MAMMOTH and Rattlesnake Workshop

Reactor physics tools for advanced multiphysics simulations
## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAMMOTH and Rattlesnake Overview</td>
<td>3</td>
</tr>
<tr>
<td>Getting Started</td>
<td>17</td>
</tr>
<tr>
<td>Building a Model</td>
<td>24</td>
</tr>
<tr>
<td>Mesh Generation</td>
<td>26</td>
</tr>
<tr>
<td>Data Preparation</td>
<td>35</td>
</tr>
<tr>
<td>Preparing the Input</td>
<td>40</td>
</tr>
<tr>
<td>The Transport Systems</td>
<td>41</td>
</tr>
<tr>
<td>Neutronics Materials</td>
<td>48</td>
</tr>
<tr>
<td>Using the Aux System</td>
<td>54</td>
</tr>
<tr>
<td>Executioners</td>
<td>69</td>
</tr>
<tr>
<td>Coupling other MOOSE Applications</td>
<td>72</td>
</tr>
<tr>
<td>Strongly Coupled Physics</td>
<td>72</td>
</tr>
<tr>
<td>Tightly Coupled Physics</td>
<td>82</td>
</tr>
<tr>
<td>Detailed Examples</td>
<td>88</td>
</tr>
<tr>
<td>More Examples</td>
<td>97</td>
</tr>
<tr>
<td>References</td>
<td>100</td>
</tr>
</tbody>
</table>
MAMMOTH and Rattlesnake Team

- Mark DeHart
  - Mark.DeHart@inl.gov

- Yaqi Wang (Developer)
  - Yaqi.Wang@inl.gov

- Frederick Gleicher (Developer/Analyst)
  - Frederick.Gleicher@inl.gov

- Javier Ortensi (Developer/Analyst)
  - Javier.Ortensi@inl.gov

- Sebastian Schunert (Developer)
  - Sebastian.Schunert@inl.gov

- Benjamin Baker (Analyst)
  - Benjamin.Baker@inl.gov
What is MAMMOTH?

- a MOOSE-based reactor physics application
- seeks to perform large scale integrated reactor multi-physics calculations on matching and non-matching geometric domains,
- built on MOOSE and inherits all the MOOSE finite element functionality,
- enables a variety of multi-physics coupling schemes (loose, tight and strong)
- able to leverage different MOOSE applications to perform large scale reactor simulations,
- being developed as a general and flexible environment to perform reactor physics analysis to support the following engineering applications:
  - core safety (ATR, TREAT, etc.)
  - transient and experiment design (TREAT),
  - core management (depletion and optimization),
  - fuel performance,
  - advanced reactor concepts (ARC) support,
  - structural Integrity,
  - spent fuel criticality and decay heat analysis
What’s inside MAMMOTH?

The MAMMOTH executable is linked to the following MOOSE applications:

- Rattlesnake: radiation transport
- Yak*: radiation transport module
- BISON: fuel performance
- Relap-7: thermal hydraulics
- Raven: reactor system and safety analysis
- Marmot: micro-structure evolution
- Thermo-chimica: chemistry
**What is Rattlesnake?**

- a MOOSE-based high-fidelity radiation transport application
- solves the linearized Boltzmann transport equation
- solvers and spatial discretizations:
  - CFEM: diffusion, self-adjoint angular flux (SAAF), least squares (LS)
  - DGFEM: diffusion, 1\textsuperscript{st} order
- angular discretizations:
  - spherical harmonics ($P_N$)
  - discrete ordinates ($S_N$)
- arbitrary number of energy groups
- arbitrary anisotropic scattering order
- void treatment in the SAAF ($2^{nd}$ order) formulation
- the method of manufactured solutions is built into the transport framework
Current MAMMOTH and Rattlesnake Capabilities

- Forward and adjoint eigenvalue problems
- Transient problems with various time integrators including the improved quasi static (IQS)
- Criticality search
- Macroscopic depletion
- Decay Heat (ANS standard based on local burnup)
- Computation of PKE parameters during transients
- Fluence determination for reactor internals and RPV ($S_N$ solver)
Future MAMMOTH and Rattlesnake Capabilities

- Microscopic depletion
- Improved decay heat (based on local heating from all isotopes)
- Multi-scale capabilities
- Customizable nuclide inventory
- Gamma transport
- Integrated cross section generation
- Multi-cycle and equilibrium cycle analysis (reshuffling)
LWR Analysis - 2-D Core Assembly-Wise Eigenvalue

- IAEA-2D benchmark
- Standard PWR reactor
- 2 energy groups
- Rattlesnake model:
  - cross sections from the benchmark
  - assembly homogenized
  - 241 elements, quadratic shape functions
  - run-time is < 1 second
- Eigenvalue 1.029696
LWR Analysis - 3-D Core Pin-Cell-Wise Eigenvalue

- Standard PWR reactor
- Start-up core (Cycle 1)
- Rattlesnake model
  - cross sections from DRAGON5
  - 3-D full core mesh
  - pin-cell homogenized
  - 8 energy groups
  - SPH corrected CFEM diffusion
  - 68 million DoFs
  - 35 minutes on 480 CPUs

- Eigenvalues
  - MIT - 0.99920 (OpenMC)
  - INL - 0.99994

INL error in detector fission rates
MIT error in detector fission rates
LWR Analysis - Transient

- LRA-2D benchmark (14-A1)
- BWR model
- super prompt-critical transient
- 2 energy groups
- 2 delayed neutron groups
- diffusion
- adiabatic heat-up
- assembly homogenized cross sections with Doppler feedback

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Averaged power density (W/cm³)</th>
<th>Averaged temperature (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.996368</td>
<td>1094.15</td>
</tr>
<tr>
<td>0.5</td>
<td>1.015445</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5455.46</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>1.44112</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>795.498</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>2.00164</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.38437 \times 10^{-6}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.07545 \times 10^{-6}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.67668 \times 10^{-6}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>432.518</td>
<td></td>
</tr>
<tr>
<td></td>
<td>795.126</td>
<td></td>
</tr>
<tr>
<td></td>
<td>98.4276</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1094.15</td>
<td></td>
</tr>
</tbody>
</table>

Initial k-effective
k-effective with the control rod completely out
The first averaged peak power (W/cm³)
The occurrence of the first peak (s)
The second normalized peak power (W/cm³)
The occurrence of the second peak (s)
Averaged power density at 0.4s (W/cm³)
Averaged power density at 0.8s (W/cm³)
Averaged power density at 1.2s (W/cm³)
Averaged power density at 1.4s (W/cm³)
Averaged power density at 2.0s (W/cm³)
Averaged power density at 3.0s (W/cm³)
Averaged fuel temperature at 3.0s (K)

Geometry

Transient

Crank-Nicholson convergence
HTR Analysis

- OECD/NEA MHTGR-350 Benchmark
  - Rattlesnake model
    - cross sections from DRAGON5
    - 3-D 1/3\textsuperscript{rd} core mesh
    - block homogenized
    - Triangular mesh
    - 26 energy groups
    - 26 million DoFs
  - Eigenvalues
    - Mean value - 1.06743 $\pm$ 92 pcm
    - Rattlesnake - 1.06692

- VHTRC Benchmark
  - Rattlesnake model
    - cross sections HELIOS/SERPENT
    - 3-D full core mesh
    - 1/3\textsuperscript{rd} block homog. and pin resolved
    - 26 energy groups
    - 0.13 and 40 million DoFs
  - Eigenvalues
    - Benchmark - 1.01100
    - Rattlesnake - 1.02036
      (ENDF/B-VII.r0)
The mesh is 3D with two lobes of ATR: (The entire ATR core contains 40 lobes.)

There are 81,004 elements of mixed types.

The 12-group cross sections with P5 scattering was obtained from Serpent.

P5 and no refinement

<table>
<thead>
<tr>
<th>Sn order</th>
<th>k-eff</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1.1746762</td>
</tr>
<tr>
<td>8</td>
<td>1.1745569</td>
</tr>
<tr>
<td>12</td>
<td>1.1745146</td>
</tr>
</tbody>
</table>

The 12-th group flux:

| k-effective results: |

S8 and no refinement

<table>
<thead>
<tr>
<th>Scattering order</th>
<th>k-eff</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.2772911</td>
</tr>
<tr>
<td>1</td>
<td>1.1719575</td>
</tr>
<tr>
<td>2</td>
<td>1.1746945</td>
</tr>
<tr>
<td>3</td>
<td>1.1745509</td>
</tr>
<tr>
<td>4</td>
<td>1.1745562</td>
</tr>
<tr>
<td>5</td>
<td>1.1745569</td>
</tr>
</tbody>
</table>

Diffusion and no refinement

<table>
<thead>
<tr>
<th>Order</th>
<th>Refine</th>
<th>k-eff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1.2471145</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1.2476926</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1.2486009</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1.2481176</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1.2489132</td>
</tr>
</tbody>
</table>

S12-P3 is a good combination. Mesh needs to be refined. We can use AMR to generate a better mesh.
Test Reactor Analysis - TREAT

- Rattlesnake model
  - cross sections from SERPENT
  - 3-D full core mesh
  - block homog. or explicit channels
  - mixed quad and wedge elements
  - 11 energy groups
  - million DoFs
- Minimum Critical Eigenvalue
  - Rattlesnake - 1.00575

![Graphs showing power and fuel temperature over time](image-url)
Pin coupling

- **Rattlesnake model**
  - cross sections from SERPENT
  - 3-D full meshed pellets
  - void gaps and water region explicitly meshed
  - 20 radial regions to capture plutonium rim effect
  - depleted at a constant power level for a year
  - explicit dishing and chamfering

- **BISON model**
  - Fuel and cladding
  - $UO_2$ thermo-material properties internal to BISON

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel Pellet Radius</td>
<td>4.09575 mm</td>
</tr>
<tr>
<td>Fuel Pin Initial Temperature</td>
<td>600 (K)</td>
</tr>
<tr>
<td>Outer Cladding Thickness</td>
<td>0.5715 (mm)</td>
</tr>
<tr>
<td>Initial Gap Thickness</td>
<td>0.08255 (mm)</td>
</tr>
<tr>
<td>$UO_2$</td>
<td>4.45 wt %</td>
</tr>
<tr>
<td>Cladding Initial Temperature</td>
<td>600 (K)</td>
</tr>
<tr>
<td>Water Temperature</td>
<td>585 (K)</td>
</tr>
<tr>
<td>Pitch</td>
<td>12.5984 (mm)</td>
</tr>
<tr>
<td>Linear Heat Rate</td>
<td>19.2 Wm</td>
</tr>
</tbody>
</table>

Rattlesnake and BISON mesh

Power Density and Fuel Temperature profile at 350 EFPF.
Getting Started
Access to MAMMOTH

- MAMMOTH, Rattlesnake, Yak, Bison, Relap-7, and Pronghorn are hosted on an INL internal gitlab server.
- Moose is hosted on github and is publicly available.
- You must meet requirements to access these codes (no exceptions):
  - You must work for INL or your organization must have a license agreement with INL.
  - INL HPC account: If you work for INL, you can request an account. If you do not work for INL, you have to have a sponsor inside INL.
- Rattlesnake, Yak, Bison, Relap-7, and Pronghorn are available through MAMMOTH if access is available.
- Process to obtain access to MAMMOTH:
  1. (Foreign nationals only) All apps that your request have to be added to your security plan.
  2. You have to apply for an HPC account.
  3. The developers have to add your as a member to access the project (the “animal”) on gitlab. You need to be a member of each application that you will use!
Getting Started - I

- If you are within the INL domain you can access the INL gitlab server here: https://hpcgitlab.inl.gov
- If you are not connected to the INL domain:
  - Set up a SOCKS proxy with localhost set to 5555.
  - Connect to INL HPC.
    ssh -D 5555 username@hpclogin.inl.gov
- Login with HPC username and password.

- In your projects page you will see a list of "animals" that you can access.
- If you cannot find the idaholab/animal you are looking for in the list, you do not have access granted.
- Select idaholab/mammoth.
A “fork” is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

From the application menu click on the “Fork” button.

The new page should indicate “Forked from idaholab.”

Copy the SSH URL.

Getting Started-III

1. For local installations follow the link in the MOOSE wiki page of your platform to install the redistributable packages.

2. Change directory into projects and use git to clone the code into ∼/projects by pasting the SSH URL.

   ```
   > cd ∼/projects
   > git clone git@hpcgitlab.inl.gov:USERNAME/mammoth.git
   ```

3. Initialize the submodules that you will need and build libMesh.

   ```
   > cd ∼/projects/mammoth
   > git submodule update --init -- ./moose/
   > git submodule update --init -- ./rattlesnake/
   > git submodule update --init -- ./yak/
   ... initialize other submodules as necessary (i.e. bison, pronghorn, relap-7)
   ```

4. The submodule update might fail with the following error message "SSL certificate problem: Invalid certificate chain". In this case do the following:

   ```
   > cd .git
   > Open config in your text editor of choice.
   > Replace "url = https://hpcgitlab.inl.gov/idaholab/mammoth.git"
   by
   "url = git@hpcgitlab.inl.gov:idaholab/mammoth.git"
   ```
5. Build libMesh.

```bash
>cd ~/projects/mammoth/moose/scripts
>./update_and_rebuild_libmesh
```

6. Build MAMMOTH.

```bash
>cd ~/projects/mammoth
>make -j8 (Add -j8 to compile using 8 processors.)
```

7. Run the regression tests to verify that MAMMOTH was build correctly.

```bash
>./run_tests -j8
```

8. Run the regression tests for all compiled applications with:

```bash
>git submodule foreach 'make -j8 test' (FIX ME SEBASTIAN)
```

9. Done.
Keeping the Applications up to Date

1. Update MAMMOTH
   
   `>git pull --rebase upstream devel`

2. Update the submodules
   
   `>git submodule update`

3. Compile
   
   `>make -j8 (Add -j8 to compile using 8 processors.)`

If libmesh has been updated or significant changes have occurred, you might need to clean all applications before building:

`>make cleanall`
`>make -j8 (Add -j8 to compile using 8 processors.)`
Building Models
Building Models - Overview

There are three general items needed for a MAMMOTH or Rattlesnake simulation

- **Mesh(s)**
  - some MOOSE applications might require independent meshes
  - this will require the projection of the solution space
  - try to keep the mesh as consistent as possible to minimize interpolation errors
  - for large # of cross section regions use a material ID variable in the mesh

- **Cross sections**
  - these can originate from any source, but need to be converted to our internal format
  - we provide a basic converter for Serpent, that will be expanded to other codes
  - we plan to add some limited cross section generation capability in the future

- **Input file(s)**
  - some multiphysics problems might require separate input files for each MOOSE application
Generating a Mesh
Meshing - Tools Available

- MOOSE and MAMMOTH based - INTERNAL
  - provide build-in generators for regular **Cartesian** and **Hexagonal** geometries

- INSTANT Mesh Generator - INTERNAL
  - A separate nodal neutronics package (INSTANT) contributes to Yak.
  - Included in *rattlesnake/contrib/instant*
  - Provides a few more geometries like LWR, and PSLG (Planar Straight Line Graph). See INSTANT documentation for more details.
  - INSTANT also provides a mesh utility that analyzes the mesh for hanging nodes or separated domains.

- CUBIT - EXTERNAL
  - U.S. contractors or government facilities have access to CUBIT
  - U.S. universities can obtain university licenses
  - some of our python scripts will be included in *mammoth/contrib/mesher*

- Other EXTERNAL (MeshKit, Gmsh, etc.)
Cartesian Mesh

- Extensions of MOOSE’s GeneratedMesh
- GeneratedBIDMesh - has fixed element sides
- CartesianMesh - allows flexible element sides
- Includes a subdomain parameter, which allows to assign a block (subdomain) ID to each mesh cell.

```
[Mesh]
type = GeneratedBIDMesh
dim = 2
xmin = 0
xmax = 80
ymin = 0
ymax = 80
elem_type = QUAD9
nx = 10
ny = 10
subdomain='1 1 1 2 2 2 2 1 1 1
1 1 1 2 2 2 2 1 1 1
1 1 1 2 2 2 2 1 1 1
2 2 2 3 3 3 3 1 1 1
2 2 2 3 3 3 3 1 1 1
2 2 2 3 3 3 3 1 1 1
2 2 2 3 3 3 3 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1'
uniform_refine = 0
[]
```
Prismatic Mesh - 1/3 Core

- Builds 1/3 core geometry based on material map file.
- Stack maps in material map file to assign SubdomainID to various layers.

```
<Mesh>
type=PrismaticReactorMeshThird
material_map_file = m3.dat
generated_mesh_filename = 'Hex_3.e'
mesh_pattern = SIMPLE
rotation = 60
fuel_assembly_pitch = .36
delta_z = '4'
axial_divisions = '4'
left = 1
right = 2
outer = 3
bottom = 4
top = 5
[
```

Contents of m3.dat

```
75 76 77 78 79 80 81 82 83  
59 60 61 62 63 64 65 66 84  
45 46 47 48 49 50 51 67 85  
33 34 35 36 37 38 52 68 86  
23 24 25 26 27 39 53 69 87  
15 16 17 18 28 40 54 70 88  
 9 10 11 19 29 41 55 71 89  
 5  6 12 20 30 42 56 72 90  
 3  7 13 21 31 43 57 73 91  
 1  2  4  8 14 22 32 44 58 74  
```
Prismatic Mesh - 1/6 Core

- Builds 1/6 core geometry based on material map file.

```
<Mesh>
type=PrismaticReactorMeshSixth
material_map_file = m6.dat
generated_mesh_filename = 'Hex_6.e'
mesh_pattern = SIMPLE
rotation = 30
fuel_assembly_pitch = .36
delta_z = '4'
axial_divisions = '4'
left = 1
right = 2
outer = 3
bottom = 4
top = 5
[]
```

Contents of m6.dat

```
1 2 4 8 14 22 32 44 58 74
3 7 13 21 31 43 57 73 91
6 12 20 30 42 56 72 90
12 20 30 42 56 72 90
19 29 41 55 71 89
28 40 54 70 88
39 53 69 87
52 68 86
67 85
84
```
Prismatic Mesh - 1/12 Core

- Builds 1/12 core geometry based on material map file.

```
<Mesh>
  type=PrismaticReactorMeshTwelfth
  material_map_file = m12.dat
  generated_mesh_filename = 'Hex_12.e'
  mesh_pattern = SIMPLE
  rotation = 30
  fuel_assembly_pitch = .36
  delta_z = '4'
  axial_divisions = '4'
  left = 1
  right = 2
  outer = 3
  bottom = 4
  top = 5
[]
```

Contents of m12.dat

```
1 2 4 8 14 22 32 44 58 74
  7 13 21 31 43 57 73 91
  20 30 42 56 72 90
    42 56 72 90
      71 89
```
Prismatic Mesh - Burnable Poisons and Control Rods

- Control Rods can be added with the following additional entries.
- Here subID 31 in m12.dat is replaced by a list of 6 new materials for CRs.

```
.Mesh
  control_rod_block = '31'
  control_rod_map = '1 2 3 4 5 6'
[]
```

- BP regions can be added with the following additional entries
- Here subID 31 in m12.dat is replaced by a list of 7 new materials for BPs.

```
.Mesh
  burnable_poison_block = '31'
  burnable_poison_map = '1 2 3 4 5 6 7'
[]
```
This exercise teaches the student how to build a mesh with GeneratedBIDMesh.

- **use the sample input block provided →**
- type is GeneratedBIDMesh
- build a 2-D mesh with:
  - dimensions x [0,100] y [0,50]
  - with interval x 10 and y 5
  - element type QUAD4
- **note** that the mesh is in centimeters

```plaintext
<Mesh>
type =
dim =
xmin =
xmax =
ymin =
ymax =
xn =
yny =
elem_type =
subdomain='1 1 1 1 1 1 1 1 1 3
  1 1 1 1 1 1 1 1 1 3
  1 1 1 1 1 1 1 1 2 3
  1 1 1 1 1 1 1 2 2 3
  3 3 3 3 3 3 3 3 3 3'
uniform_refine = 0
[]
```
Meshing - Checking your Mesh

- Often times convergence or solution shape problems are due to the mesh
- Using the INSTANT mesh checker

```bash
# to compile INSTANT and the mesh checker
~/projects/mammoth/rattlesnake>make instant -j8

# to execute the mesh checker under rattlesnake
~/projects/mammoth/rattlesnake/contrib/instant/instant_mesh_generator-opt inp.xml

# see the INSTANT manual for details on how to prepare the input file in
```

- Use the Debug block in MOOSE with:

```yaml
[Debug]
    check_boundary_coverage = 1
    print_block_volume = 1
[]
```
Data Preparation
MAMMOTH

- A multi-group reactor physics database includes among other items these basic datasets:
  - neutron and gamma cross sections (cross sections, diffusion coefficients, velocities)
  - delayed neutron (delayed neutron fractions and spectrum)
  - depletion data (decay constants, branching ratios, Q values, other decay chain information)

- Present state and Future Development
  - MAMMOTH and Rattlesnake support both INSTANT and YAKXS formats (both internal INL formats) for cross sections and delayed neutron data.
  - We are working on our decay and reaction library formats.
  - Preliminary capability to convert SERPENT to YAKXS and INSTANT format is available.
  - Support planned for the following external neutron cross section formats: DRAGON-5, SERPENT (branch), OpenMC and AMPX.
YAKXS

- YAKXS (Yak cross section library) is a modern XML (eXtendable Markup Language) multigroup cross section library format.
- Cross section processing functions, including interpolation, mixing, etc., are provided along with the library format.
- It is designed to provide a general tool for radiation transport calculations but not limited to be used only by Yak.
- The source files are located under `rattlesnake/contrib/yakxs`.
- The main source file is `yakxs.C`, which defines the MultigroupLibrary, MixingTable and Mixture.
- A separate document for YAKXS can be found under `rattlesnake/contrib/yakxs/doc`.
YAKXS Graphical View

YAKXS

MultigroupLibrary

\[ \sigma_{x,i}(V_1, V_2, V_3, V_4, \ldots), \quad i = 1, \ldots, N, \quad x = 1, \ldots, K(G) \]

YAKXSCreator

MixedMultigroupLibrary

\[ \Sigma_x(V_2, V_3, V_4, \ldots), \quad x = 1, \ldots, K(G) \]

Mixture

\[ \Sigma_x(V_1, V_2, V_3, V_4, \ldots), \quad x = 1, \ldots, K(G) \]

Interpolation

MixingTable

\[ \sigma_{x,i}(V_1, V_2, V_3, V_4, \ldots), \quad i = 1, \ldots, N, \quad x = 1, \ldots, K(G) \]

Reduction

Mixing

Macroscopic cross section holders:

InputXS

FunctionInputXS

PerturbedInputXS
YAKXS - Parametric Micro/Macroscopic Cross Sections

```xml
<Table>
  <GridIndex Name="1 1 1 1"></GridIndex>
  <Isotope Name="pseudo">L="1", I="6" NS="1"/></Isotope>
  <Total index="g">4.42858E+01 7.27638E-01</Total>
  <Removal index="g">2.93925E+02 3.25622E-01</Removal>
  <Transport index="g">2.41737E-01 7.39292E-01</Transport>
  <nuFission index="p">2.3875E-02 6.09648E-01</nuFission>
  <Fission index="g">1.0202E-02 2.50091E-01</Fission>
  <kappeFission index="g">3.32101E-13 7.77362E-12</kappeFission>
  <FissionSpectrum index="g">1.08006E+00 1.36579E+00</FissionSpectrum>
  <Absorption index="g">2.86519E-02 3.24470E-01</Absorption>
  <Scattering index="p1">profile=1', han21=false'
    <Profile>
      1 2 1 2 1 2
      <Value>
      </Value>
    </Profile>
  </Scattering>
  <DNSFraction index="g">2.26657E-04 1.2129E+03 1.83565E-03 2.75113E-03 1.23971E-03 5.12022E-04</DNSFraction>
  <DNSpectrum index="g">1.00000E+00 1.45916E-07 9.99999E-01 6.49216E-07 1.00000E+00 4.64781E-07 1.00000E+00 3.00498E-07
    <NeutronVelocity index="g">1.98558E+00 1.00000E+00 1.40234E-07</NeutronVelocity>
  </DNSpectrum>
</Table>
```
Preparing the Input
**MAMMOTH versus Rattlesnake Recap**

- **Rattlesnake** solves the multigroup neutron transport equation
  - forward and adjoint eigenvalue
  - steady state source problem
  - time dependent with precursor equations (0-D PKE and 3-D spatial kinetics)

- **Rattlesnake** can also solve the radiative transfer equation (RTE) (but not covered in this training)

- **MAMMOTH** controls the execution of various calculations needed in multiphysics simulations
  - perform upper level computations that require multiple solution spaces (e.g. depletion, feedback, SPH correction, optimization)
  - calls Rattlesnake to obtain flux solutions (e.g. eigenvalue calculation)
  - can call BISON, Relap-7 or Pronghorn to obtain thermal and fluid fields
The Transport Systems
Multigroup Neutron Transport Equation

The neutron transport physics can be modeled with the multigroup neutron transport equation:

\[
\frac{1}{v_g} \frac{\partial}{\partial t} \psi_g(\vec{r}, \vec{\Omega}, t) + \vec{\Omega} \cdot \vec{\nabla} \psi_g + \sigma_{t,g}(\vec{r}) \psi_g = S_{\text{ext},g}(\vec{r}, \vec{\Omega}, t) + \sum_{g'=1}^{G} \frac{\chi_{p,g}(\vec{r})}{4\pi} \int_{4\pi} \psi_{g'}(\Omega') d\Omega' + \sum_{d=1}^{I} \chi_d, g(\vec{r}) \lambda_d C_d(\vec{r}, t) + \sum_{g'=1}^{G} \sigma_s, g'(\vec{r}) \int_{4\pi} \rho_{g' \rightarrow g}(\vec{r}, \vec{\Omega}' \cdot \vec{\Omega}) \psi_{g'}(\Omega') d\Omega', \quad g = 1, \ldots, G.
\]

- \( g \) - group index. \( p, d \) - prompt, delayed. \( G \) - number of energy groups. \( I \) - number of delayed neutron groups.
- independent variables: \( t \) - time; \( \vec{r} \) - position; \( \vec{\Omega} \) - streaming direction.
- \( \psi_g \): angular group flux in group \( g \) (the primary variable!).
- \( C_d \): neutron precursor concentration.
- \( v_g, \sigma_{t,g}, \chi_g, \nu, \sigma_{f,g}, \sigma_{s,g} \): group averaged neutron speed, total cross section, fission spectrum, fission yield, fission cross section, scattering cross section.
- \( \rho_{g' \rightarrow g}(\vec{r}, \vec{\Omega}' \cdot \vec{\Omega}) \): scattering kernel.
- \( S_{\text{ext},g} \): external source.
- Notes:
  - fission and fission-related delayed neutron production are special for neutron transport;
  - the equations for delayed neutron precursors are not given here;
  - boundary conditions on the incoming angular fluxes are required.
  - because of the fission production with \( (\nu > 1) \), neutron population of a system can be sustained without external source;
  - scattering can be treated with spherical harmonics expansion with relatively a low scattering order for neutron transport;
  - multiphysics feedback on neutron distribution are in place through those cross sections.
Choosing the Discretization Schemes

- The multigroup neutron transport equation must be discretized to be solved numerically.
- FEM (finite element methods) are used exclusively for spatial discretization; $S_N$ (discrete ordinates method) or $P_N$ (spherical harmonics expansion method) are used for angular discretization (Diffusion can be thought as a special case of $P_N$); Method of lines is applied to time, i.e. time integration schemes are independent on neutron transport and any of them can be used.

- Currently supported schemes by Rattlesnake:

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Mathematical adjoint</th>
<th>Neutron</th>
<th>Thermal radiation</th>
<th>Transient</th>
<th>Multiscale</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFEM-Diffusion</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DFEM-Diffusion</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>SAAF-CFEM-SN</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SAAF-CFEM-PN</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LS-CFEM-SN</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>LS-CFEM-PN</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>DFEM-SN</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>DFEM-PN</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Angular Discretization Methods in Rattlesnake

- **Discrete ordinates (S\textsubscript{N})**
  - Transport equation solved along discrete streaming directions.
  - The directions in the streaming term are decoupled.
  - The computing effort is increased about linearly with the increased number of streaming directions.
  - Accuracy depends on order and type of angular quadrature.
    - Level symmetric (maintains symmetry - cheap since less unknowns)
    - Gauss-Chebyshev (allows specification of polar and azimuthal directions)
    - Bickley3-Optimized (production quadrature for 2D)
      - Prone to ray effects (or solution singularities).

- **Spherical harmonics expansion (P\textsubscript{N})**
  - Directional dependence approximated by series of spherical harmonics functions.
  - Unknowns are the directional expansion coefficient or angular moments.
  - Streaming term couples all angular moments.

- **S\textsubscript{N}** is typically more suitable for heterogeneous calculations, while **P\textsubscript{N}** is mainly for problems with the significant spatial homogenization.
FEM for Spatial Discretization in Rattlesnake

- Continuous FEM
  - stabilization scheme needs to be applied for hyperbolic system like the neutron transport equation
  - pros most of existing mesh frameworks support directly; AMG (algebraic multi-grid method) can be used.
  - cons having requirements on the mesh quality.
  - cons memory intensive because of storing the preconditioning matrix.
  - cons lack of local conservation.
  - cons high-order expansion functions are expensive.

- Discontinuous FEM
  - simple upwind scheme
  - DFEM can be solved with the transport sweep
  - pros highly memory efficient with the matrix-free scheme;
  - cons difficulties associated with transport sweep: massive parallelization, unstructured mesh, etc.

- Rattlesnake currently does not have a good transport sweeper. So for large size applications you will want to choose CFEM.
The Input Syntax of Transport Systems - Overview

- Allows users to specify:
  - transport equation to solve
  - discretization scheme
  - spatial domain where transport equation is solved
  - spatial functional space
  - angular discretization
  - angular discretization settings
  - scattering order (number of anisotropy)
  - and etc...

- Multiscale can be easily turned on by providing multiple block-restricted discretization sub-blocks.
Neutronics Materials
Neutronics Materials - Overview

- It declares and evaluates material properties (cross sections) required by neutronics computations for both diffusion and transport, steady-state and transient.
- Types of material (Rattlesnake user manual contains full list):
  - **Constant** - specifies a constant set of macroscopic cross sections
  - **Mixed** - creates a fixed set of macroscopic cross sections by interpolating and mixing from a tabulated cross section set
  - **CoupledFeedback** - similar to Mixed, but with on-the-fly interpolation and mixing from a tabulated cross section set
  - **Function** - all macroscopic cross sections can be a function varying in space and time. Suitable for doing MMS or some complicated spatial kinetics benchmark calculations
  - **CRodded** - to model control rod movement within a block
- Volume correction of cross sections is available if the meshed volume is different from the 'real' volume.
- We will discuss two materials: 1) **Constant** and 2) **CoupledFeedback**.
Neutronics Materials - Assigning cross sections

1. Using the block number
   - simplest XS assignment
   - uses the 'block' parameter to match 1 block # to 1 XS ID in library.
   - used for a small # of XS regions.
   - can be automated using the last two digits of the block # to match library ID.

2. Using a material ID variable
   - when the number of material regions in the problem is large, the material input block can become extremely long.
   - the memory usage increases for large numbers of blocks (> 100).
   - assigns cross sections based on a material ID variable written to the mesh.
Can input cross sections by either:

- reading from a file
- direct input

In direct input we specify the values of the main cross sections: total, scattering, nu fission, fission spectrum, diffusion coefficients, removal, fission, and kappa fission.

Note that some of these can be internally calculated - depending on calculation type (diffusion vs transport).

Other data like neutron speed and delayed neutron data is also included in the material specification.
**Neutronics Materials - CoupledFeedback**

- **CoupledFeedbackNeutronicsMaterials** is used with parametric data $\Sigma(\text{Burnup, fueltemp, modtemp, bor, ...})$.
- Tied to a BaseLibObject (userobject), which loads cross section data and interacts with the material.
- The material will use the values of MOOSE variables (burnup, fuel temperature) at each quadrature point to obtain an interpolated table.
- The MOOSE variables can either be primal or Aux, but order of execution will be important for the latter.
- The material will call a mixer to generate a set of macroscopic cross sections.
- Requires the isotope names and number densities of initial isotopes. The use of isotope "pseudo" indicates they are macro cross sections with number density 1.0.
Some specialized versions of the neutronics materials allow automatic assignment of the material_id based on either:

- last two digits of the block number
- the value of a material_id variable in the mesh file

This specialized neutronics materials use the same naming convention but with "MatID" inserted after the base name:

- ConstantMatIDNeutronicsMaterial
- MixedMatIDNeutronicsMaterial
- CoupledFeedbackMatIDNeutronicsMaterial

If reading a material ID variable from the mesh file make sure that the mesh distribution is serial. Failure to do this will:

- affect the automated assignment of cross sections
- produce wrong answers
User Objects, Postprocessors, AuxVariables and AuxKernels
Integrate a Variable over the Domain

- You have the scaled power density: total_power_density \( (p(\vec{r}, t)) \) and want to compute total power \( P(t) \)

\[
P(t) = \int_V dV p(r, t)
\]

- Simple example of using ElementIntegralVariablePostprocessor

```
[./power]
  type = ElementIntegralVariablePostprocessor
  block = fuel1
  # this is the scaled power density added by the DepletionAction in MAMMOTH (see AuxVariables section)
  variable = total_power_density

[../]
```
**Subdomain Integrated Fluxes**

- If you want the sum of fluxes integrated over a region (block) in space

\[
\text{FluxIntegral} = \sum_{g=g_s}^{g_e} \int_{V} dV \phi_g
\]

use the FluxIntegral postprocessor.
- Example computes flux integral of the sum of groups 1 and 2 over blocks (regions) 0 and 1 and multiplies it by 1.2.

```plaintext
[./flux_integra]
  type = FluxIntegral
  coupled_flux_groups = 'phi_g0 phi_g1'
  block = '0 1'
  scaling_factor = 1.2
[../]
```
Subdomain Integrated Reaction Rates

- If you want the reaction rates integrated over a region in space and over a sum of groups:

\[
\text{ReactionRate} = \sum_{g=g_s}^{G} \int_{V} dV \Sigma_g \phi_g
\]

- FluxReactionIntegral: vector of scalar material properties ($\Sigma_t$).
- FluxRxnIntegral: vector material property ($\nu \Sigma_f$).

- Example computes the integral of the total reaction rate over groups 2 and 3 and block 1.

```plaintext
[./flux_reaction]
  type = FluxReactionIntegral
  coupled_flux_groups = 'phi_g1 phi_g2'
  cross_section = 'sigma_total_g1 sigma_total_g2'
  block = '1'
[../]
```

- Example computes the integral of the fission neutron production over groups 2 and 3 and block 1.

```plaintext
[./flux_rxn]
  type = FluxRxnIntegral
  coupled_flux_groups = 'phi_g1 phi_g2'
  cross_section = nu_sigma_fission
  initial_group = 2
  block = '1'
[../]
```
Normalization Factor for Eigenvalue Calculations

- Flux obtained in eigenvalue calculations can be arbitrarily scaled
- Scaling factor is obtained by matching total reactor power
- If Depletion block is used the provided reactor power total_reactor_power is correctly scaled
- Example of obtaining scaled power manually (assume two group diffusion):

```plaintext
[AuxVariables]
    ./scaled_power
        order = CONSTANT
        family = MONOMIAL
    ../

[Postprocessors]
    ./scale_pp
        type = PowerModulateFactor
        power_pp = raw_power_pp
        rated_power = 3000e6
        execute_on = 'timestep_begin timestep_end'
        power_modulating = PowerModulator
    ../

    ./raw_power_pp
        type = FluxRxnIntegral
        coupled_flux_groups = 'sflux_g0 sflux_g1'
        cross_section = kappa_sigma_fission
        # block needs to be set to wherever kappa_fission exists
        # block = <where kappa fission exists>
        # this is the correct setting for depletion; always true?
        execute_on = 'timestep_begin timestep_end'
    ../
```


Another example for obtaining scaled fast flux can be found in AuxVariable section.

Scaling in MAMMOTH is tricky because the user can run into cyclic dependencies.
Obtaining Reaction Rates and Fluxes Everywhere

- Use UserObjects VariableCartesianCoreMap and FluxCartesianCoreMap
- **VariableCartesianCoreMap**: Evaluates integrated values of variables on blocks, materials, regions, assemblies, pins, and pin rings/sectors
- **FluxCartesianCoreMap**: Interacts with the TransportSystem to compute the following reaction rates: fission neutron production, power density, neutron absorption, total fluxes, group fluxes
- Can output to dedicated file and/or screen
- Assemblies, pins, and pin rings/sectors can be inferred in two different ways:
  - Define a regular Cartesian grid using input options of UserObject
  - Load variable id values from the mesh file into elemental AuxVariables. (NOTE: variable names are fixed, see manual)
** FluxCartesianCoreMap - Example **

- **Using a cartesian mesh**

```plaintext
[./flux_map]
  type = FluxCartesianCoreMap
  transport_system = diff
  output_in = flux
  regular_grid = true
  grid_coord_x = ‘0 15 30 45 60 75 90 105 120 135’
  grid_coord_y = ‘0 15 30 45 60 75 90 105 120 135’
  execute_on = ‘initial timestep_end’

[../]
```

- **Using elemental variables (see manual for variable names)**

```plaintext
[./flux_map]
  type = FluxCartesianCoreMap
  transport_system = saaf
  print = ‘assembly pin’
  power_map_from = kappa_sigma_f
  execute_on = ‘initial timestep_end’

[../]
```
Auxiliary Variables in MAMMOTH

- Auxiliary Variables $\equiv$ Variables that can be readily computed from other quantities without nonlinear solve
- Auxiliary variables are computed by AuxKernels
- AuxKernels can use nonlinear variables, material properties, postprocessors, etc.
- Example: reaction rates, burnup in macroscopic depletion, normalized reactor power, decay heat, boron content, temperatures (in certain models), scaled fast flux etc.
Macroscopic Depletion & Decay Heat Auxiliary Variables

- Macroscopic depletion solves eigenvalue problems at each time step.
- Requires scaling of the power to get burnup and decay heat magnitude right.
- Depletion block automatically adds:
  - Scaled power density: total_power_density
  - Scaling factor (postprocessor): power_scaling \equiv \text{number to multiply computed flux to get actual reactor flux}
  - Burnup: burnup (currently only one unit - select FIMA — MWD/kg — EFPD — J/cm³ (default))
  - **NOTE**: your library must be tabulated in the unit you select here.
  - This will soon be generalized.
- DecayHeat block automatically adds:
  - Delayed power density: decay_heat_power_density
  - Prompt power density: prompt_power_density
  - Total reactor power density \equiv \text{decay_heat_power_density + prompt_power_density}
Macroscopic Depletion & Decay Heat Actions - Example

[Depletion]
# you have to provide the name of the transport system, in this case ./diff
transport_system = diff
family = MONOMIAL
order = CONSTANT
# you provide a modulating function for the power (0 <= mod <=1).
# power_modulator * rated_power = Reactor operating power
# shutdown mode is entered if this function’s value is zero
power_modulating_function = PowerModulator
# rated power in MW: 3000MW thermal for 50,000 fuel pins
rated_power = 0.06
# unit that burnup will be computed as, currently only one unit and one fuel type can be selected
burnup_unit = MWd/kg
fuel_volume = 150
# in kg / m^3
fuel_density = 10421.5
heavy_metal_isotopes = ’U234 U235 U236 U238’
weight_percentages = ’0.04 4.11 0.0 95.85’

[DecayHeat]
fract_power_file = ’path/to/fp.xml’
decay_heat_blocks = ’1 2 3 4 5 6’
# total_power_density and burnup are added by depletion action
power_density_variable = total_power_density
burnup_variable = burnup
order = CONSTANT
family = MONOMIAL

- Limitations: currently only one fuel type and one burnup unit can be selected
- This limitation will be removed soon
Setting Temperature: Constant and Adiabatic Model

- Setting temperature to a constant value (no aux kernel required):

```
[AuxVariables]
  ./Tfuel
    order = CONSTANT
    family = MONOMIAL
    initial_condition = 566.0
[../]
```

- Use the adiabatic model (local deposition of energy) for temperature

\[
\frac{d(c_p \rho T)}{dt} = p(t) \Rightarrow T(t) = \frac{1}{c_p \rho} \int_0^t d\tau p(\tau)
\]

```
[AuxKernels]
  ./temperature_aux
    type = VariableTimeIntegrationAux
    variable = Tfuel
    variable_to_integrate = total_power_density
    coefficient = 0.732
    execute_on = timestep_end
    # order of the time integration. 2 is default so it’s commented here
    # order = 2
  [../]
```
Setting Boron using a Time-dependent Function

- Boron concentration is often defined using a boron letdown curve
- Boron content is set using a piecewise linear function defined at times $t = 0, 50, 100$ seconds

```bash
# variable definition
[AuxVariables]
[./Boron]
  order = CONSTANT
  family = MONOMIAL
  initial_condition = 500.0
[../]

# function definition
[Functions]
[./BoronContent]
  type = PiecewiseLinear
  x = '0 50.0 100.0'
  y = '500.0 400.0 100.0'
[../]

# AuxKernel definition
[AuxKernels]
[./SetBoronAux]
  type = FunctionAux
  variable = Boron
  function = BoronContent
[../]
```
Extracting the Scaled Fast Flux for BISON

- BISON requires the fast flux for estimating radiation damage on
- NOTE: fast flux from eigenvalue calculation is not properly scaled!
- Depletion actions does not set up the properly scaled fast flux so you have to use FluxNormalizationAux and do it yourself

```plaintext
# variable definition
[ AuxVariables ]
    [. / scaled_fast_flux ]
    # use the same shape function as the primal variable
    [ .. / ]
[

# [ AuxKernels ]
    [. / scaled_fast_flux_aux ]
    type = FluxNormalizationAux
    # this must be the correct name that the transport system assigns to scalar fluxes
    # diffusion: sflux_g<> otherwise flux_moment_g<>_L0_M0
    source_variable = sflux_g0
    variable = scaled_fast_flux
    # this postprocessor is added automatically if Depletion action is used
    normalization = power_scaling
    # execute this when BISON subapp is executed
    execute_on = timestep_end
    [ .. / ]
[
```
Cyclic Dependencies (Advanced)

- Users could run into the dependency issue:
  - MOOSE provides several `execute_on` options: initial, linear, nonlinear, timestep_begin, timestep_end
  - MOOSE provides (limited) dependency resolution capabilities
  - MOOSE execution order for each `execute_on` option: 
    - PPs (PREAUX) → AuxKernels → PPs (POSTAUX)
  - PPs are POSTAUX by default and only if AuxKernel depends on them, they become PREAUX

- Illegal (cyclic) dependencies AuxKernel → PP → AuxKernel on the same `execute_on`

- Problematic dependency: PP1 → PP2 → AuxKernel, AuxKernel does not know about PP1 and it remains in POSTAUX

- MOOSE does not currently check for this (will be added soon)! User must ensure input is valid

- **NOTE:** The following dependency is correctly resolved:
  PP → AuxKernel → P

- Ways to resolve the cyclic dependency:
  - make them have different `execute_on`
  - try making AuxKernel also depend on PP1 in case PP1 → PP2 → AuxKernel (see FissionSource.C)
Executioners
Executioners - Overview

- List of available executioners or solvers:
  - **Steady**: an executioner in MOOSE framework; for solving steady-state source problem; preconditioned Jacobian-free Newton-Krylov (PJFNK) method through PETSc.
  - **Transient**: an executioner in MOOSE framework; for solving transient problem; preconditioned Jacobian-free Newton-Krylov (PJFNK) method through PETSc with various time integration schemes; Picard iteration for multiphysics can be turned on at each time step.
  - **InversePowerMethod**: an executioner in MOOSE framework; for solving the generalized nonlinear eigenvalue problem; Jacobian-free Krylov method wrapped with the inverse power iterations; Chebyshev acceleration for the power iteration is available.
  - **NonlinearEigen**: an executioner in MOOSE framework; for solving the generalized nonlinear eigenvalue problem; preconditioned Jacobian-free Newton-Krylov (PJFNK) method through PETSc; free inverse power iterations can be activated to ensure the solution converges to the fundamental mode.
  - **Depletion**: a MAMMOTH executioner; for depletion calculations.

- We will discuss **NonlinearEigen**.
Parameters for NonlinearEigen

[Executioner]
[./NonlinearEigen]

bx_norm = (required)

# convergence control
source_abs_tol = 1e-06
source_rel_tol = 1e-50

# advanced
auto_initialization = 1
free_power_iterations = 4
k0 = 1
pfactor = 0.01
time = 0
output_after_power_iterations = 1

# normalization
normal_factor =
normalization =
output_before_normalization = 1

[]

- Name of the postprocessor for evaluating the $|Bx|$ for the eigenvalue
- Absolute tolerance on residual norm
- Relative tolerance on residual norm after free power iterations
- True to ask the solver to set initial
- The number of free power iterations
- Initial guess of the eigenvalue
- The factor of linear residual norm to be reduced per free power iteration or per nonlinear step
- System time
- True to output solution after free power iterations
- Normalize x to make $|x|$ equal to this factor
- Name of the postprocessor for evaluating $|x|$ for normalization
- True to output a step before normalization

- It typically requires few free power iterations at the beginning to get a better initial guess for converging to the fundamental mode.
- Transport system will automatically add the eigen postprocessor and link it to bx_norm.
- Extra parameter can be added for tuning PETSc.
Physics Coupling - Definitions

- **Strong** coupling here refers to the implicit solve for all the unknowns without operator splitting - 1 multiphysics matrix.

- **Tight** coupling refers to the split operator for different grouped physics with an external iteration to resolve nonlinearities.

- **Loose** coupling refers to the split operator where a group of physics is lagged in time.
Strong Coupling
Strong Coupling - Overview

- Currently the physical domain has to be shared and meshing has to match for the implicit solve.
- Currently all physics must be resolved on the same time discretization.
- The Transport Systems will add all neutronics primal variables and kernels automatically.
- Neutronics materials that allow coupling (interpolation of XS) must be defined.
- The rest of the physics (e.g. thermal-fluids, thermo-mechanics, etc.) need to be specified in the same input:
  - Primal variables
  - Primal kernels
  - Materials defining the coefficients for the coupled physics
  - Boundary conditions
Strong Coupling - Exercise

Problem Statement:
- Quarter Fuel Pin with six neutron sub-regions in the fuel pellet
- Temperature obtained in Fuel, Gap, and Clad
- Thermal conduction with heat generated from fission
- Dirichlet Temperature condition specified on the Clad boundary (600K)
- Neutron flux obtained in Fuel, Gap, Clad and Water
- Neutron steady state eigenvalue problem
- Reflective Neutron flux condition specified top, bottom, left and right side boundary
Strong Coupling - Syntax

- Open file exercise_sc_1.i
- Open mesh file to look at blocks and sidesets
- Enter the following:
  - Neutron eigenvalue problem
  - Coarse energy groups - 2
  - B.C. - Reflecting on sidesets ’1 2 3’
  - Solver scheme - CFEM-Diffusion
  - Spatial functional family - LAGRANGE
  - Spatial functional order - FIRST

```plaintext
[TransportSystems]
  particle =
  equation_type =
  G =
  ReflectingBoundary =

  ./diff
    scheme =
    family =
    order =
  [.//]

[]
```
Strong Coupling - Syntax

- Add the primal variable *TempFuel*
  - first order Lagrange family
  - in all blocks 1-8
  - with initial condition set to 800 K.

- Must add additional kernels:
  - *HeatConduction* operating on *TempFuel* in blocks 1-8
  - *CoupledForce* operating on *TempFuel* in blocks 1-8 coupled to the *ScaledPowerDensity*
Strong Coupling - Additional Syntax

• Material - Javinium
  – is a `HeatConductionMaterial`
  – depends on `TempFuel`
  – has a thermal conductivity of 3.4 W/m/K
  – in blocks 1-6

• Material - Javinium2
  – is a `HeatConductionMaterial`
  – depends on `TempFuel`
  – has a thermal conductivity of 100.0 W/m/K
  – in blocks 7-8

• About Javinium’s density
  – constant density (use `GenericConstantMaterial`)
  – the property name is ’density’
  – with a value of of 10900.0 kg/m³
  – in blocks 7-8

```
[Materials]
./Javinium
  type =
  temp =
  thermal_conductivity =
  block =
[../]
./Javinium2
  type =
  temp =
  thermal_conductivity =
  block =
[../]
./JaviniumDensity
  type =
  prop_names =
  prop_values =
  block =
[../]
[]
```
The thermal system requires BCs on TempFuel
- place a NeumannBC on sidesets 1001 1002.
- place a DirichletBC on sideset 1003 with a value of 600 K.

```plaintext
[BCs]
active = 'WaterEdgeTempBC FuelEdgeTempBC'
[./FuelEdgeTempBC]
type =
variable =
boundary =
value =
[../]
[./WaterEdgeTempBC]
type =
variable =
boundary =
value =
[../]
[
]
Tight Coupling
Tight Coupling - Overview

- Tight coupling allows different physics systems to be solved for on different domains and different meshes.
- Tight coupling allows for different time discretization to be applied to different physics.
- Implies one master application and many sub applications (slaves).
- Currently, one must prepare one input file for the master and slaves.
- The master
  - does not necessarily have to be the neutronics,
  - contains the execution call of the sub-application via MOOSE’s Multi-App system,
  - contains the transfer of information to and from the slave,
  - contains aux variables that store the solution information.
- The Transport Systems will add all neutronics primal variables and kernels automatically.
- Neutronics materials that allow coupling (interpolation of XS) must be defined.
Tight Coupling - Problem Statement Neutronics

- Quarter Fuel Pin with six neutron sub-regions in the fuel pellet
- Neutron flux obtained in Fuel, Gap, Clad and Water
- Neutron steady state eigenvalue problem
- Reflective Neutron flux condition specified top, bottom, left and right side boundary
- Here to leverage the Picard iteration system we leverage the Depletion executioner with two timesteps.
Tight Coupling - Problem Statement Thermal Mechanics

- Temperature obtained in Fuel and Clad
- Fuel is allowed to undergo thermal expansion
- Thermal conduction with heat generated from fission
- Dirichlet Temperature condition specified on the Clad boundary (600K)
- Small heat capacity to minimize impact of the temperature time derivative
Add the primal variable *TempFuel*
- first order Lagrange family
- in all blocks 1-8
- with initial condition set to 900 K.
- values are obtained from the multiapp transfer

Add call for multi-app with input file

```plaintext
[AuxVariables]
./TempFuel
  family =
  order =
  block =
  initial_condition =
  ../
[

[MultiApps]
./sub
  type = TransientMultiApp
  app_type = MammothApp
  positions = '0.0 0.0 0.0'
  input_files = exercise_tc_l_sub_solution.i
  ../
[
```
Tight Coupling - Syntax Neutronics and Fuel Performance File

- Add call for multi-app transfers

```
[Transfers]
[./tosub]
type = MultiAppInterpolationTransfer
direction = to_multiapp
multi_app = sub
source_variable = ScaledPowerDensity
variable = ScaledPowerDensity
[../]
[./fromsub]
type = MultiAppInterpolationTransfer
direction = from_multiapp
multi_app = sub
source_variable = TempFuel
variable = TempFuel
[../]
[]
```

- Fuel Performance: Add auxvariables for multi-app transfers

```
[AuxVariables]
[./ScaldedPowerDensity]
[../]
[]
```
Examples - Kobayashi Benchmark

- one-group source problem with void
- Cross sections and source strength:

<table>
<thead>
<tr>
<th>Region</th>
<th>$S$ ($n \cdot cm^{-3} \cdot s^{-1}$)</th>
<th>$\Sigma_f$ ($cm^{-1}$)</th>
<th>$\Sigma_s$ ($cm^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>$10^{-4}$</td>
<td>$0.5 \times 10^{-4}$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

- Reflective boundary conditions are used at the boundary planes $x = 0$, $y = 0$ and $z = 0$, and vacuum boundary conditions at all outer boundaries

<Mesh>
  type = CartesianMesh
dim = 3
dx = ’10 20 10 20’
dy = ’10 40 10 40’
dz = ’10 20 10 20’
ix = ’2 4 2 4’
iy = ’2 8 2 8’
iz = ’2 4 2 4’
subdomain_id = ’
  1 3 3 3
  2 3 3 3
  2 2 2 3
  3 3 2 3
</Mesh>

<TransportSystems>
  particle = common
equation_type = steady-state
  G = 1
  VolumetricSourceBlock = ’1’
  VolumetricSource = ’1.0’
  VacuumBoundary = ’right front top’
  ReflectingBoundary = ’left back bottom’

  [./saaf]
  scheme = SAAF-CFEM-SN
  AQtype = Level-Symmetric
  AQOrder = 8
  order = FIRST
  hide_angular_flux = true
  second_order = false
[../saaf]
</TransportSystems>
[Materials]
[./region13]
  type = ConstantNeutronicsMaterial
  block = '1 3'
  sigma_t = 0.1
  sigma_s = 0.0
#  sigma_s = 0.05

[../]
[./region2]
  type = ConstantNeutronicsMaterial
  block = 2
  sigma_t = 0.0001
  sigma_s = 0.0
#  sigma_s = 5e-5

[../]

[Postprocessors]
[./norm]
  type = ElementL2Norm
  variable = flux_moment_g0_L0_M0

[../]

[./3A01]
  type = PointValue
  variable = flux_moment_g0_L0_M0
  point = '5 5 5'

[../]

[Executioner]
  type = AMGUpdate
  richardson_max_its = 10
  richardson_abs_tol = 1e-8
  debug = false
  amg_tol = 1e-3
  amg_abs_tol = 1e-9
#  type = Steady
#  petsc_options_iname = '-pc_type -pc_hypre_type
#    -ksp_gmres_restart'
#  petsc_options_value = 'hypre boomeramg 10'
#  l_max_its = 50
#  nl_rel_tol = 1e-12

[Outputs]
  exodus = true
  print_perf_log = true
[./csv]
  type = CSV
  file_base = kobayashi_out
  align = true
  precision = 6
  execute_on = timestep_end

[../]
Examples - Kobayashi Benchmark Results

Scalar flux with ParaView

\[ x = z = 5cm \]

\[ y = 55cm, z = 5cm \]

\[ y = 95cm, z = 35cm \]
Examples - C5G7-2D Benchmark

- benchmark on deterministic transport calculations without spatial homogenization.
- two UOX and two MOX assemblies surrounded by moderator.
- each assembly is made up of 17-by-17 grid of pin cells with 264 fuel pins, 24 guide tubes and 1 centering fission chamber.
- seven-group are given for 7 different materials.
- three of seven groups are fast, i.e. no upscattering; four are thermal:

<table>
<thead>
<tr>
<th>Group</th>
<th>Upper Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20 MeV</td>
</tr>
<tr>
<td>2</td>
<td>1 MeV</td>
</tr>
<tr>
<td>3</td>
<td>500 keV</td>
</tr>
<tr>
<td>4</td>
<td>3 eV</td>
</tr>
<tr>
<td>5</td>
<td>0.625 MeV</td>
</tr>
<tr>
<td>6</td>
<td>0.1 MeV</td>
</tr>
<tr>
<td>7</td>
<td>0.02 MeV</td>
</tr>
</tbody>
</table>

- left and bottom are reflective; right and top are vacuum.
All sections are contained in Geometry.

The Pins section describes all cylindrical and rectangular pin cells.

The Assemblies section describes the pin arrangement of each assembly.

The Core section describes the assembly arrangement, boundary conditions and homogenization level.

The Controls section describes the control parameters for mesh generation.
Examples - C5G7-2D Benchmark - Mesh Generation

Mesh08 (56.6)

Mesh16 (212.0)

Mesh32 (824.3)

Mesh64 (3370.1)

Reflector

- The previous XML geometry file is processed by the INSTANT mesh generator by calling Triangle to generate an ExodusII mesh file.
- Fuel rods are approximated with line segments while volumes are preserved.
- We will use Mesh08.
<Materials>
  <Macros NG="7">
    <material ID="1" NA="0" fissile="true">
      <name>UO2 fuel-clad</name>
      <TotalX5>1.77949E-01 3.29805E-01 4.80386E-01 5.54367E-01 3.11801E-01 3.95168E-01 5.64406E-01</TotalX5>
      <NuFissionX5>2.00592E-01 2.073029734E-03 1.5705991756E-02 4.5183010248E-02 4.3342083920E-02 2.0209009624E-01 5.297105325E-01</NuFissionX5>
      <ChiX5>5.87910E-01 4.11760E-01 3.39060E-04 1.17610E-07 0.00000E+00 0.00000E+00 0.00000E+00</ChiX5>
      <FissionX5>7.21206E-03 0.19301E-04 6.45320E-03 1.85646E-02 1.70094E-02 0.30346E-02 2.16004E-01</FissionX5>
      <KappaFissionX5>7.21206E-03 0.19301E-04 6.45320E-03 1.85646E-02 1.70094E-02 0.30346E-02 2.16004E-01</KappaFissionX5>
    </material>
    <Profile>
      1 1
      1 2
      1 3
      1 5
      4 6
      5 7
    </Profile>
  </Macros>
  <ScatteringX5>
    1.27537E-01
    4.23780E-02 3.24456E-01
    9.43740E-06 1.63140E-03 4.50940E-01
    5.51630E-09 3.14270E-09 2.67920E-03 4.52565E-01 1.25250E-04
    5.56640E-03 2.71401E-01 1.29560E-03
    1.02550E-02 2.65802E-02 8.54850E-03
    1.00210E-08 1.60905E-02 2.73080E-01
  </ScatteringX5>
</Materials>

<material ID="2" NA="0" fissile="true">
<material ID="3" NA="0" fissile="true">
<material ID="4" NA="0" fissile="true">
<material ID="5" NA="0" fissile="False">
<material ID="6" NA="0" fissile="False">
<material ID="7" NA="0" fissile="False">
</Materials>
Examples - C5G7-2D Benchmark

- Set up the input for direct transport solve with SAAF-CFEM-SN.

\[ L[\psi] = S[\Phi] + \frac{1}{k} F[\Phi] \]

- Convert the direct transport solve for transport update for solving:

\[ L\left[\psi^{(n+1)/2}\right] = S\left[\Phi^{(n)}\right] + \frac{1}{k^n} F\left[\Phi^{(n)}\right] \]

- Set up the low order diffusion system and link it to the transport update.

\[ A(\psi^{(n+1)/2}) \left[\Phi^{(n+1)}\right] = S\left[\Phi^{(n+1)}\right] + \frac{1}{k} F \left[\Phi^{(n+1)}\right] \]
Examples - C5G7-2D Benchmark

- Generated with Mesh32, quadratic shape functions, 4 polar directions and 64 azimuthal directions in each octant.
- Axial asymmetry of the fission rate can be observed.
- MOX fuel pins exhibit steeper variation of the fission rate.
More Examples
TREAT PKE - Overview

- Fully implicit adiabatic heating and PKE
- Uses scalar variables and auxiliary variables
- Uses functions with auxiliary variables and primal variables
- Uses initial conditions and array of initial conditions (special for the PKE set)
TREAT Element - Overview

- 3-D TREAT fuel element with homogenized fuel, gap and clad, but with explicit air channels
- Reflective B.C. on sides and Vacuum top and bottom
- CFEM diffusion solution in 11 energy groups
- Uses the transfer system to generate the I.C. form an eigenvalue calculation
- Transient case via boron dilution
- Includes a fully implicit (strongly coupled) thermal field solution
- thermal properties provided with functions
References
References

1. E.E. Lewis and W.F. Miller, Jr., Computational Methods of Neutron Transport.


References


13. Isotopic Analysis of High-Burnup PWR Spent Fuel Samples From the Takahama-3 Reactor (NUREG/CR-6798).


THE END