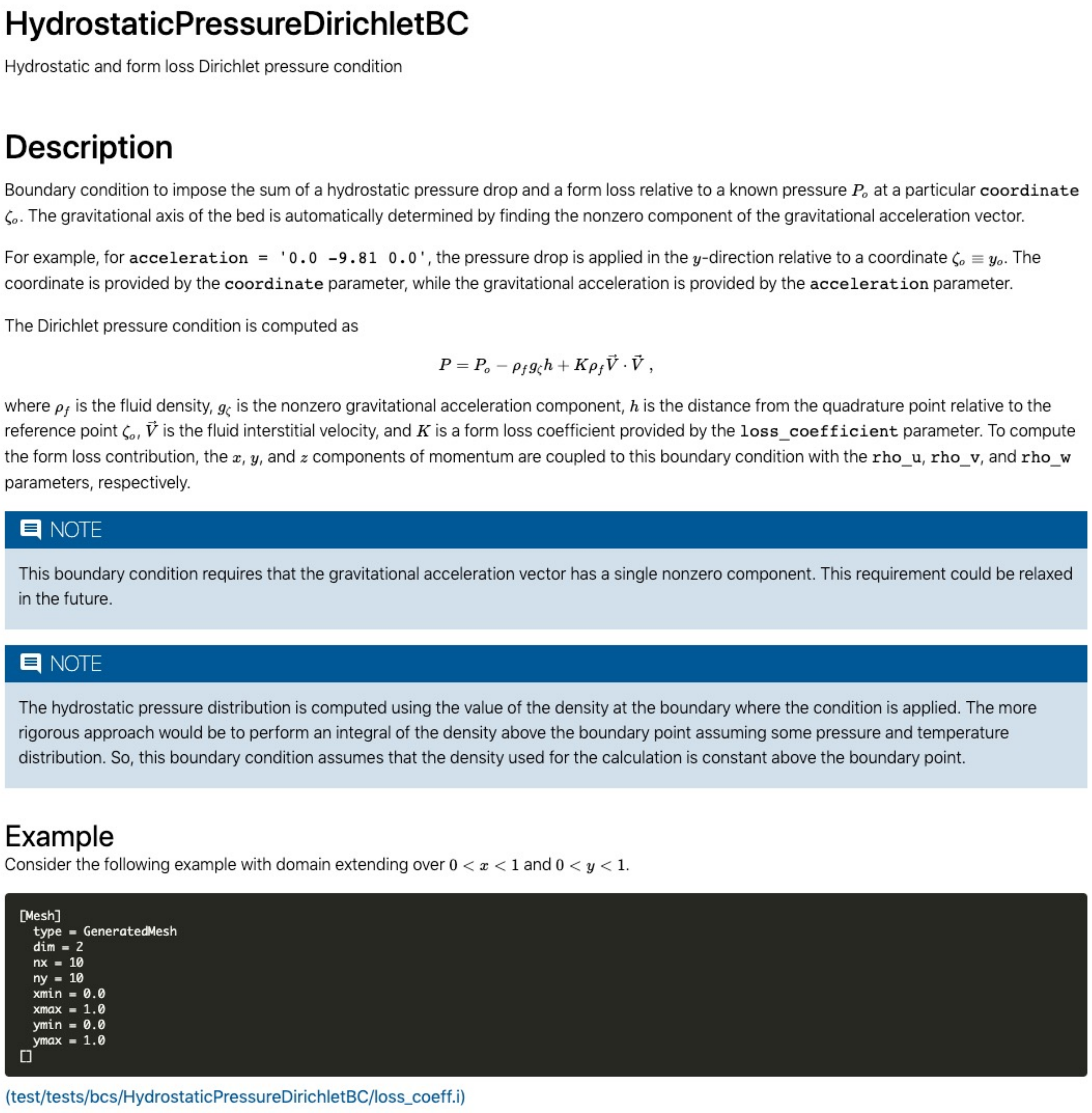
Software Requirement Specification Entry



1.3.7: The system shall compute the correct hydrostatic pressure
Specification(s): hydrostatic_bc
Design: HydrostaticPressureDirichletBC
Issue(s): #249
Collection(s): FUNCTIONAL
Type(s): Exodiff

Link to design document (next slide)

The Requirement Traceability Matrix (RTM) captures all requirements and maps each to the associated design documentation and associated test case



HydrostaticPressureDirichletBC

Hydrostatic and form loss Dirichlet pressure condition

Description

Boundary condition to impose the sum of a hydrostatic pressure drop and a form loss relative to a known pressure P_o at a particular `coordinate` ζ_o. The gravitational axis of the bed is automatically determined by finding the nonzero component of the gravitational acceleration vector.

For example, for `acceleration = '0.0 -9.81 0.0'`, the pressure drop is applied in the y-direction relative to a coordinate $\zeta_o \equiv y_o$. The coordinate is provided by the `coordinate` parameter, while the gravitational acceleration is provided by the `acceleration` parameter.

The Dirichlet pressure condition is computed as

$$P = P_o - \rho_f g_\zeta h + K\rho_f \vec{V} \cdot \vec{V} \ ,$$

where ρ_f is the fluid density, g_ζ is the nonzero gravitational acceleration component, h is the distance from the quadrature point relative to the reference point ζ_o, $\vec{V}$ is the fluid interstitial velocity, and K is a form loss coefficient provided by the `loss_coefficient` parameter. To compute the form loss contribution, the x, y, and z components of momentum are coupled to this boundary condition with the `rho_u`, `rho_v`, and `rho_w` parameters, respectively.

NOTE

This boundary condition requires that the gravitational acceleration vector has a single nonzero component. This requirement could be relaxed in the future.

NOTE

The hydrostatic pressure distribution is computed using the value of the density at the boundary where the condition is applied. The more rigorous approach would be to perform an integral of the density above the boundary point assuming some pressure and temperature distribution. So, this boundary condition assumes that the density used for the calculation is constant above the boundary point.

Example

Consider the following example with domain extending over $0 < x < 1$ and $0 < y < 1$.

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 10
  ny = 10
  xmin = 0.0
  xmax = 1.0
  ymin = 0.0
  ymax = 1.0
[]
```

(test/tests/bcs/HydrostaticPressureDirichletBC/loss_coeff.i)

Concluding Comments

- MOOSE-based tools have been and are being developed to support multiphysics simulations of complex systems.
- MOOSE is a game-changing concept and was a recipient of an R&D 100 Award in 2014. Its open-source distribution has been retrieved world-wide.
- MOOSE's original design and application has evolved since its creation to meet the needs of increasingly complex multiphysics modeling concepts.
- MOOSE-based tools can be rapidly developed based on the PDEs that describe a given set of physical phenomena.
- These tools can be developed for standalone needs, but MOOSE really shines is coupled simulations of different but co-dependent physics.

INL

Idaho National Laboratory

WWW.INL.GOV