Basic Griffin Inputs

Mustafa K. Jaradat , Ph.D.

4/21/2024

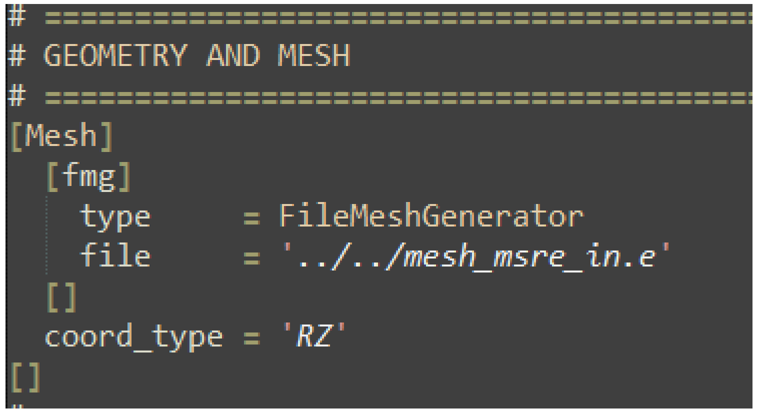
System Overview

- MOOSE syntax for all systems is block based
  - Includes extensive parameter definition to customize based on user needs
  - Check application documentation for available parameters

- MOOSE systems and their function
  - **Mesh** – mesh operations (build, read, modify, … mesh)
  - **Primal/Primary** -- generates variables and physics kernels that represent our PDEs
  - **Auxiliary** -- generates variables and kernels that store or compute data not in the primal system like power density, temperatures, etc.
  - **Material** -- used to assign the values of the coefficients in the PDEs (e.g., cross sections for Griffin)
  - **Output** -- storage of solutions
  - **Action** -- used for input simplification and special problem setup
  - **MultiApps & Transfers** -- used for multiphysics coupling between different application to transfer variables among each other.

MSRE Mesh

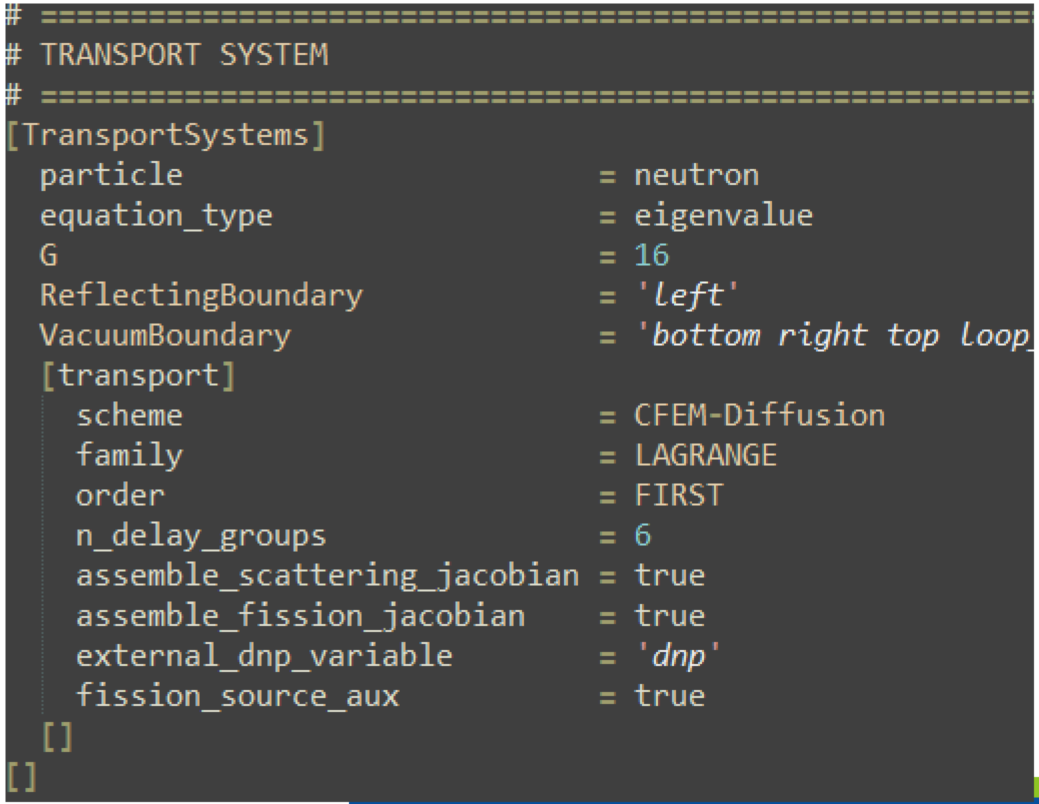
- Mesh System
  - Mesh operations (build, read, modify, … mesh)
- The example below:
  - Uses FileMeshGenerator to read an Exodus formatted mesh file
  - Loads a variable named material_id from the mesh file
  - Sets the coordinate type of the problem



```
# ==========================================
# GEOMETRY AND MESH
# ==========================================
[Mesh]
  [fmg]
    type      = FileMeshGenerator
    file      = '../../mesh_msre_in.e'
  []
  coord_type = 'RZ'
[]
```

Primal System

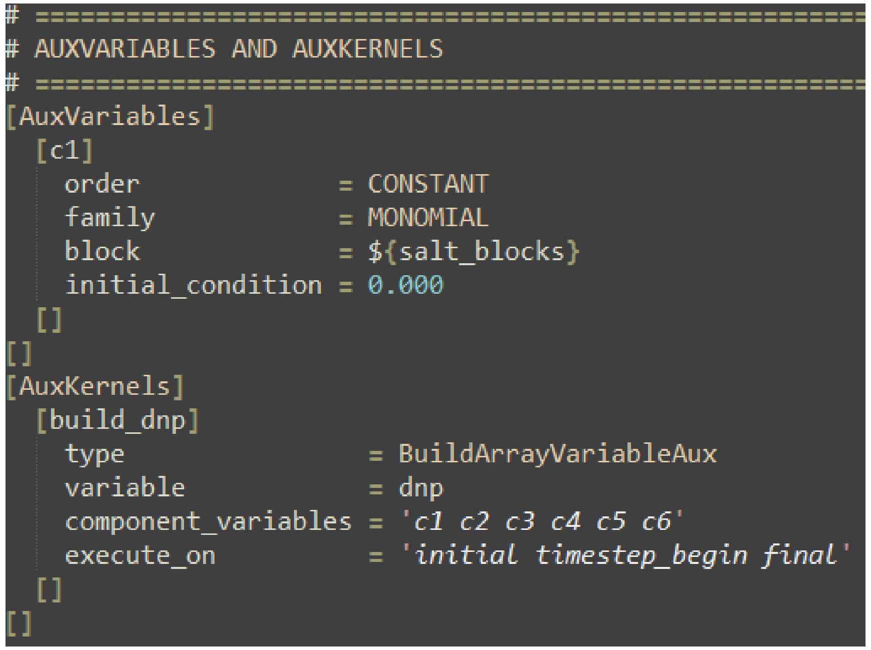
- Primal System
  - In Griffin we use an action to setup the primal system since setting up neutron transport problems requires many kernels
  - In the example we use the TransportSystems action to setup a diffusion problem
    - Particle type
    - Equation type
    - No. neutron energy groups
    - Boundary conditions
    - No. delayed neutron precursor groups
    - Quadratic Lagrange family
    - Jacobian parameters for improved convergence



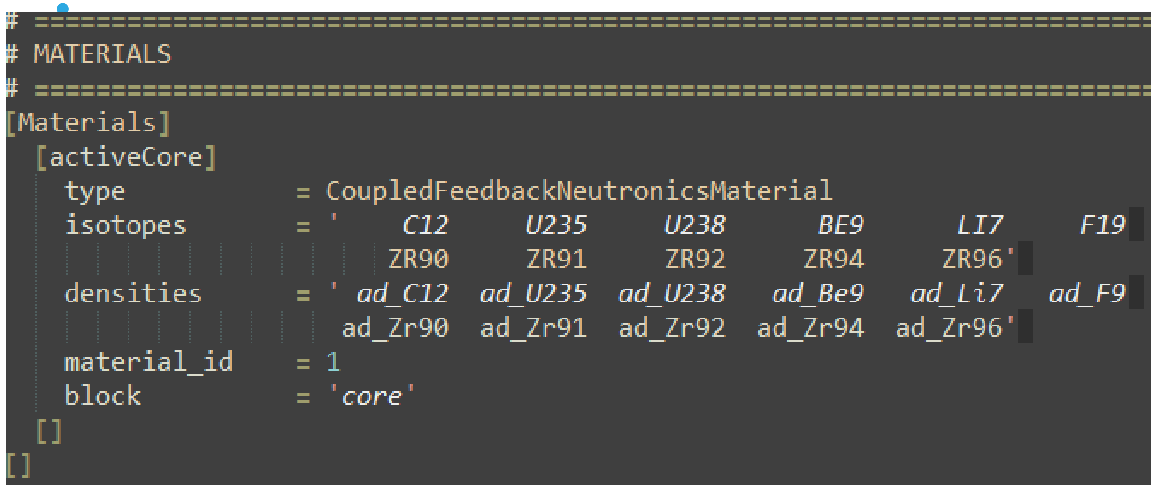
```
# ============================================================
# TRANSPORT SYSTEM
# ============================================================
[TransportSystems]
  particle                           = neutron
  equation_type                      = eigenvalue
  G                                  = 16
  ReflectingBoundary                 = 'left'
  VacuumBoundary                     = 'bottom right top loop
  [transport]
    scheme                           = CFEM-Diffusion
    family                           = LAGRANGE
    order                            = FIRST
    n_delay_groups                   = 6
    assemble_scattering_jacobian     = true
    assemble_fission_jacobian        = true
    external_dnp_variable            = 'dnp'
    fission_source_aux               = true
  []
[]
```

Auxiliary & Material Systems

- Auxiliary System
  - Add DNPCs Aux
    - Add an aux variable named ‘c1’
    - Define mesh block(s) where to evaluate the tally
    - Add an aux kernel named ‘build_dnp’ and store it in variable name ‘dnp’
    - Group all DNPCs ‘c1 c2 c3 c4 c5 c6’ and store it in ‘dnp’ victor variable
    - ‘dnp’ is assigned ‘external_dnp_variable’ in the Primal System
    -



```
# ==========================================================
# AUXVARIABLES AND AUXKERNELS
# ==========================================================
[AuxVariables]
  [c1]
    order             = CONSTANT
    family            = MONOMIAL
    block             = ${salt_blocks}
    initial_condition = 0.000
  []
[]
[AuxKernels]
  [build_dnp]
    type                = BuildArrayVariableAux
    variable            = dnp
    component_variables = 'c1 c2 c3 c4 c5 c6'
    execute_on          = 'initial timestep_begin final'
  []
[]
```

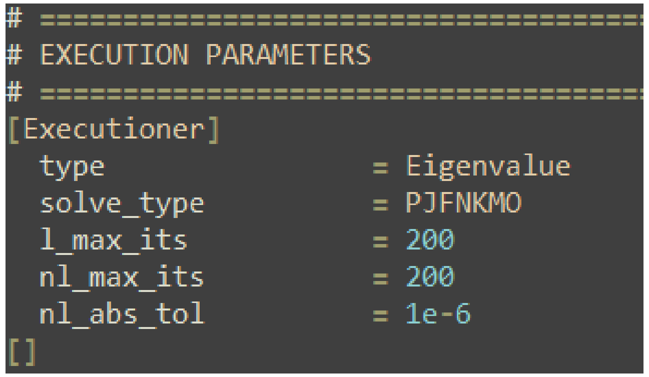


```
# ======================================================================
# MATERIALS
# ======================================================================
[Materials]
  [activeCore]
    type           = CoupledFeedbackNeutronicsMaterial
    isotopes       = '     C12      U235      U238       BE9       LI7      F19
                            ZR90      ZR91      ZR92      ZR94      ZR96'
    densities      = ' ad_C12  ad_U235  ad_U238    ad_Be9    ad_Li7    ad_F9
                       ad_Zr90  ad_Zr91  ad_Zr92   ad_Zr94   ad_Zr96'
    material_id    = 1
    block          = 'core'
  []
[]
```

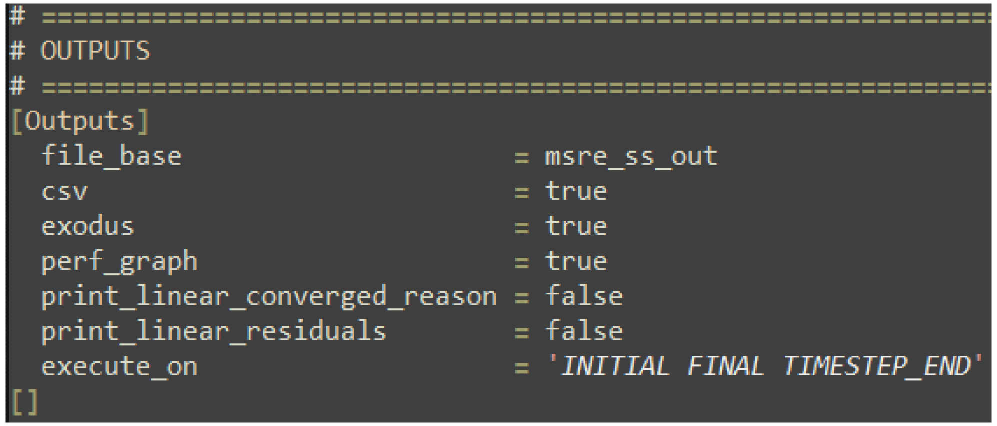
- Material System
  - Use a Griffin material to define the mesh block, isotope names, isotope number densities

Executioner & Output Systems

- Executioner System
  - Eigenvalue (nonlinear power iteration method)
  - PJFNKMO – Matrix only PJFNK
  - Nonlinear tolerance (for Newton iteration)
  - Maximum number of iteration



```
# ====================================
# EXECUTION PARAMETERS
# ====================================
[Executioner]
  type                     = Eigenvalue
  solve_type               = PJFNKMO
  l_max_its                = 200
  nl_max_its               = 200
  nl_abs_tol               = 1e-6
[]
```

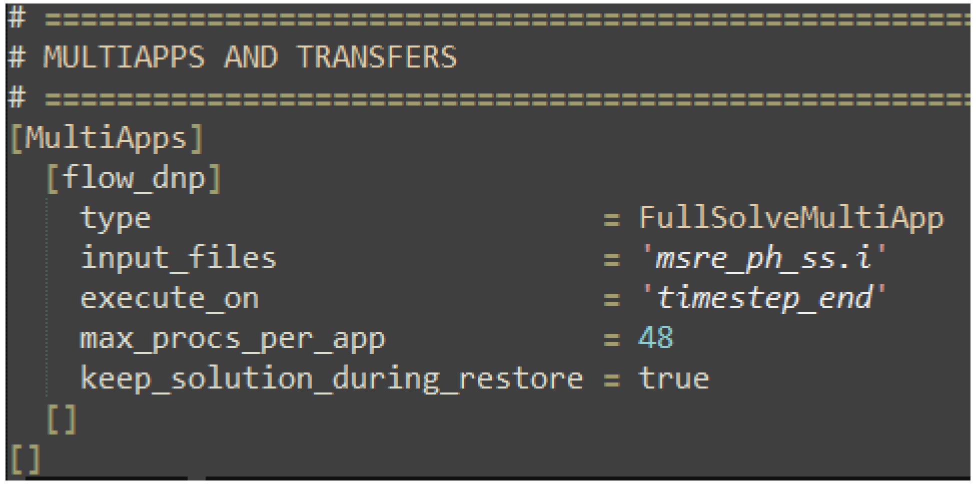


```
# ==============================================================
# OUTPUTS
# ==============================================================
[Outputs]
  file_base                    = msre_ss_out
  csv                          = true
  exodus                       = true
  perf_graph                   = true
  print_linear_converged_reason = false
  print_linear_residuals       = false
  execute_on                   = 'INITIAL FINAL TIMESTEP_END'
[]
```

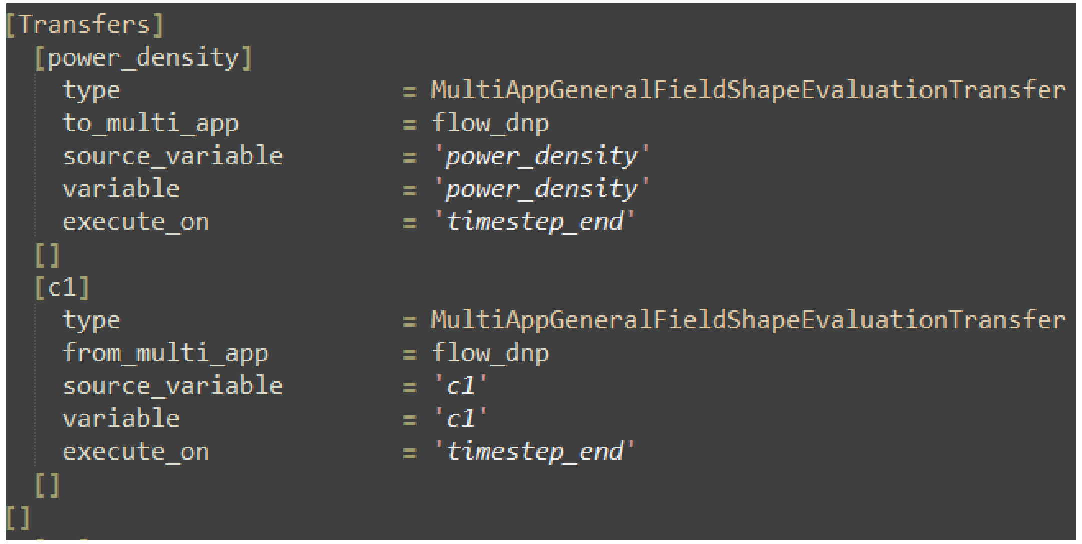
- Output System
  - **File_base** is the root name for output files
  - Outputs requested in Exodus II format and CSV
  - **Perf_graph** is the table with performance metrics
  - User chooses the execution of outputs (initial, timestep_begin, timestep_end, final)

MultiApps & Transfers Systems

- MultiApps System
  - Type of the app (FullSolveMultiApp, TransientMultiApp,)
  - Input file of subapp
  - Execution time (initial, final, timestep_end,)
  - Maximum number of processors
  - Takes the final solution from the previous coupling iteration and re-uses it as the initial guess



```
# ==========================================================
# MULTIAPPS AND TRANSFERS
# ==========================================================
[MultiApps]
  [flow_dnp]
    type                           = FullSolveMultiApp
    input_files                    = 'msre_ph_ss.i'
    execute_on                     = 'timestep_end'
    max_procs_per_app              = 48
    keep_solution_during_restore = true
  []
[]
```



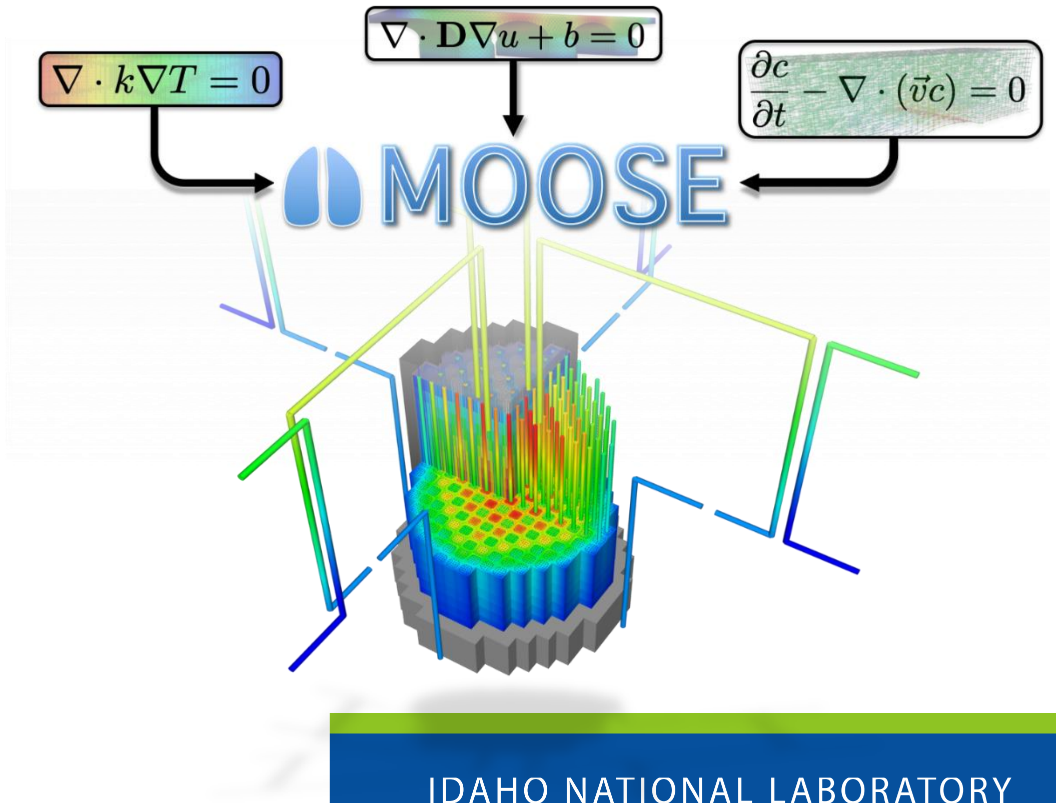
```
[Transfers]
  [power_density]
    type              = MultiAppGeneralFieldShapeEvaluationTransfer
    to_multi_app      = flow_dnp
    source_variable   = 'power_density'
    variable          = 'power_density'
    execute_on        = 'timestep_end'
  []
  [c1]
    type              = MultiAppGeneralFieldShapeEvaluationTransfer
    from_multi_app    = flow_dnp
    source_variable   = 'c1'
    variable          = 'c1'
    execute_on        = 'timestep_end'
  []
[]
```

- Transfers System
  - Type of the transfer
  - Direction of the transfer (from_multi_app, to_multi_app)
  - Transferred and received variables
  - Execution time (initial, final, timestep_end,)

Idaho National Laboratory

MOOSE BASICS

- Multiphysics Object-Oriented Simulation Environment (MOOSE)
- Various spatial discretization methods
  - Finite Element: Continuous and Discontinuous Galerkin
  - Finite Volume: Cell centered variables, interpolation to faces with limiters, orthogonality & skewness correction
- 1D, 2D and 3D Cartesian, 2D RZ, 1D R-Spherical
- Unstructured Mesh
  - Many shapes (polygons)
  - Higher order geometry (curvilinear)
  - Reads and writes multiple formats
- Parallel
- Built-in Postprocessing


$\nabla \cdot \mathbf{D}\nabla u + b = 0$
$\nabla \cdot k\nabla T = 0$
$\frac{\partial c}{\partial t} - \nabla \cdot (\vec{v}c) = 0$
MOOSE

MOOSE BASICS

- Our main purpose is to solve PDEs (like the neutron transport equation)

$$\underbrace{\frac{1}{v}\frac{\partial\psi}{\partial t}}_{\text{Time derivative}} + \underbrace{\Omega\cdot\nabla\psi}_{\text{Streaming}} + \underbrace{\Sigma_t\psi}_{\text{Collision}} = \underbrace{\int_0^\infty\int_{4\pi}\Sigma_s(\Omega'\cdot\Omega, E'\rightarrow E)\psi(\Omega', E')d\Omega' dE'}_{\text{Scattering}} +$$

$$\underbrace{\frac{\chi_p}{k_{\text{eff}}4\pi}\int_0^\infty \nu\Sigma_f(E')\phi(E')dE'}_{\text{Fission}} + \underbrace{\frac{\chi_d}{4\pi}\sum_{i=1}^{I}\lambda_i C_i}_{\text{Delayed Neutron}}$$

- For this purpose, we require
  - A finite element mesh where to solve the PDEs
  - Coefficients for the PDEs (i.e., material properties, neutron cross sections)
  - Instructions on what PDEs to solve (Physics/MOOSE kernels)
  - Instructions on what output to generate
- MOOSE provides various systems to handle these requirements