INL/JOU-23-75569-Revision-0

MOOSE Reactor Module: An Open-Source Capability for Meshing Nuclear Reactor Geometries

November 2022

Emily Shemon, Shikhar Kumar, Shikhar Kumar , Kun Mo, Yeon Sang Jung, Aaron Oaks, Scott Richards, Guillaume Louis Giudicelli, Logan H Harbour, Roy Hulen Stogner

Changing the World's Energy Future

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

INL/JOU-23-75569-Revision-0

MOOSE Reactor Module: An Open-Source Capability for Meshing Nuclear Reactor Geometries

Emily Shemon, Shikhar Kumar, Shikhar Kumar , Kun Mo, Yeon Sang Jung, Aaron Oaks, Scott Richards, Guillaume Louis Giudicelli, Logan H Harbour, Roy Hulen Stogner

November 2022

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

MOOSE Reactor Module: An Open-Source Capability for Meshing Nuclear Reactor Geometries

Emily Shemon,[a]* Yinbin Miao[b], Shikhar Kumar [a], Kun Mo[b], Yeon Sang Jung[a], Aaron Oaks[b], Scott Richards[a], Guillaume Giudicelli[c], Logan Harbour[c], and Roy Stogner[c]

[a] *Argonne National Laboratory, Nuclear Science and Engineering Division, Lemont, Illinois*

[b] *Argonne National Laboratory, Chemical and Fuel Cycle Technologies Division, Lemont, Illinois*

[c] *Idaho National Laboratory, Computational Frameworks Group, Idaho Falls, Idaho*

*Corresponding Author: Emily Shemon

E-mail: eshemon@anl.gov

Address: 9700 S. Cass Avenue, Argonne National Laboratory, Lemont, IL 60439

Tel: (630) 252-4477

Fax: (630) 252-4780

Total number of pages (including cover page): 48

Tables: 1

Figures: 31

Color Print: No (Color online only)

MOOSE Reactor Module: An Open-Source Capability for Meshing Nuclear Reactor Geometries

The Department of Energy (DOE) Nuclear Energy Advanced Modeling and Simulation (NEAMS) program has developed numerous physics solvers utilizing the open-source Multiphysics Object-Oriented Simulation Environment (MOOSE) framework for multiphysics reactor analysis. These solvers require input finite element meshes representing the discretized spatial domain. Typically, reactor analysts turn to licensed tools for creation of reactor geometry meshes. Recently, open-source functionality has been added to the MOOSE framework to mesh common reactor geometries and improve MOOSE-based nuclear reactor application user workflows. The new functionality is primarily contained in the new Reactor Module of MOOSE and includes support for hexagonal pins, assemblies, and cores, extended Cartesian geometry support, options for modelling static and rotating control drums within a hexagonal assembly, core periphery triangulation, and automatic tagging of pin, assembly, plane, and depletion regions for easier post-processing of physics results. A set of reactor geometry mesh builder objects further streamlines the construction of hexagonal and Cartesian cores and allows mapping of materials to regions during mesh generation. The meshes produced with MOOSE's Reactor Module may be used directly within MOOSE-based applications or exported as Exodus II files for use in other finite element solvers. The tools have been demonstrated and verified using a variety of NEAMS physics solvers on a range of reactor applications including a sodium-cooled fast reactor core analysis using Griffin, a fast reactor assembly thermal deformation analysis using MOOSE Tensor Mechanics, and a heat-pipe cooled microreactor coupled analysis using Griffin, Bison, and Sockeye. MOOSE's Reactor module provides significant advantages compared to the use of external meshing tools when analyzing Cartesian and hexagonal reactor lattices using MOOSE-based applications: immediate accessibility (open-source) to the end user, low barrier to entry for new users, speed of mesh generation, volume preservation of meshed fuel pins, and simplification of analysis workflow when used in conjunction with MOOSE-based applications.

Keywords: MOOSE, finite element, reactor, meshing, open source

I. INTRODUCTION

The Multiphysics Object Oriented Simulation Environment (MOOSE) [1] framework is an open-source, parallel finite element framework designed to permit rapid development of robust multi-physics modeling capabilities. The Department of Energy Nuclear Energy Advanced Modeling and Simulation (DOE-NEAMS) program [2] heavily leverages MOOSE to develop and couple advanced physics and engineering applications such as the reactor physics code Griffin [3] and fuel performance code BISON [4]. MOOSE-based physics applications generally require an input finite element mesh on which the physics solution is calculated, reported, and transferred to other physics codes. The mesh is a discretized representation of the physical geometry onto which material properties are assigned and on which basis functions for representing the solution live. Creation of the finite element mesh is often one of the most time-consuming stages of input preparation, particularly for reactor physics analysis where the entire core (and possibly more) needs to be modeled rather than isolated components (single pins, subchannels, or ex-core components). High-fidelity geometry modeling (explicit pins, gap regions, cladding) requires elaborate tracking of groups of elements for material property assignment and output reporting which can be considerably complex for the user to manage.

Licensed, general purpose external tools like Cubit [5] and Ansys [6] are available for creation of reactor geometry meshes. However, these tools require significant user expertise to build even basic meshes, and downstream physics input modifications are sometimes required to adjust for differences in meshed and actual fuel volumes in which mass preservation is crucial. In the past decade, reactor-focused meshing tools such as RGG MeshKit [7] and Argonne Mesh Tools [8] have been developed to accelerate the construction of predictable patterns common in reactors

(Cartesian and hexagonal lattices). These tools greatly reduced user barriers to employing finite element solvers for reactor analysis. MeshKit leverages numerous external dependencies and requires a Cubit license for full functionality. Argonne Mesh Tools is targeted at generating meshes for the PROTEUS [9] transport code and requires a license, although its meshes can be converted into the widely used Exodus II format for use in other tools. There is a need for a simplified, reactor-geometry focused, open-source meshing tool directly compatible with MOOSE-based solvers.

The recently developed Reactor Module in MOOSE [10-13] addresses this need by significantly enhancing MOOSE's ability to mesh common reactor geometries. Support for hexagonal pins, assemblies, and cores has now been added, and Cartesian support has been extended. Options for modeling static and rotating control drums within a hexagonal assembly are now available. Pin, assembly, plane, and depletion regions are distinguished through automatically applied tags on the mesh called "reporting IDs" for easier post-processing of physics results. An external open-source triangle routine [14] has been leveraged within a new MOOSE object to mesh core periphery zones. A set of reactor geometry mesh builder (RGMB) routines further streamlines the construction of hexagonal and Cartesian pin, assembly, and core meshes and conveniently enables the assignment of materials to regions during geometry definition. The meshes generated with the new objects may be used directly within MOOSE-based applications or exported as Exodus II files for use in other finite element solvers.

An overview of the capabilities is given first, followed by a series of examples which demonstrate the use of the Reactor Module for various reactor applications.

II. REACTOR MODULE MESH CONSTRUCTION OVERVIEW

MOOSE leverages the finite element framework provided by the libMesh [15] library. In particular, MOOSE's existing Mesh System [16] packages libMesh operations into useful objects referred to as *Mesh Generators* which allow the user to create a simple mesh and perform operations on it (e.g., rotation, block deletion, stitching) in order to build up larger meshes. The Reactor Module contains specialized Mesh Generator objects which greatly accelerate the construction of patterned nuclear reactor geometries.

The procedure to build up a conventional patterned nuclear reactor core geometry and mesh using the Reactor Module consists of the following steps which act on meshes built in previous steps:

1. Define unique pins in 2D (if applicable)
2. Pattern pins into an assembly, including addition of duct and extra coolant regions (if applicable)
3. Pattern assemblies into a core
4. Delete extraneous assemblies or zones
5. Add peripheral circular reflector or barrel around the core if present
6. Extrude to 3D
7. Label sidesets for boundary condition assignments

Steps 1-2 can be replaced by a homogeneous assembly model if desired. The culmination of this series of steps is a 3D reactor core finite element mesh. Some more advanced features include mesh trimming, which reduces domain size by leveraging symmetry, and identification of "output" zones such as axial zones along a single fuel pin in a specific assembly. These types of operations have been embedded into specialized Reactor Module mesh generators. However, before being processed by a physics solver, materials (and their properties) must be mapped to all zones of the mesh.

Material assignment has also been encapsulated into a streamlined "Reactor Geometry Mesh Builder" set of Mesh Generators for users who want the simplest possible set of input options and workflow. The next several sections of this paper discuss the newly developed Mesh Generator objects which perform the actions in Steps 1-7, as well defining output zones and permitting material assignment.

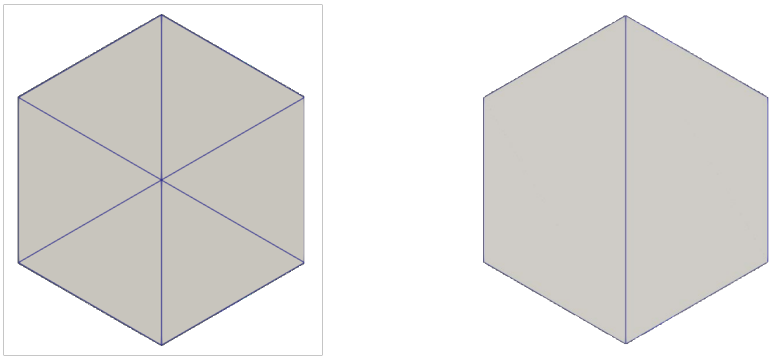
III. HEXAGONAL GEOMETRY MESHING

Hexagonal geometry is commonly found in liquid-metal cooled fast reactor (LMFR) and other advanced nuclear reactor systems such as prismatic gas-cooled reactor (P-GCR) and microreactor (MR) designs. In LMFRs, pins are concentric cylinders representing fuel, bond (or gap) and cladding. A pin cell is the pin surrounded by a hexagonal coolant zone – while not necessarily a physical structure (coolant does not stay within its hexagonal zone), pin cells are essentially repeated units throughout the assembly and therefore are commonly modelled in debugging analysis and multigroup cross section generation. Assemblies consist of pin cells patterned into a triangular lattice; these clusters of pins are often bundled within hexagonal ducts. Cores consist of assemblies placed in a triangular lattice, with neighboring assemblies separated by inter-assembly coolant gaps. A cylindrical reflector region commonly surrounds the core periphery. Several new capabilities are now available to mesh these building block geometries and assemble them into larger cores.

III.A Hexagonal Pin Cells

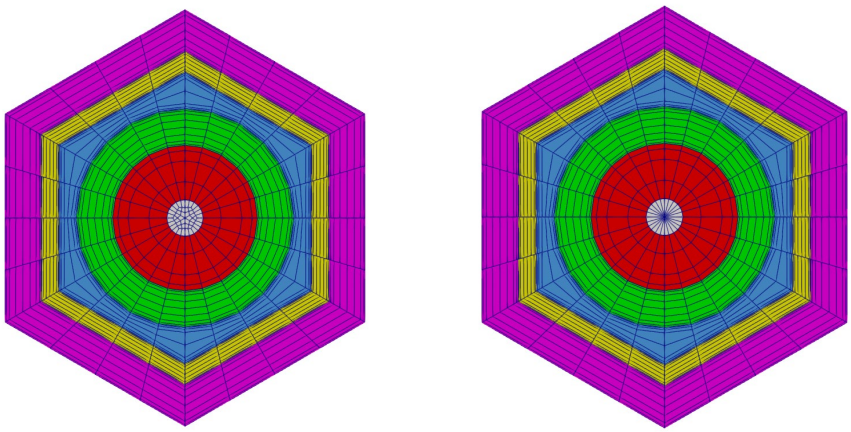
Two primary objects are available for creating hexagonal pin cells: SimpleHexagonGenerator and PolygonConcentricCircleMeshGenerator (abbreviated in subsequent references as *PCCMG*). SimpleHexagonGenerator creates a hexagon subdivided into six equilateral triangles or two quadrilaterals as shown in Figure 1 with

just one required input parameter (size of hexagon) and an option to override the default TRI element option with QUAD elements. The resulting block of elements and external boundary can optionally be renamed and/or renumbered by the user. This type of mesh is useful for homogenized assembly simulations and coarse mesh acceleration techniques. Input syntax for all mesh generators can be found on the MOOSE website [10].



(a) (b)

Figure 1: SimpleHexagonGenerator creates a homogenized pin cell or assembly with (a) six equilateral triangle elements or (b) two quadrilateral elements.



(a) (b)

Figure 2: PolygonConcentricCircleMeshGenerator (PCCMG) creates a heterogeneous pin cell or assembly with optional ring(s) and duct(s) by using (a) quadrilateral elements only, or (b) quadrilateral mesh with triangular center elements. Both meshes have boundary layers defined in all the blocks.

Figure 2 depicts typical meshes generated with PCCMG. This object is generalized for any regular polygon shape and thus can also create square Cartesian pin cells if 4 sides are specified rather than 6 (*num_sides*). By default, the elements in the center zone of the mesh generated by PCCMG are triangular. The user may alternatively select to use quadrilateral elements in the center zone by setting the corresponding parameter (*quad_center_elements)* as *true*. This will result in a fully quadrilateral pin cell mesh. The number of rings and ducts are arbitrary, while the background region (shown in blue) is always present. If rings are present, the meshed volume of the rings are optionally preserved (*preserve_volumes*). The number of azimuthal segments may be selected uniquely for each polygon side. The user has fine-grained control over block and boundary naming and numbering. The typical uses of this mesh generator are to create homogeneous pin cells or assemblies, heterogeneous pin cells (rings present), partially heterogeneous assemblies (ducts present), and control drum (rings and duct present).

PCCMG also supports boundary layer meshing as well as biasing in the radial direction. On both the inner (closer to center) and outer (closer to exterior) sides of each block, the user may define boundary layers by providing layer thickness and radial biasing growth factor values. Boundary layer control is particularly useful for thermal hydraulics simulations. The user can also provide a general growth factor for each block to introduce biased radial meshing in the regions that are not defined as boundary layers.

The meshes generated with SimpleHexagonGenerator and PCCMG contain mesh metadata specifying them as pin-cell level objects. This allows downstream mesh generators to understand information about their compatibility and stitchability.

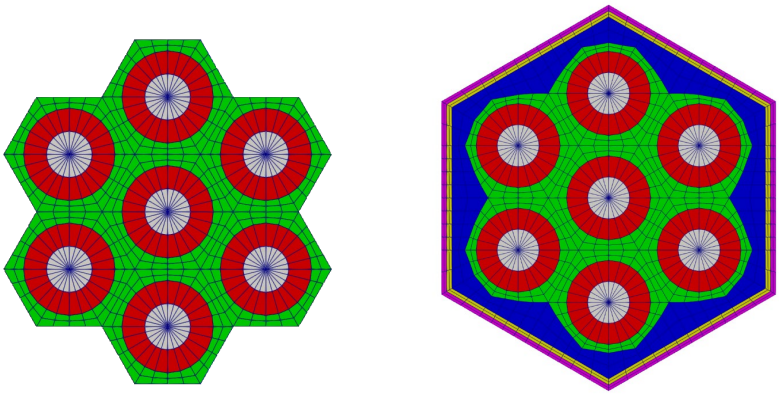
III.B Hexagonal Assemblies and Cores

A few options are available for constructing hexagonal assemblies. PatternedHexMeshGenerator assembles multiple pin cells into a triangular lattice and optionally applies background coolant, duct, and inter-assembly gap regions on the periphery. Assemblies not consisting of repeated pin cells are handled by other mesh generators. SimpleHexagonGenerator generates a coarsely meshed homogenized assembly which can be stitched to other like objects into a core. PCCMG generates a duct heterogeneous assembly (or assembly with one central pin) which can be stitched to other PCCMG objects. Two other assembly mesh generators are available for constructing assembly objects with specialized mesh discretization and geometry, respectively.

III.B.1 Conventional Assemblies and Cores

PatternedHexMeshGenerator assembles hexagonal meshes into a hexagonal grid pattern and optionally adds a background region and ducts around the periphery of the grid. This mesh generator can be used to mesh a hexagonal assembly containing multiple pins, background, and duct regions (*pattern_boundary = hexagon*), or a reactor core containing multiple assemblies (*pattern_boundary = none*). Input meshes to be used in the pattern are specified via the inputs parameter and placed into the grid specified by pattern. The pattern array must be a perfect hexagonal grid – no missing entries are currently permitted. Dummy meshes must be used to fill in empty slots and deleted later if the desired pattern is not a perfect hexagon. This mesh generator stitches together multiple SimpleHexagonGenerator objects or multiple PCCMG objects or a combination of HexagonConcentricCircleAdaptiveBoundaryMeshGenerator and PatternedHexMeshGenerator objects. It also works with TriPinHexAssemblyGenerator (see Section III.B.3). These limitations are due to the mesh metadata included in single

pin cell vs. assembly-type objects with multiple pins which are required for the stitching object to understand the input meshes.



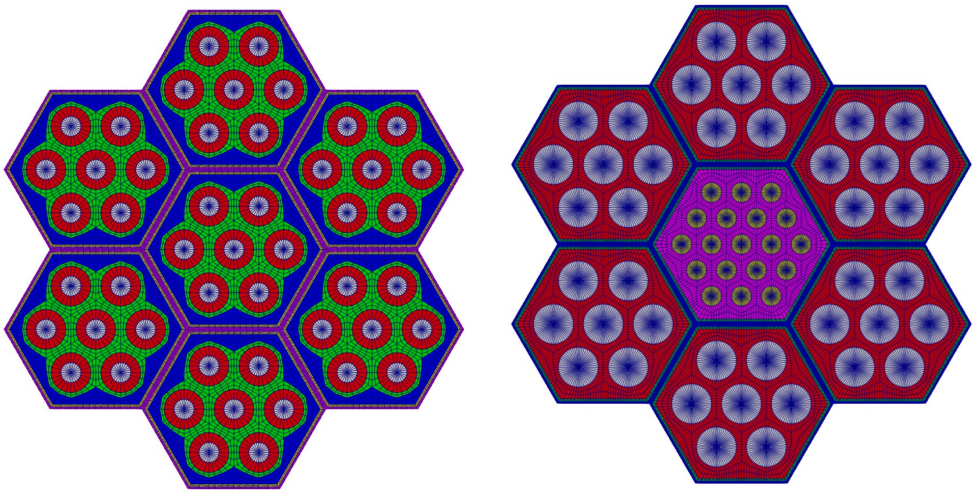
(a) (b)

Figure 3: PatternedHexMeshGenerator stitches pin cells into a triangular lattice with (a) no additional zones added or (b) additional boundary coolant and ducted zones added.

If *pattern_boundary* = *none*, the input meshes will be stitched together and the resulting mesh will have a zig-zag boundary (e.g., reactor core) as shown in Figure 3(a). If *pattern_boundary* = *hexagon*, then an extra layer of background material will be stitched around the pattern according to *hexagon_size* so that the outer boundary of the pattern is a hexagon rather than a zig-zag boundary as shown in Figure 3(b). Optional duct regions may also be added via *duct_sizes* to specify the inner duct boundaries and *duct_intervals* to describe the meshing resolution. The external boundary sideset/nodeset is generated automatically.

PatternedHexMeshGenerator can be used recursively to stitch its own output together, as shown in Figure 4(a-b), provided the user enables the Boolean-type input parameter *generate_core_metadata* = *true.* If the constituent input meshes contain different numbers of pins, as shown in Figure 4(b), the user must ensure conformality by strategically setting azimuthal discretization values. Alternatively, the user can use PatternedHexPeripheralModifier to modify the assembly meshes before assembling into

a core pattern, so that the external boundaries consist of a user-specified number of uniformly spaced nodes.



(a) (b)

Figure 4: PatternedHexMeshGenerator stitches (a) identical hexagonal meshes or (b) different hexagonal meshes recursively into a core pattern.

III.B.2 Assemblies Consisting of One Pin Cell

The HexagonConcentricCircleAdaptiveBoundaryMeshGenerator object creates a hexagonal pin cell similar to PCCMG, but with the additional option to match azimuthal discretization to other input meshes to enable conformal mesh stitching. Figure 5 depicts a mesh with forced azimuthal discretization on one edge of the mesh to match an input mesh (not shown). Outputs from HexagonConcentricCircleAdaptiveBoundaryMeshGenerator are treated as assembly objects and may be directly stitched with assembly meshes using PatternedHexMeshGenerator. This object is particularly useful for meshing a rotating control drum which essentially has one large pin in the center as produced with PCCMG, but needs to be stitched to assembly objects rather than other pin cells.

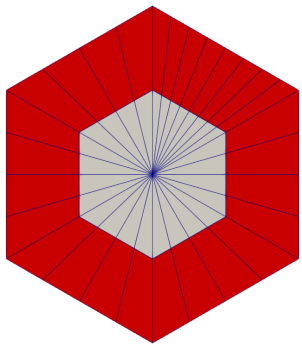


Figure 5: HexagonConcentricCircleAdaptiveBoundaryMeshGenerator generates a PCCMG-like mesh but assembly stitchability and option to force mesh matching on various sides (upper right side in this figure).

III.B.3 Assemblies with Three Corner Pins

Some hexagonal assemblies contain pins arranged in unconventional patterns, such as pins placed in alternating corners of the assembly. The TriPinHexAssemblyGenerator creates a hexagonal assembly with three diamond sections. One pin per diamond may be placed along the line from vertex to center of the hexagon with a specified offset. The orientation of the assembly can be designated as *pin_up* as shown in Figure 6 where the top vertex has a pin, or *pin_down* where the lower vertex has a pin. This mesh is stitchable to other assembly objects using PatternedHexMeshGenerator. This mesh generator was recently used [17] to create a geometry for the High Temperature Test Reactor (HTTR) [18].

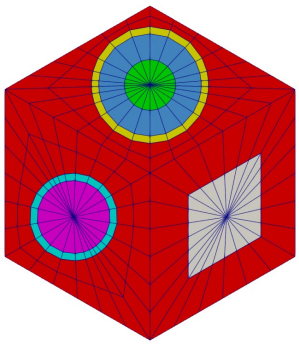


Figure 6: TriPinHexAssemblyGenerator generates a mesh with three corner pins which has assembly stitchability properties.

III.B.4 Assemblies with Control Drums

Rotating control drums are utilized in microreactor and other core designs for reactivity control. Rather than control rods which are inserted and withdrawn axially, control drums can be rotated to position the control material closer or further away from the core center. The control drum absorber geometry is an annular segment (arc) which traverses around the center along a circular path. A typical control drum geometry is shown in Figure 7(a). Figure 7(b) depicts the modifications enabled by the AzimuthalBlockSplitGenerator, which modifies a mesh generated by PCCMG or HexagonConcentricCircleAdaptiveBoundaryMeshGenerator. AzimuthalBlockSplitGenerator modifies a sector of the hexagonal mesh according to a user defined start angle, angle range, and list of the block IDs and block names to modify. Volumes in each arc segment may optionally be preserved. The external boundary nodes may optionally be moved or kept stationary to facilitate stitching to other mesh objects. This mesh generator thus may be used to create static control drum geometries. Currently, the control drum geometry must be contained within a hexagonal cell.

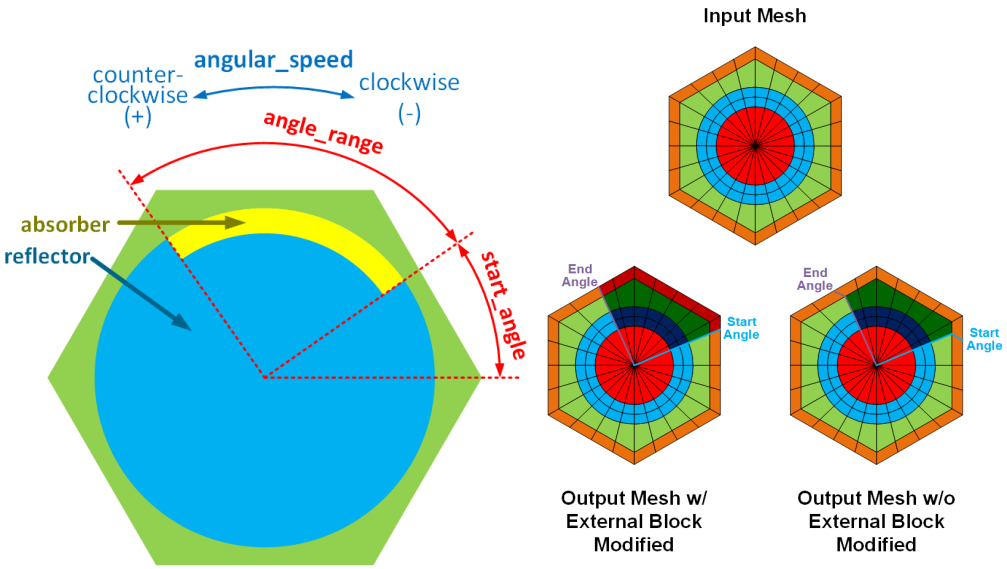
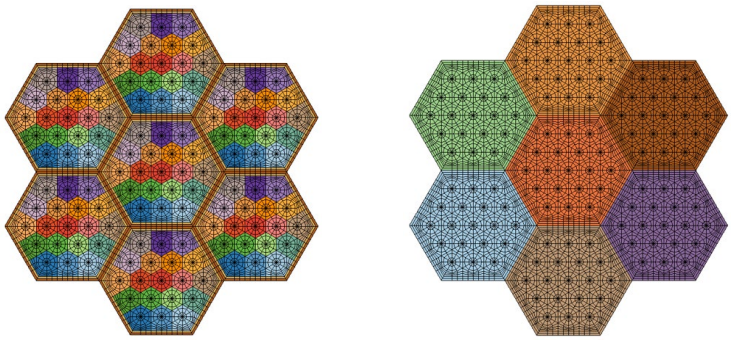


Figure 7: Typical control drum structure and important geometrical parameters as well as mesh modifications enabled by AzimuthalBlockSplitGenerator.

To handle actively rotating control drums, individual control drums may be meshed using HexagonConcentricCircleAdaptiveBoundaryMeshGenerator with *is_control_drum = true*. The individual control drum objects can be stitched with other assemblies into a core configuration using PatternedHexMeshGenerator. Finally, the MultiControlDrumFunction was developed to calculate and return absorber percent volume fractions within each element of the control drum as a function of position and time. This function relies on a series of meta-data in the mesh generated by PatternedHexMeshGenerator, and it is intended to work with either MOOSE's FunctionAux to assign values to an elemental auxiliary variable, or GenericFunctionMaterial to assign values to a material property. If a mesh element is fully contained with the absorber's current geometric position, the returned value is 100, and if the mesh element is fully outside the absorber, the returned value is 0. Intermediate values are returned for mesh elements which are partially contained in the absorber.

IV. IDENTIFICATION OF COMPONENTS USING "REPORTING IDS"

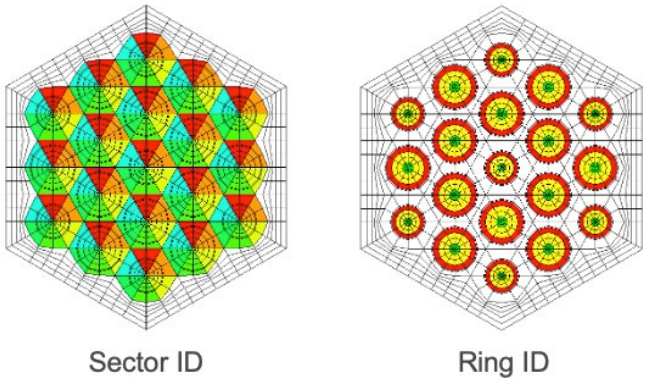
Typical reactor cores have a hierarchical structure consisting of multiple levels: pin, assembly, core, and axial plane. By assigning corresponding "reporting IDs" (which are extra integer tags) on each element of the mesh for those levels, the user can easily identify mesh elements belonging to geometric components and use this information to tally component-wise values, such as pin axial power distribution. The reporting ID capability was embedded into Cartesian and hexagonal geometry patterned mesh objects (CartesianIDPatternedMeshGenerator and HexIDPatternedMeshGenerator); these objects inherit the functionality of the standard objects that perform patterned stitching (PatternedMeshGenerator and PatternedHexMeshGenerator). When using the ID-version of these stitching objects, pin IDs are applied during assembly creation, and assembly IDs are applied during core creation, as shown in Figure 8, according to one of several numbering schemes. The default scheme begins at 0 in the top left corner of the pattern. Other numbering options including the assignment of unique IDs for each pin cell, unique IDs for each pin type, or use of a manual numbering scheme. When assembly duct regions are present, the unique pin IDs are first sequentially assigned, then duct regions are numbered sequentially starting from the inner-most region to the outer-most region. Selected regions may be excluded from the numbering system if desired. Axial plane IDs are applied to 3D extruded meshes using the PlaneIDMeshGenerator object.



(a) (b)

Figure 8: Automatically generated (a) pin and (b) assembly reporting IDs are differentiated by color on each mesh element on a 7-assembly core.

During the pin generation stage of the PCCMG object, users can optionally assign reporting IDs to each ring and azimuthal sector, as illustrated in Figure 9. Users can utilize the ring and sector IDs to identify and manipulate sub-pin level information in the mesh generation and the post-processing stages of the simulation.



(a) (b)

Figure 9. Automatically generated (a) sector and (b) ring reporting IDs are differentiated by color on each mesh element in individual pins.

By making use of reporting IDs together with material IDs (which map physical materials to the mesh), depletion IDs can be automatically prepared with the

DepletionIDGenerator object for Cartesian and hexagonal cores. DepletionIDGenerator assigns depletion IDs on individual elements by finding unique combinations of user-specified IDs. Users can easily control the level of detail for which depletion IDs are assigned through the selection of reporting IDs used in the depletion ID generation. In a pin-level depletion case, as illustrated in Figure 10, the individual pins can be identified by the pin and assembly IDs, and the detailed depletion regions within a pin can be further divided by material IDs. For assembly-wise depletion, the user can set up the depletion IDs by combining the assembly and material IDs.

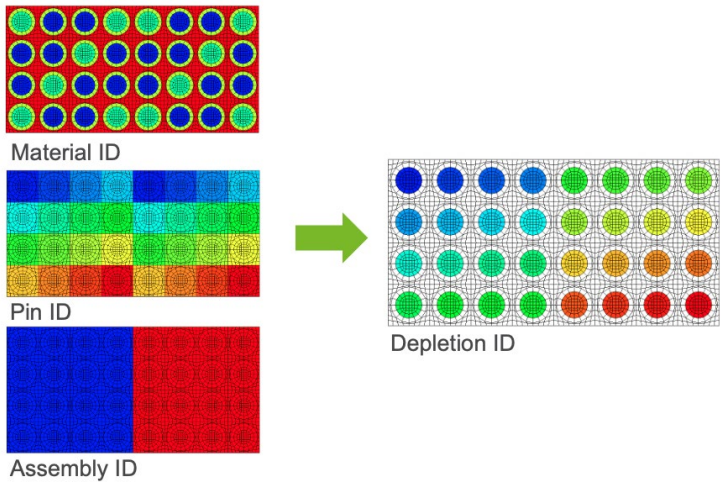


Figure 10. Depletion ID generation is based on finding groups of elements with unique combinations of reporting IDs.

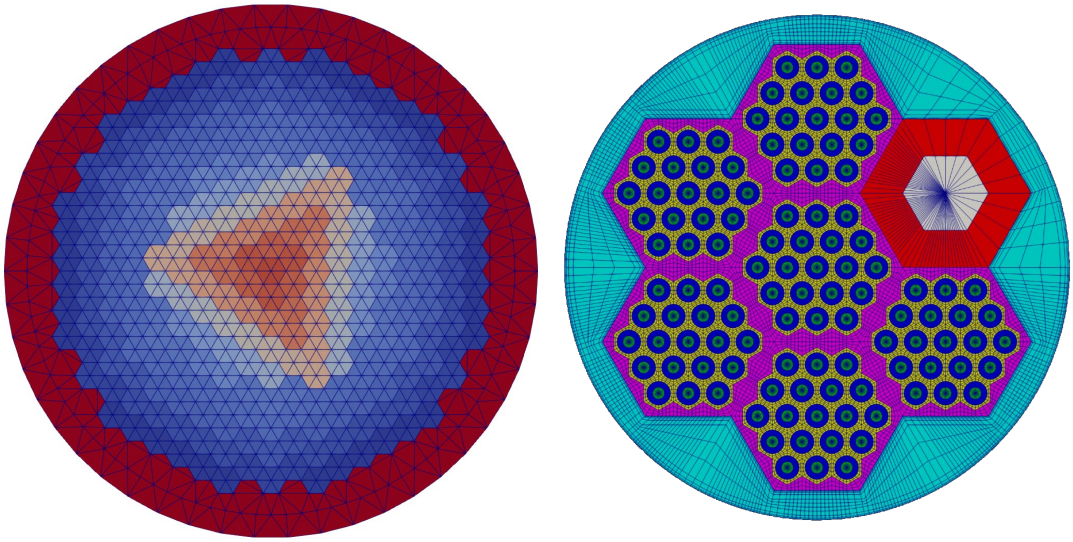
MOOSE VectorPostprocessors can be used to query data on elements with these extra IDs, simplifying output processing significantly since collections of elements forming pins and assemblies are now identifiable without providing information on their physical location. ExtraIDIntegralVectorPostProcessor integrates solution variables over zones identified by combinations of reporting IDs. For reactor applications,

component-wise values such as pin-by-pin power distribution can be easily yielded by specifying integration over pin and assembly reporting IDs.

V. CORE PERIPHERY MESHING CAPABILITY

A group of hexagonal or Cartesian assemblies comprising a reactor core is often surrounded by a peripheral cylindrical reflector region which may require meshing. Cubit's advanced triangulation algorithms are typically used to mesh this complex-shaped region which has a zig-zag inner boundary. To avoid use of an external tool, the open source poly2tri library [14], with a MOOSE-compatible license, was integrated within a new MOOSE mesh generator (PeripheralTriangleMeshGenerator) to test functionality for meshing this geometry. The library allows the addition of Steiner (extra triangulation) points to improve the quality of the triangulation and/or to bias the mesh radially. Element quality is observed to be good for coarsely meshed assemblies. The element quality deteriorates when pin-by-pin assembly meshes are used, and consequently, a general Delaunay triangulation algorithm has been developed and is planned to be merged into MOOSE [19]. Figure 11(a) depicts a core with assemblies generated using SimpleHexagonGenerator and core periphery generated using PeripheralTriangleMeshGenerator, using Steiner points to improve quality.

Alternatively, the PeripheralRingMeshGenerator object adds a cylindrical reflector region to a given 2D core mesh using *quadrilateral* elements. PeripheralRingMeshGenerator also features conformal boundary layer generation and radial mesh biasing options as indicated in Figure 11(b).



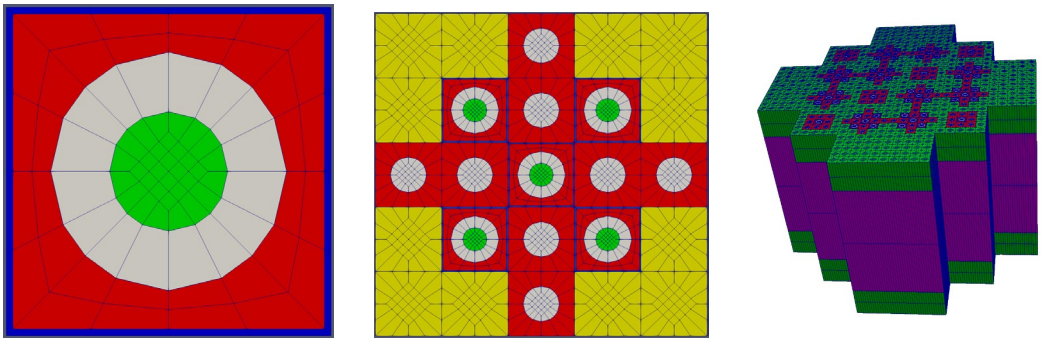
(a) (b)

Figure 11: Sample full core meshes with a peripheral region consisting of (a) triangular elements and (b) quadrilateral elements.

VI. REACTOR GEOMETRY MESH BUILDER CAPABILITY

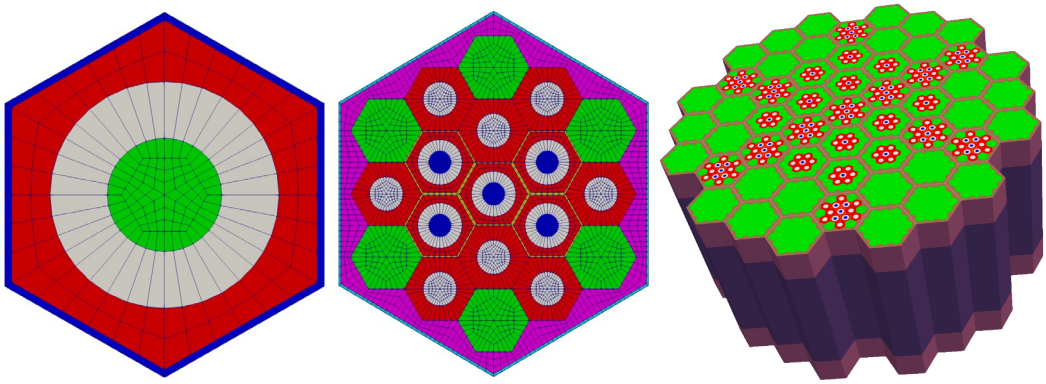
Reactor analysts for conventional Cartesian and hexagonal reactor cores typically want to specify both geometry and materials simultaneously, as these two parts of the input need to be mapped together to specify the physics problem. PinMeshGenerator, AssemblyMeshGenerator, and CoreMeshGenerator objects, collectively known as the Reactor Geometry Mesh Builder (RGMB) capability, have been developed to further streamline meshing input for conventional Cartesian and hexagonal reactor pins, assemblies, and cores by introducing reactor terminology input words and permitting material assignment tags to be placed directly on the mesh during mesh generation. The material assignment tags are placed into an extra element integer placeholder called *region_id* rather than *material_id* which is the commonly used placeholder for materials. Extrusion to 3D can be performed at any stage (pin, assembly, core), and axial discretization information is inherited across the core to

reduce extraneous input. The reactor geometry mesh builder capability both leverages and augments the functionality of other objects in the Reactor module (and MOOSE framework objects) to eliminate as much work for the user as possible. Figure 12 and Figure 13 depict Cartesian and hexagonal pins, assemblies, and cores generated using these objects. These meshes include pin, assembly, and plane IDs as well as material IDs on each element, stored as extra element integers on the mesh. The core peripheral meshing capability has been integrated into this series of mesh generators, and at time of publication is undergoing final review to be merged into MOOSE. RGMB capabilities are demonstrated in Section VIII.A to streamline mesh generation and identification of like material regions (regions with identical cross section properties) in a pin heterogeneous 3-D assembly problem.



(a) (b) (c)

Figure 12: Example Cartesian (a) pin, (b) assembly and (c) 3D core meshes produced by MOOSE's RGMB capability.



(a) (b) (c)

Figure 13: Example hexagonal (a) pin, (b) assembly and (c) 3D core meshes produced by MOOSE's RGMB capability.

The implementation of the RGMB capabilities was enabled by recent improvements in the MOOSE framework which allows a MeshGenerator object to create its own "sub-generator" objects without the need for additional user input. The RGMB mesh generator objects use sub-generators under the hood to pass information from the original mesh generator object through a series of additional objects hidden from the user. This allows the user to see a simple and concise input structure while performing complex operations behind the scenes.

VII. ADVANCED MESH OPERATIONS IN MOOSE

Advanced features which have been implemented or ready in the near future are described in this section.

VII.A "Transition Layer" Meshing

A mesh utility tool named FillBetweenPointVectorsTools has been developed to enable generation of a 2D "transition layer" mesh to connect two given curves (i.e., two sets of coplanar points), as shown in Figure 14.

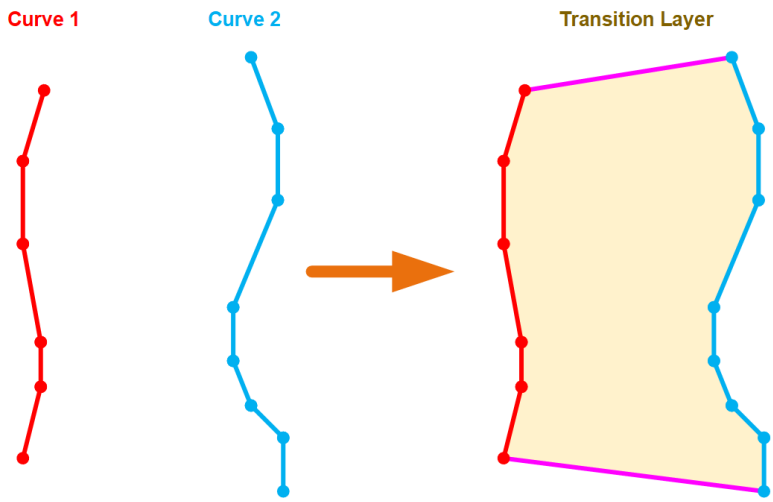


Figure 14: A schematic drawing showing the fundamental functionality of the FillBetweenPointVectorsTools.

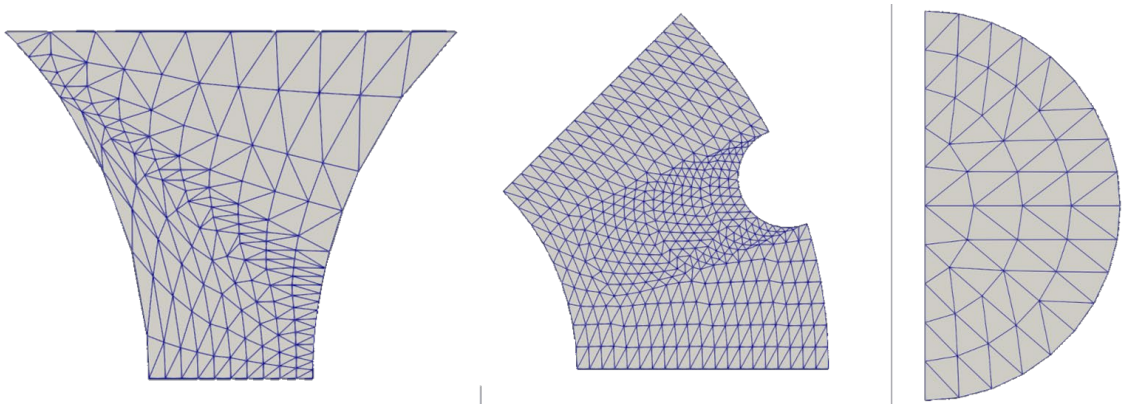


Figure 15: Some representative meshes generated by FillBetweenPointVectorsTools.

The tool provides a versatile meshing capability in 2D space, as illustrated in Figure 15. Currently available as a development tool in MOOSE (only directly accessible by MOOSE developers), end users benefit from using mesh generators that adopt this toolset, including FillBetweenPointVectorsGenerator and FillBetweenSidesetsGenerator (see Figure 16).

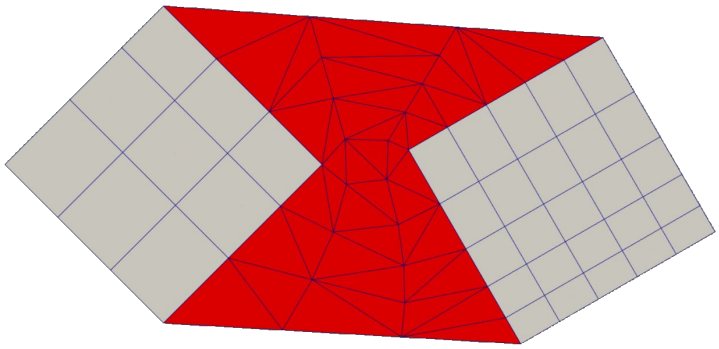


Figure 16: A typical example of using FillBetweenSidesetsGenerator to connect two square meshes together.

VII.B Hexagonal Mesh Trimming (Upcoming Capability)

An object tentatively termed HexagonMeshTrimmer is under development to trim hexagonal assembly or core meshes generated by PatternedHexMeshGenerator and its derived classes, or outputs of the two objects which add peripheral ring meshes. Both peripheral trimming and center trimming will be supported. Peripheral trimming slices off peripheral region(s) of one or multiple sides of the input hexagonal assembly mesh. To be specific, for each side, half of the unit pin meshes are trimmed off, as shown by the purple lines in Figure 17. Peripheral trimming is needed to generate meshes with pins on the unit assembly boundaries.

Center trimming removes azimuthal sectors from the input hexagonal assembly or core mesh in order to leverage symmetry and reduce model size. The mesh may be trimmed along lines of symmetry in the input mesh. Valid hexagonal input meshes may be trimmed at twelve possible center trimming lines, indexed from 0 to 11 as the blue lines shown in Figure 17. Envisioned uses of this trimming utility are depicted in Figure 18.

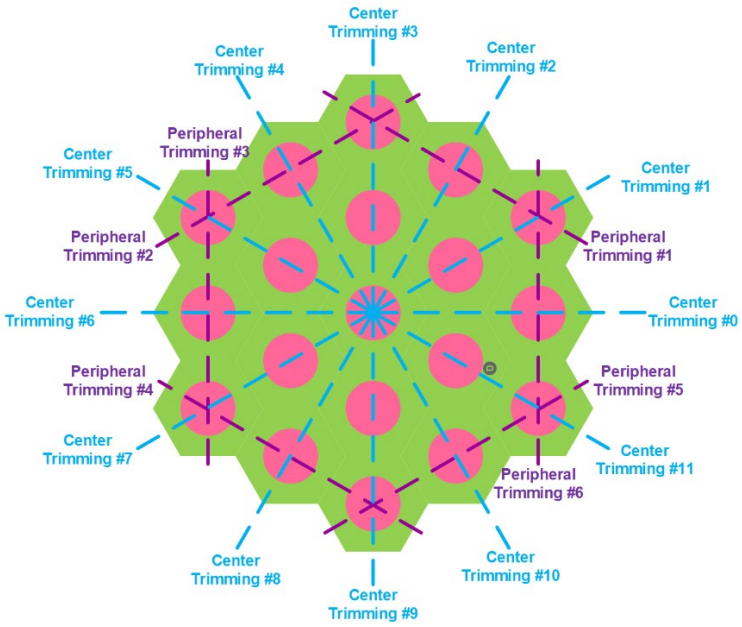


Figure 17: Trimming schemes under development for hexagonal mesh geometry.

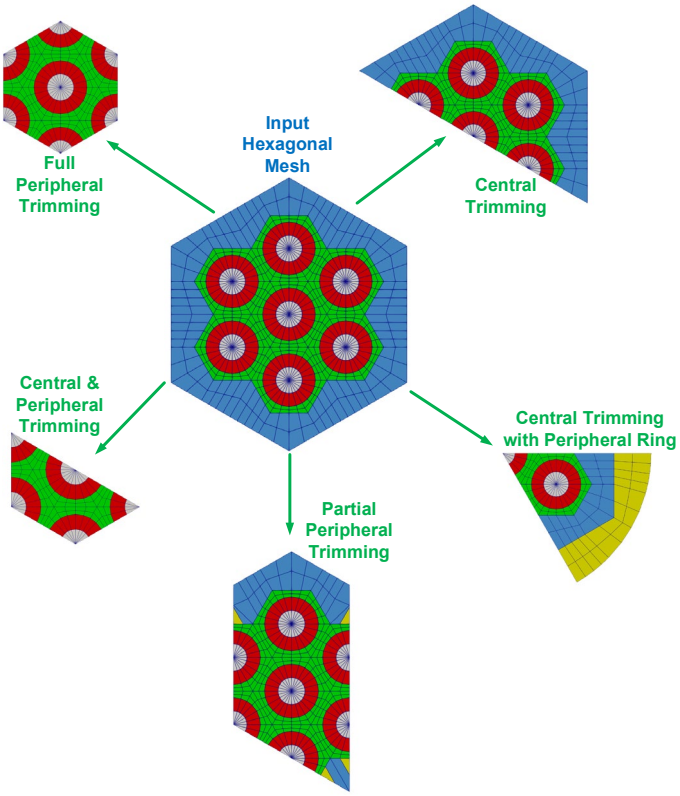


Figure 18: Example outputs of the HexagonMeshTrimmer.

VII.C Advanced Extrusion Options (Upcoming Capability)

A series of capability enhancements has been made to MOOSE's FancyExtruderGenerator, which extrudes 2D meshes along a user-specified vector. An on-the-fly faulty element fixer has been added to ensure that the 3D elements generated during extrusion have the correct node convention. In each layer of extrusion, the axial meshing density can now be biased using a specific growth factor (see Figure 19). Additionally, the original subdomain swap feature has been expanded to extra element integer IDs so that reporting IDs on the 2D input mesh can be maintained or changed during extrusion on each axial level.

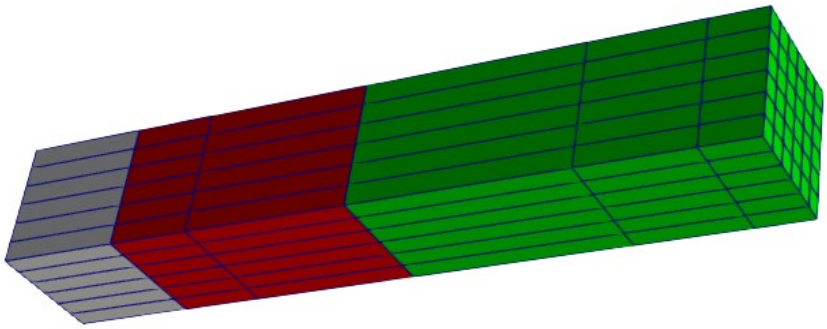


Figure 19: An extruded square mesh with axially-biased mesh density.

Meanwhile, a series of updates have also been made to improve the boundary definition functionality of FancyExtruderGenerator. Two types of boundaries are involved in the extruded mesh: (1) boundaries parallel to the extraction vector which originate from the lower-dimensional boundaries in the input mesh; and (2) boundaries that are not parallel to the extrusion vector. Both types of boundaries can be defined for each elevation independently using a strategy similar to the subdomain swap.

VIII. VERIFICATION OF REACTOR MODULE CAPABILITIES

To verify the functionality of the new Reactor module meshing capabilities for hexagonal reactor geometries, advanced reactor physics verification cases were analyzed with MOOSE-based applications (Griffin, BISON, Sockeye [20], MOOSE Tensor Mechanics Module). Two specific analysis examples are given here. Results generated by these physics applications using MOOSE-based meshes were compared to results derived using externally generated meshes from either Cubit or the Argonne Mesh Tools system. Excellent agreement was observed between MOOSE-based meshes and externally based meshes, as described in the following sections. Additionally, the elimination of external, licensed meshing tools significantly streamlines the workflow for the end user.

VIII.A. Fast Reactor Assembly Neutronics Example

Griffin, a NEAMS-developed, MOOSE-based application developed for supporting radiation transport calculations, was employed to test the new mesh generation functionality introduced in the Reactor module. Griffin was compiled by linking directly to the MOOSE Reactor module to access the new capabilities. To demonstrate the benefits of using the Reactor module, a 3-D lead-cooled fast reactor (LFR) assembly example is studied. The geometry and design specifications of the assembly are based on inner zone fuel assembly of a 950 MWth liquid lead-cooled, fast neutron spectrum core designed by Westinghouse Electric Company (WEC) [21]. The assembly consists of 7 rings of hexagonal pins discretized into 10 axial zones based on material heterogeneity. MOX fuel composed of depleted uranium and enriched plutonium is used in the inner fuel zones of the fuel pins in the assembly.

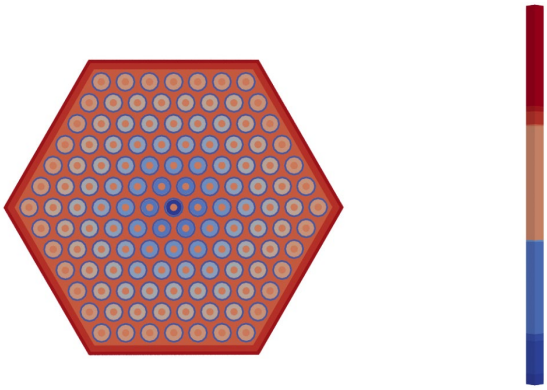
VIII.A.1. Mesh Generation

Figure 20 illustrates the top-down and side view of the assembly geometry generated using RGMB objects. Each hexagonal pin region is discretized radially into 4 rings representing a central helium hole surrounded by annular rings of fuel, helium gap, and cladding, plus a background region of coolant. Two duct regions surround the entire assembly, representing the solid duct and the inter-assembly coolant gap. Axially, the 2D mesh is extruded into 3D with 10 regions of varying material composition. The general steps for defining the mesh are outlined as follows:

1. Define each unique 3-D pin including axial and radial region IDs using PinMeshGenerator
2. Assign each pin to the appropriate assembly lattice position, and define assembly background and duct regions using AssemblyMeshGenerator

In step 1, the ring radii, pin pitch, and radial and azimuthal discretizations of the pin are specified. In addition, the region IDs corresponding to material zones in the pin are specified through the *region_ids* parameter. This generates a map of all radial and axial region IDs within the pin as an extra element integer map, which the Griffin code can use downstream to set material properties directly instead of relying on a separate mapping input for blocks to materials. In step 2, each of the pins defined in step 1 are placed in a hexagonal lattice, and the *extrude=true* option is used to inform the mesh generator to extrude the geometry into 3-D. At this stage, the dimensions and region IDs of the assembly duct and background regions are also provided. Upon completion, AssemblyMeshGenerator generates the 3-D assembly mesh and automatically produce the extra element integers related to pin IDs (unique id given to each pin region in the assembly), plane IDs (unique id given to each axial region in the assembly), and region IDs (unique id given to each material region, specified through the user input). These reporting IDs can be queried by MOOSE's ExtraIDIntegralVectorPostprocessor to

compute integral quantities of interest such as scalar flux and fission source based on pin location, axial location, and/or material region.



(a) (b)

Figure 20: Top-down (a) and side (b) views of LFR heterogeneous assembly, created using RGMB objects in the Reactor Module. Each color represents a unique material region in the assembly.

VIII.A.2. Computation and Results

The neutronic behavior for the LFR assembly was simulated with Griffin, and the region_id extra element integer map generated by the RGMB mesh generators is used directly to specify material regions in Griffin. The multigroup cross sections for each material region were generated prior to the Griffin simulation with MC^2-3 [22]. For this verification test, three avenues for mesh generation are explored: (1) using the RGMB mesh generators described above, (2) using the base Reactor Module mesh generators that are called by the RGMB mesh generators, and (3) using Argonne Mesh Tools (external software used for generating reactor-based meshes) to generate the mesh. All neutronics parameters are kept the same across Griffin runs: 9 energy groups, P1A3 Gauss-Chebyshev cubature, and a solver scheme that utilizes the discontinuous finite element method (DFEM) with discrete ordinates (S_N). Vacuum boundary

conditions are applied to the top and bottom of the assembly; reflective boundary conditions are applied to the radial boundary. Coarse mesh finite differences (CMFD) is used to accelerate the transport problem convergence, and the input coarse mesh for this acceleration scheme – also defined with MOOSE mesh tools - homogenizes each pin region into a single hexagonal region composed of 12 triangles.

K-effective is used as a metric for comparing neutronics results between the three methods for mesh generation described above, and these results are summarized in Table 1. The MOOSE-based mesh generators produce identical eigenvalues while there is a 20pcm difference in the Griffin simulation that uses Argonne's Mesh Tools for mesh generation, which is due to the minor variation in the discretization schemes employed by MOOSE and Argonne's Mesh Tools in the assembly background region. This is described in Ref [11]. While the two MOOSE-based mesh generation approaches produce identical results, the RGMB-based procedure offers several advantages. First, the parameters within PinMeshGenerator and AssemblyMeshGenerator have been optimized to simplify input generation to focus solely on parameters of interest to the reactor analyst while masking any extraneous parameter definitions in base Reactor module mesh generators that would otherwise need to be specified. Secondly, material ID assignments can be made directly onto the resultant RGMB mesh, which Griffin can natively read to greatly simplify the procedure of mapping cross section materials to elements within the input mesh. Base mesh generators defined in the Reactor module do not have this capability, and instead material assignments must be made based on block ID assignments of the mesh, which can be a very cumbersome process for problems with a large degree of radial and axial material heterogeneity. Finally, RGMB mesh generators are seamlessly integrated with

reporting ID generation (pin, assembly, plane, and region IDs), and these reporting IDs streamline the definition of tally regions commonly used by reactor analysts.

Table 1: Griffin-computed k-effective results for LFR Assembly Problem: MOOSE mesh vs. Argonne Mesh Tools.

Mesh Generation Tool	**Griffin Solver Scheme**	**K-Effective**
MOOSE RGMB Mesh Generators	DFEM-SN with CMFD Acceleration	1.17142
MOOSE Base Reactor Module Mesh Generators	DFEM-SN with CMFD Acceleration	1.17142
Argonne Mesh Tools	DFEM-SN with CMFD Acceleration	1.17122

To illustrate how automatically-generated reporting IDs from RGMB mesh generators can be used for reactor analysis workflows, the ExtraIDIntegralVectorPostprocessor is used in conjunction with the RGMB reporting IDs to inspect pin-by-pin tallies of interest. For this specific problem, scalar flux was specified as the tally variable, and both pin ID and axial plane ID were set as the reporting IDs that tell Griffin where to tally. Based on these specifications, an output CSV file is produced by Griffin, which tabulates the scalar flux for each unique combination of pin ID and axial plane ID. The data from this file can be used with a custom processing script to aggregate the scalar flux over like reporting IDs to generate pin-wide and axial plane-wide flux distributions over the entire core. For reference, the axially integrated pin scalar flux map produced from the processing script is illustrated in Figure 21.

Thus, this LFR verification example illustrates how the Reactor Module can streamlineGriffin workflow to execute a 3-D reactor analysis from input mesh construction, simulation execution, and output data generation for post-processing neutronics assembly or core distributions.

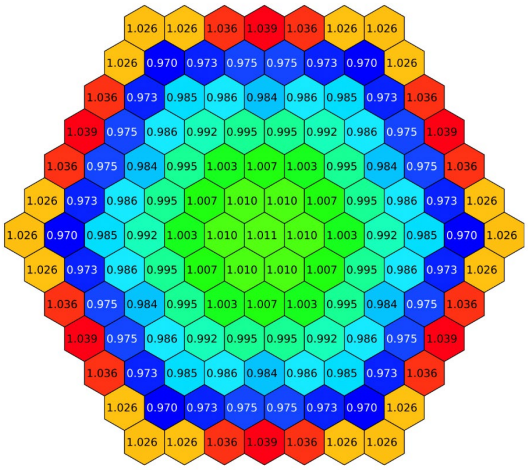


Figure 21: Axially integrated pin-by-pin group 0 normalized scalar flux distribution for the LFR assembly example.

VIII.B. Heat-Pipe Cooled Microreactor Multiphysics Example

Microreactor concepts have received significant attention in the United States due to reduced capital investment, siting flexibility and high mobility. Under NEAMS, an on-going effort to demonstrate microreactor multiphysics simulation capabilities by coupling multiple MOOSE based codes BISON (fuel performance analysis), Griffin (neutronics), Sockeye (heat pipe analysis), and SAM (system analysis) [23] is underway [24]. The reactor example studied in this activity is an TRISO-fueled heat pipe microreactor that includes control drums, reflectors, graphite monolith, heat pipes, moderators, fuel, and helium/air gaps. The Reactor Module has been used to generate the full core microreactor mesh which was formerly meshed with Cubit.

VIII.B.1. Mesh Generation

Development of the microreactor core mesh is hierarchical, following mesh generation of (1) pin-cell structures using PCCMG, (2) assemblies using PatternedHexMeshGenerator, and (3) reactor core using PatternedHexMeshGenerator (Figure 22a-b). In addition, hexagonal 2D pin-cell structures, including control drums, reflectors, dummy, and air center block, were produced and meshed using HexagonConcentricCircleAdaptiveBoundaryMeshGenerator and AzimuthalBlockSplitGenerator (control drum only). After generating the 2D full core mesh, dummy blocks were removed using BlockDeletionGenerator, and the 2D full core mesh was sliced into a 1/6 core by combining ParsedSubdomainMeshGenerator and BlockDeletionGenerator. Extrusion to 3D was performed using FancyExtruderGenerator. Top and bottom reflectors were defined by ParsedSubdomainMeshGenerator (Figure 22c). Finally, sidesets needed for applied boundary conditions in simulations were generated and renamed by using SideSetsBetweenSubdomainsGenerator and RenameBoundaryGenerator, respectively.

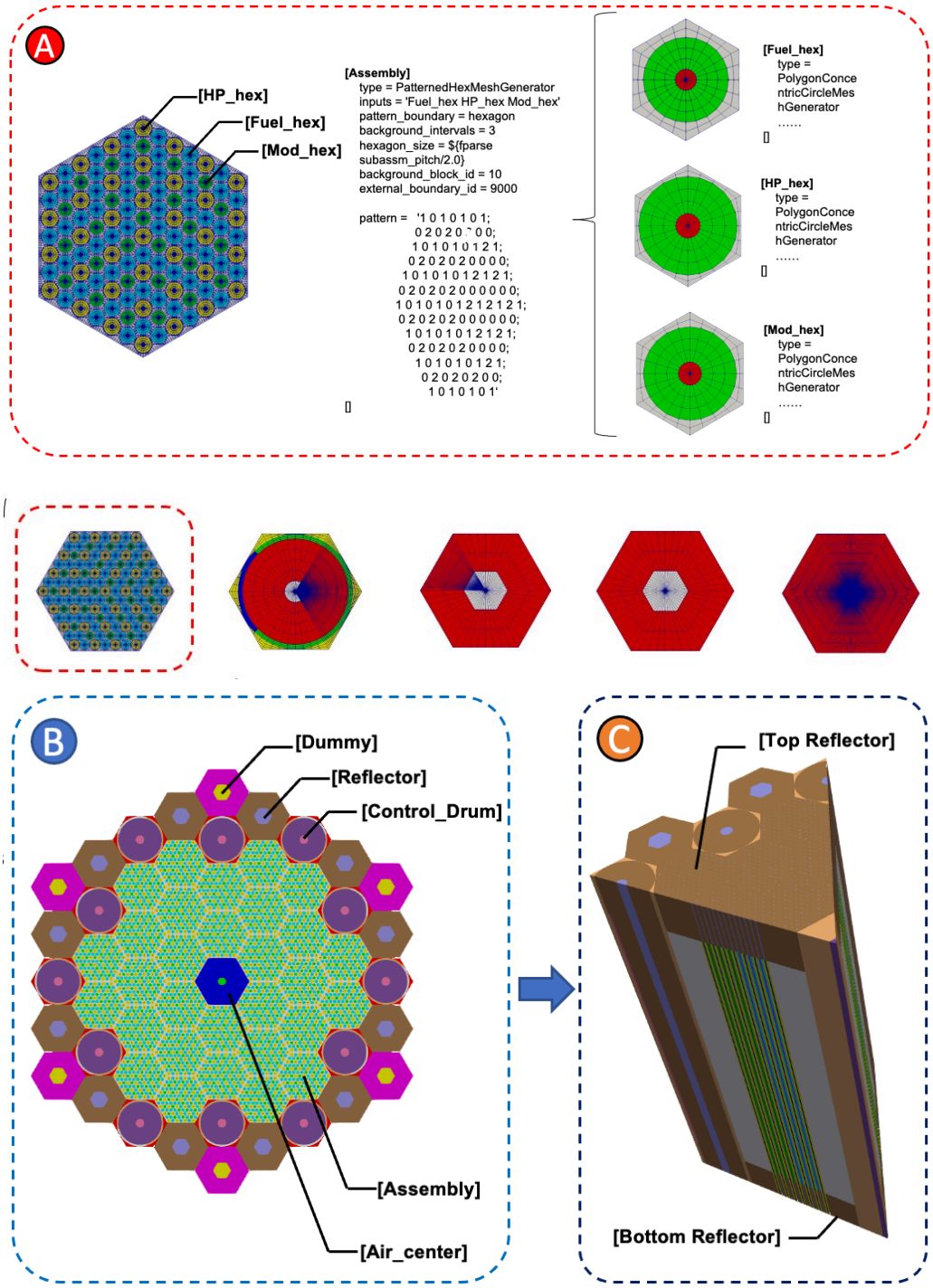


Figure 22: The 3D 1/6 microreactor core was generated using MOOSE by (a) creating individual assemblies, (b) patterning them into the core, and (c) extruding the 2D geometry into 3D and slicing the mesh to yield a 1/6 sector.

The Cubit mesh was also generated by extrusion of the 2D mesh into 3D. The 2D mesh was generated by using the "pave" function which automatically meshes a surface with an unstructured quadrilateral mesh. Figure 23 shows the comparison of the meshes generated by Cubit and MOOSE. All the microreactor components in the Cubit mesh were reproduced and meshed using MOOSE mesh generators. MOOSE provides more flexibility to refine the mesh in targeted areas, thereby reducing the overall mesh size compared to Cubit.

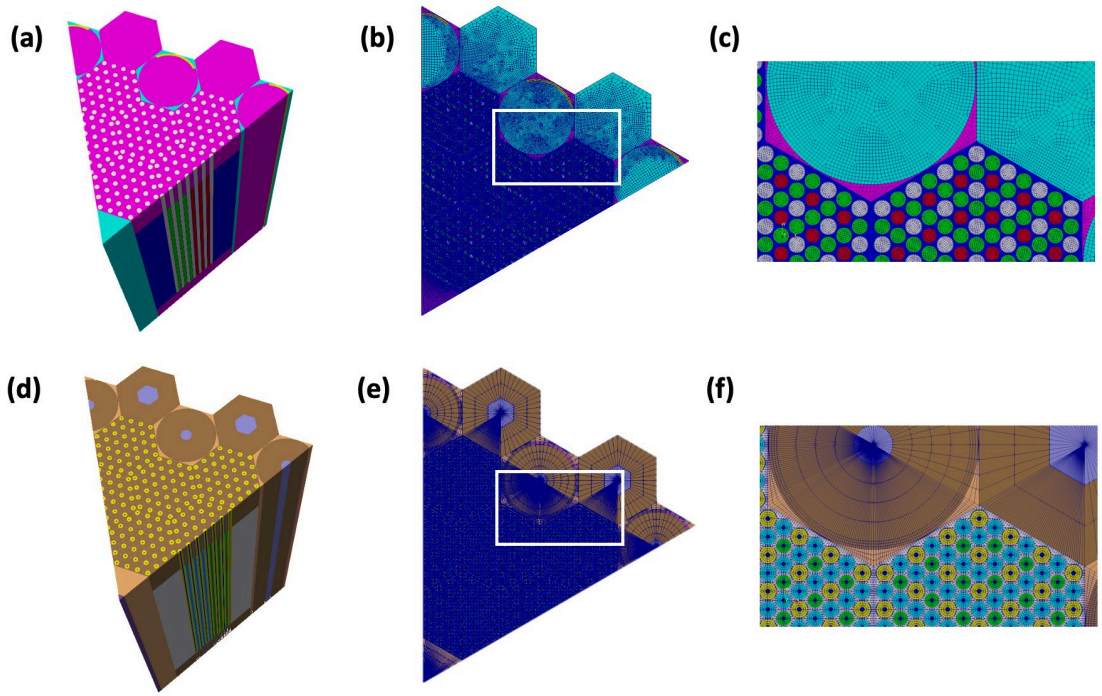


Figure 23: Microreactor core meshes produced by Cubit (number of nodes: 1.7×10^6): (a) 1/6 symmetric core mesh, (b) cross-section of the core, and (c) zoom-in region of (b); and MOOSE mesh generators (number of nodes: 1.0×10^6): (d) 1/6 symmetric core full core mesh, (e) cross-section of the core, and (f) zoom-in region of (e)

VIII.B.2. Computation and Results

Two simulations were conducted to examine the performance of 3D 1/6 symmetric core mesh generated by MOOSE in comparison to that generated by Cubit. A BISON simulation first solves a simple heat conduction problem with no input power. The core

was simulated to establish a “steady-state” which is also the starting status of the multiphysics simulation. The temperature of the core at the initial state was set to 300K uniformly. Without input power, the temperature distribution relies on the boundary conditions: the heat pipe surfaces with external temperature T_{ext} = 800K are the only heat sources, and external surfaces (including top, bottom and outer surfaces of reflector, and top surface of helium gap) with T_{ext} = 300K become the heat sink.

Figure 24(a) and (b) show the temperature distribution of 1/6 symmetric core at the last time step (20 sec) of simulation. Similar temperature distributions were developed using the meshes generated by Cubit and MOOSE mesh generators. Temperature evolutions of fuel and heat pipe surfaces are selected to compare computations with the meshes generated by Cubit and MOOSE mesh generators (Figure 24(c) and (d)). As seen, the difference in temperature evolutions is negligible, indicating both meshes are reliable in the heat conduction simulation.

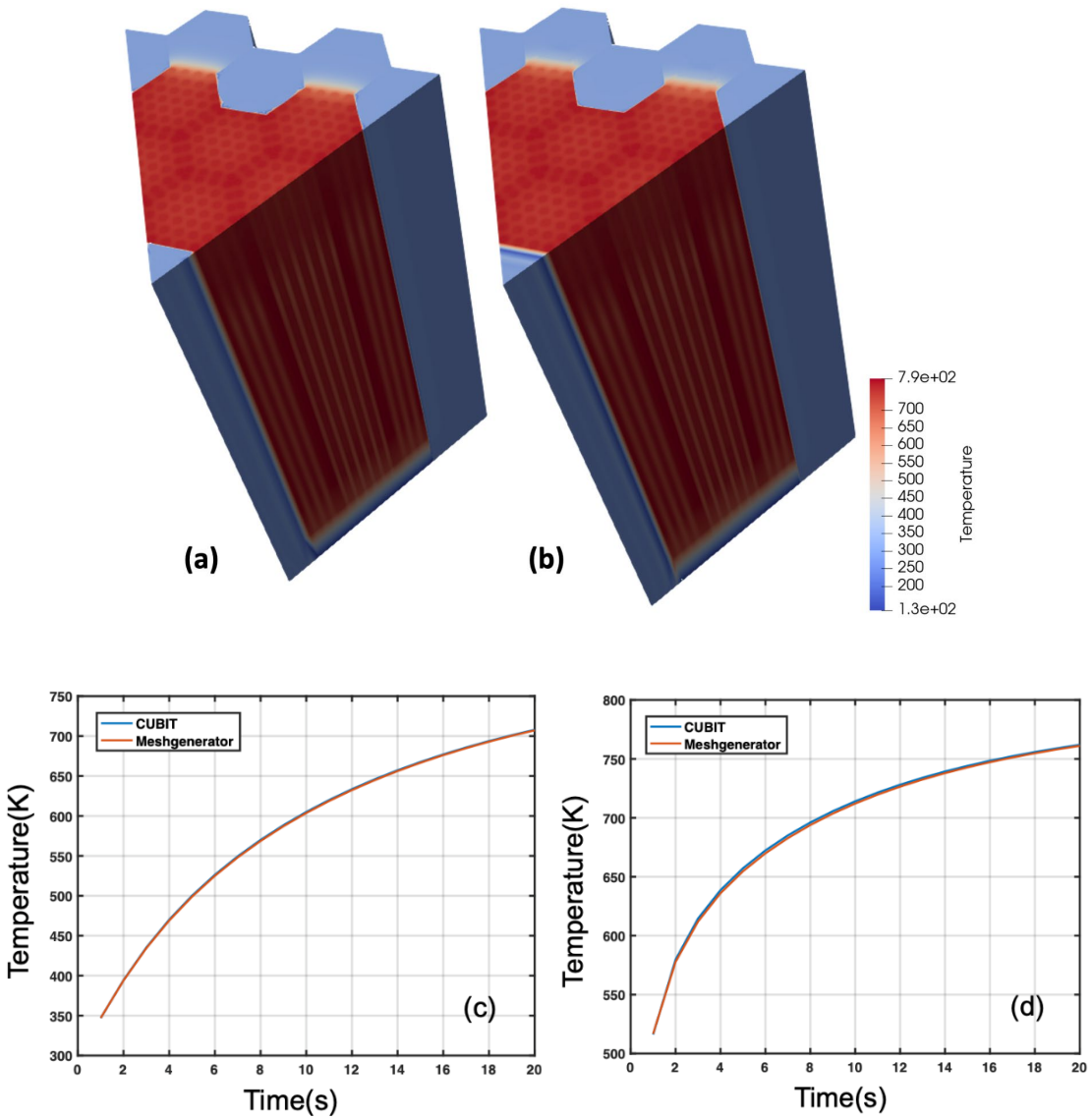
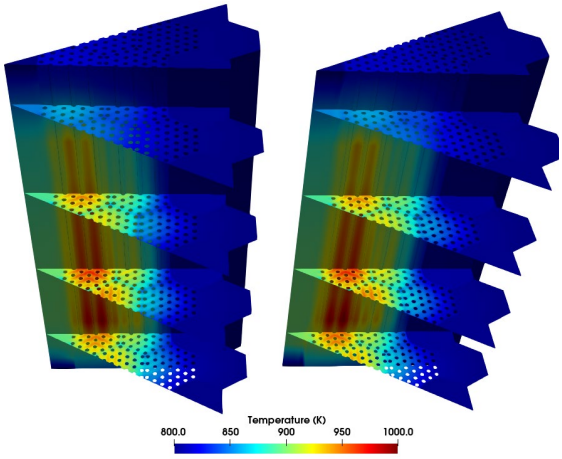


Figure 24: Simulated temperature distribution of the 1/6 symmetric core at the last time step (20 sec) of simulation using (a) Cubit mesh (b) MOOSE mesh generators for (c) heat pipe and (d) fuel.

Consistency between the Cubit-generated mesh and MOOSE-generated mesh was further investigated using a multiphysics approach with three MOOSE applications: Griffin, BISON, and Sockeye. Griffin, the main application, performs steady state eigenvalue calculation using the generated 3D core meshes and the diffusion solver option. Griffin is coupled with a BISON sub-application using the same meshes through

Picard iteration. Griffin provides power density distribution to the BISON sub-application and gets fuel temperature profile back during each iteration. Additionally, for each single heat pipe, the BISON sub-application has its own second-level Sockeye sub-application to simulate the heat pipe cooling performance. Consistent temperature profile at the steady state were predicted by the Griffin-BISON-Sockeye approach using the Cubit and MOOSE meshes as shown in Figure 25(a). Figure 25(b) compares specific quantities: the k-effective differs by less than 100 pcm by using the Cubit mesh and MOOSE mesh, and the temperature difference is usually lower than 5 K for different components of the reactor core. This is excellent agreement considering that the Cubit and MOOSE meshes have vastly different discretizations (owing to the "pave" algorithm in Cubit which is not easily controlled by the user). The differences in results are caused by the mesh density. While the mesh generators in the recently developed MOOSE Reactor module can auto-preserve the circular block volumes (including fuels, moderators, and heat pipes in this mesh), Cubit does not provide such functionality to preserve block volumes, and thus may contribute to some level of error unless the user adjusts densities for each mesh. The run-time for the MOOSE mesh simulation is shorter than that of the Cubit mesh simulation due to the difference in mesh density.



RESULTS	CUBIT	MOOSE
k_{eff}	0.93078	0.93165
T_{max}^{fuel}	982.50	987.62
T_{avg}^{fuel}	873.07	874.01
T_{min}^{fuel}	804.41	804.53
$T_{max}^{moderator}$	971.59	975.58
$T_{avg}^{moderator}$	870.92	871.89
$T_{min}^{moderator}$	806.11	806.34
$T_{max}^{monolith}$	978.38	983.18
$T_{avg}^{monolith}$	866.66	867.64
$T_{min}^{monolith}$	803.62	803.79

(a) (b)

Figure 25: Comparison of microreactor multiphysics simulation results: (a) 3D temperature profile; and (b) eigenvalues and key temperature values

VIII.C. Additional Reactor Mesh Generation Examples

VIII.C.1 Ducted Fast Reactor Assembly with Load Pads

A fast reactor ducted assembly with load pads was meshed using MOOSE's Reactor Module and utilized on the Verification Problem 1 (VP1) included in a set of benchmarks published by the International Working Group on Fast Reactors (IWGFR) acting under The International Atomic Energy Agency (IAEA) [25]. This IAEA VP1 example examines the free thermal bowed deformation of a single, hexagonal assembly with above core load pad (ACLP) and linearly varying thermal gradient in the active core region. A linearly varying thermal gradient is applied along the axial direction in the active core region as shown in Figure 26.

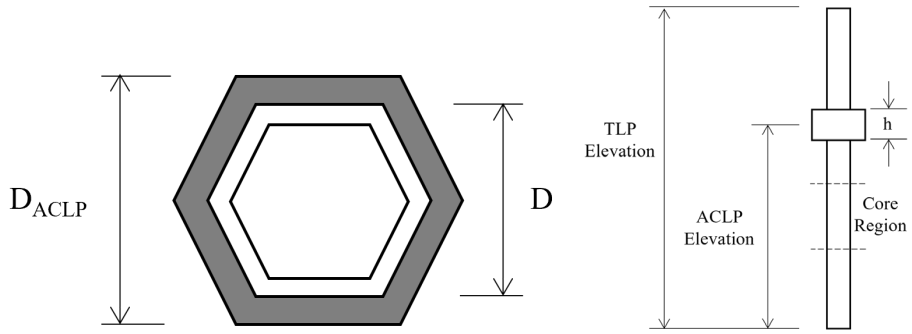


Figure 26: Geometry for hexagonal ducted assembly: (a) 2D cross section at load pad, and (b) vertical cross section showing location of load pad and active core.

First a 2D duct is created with the PCCMG object and then the duct interior is removed with with BlockDeletionGenerator. The 2D duct is extruded to 3D using FancyExtruderGenerator. The same steps are taken to create the load pad, which is moved axially along the duct to the correct position using TransformGenerator and stitched to the duct with StitchedMeshGenerator. Sidesets can be renamed using RenameBoundaryGenerator, which can be useful to name the load pad faces for creating contact sets or for post-processing. The final mesh is depicted in Figure 27 and compared to a Cubit generated mesh to show similar mesh properties.

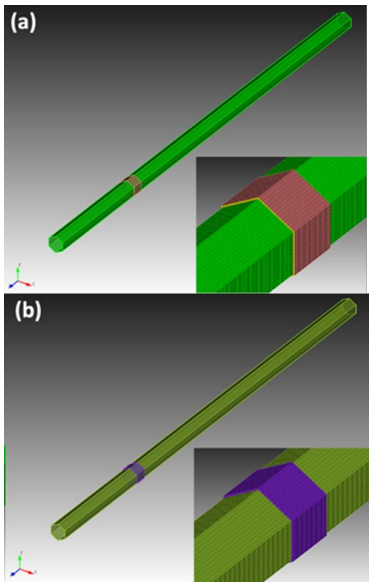


Figure 27: Ducted hexagonal assembly mesh generation with (a) mesh generated by Cubit and (b) mesh generated by MOOSE.

Detailed analysis results of the behavior of the ducted assembly under thermal load are available [13].

VIII.C.2 Gas-Cooled Microreactor Mesh

An assembly mesh of a Gas-Cooled Microreactor (GCMR) concept was generated for multiphysics simulations. The hierarchical process to generate the GCMR mesh as shown in Figure 28 starts with the hexagonal unit meshes for burnable poison, coolant hole, channel, yttrium hydride (YH) pin, TRISO fuel, and control rod hole. The hexagonal unit meshes generated by PCCMG objects were assembled by PatternedHexMeshGenerator and extruded to 3D using FancyExtruderGenerator. Finally, 3D blocks and sidesets were defined using ParsedSubdomainMeshGenerator and SideSetsBetweenSubdomainsGenerator, respectively. Details of the multiphysics simulation can be found in [26,27].

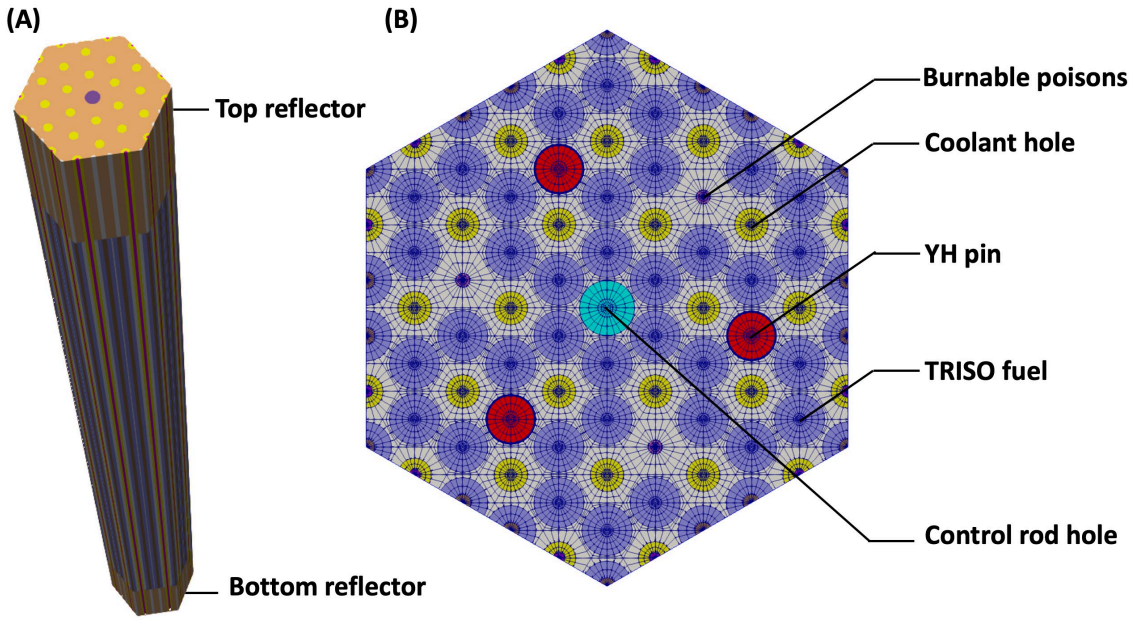


Figure 28: Description of GCMR Assembly Mesh: (a) constructed 3D mesh consisting of (b) hexagonal (pin-cell) unit meshes.

VIII.C.3 KRUSTY Heat-Pipe Microreactor Mesh

MOOSE-based mesh generation using the newly available tools was also performed for KRUSTY, a prototypic nuclear-powered test of a fission space reactor [28]. The reactor was fueled with U8Mo (the actual weight fraction of Mo was 7.65% to produce 1 kW electric power). The heat generated in the core was carried to the Stirling converters using heat pipes for electric power generation.

Generation of a full core mesh was performed in preparation for multiphysics (neutronics, thermo-mechanics, and heat pipe analysis) simulation of KRUSTY. Due to the geometrical complexity of KRUSTY, the core model was simplified and initially meshed with Cubit by Los Alamos National Laboratory [29] for multiphysics simulation. Due to high demand for computation resource, a half symmetric core was meshed as shown in Figure 29(a-b). The LANL model was rebuilt using the MOOSE Mesh System as shown in Figure 29(c-d). Unlike the examples of HPMR and GCMR, the process to build the KRUSTY core is not hierarchical. Construction of the KRUSTY

mesh relies on the FillBetweenPointVectorsGenerator and FillBetweenSidesetsGenerator objects (described in Section VII.A) to generate transition layers to connect vectors/boundaries to form irregular geometries like the fuel zones with heat pipes. The completed mesh is 1/16 angular symmetric and could be divided into half, quarter, 1/8, & 1/16 meshes using PlaneDeletionGenerator. Figure 29 shows the examples of full core, quarter and 1/16 meshes, all of which have been utilized in multiphysics computation. Leveraging symmetry to reduce the mesh size significantly decreases computational resource requirements for performing the physics simulation, which is particularly useful in the code testing stages and the steady-state simulation.

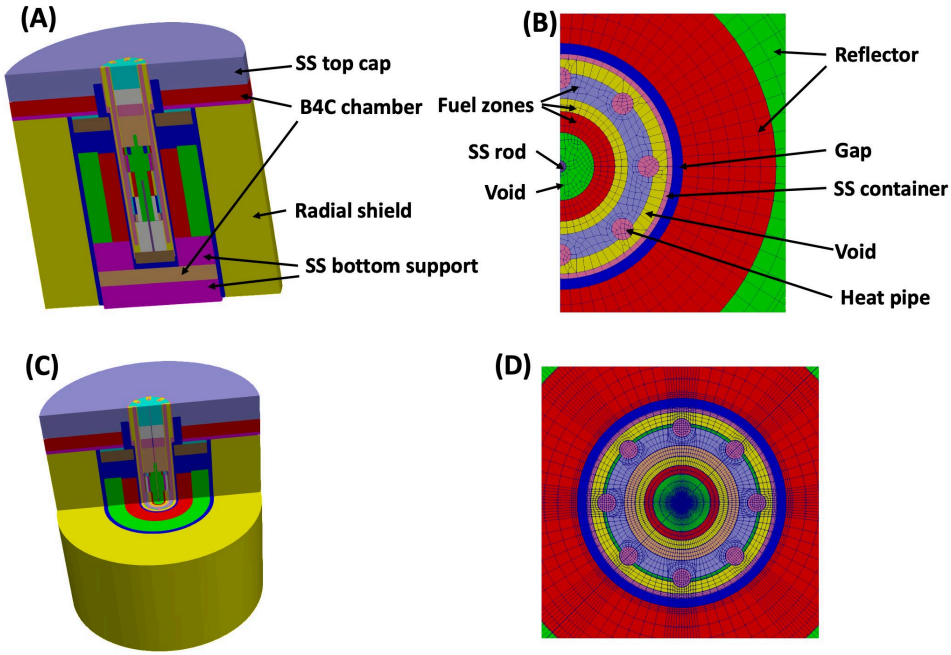


Figure 29: KRUSTY meshes: (a) half-core Cubit mesh; (b) cross-section of Cubit mesh near the fuel zone (core center); (c) full core mesh MOOSE mesh (a quarter of core was removed to show the internal structures); and (d) cross-section of MOOSE mesh near the fuel zone (core center).

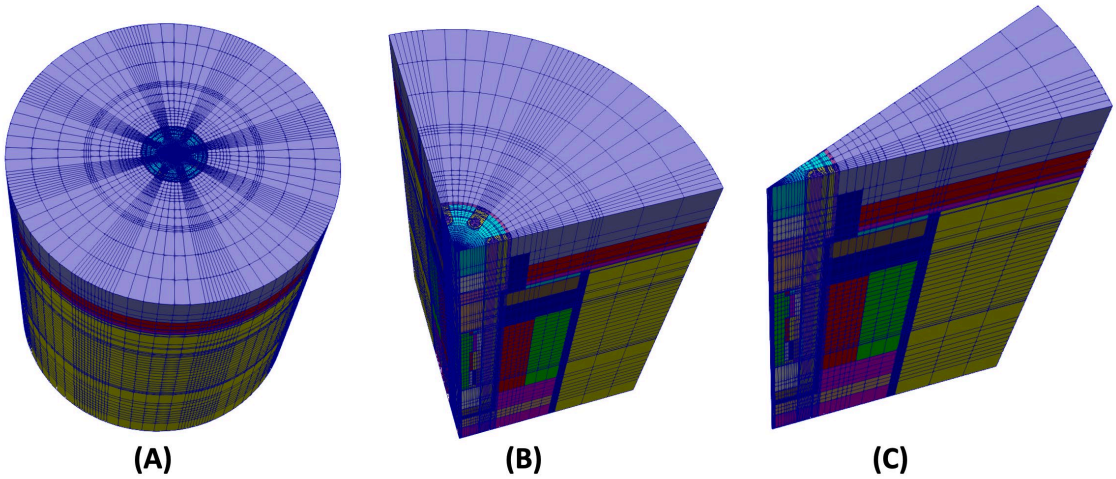


Figure 30: KRUSTY meshes generated using MOOSE: (a) full core, (b) quarter core, and (c) 1/16 core meshes

VIII.C.4 Molten Salt Reactor Experiment

The Molten-Salt Reactor Experiment (MSRE) is an 8MW_{th} molten salt reactor developed at Oak Ridge National Laboratory (ORNL) and operated in 1960s [30]. The experiments conducted at the MSRE provides invaluable benchmark experimental data for modelling and simulation of molten salt reactors. Fuel depletion computation using MSRE data was conducted at Argonne based on MSRE lattice [31] with plans to employ a full core model soon. Figure 31 shows the MSRE mesh generated by MOOSE mesh generators for fuel depletion computation. More details of the model can be found in [32]. Except for the control rods, the full core mesh was built hierarchically from the MSRE unit lattice containing graphite and fuel. The control rod regions were built with the FillBetweenSidesetsGenerator object and stitched to other MSRE lattices to form the 2D mesh as shown in Figure 31(b). The 3D core mesh in Figure 31(a) was completed by extruding the 2D mesh using FancyExtruderGenerator.

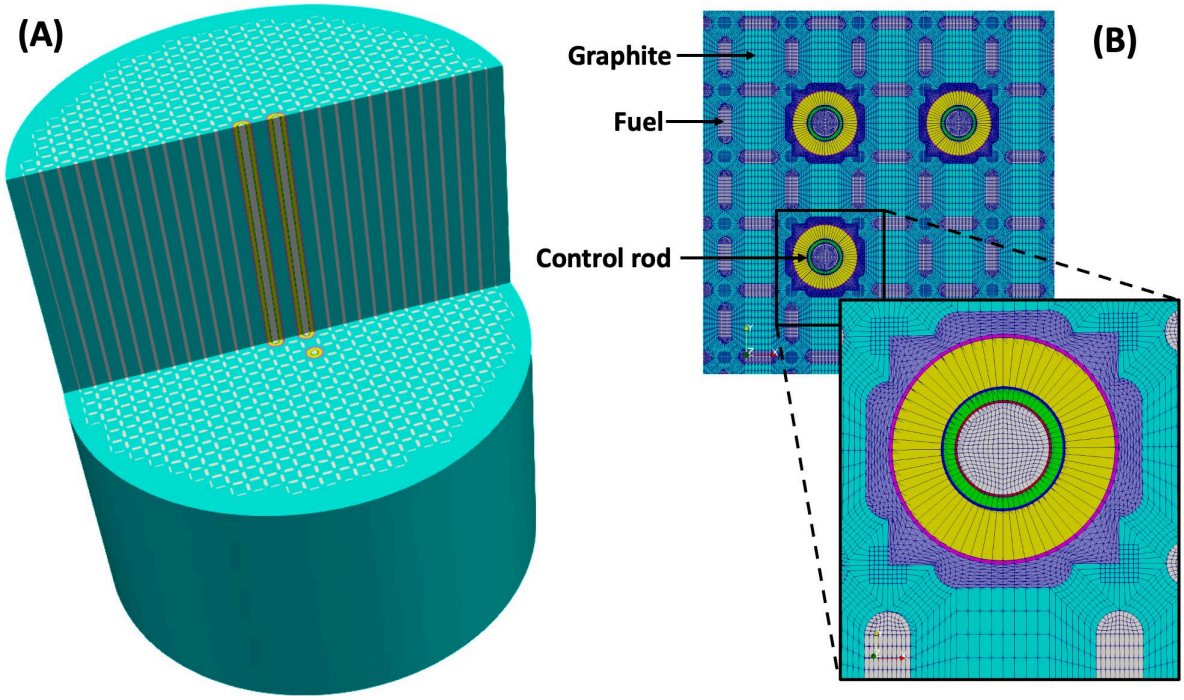


Figure 31: MOOSE-generated MSRE mesh: (a) 3D full core (about a quarter of core was removed to show the internal structures); and (b) cross-section near the fuel zone (core center) and view of control rod zone

IX. CONCLUSIONS

Capabilities to mesh common reactor geometries have been added to the open-source MOOSE framework in the new 'Reactor' module. These capabilities include support for hexagonal geometry, control drum rotation, tagging reactor components like pins, assemblies, and planes on the mesh for output processing, core periphery meshing, and streamlining the construction of Cartesian and hexagonal cores by permitting material assignment at mesh generation time. These open-source meshing capabilities are available to any MOOSE-based physics solver and may also be used to export Exodus II meshes of reactor geometries for use in other finite element-based solvers. The tools are currently most beneficial for producing conformal 2D or 3D (extruded) meshes for regular Cartesian and hexagonal geometries. Non-conformal meshing and

generation of axial features outside simple extrusion are not currently supported by these tools.

Physics simulations using MOOSE-based meshes have been verified to match results which leverage external meshing software such as Cubit or Argonne's Mesh Tools. MOOSE's Reactor module provides significant advantages compared to the use of external meshing tools when analyzing Cartesian and hexagonal reactor lattices using MOOSE-based applications: accessibility to the end user, low barrier to entry for new users, speed of mesh generation, volume preservation of meshed fuel pins, and simplification of analysis workflow when using MOOSE applications. The application of MOOSE's meshing tools including the Reactor Module have thus far been demonstrated on liquid metal cooled fast reactors, heat pipe microreactors, gas cooled reactors, and molten salt reactors. The use of the new reporting IDs enabled by the Reactor module permit users to generate neutronics tallies of interest on specific geometric regions like pins, assemblies, and planes. In summary, the newly developed Reactor module of MOOSE allows physics analysts to quickly generate accurate finite element meshes for many reactor problems of interest. Additional activities are underway to expand the meshing functionality of MOOSE and further streamline the workflow for reactor analysts seeking to perform MOOSE-based reactor analysis.

ACKNOWLEDGMENTS

This work was funded by the Department of Energy Office of Nuclear Energy Advanced Modeling and Simulation Program (DOE-NEAMS). The authors gratefully acknowledge the computing resources provided on Bebop, a high-performance computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory. The submitted manuscript has been created by UChicago

Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. This manuscript was also authored by Battelle Energy Alliance LLC under contract no. DE-AC07-05ID14517 with the U.S. Department of Energy (DOE). The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan.

REFERENCES

1. C. Permann, et al, "MOOSE: Enabling massively parallel multiphysics simulation," *SoftwareX,* **11**(2352-7110), pp. 100430 (2020).
2. C. Stanek, "Overview of DOE-NE NEAMS Program", LA-UR-19-22247, Los Alamos National Laboratory, Los Alamos, NM (2019).
3. C. Lee, et al, "Griffin Software Development Plan," INL/EXT-21-63185 & ANL/NSE-21/23, Idaho National Laboratory and Argonne National Laboratory Technical Report (2021).
4. R. Williamson, et al, "BISON: A Flexible Code for Advanced Simulation of the Performance of Multiple Nuclear Fuel Forms," *Nuclear Technology*, **207**(7), pp. 954-980 (2021).
5. "The CUBIT Geometry and Meshing Toolkit", https://cubit.sandia.gov/ (2021).
6. "Ansys Mechanical Finite Element Analysis (FEA) Software for Structural Engineering", https://www.ansys.com/products/structures/ansys-mechanical (2022).
7. T. Tautges and R. Jain, "Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach," *Engineering with Computers,* **28**(4), pp. 319-329 (2012).
8. M. Smith and E. Shemon, "User Manual for the PROTEUS Mesh Tools," ANL/NE-15/17 Rev. 1.0, Argonne National Laboratory, Argonne, IL (2015).
9. Y. Jung, C. Lee, and M. Smith, "PROTEUS-MOC User Manual," ANL/NSE-18/10, Argonne National Laboratory, Argonne, IL (2018).
10. MOOSE Framework Website, "Reactor Module". https://mooseframework.inl.gov/modules/reactor/index.html (2022)
11. E. Shemon, Y.S. Jung, S. Kumar, Y. Miao, K. Mo, A. Oaks, and S. Richards, "MOOSE Framework Meshing Enhancements to Support Reactor Physics Analysis," ANL/NSE-21/43, Argonne National Laboratory, Argonne, IL (2021).

12. E. Shemon et al., “MOOSE Framework Enhancements for Meshing Reactor Geometries,” PHYSOR 2022, Pittsburgh, PA, May 15-20 (2022).
13. S. Kumar, et al, “Physics Demonstration and Verification of MOOSE Framework Reactor Module Meshing Capabilities,” PHYSOR 2022, Pittsburgh, PA, May 15-20 (2022).
14. J. N. Hasse, “Poly2Tri Github Site,” https://github.com/jhasse/poly2tri (2021)
15. B.S. Kirk, J.W. Peterson, R.H. Stogner and G.F. Carey, "libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations," *Engineering with Computers*, **22**, 237-254 (2006).
16. D. R. Gaston, "Parallel, asynchronous ray-tracing for scalable, 3D, full-core method of characteristics neutron transport on unstructured mesh." Doctoral dissertation, Massachusetts Institute of Technology, 2020.
17. V. Laboure, private communication, May 2022.
18. Y. Fujiwara, et al, “Design of High Temperature Engineering Test Reactor (HTTR), JSME Series in Thermal and Nuclear Power Generation, High Temperature Gas-Cooled Reactors, V5, pp 17-177 (2021).
19. R. Stogner, private communication, June 2022.
20. J. Hansel, et al, “Sockeye: A One-Dimensional, Two-Phase, Compressible Flow Heat Pipe Application,” Nuclear Technology, 207(7), pp. 1096-1117 (2021).
21. G. Grasso, A. Levinsky, F. Franceschini, P. Ferroni, “A MOX-Fuel core Configuration for the Westinghouse Lead Fast Reactor”, International Congress on Advances in Nuclear Power Plants (ICAPP), France, May 12-15, 2019.
22. C. H. Lee and W. S. Yang, “MC2-3: Multigroup Cross Section Generation Code for Fast Reactor Anaysis,” ANL/NE-11-41 Rev. 1, Argonne National Laboratory (2012).
23. R. Hu, et al, “SAM Theory Manual,” ANL/NE-17/4 Rev. 1, Argonne National Laboratory (2021).
24. N. Stauff, et al, “Detailed Analyses of a TRISO-fueled Microreactor: Modeling of a Micro-Reactor System using NEAMS Tools,” ANL/NEAMS-21/3, Argonne National Laboratory (2021).
25. “Verification and Validation of LMFBR Static Core Mechanics Codes Part I,” IWGFR/75, International Atomic Energy Agency, Vienna, Austria (1990).
26. A. Abdelhameed, Y. Cao, D. Nunez, Y. Miao, K. Mo, C. Lee, E. Shemon, N. E. Stauff, “High-Fidelity Multiphysics Modeling of Load Following for 3-D Gas-Cooled Microreactor Assembly using NEAMS Codes,” 2022 ANS Winter Meeting and Technology Expo, 2022, *accepted*.
27. N. E. Stauff, A. Abdelhameed, Y. Cao, D. Nunez, Y. Miao, K. Mo, C. Lee, C. Matthews, E. Shemon, J. Thomas, “Applications of NEAMS Codes for Multiphysics Modeling of Several Microreactors Problems,” 2022 ANS Winter Meeting and Technology Expo, 2022, *accepted.*
28. D. I. Poston , M. A. Gibson , T. Godfroy & P. R. McClure, “KRUSTY Reactor Design”, J. Nucl. Tech., V206, S13-S30, 2020.
29. B. Wilkerson et al., “Thermo-Mechanical-Neutronic Considerations for DireWolf Modeling of the KRUSTY Criticality Experiment,” LA-UR-21-30496, Los Alamos National Laboratory, October 21, 2021.
30. P. N. Haubenreich and J.R. Engel, “Experience with the molten-salt reactor experiment”, J. Nuclear Applications and Technology, V8(2):118–136, 1970.
31. T. Fei, S. Shahbazi, J. Fang, D. Shaver, “Validation of NEAMS Tools Using MSRE Data”, ANL/NSE-22/48, Argonne National Laboratory, August 2022.

32. M. Fratoni, D. Shen, G. Ilas, and J. Powers, "Molten Salt Reactor Experiment Benchmark Evaluation," DOE-UCB-8542, University of California-Berkeley and Oak Ridge National Laboratory, 2020.