ANL/NEAMS-20/2 Rev. 1

Status of the NEAMS and ARC fast reactor tools integration to the NEAMS Workbench

FY-2022 progress report

Nuclear Science and Engineering Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

ANL/NEAMS-20/2 Rev. 1

Status of the NEAMS and ARC fast reactor tools integration to the NEAMS Workbench

FY-2022 progress report

prepared by
N. Stauff, M. Atz, A. Abdelhameed, K. Kiesling, S. Kumar, Y. Jung
Nuclear Science and Engineering Division, Argonne National Laboratory

P. Shriwise
Computational Science Division Division, Argonne National Laboratory

September 30, 2022

ACKNOWLEDGMENT

This work was supported by the Department of Energy – Nuclear Energy Advanced Modeling and Simulation Program (NEAMS) under the Multiphysics Applications Technical Area. Reviews from E. Shemon and R. Lefebvre (ORNL) were very appreciated.

The authors would like to acknowledge contributions from various developers and beta-testers in the past years:

- ANL: T. K. Kim, T. Fei, B. Feng
- ORNL: R. Lefebvre, B. Langley
- Summer interns: N. Gaughan, P. Lartaud, P. Seurin, K. Zeng, A. Rivas

Discussion with Griffin developers C. H. Lee and J. Ortensi (INL) was instrumental to initiate this PyGriffin plan.

EXECUTIVE ABSTRACT

The Workbench initiative was launched in FY-2017 within the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program to facilitate the transition from conventional tools to high-fidelity tools. The NEAMS Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated codes.

The integration of the Argonne Reactor Computation (ARC) suite of codes into the NEAMS Workbench through the PyARC module was initiated in FY-2017. The ARC codes, which focus on fast reactor multiphysics analysis, contain both legacy codes like DIF3D and REBUS-3 that were developed with over 30 years of experience, and newer NEAMS additions like MC^2-3, PERSENT, and PROTEUS. Recent work expended this integration to other tools to support the U.S. fast reactor community such as DASSH for sub-channel thermal-hydraulics, Griffin for high-fidelity deterministic transport calculations, and OpenMC for Monte Carlo simulations (with Shift integration planned for FY-2023).

The integration of the "extended ARC" suite of codes into the NEAMS Workbench interface relies on the PyARC and PyGriffin modules to handles the pre- and post-processing of these codes input, and the runtime environment. The PyARC module together with the NEAMS Workbench interface are both released under Open Source Software licenses.

Integrating the extended ARC codes into the Workbench directly benefits the advanced reactor modeling community by:

- Providing a set of controlled, maintained, documented and validated scripts to generate inputs, which promotes best practices, reduces the learning curve, and facilitates project collaboration.
- Improving the user experience: the Workbench interface provides assistance for building an input through auto-completion, real-time validation, document navigation, and geometry and results visualization.
- Automatize complex calculations and workflows for reactor analysis.
- Helping users transition to high-fidelity NEAMS codes, through Griffin integration within the same input logic as the legacy ARC codes.

The report provides complete description of Workbench/PyARC and PyGriffin, while highlighting recent developments. In FY-2021 and FY-2022, the effort focused on integrating DASSH (Section 3.8), Monte Carlo (in Section 3.9 with OpenMC and planned Shift integration in FY-2023) and Griffin (Section 4) simulation workflows into newly released PyARC version 2.0.0. Additional capabilities were also integrated in response to user requests, such as the automation of the reactivity coefficient post-processing for safety analyses codes (in Section 3.10.2). Significant effort was continued to train and support users from ANL, University of Michigan, Oklo, Moltex, IDOM, and Westinghouse to apply these tools for LFR, MSR, micro-reactor, and SFR core design analyses.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 Introduction

One of the objectives of the Nuclear Energy Advanced Modeling and Simulation (NEAMS) Workbench is to facilitate the deployment of the high-fidelity codes developed within the NEAMS program. The Workbench [1] initiative was launched in FY-2017 to facilitate the transition from conventional tools to high fidelity tools [2]. The Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated codes. The integration of the Argonne Reactor Computation (ARC) suite of codes into the NEAMS Workbench was initiated in FY-2017 [3][4][5][6].

The ARC suite of codes [7] gathers neutronics, thermal hydraulics, safety, and fuel behavior analysis codes. Until FY-2020, the Workbench integration of the ARC codes focused on the deterministic neutronic codes. It includes MC^2-3 [8] for multi-group cross-section processing, DIF3D [9] for flux calculation, REBUS-3 [10] for depletion and equilibrium calculations, PERSENT [11] for perturbation theory calculations (perturbation, sensitivity and uncertainty quantification), GAMSOR [12] for gamma heating calculations, PROTEUS-Nodal transport solver [13], and ORIGEN-S [14] for detailed depletion calculations. In FY-2021 and FY-2022, a wider range of fast reactor modeling capabilities with the sub-channel code DASSH [15], the Monte Carlo code OpenMC [16], and the high-fidelity deterministic code Griffin [17], were integrated in the NEAMS Workbench through PyARC and PyGriffin, as shown in Figure 1-1. For simplicity, the "extended ARC suite of codes" will be referenced and combines the legacy ARC codes together with all the reactor modeling codes integrated in PyARC (PROTEUS, Griffin, OpenMC, DASSH, ORIGEN-S, etc).

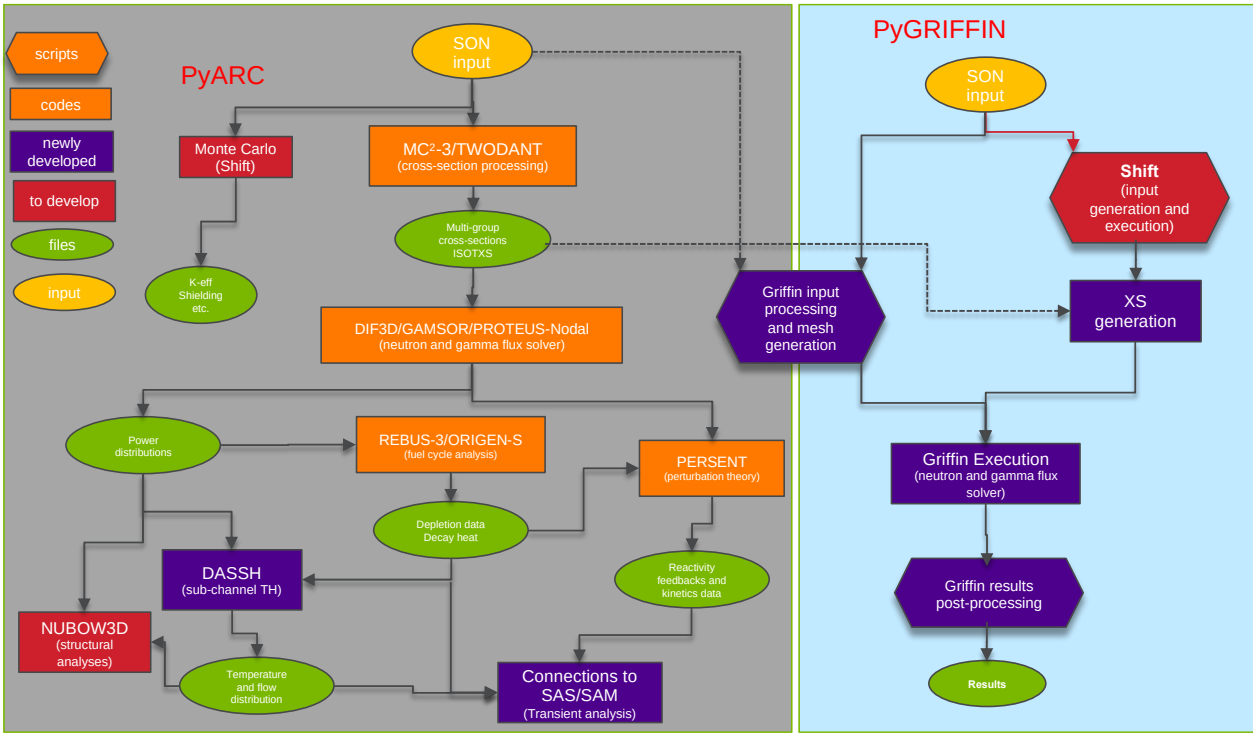

Figure 1-1. Layout of PyARC and PyGriffin with development status of different capabilities.

These “extended ARC” codes are used at national laboratories, universities, and companies for advanced reactor analyses. They gather more than 30 years of development, went through extensive validation and verification, and can solve complex physics phenomena in a very efficient way. However, these codes require knowledge on reactor physics and experience on fast reactor design in order to be familiar with the extent of their capabilities, and users mostly rely on scripts, developed based on their experiences, to generate inputs. Integrating them into the NEAMS Workbench was initiated in FY-2017 through the development of the PyARC module to address these challenges and to improve user experience with these codes by taking advantage of the various benefits brought by the Workbench interface. Both the Workbench and PyARC are being distributed under Open Source Software licenses (PyGriffin is not yet licensed for distribution).

The status of the ARC and NEAMS fast reactor modeling code integration in the NEAMS Workbench is described in this report, focusing on FY-2022 developments (highlighted in this report) and on description of the new released version **2.0.0** of PyARC. The code integration framework in the PyARC bundle of the Workbench is reminded in Chapter 2. The status of the capabilities integrated in PyARC are discussed in Chapter 3. The new PyGriffin tool is described in Chapter 4. Finally, Chapter 5 draws the conclusions and discusses future developments.

2 Framework for extended ARC Tools Integration

Figure 2-1 illustrates how the Workbench interface connects with the extended ARC codes. This is a "black box" type of integration where the Workbench must rely on an opaque runtime module (called PyARC) that conducts the native input formatting. One of the benefits of the "black box" type of integration is that the user is shielded from the original input of the legacy codes. There are several components to the integration that are described in this section:

- Workbench interface: It is developed at ORNL and several components of this interface are required for a code's integration:
  - Common input
  - Templates
  - Visualization
- PyARC module: this is a python module required for "black box" integration that contains the logic for processing the code's inputs, generating the legacy ARC code input, running them, and post-processing the outputs. This module is the glue between the Workbench interface and the extended ARC codes.
- PyGriffin module: similar to PyARC, this python module provides a "black box" integration of Griffin that contains the logic for generating the Griffin (and ultimately Shift) inputs, mesh and cross-section files, running them, and post-processing the outputs.

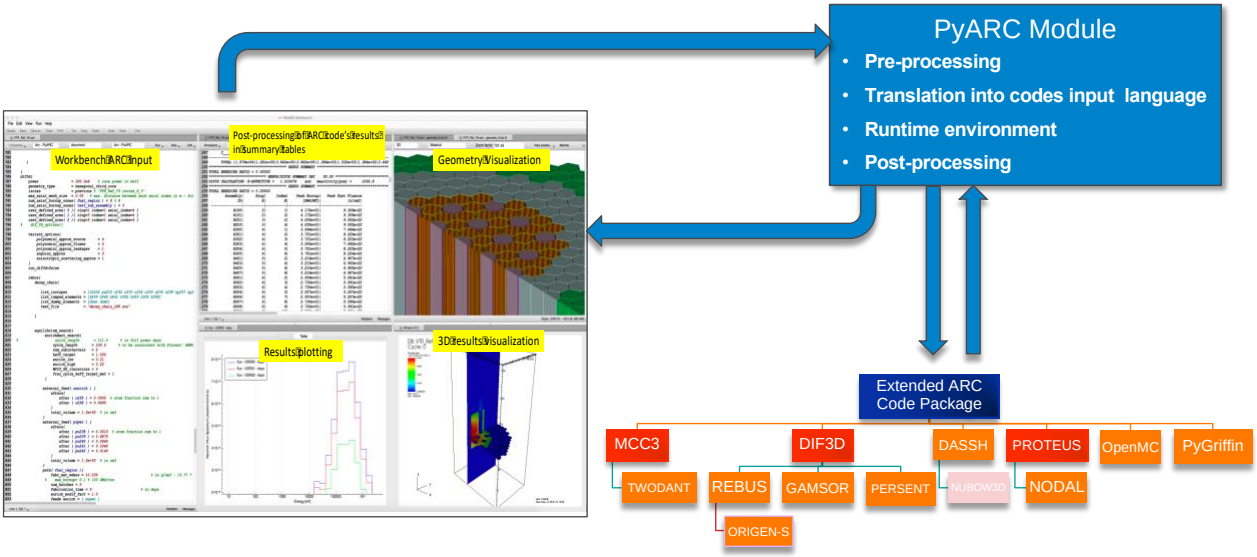

Figure 2-1. Structure of the extended ARC codes integration in the Workbench.

2.1 The Workbench Interface

The Workbench [1] interface is developed at ORNL and designed to assist new users, while not obstructing experienced ones. The Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated

codes. For instance, the user is guided by the auto-completion capability in the Workbench to build its core model in the "common input" structure.

2.1.1 Common input

The Workbench input format adopted is described as the "common input" since it is used to generate inputs for all the tools within the extended ARC for integrated problem-dependent cross-section preparation, core analysis, depletion, and sensitivity/uncertainty quantification. The main benefits of the "common input" strategy is to insure every native input uses consistent information and to facilitate project collaboration (since one PyARC or PyGriffin input contains all the problem definition, while each of the extended ARC inputs only contain part of the information).

This "common input" for PyARC allows modeling a reactor geometry in an intuitive and flexible way and was developed with continuous involvement of ARC users. It uses the open source Workbench Analysis Sequence Processor's (WASP) Standard Object Notation (SON) [20] format that enables the auto-completion, real-time input validation, and access to templates, through the Workbench interface. The structure of the PyARC input is shown in Figure 2-1 and a tutorial was developed (detailed in Section 2.4) to explain in detail the input logic to new users. The common input is defined in the "*arc.sch*" file that takes ~5000 lines of code. Detailed documentation of all the input options is provided (in the "*PyARC_README.html*" file of the PyARC package). The user has direct access to the input keyword definitions through the Workbench user-interface upon auto-completion. The input is automatically validated for correctness by the WASP Hierarchical Input Validation Engine (HIVE) upon edit by the user from within the NEAMS Workbench and upon by PyARC.

The inputs used by PyGriffin are further discussed in section 4.

2.1.2 Templates

The Workbench, through its WASP subcomponent, contains the HierarchicAL Input Template Engine (HALITE) developed to expand hierarchical input data into code-specific input. Templates are used to assist users in generating the common input within the Workbench. The common input templates were developed in parallel to the schema and the common input. Those are blocks of input with default values accessible for convenience to the user. A total of **180 templates** were generated for PyARC and 5 templates were generated for PyGriffin. Templates are also relied upon by the Dakota/PyARC coupling, as explained in Section 2.3.2.

2.1.3 Visualization

The Workbench provides different built-in types of visualization capabilities that PyARC benefits from. In particular, it provides visualization of user input problems through built-in input visualization capabilities (see Figure 2-1). This visualization capability supports 2D/3D hexagonal and Cartesian geometries.

The ParaView tool [21] is integrated into the Workbench (versions 4.0 and after) and allows direct visualization of the ARC post-processed outputs. The Workbench interface supports plotting capabilities using line plots, histograms, bar charts, etc. Two types of line plots were implemented

to display the multi-group cross sections processed by MC^2-3 (as illustrated in Figure 2-1), and the region-wise flux spectrum printed by DIF3D and REBUS-3 (also shown in Figure 2-1).

A utility script provides the user 2D plot generation of the core geometry, as shown in Figure 2-2, and of the assembly peak or integrated results obtained with DIF3D in standalone-DIF3D simulations (as illustrated in Figure 3-3) or together with GAMSOR, PERSENT, REBUS, or PROTEUS. Similar plots can be generated for DASSH thermal-hydraulics results leveraging directly DASSH native plotting capability as discussed in Section 3.8.

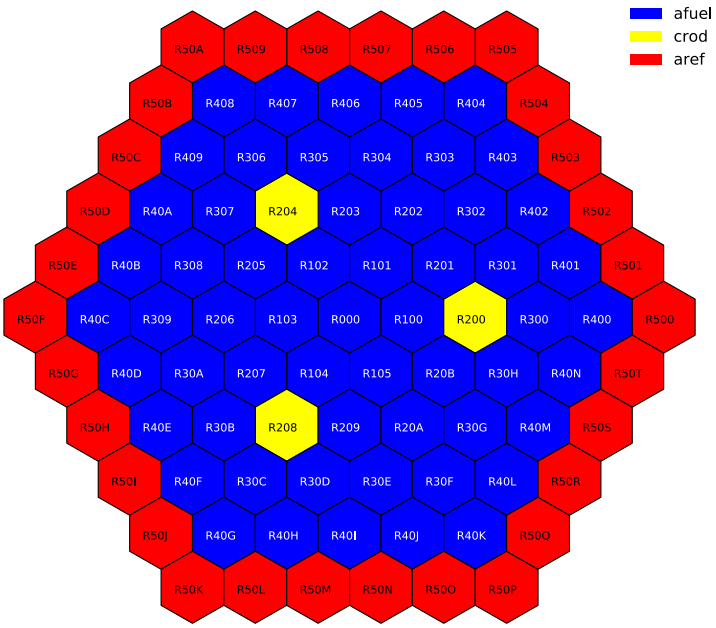

Figure 2-2. Automatic generation of core layout.

2.2 PyARC Module

2.2.1 PyARC Module Introduction

The PyARC module is the glue between the Workbench interface and the ARC codes. It is being released by ANL under an open-source software license and is bundled with the Workbench install for user convenience. For a "black box" integration, this wrapper is essential as it contains the logic to:

- process information from the common input
- perform additional verifications on the core model that the validation engine of the Workbench cannot perform
- pre-process the information, calculating for instance homogenized atom densities in different regions
- generate the ARC codes' native inputs

- handle the runtime environment, which can be very complex. For instance, running MC^2-3 elementary cell calculations in parallel to calculate fine-mesh cross-sections, followed by TWODANT to calculate region-wise flux spectrum, then by MC^2-3 for broad-mesh condensation of the cross-sections, then by REBUS to calculate depleted compositions, then by PERSENT to calculate neutronic feedback coefficients on depleted core compositions, etc.
- post-process the outputs, gathering the main results of the different codes' and creating a single summary file.

The PyARC module is developed through a collaborative environment on GitLab so that new additions are tracked and reviewed. The PyARC module relies on the following sub-modules:

- PyARCModel: loads the input, performs list of additional verification, performs pre-processing on the input
- PyARCUtils: contains utilities procedures
- PyARCUserObject: defines variables and procedures that are used throughout the code
- PyMCC3, PyREBUS, PyPERSENT, PyGAMSOR, PyREBORS, PyPROTEUS, PyDASSH, PyMCSim: contain the logic for input writing, execution, and post-processing for each code
- PyRzmflxCode: contain the logic for input writing, execution, and post-processing for TWODANT
- PyGriffinConnect: connects PyARC to PyGriffin
- The PyARC module also relies on the PySCL module that is developed by ORNL to provide the standard composition library (SCL)
- Several additional plotting or postprocessing utilities

Tests are developed for regression testing after each code modification and prior to committing and pushing modifications to the protected master branch. Currently, more than **295 unit tests** are implemented to check the common input processing, interpretation of the standard composition library, input generation (of MC^2-3, DIF3D, REBUS-3, TWODANT, PERSENT, GAMSOR, PROTEUS, ORIGEN-S, DASSH, Griffin, and OpenMC), execution of the codes, and post-processing of the outputs. Consequently, the unit tests check the pre-processing, input writing, execution, and post-processing logic of PyARC. In V2.0.0, many tests checking error messages returned to the users were implemented.

The PyARC software also uses continuous integration (CI) testing and deployment (CD) bundle infrastructure. This checks all tests at each 'push' to the code repository and ensures all features are functional on Linux and Mac operating systems, and subsequently bundles the new PyARC version into an easily deployable file.

2.2.2 PyARC Workflow

PyARC takes a ".son" input that is generated through the Workbench interface (or using any text editor) following the formatting defined in Section 2.1.1. Examples of inputs are provided in the

training material described in Section 2.4. In addition to the main input, additional input files may be requested to provide decay chain (with REBUS native structure), fission yields of lumped fission products, covariance matrix for uncertainty calculations, etc. A repository of such pre-built files is made available.

PyARC simulations are executed through the "run" button on the Workbench interface. Alternatively, the following commend can be used to execute PyARC input:

```
/link/to/Workbench/rte/entry.sh  /link/to/PyARC/PyARC.py -i my_pyarc_input.son
```

Every PyARC simulations will generate the following output files:

- "*.summary*": summary of the output files from each extended ARC simulations, with "*.extended.summary*" provided more detailed summary results when applicable.
- "*.timeline.out*": timeline of calculation for real-time check of the status of the calculation, and to save the computational time spent at every step of the workflow.
- "*.inp*": all input files generated by PyARC concatenated into one single file
- "*.out*": all output files generated from ARC runs concatenated in one single file
- "*.vtk*": region-wise results in a ParaView-readable format
- "*.zip*": gathers all input, output, isotxs, and summary files
- "*.user_info.out*": important information about the model, including errors, warnings, description of the automatically computed 1D and RZ geometries (see Section 3.1) and of the volume fractions. In particular, the list of isotopes and region IDs are detailed to facilitate advanced ARC users understanding the PyARC-generated native code inputs.

Additional output files may be generated when running different codes, as discussed in Section 3.

2.3 PyARC Applications in Complex Workflows

PyARC streamlines complex ARC workflows (exchange of files and data within MC^2-3, DIF3D, etc.), but it also can be used within complex workflows for reactor design, optimization, and uncertainty analyses. This can be done through scripting (running PyARC instances within a python script) or coupling with optimization or sensitivity analyses codes such as DAKOTA, as discussed in this section.

2.3.1 PyARC Coupling with WATTS [23]

New in V2.0.0

The WATTS (Workflow and Template Toolkit for Simulation) is an open-source workflow management tool developed at Argonne (under LDRD funding). This framework is meant at streamlining interaction within the various integrated reactor design and economics modeling tools. A plugin to PyARC was developed which provides its users knowledgeable with Python to perform a wide range of analyses as demonstrated with several tutorial examples. WATTS can be especially useful to define complex workflows (involving various PyARC steps for instance, or

involving tools outside of PyARC), and performing sensitivity/optimization analyses on such workflow.

2.3.2 PyARC Coupling with Dakota within Workbench (could be applied straightforwardly also to PyGriffin)

The Workbench provides a common user interface for model creation allowing for its integrated codes to communicate and work together with limited coupling development. The feasibility and benefits of using the Workbench as a coupling mechanism between the Dakota [24] code and PyARC was demonstrated in [25],[26],[27], and [28]. The Dakota software maintained by Sandia National Laboratory is a sensitivity analysis/uncertainty quantification (SA/UQ) and optimization toolkit with over 20 years of supporting development. Dakota provides advanced mathematical methods to vary one code's input parameters and analyze the output results for optimization and uncertainty quantification analyses.

2.3.2.1 Workflow Implemented

The workflow in Figure 2-3 was developed to allow Dakota to drive the PyARC calculations within the Workbench interface. For SA/UQ or optimization analyses, the PyARC input is perturbed by a sequence of random values from Dakota. After the ARC runs are performed with different sampled input values, Dakota evaluates the user-specified responses of interest. For SA/UQ types of analyses, Dakota performs statistical analysis on response functions. For optimization problems, it selects best performing solutions and generate new samples.

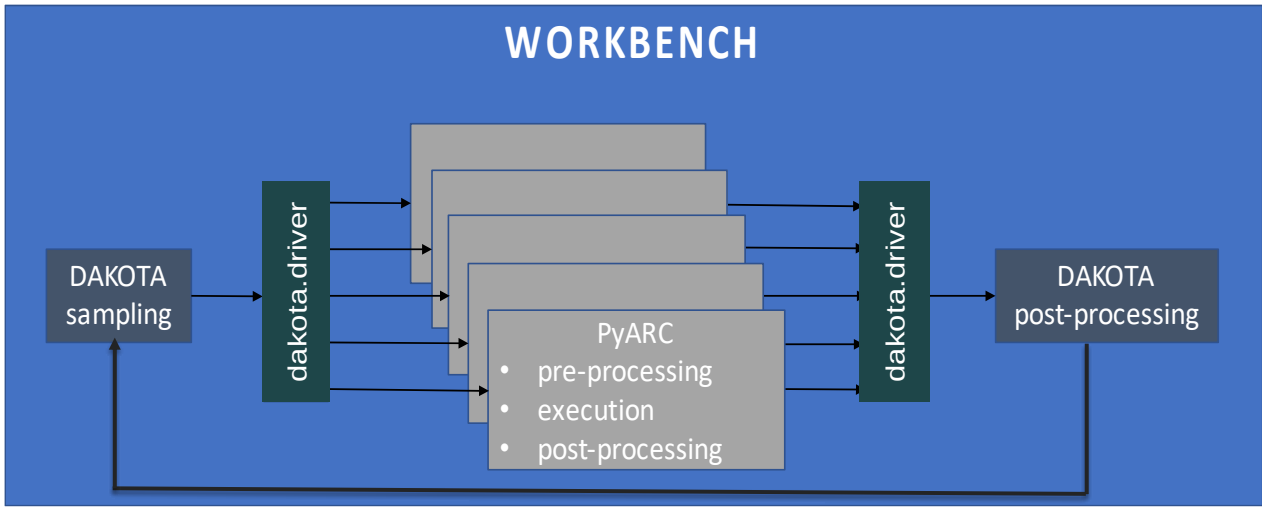

Figure 2-3. Schematic of the Dakota/PyARC coupling.

Three files are required in this workflow:

- Dakota input: Input built by the analyst with the aid of the Workbench that describes the sensitivity, uncertainty or optimization problem options.
- PyARC "common input": This is the PyARC input built by the analyst with the aid of the Workbench, but saved in a **template** format where some input values are replaced with variables defined in the Dakota input.

- PyARC.driver: Input built by the analyst with the aid of the Workbench that connects the Dakota input to the PyARC application. It contains the logic to extract (with customizable grep commands or links to post-processing scripts) different results from the ARC output summary file.

The main PyARC results of interest (core lifetime, inventory, fissile enrichment, peak power, peak fast flux, etc.) are returned in the ".summary" output file so that the user does not need to develop his own postprocessor logic to extract such results to return to Dakota. An example of Dakota/PyARC coupling is detailed in Sample #11.

2.3.2.2 Benefits of the Dakota/PyARC Coupling

The Dakota/ARC coupling would be a significant effort to set up outside of the Workbench since the individual ARC codes use different input logic. This would require developing a script that propagates the input parameters sampled throughout the different codes, which is effectively done with the PyARC module. This coupling enables new capabilities for ARC users since Dakota can be used for driving sensitivity analysis and uncertainty quantification (SA/UQ) calculations [25], [26], and core design optimization with the ARC codes [27], [28].

- ***Optimization problems***

Mathematical optimization methods can be used to investigate a space of input options and find the most promising solutions (usually a compromise between targeted performance). This is especially well suited for advanced reactor design work, where many core options need to be evaluated (size and number of fuel pins, different fuel forms, etc.) to assess their impact on core performance (irradiation testing capabilities, inherent safety performance, etc.). Dakota provides a wide range of advanced optimization methods that can be used to effectively investigate the design space and find the best performing core concepts.

- ***SA/UQ problems***

Dakota extends the SA/UQ capability currently available in the ARC codes (with PERSENT) to propagate the uncertainty on any type of input parameter, and to observe its impact on any output result. In fact, there are different benefits/challenges associated with solving SA/UQ problems through adjoint-based perturbation theory (available with PERSENT [11]) and through stochastic sampling (with Dakota [24]) making both approaches complimentary to each other. The adjoint-based method is usually cheaper in terms of computational resources, is well suited to treat a large number of uncertain parameters such as uncertainties on multi-group cross-sections, and can provide detailed information such as the impact of cross-section values in any energy group, in any core location, on different core parameters. It is usually applied to see the uncertainty impact on the eigenvalue, on reactivity effects and on reaction rates. The stochastic sampling method provides a more general approach that can be applicable to any uncertainty problem considered (including those with changes in core geometry), to analyze the impact uncertain parameters may have on any output of the problem. However, it may require many simulations to reach targeted levels of confidence. An analysis using Dakota to drive PyARC simulations for SA/UQ problems is proposed in Appendix C of [5].

2.4 Training Material

Training material is available to assist a user getting started. It is based on the Advanced Burner Test Reactor (ABTR) design [29], which is a small SFR concept designed for irradiation testing. Discussion of ABTR modeling results and accuracy of the different PyARC solvers are discussed in Appendix A. The training material consists in a list of sample problems provided within the PyARC released bundle, that are documented and that demonstrate and explain the most popular capabilities:

- Sample #1 - MC^2-3 cross-section processing with 1step approach, and modeling full-core with DIF3D diffusion.
- Sample #2 - DIF3D calculation with VARIANT for full-core simulation.
- Sample #3 - MC^2-3 cross-section processing with 1D heterogeneous model and 2steps (TWODANT) calculation, modeling full-core with DIF3D finite diffusion.
- Sample #4 - MC^2-3 cross-section processing with homogeneous - 1step calculation, and core once-through fuel depletion calculation with REBUS with DIF3D finite difference option with third core symmetry.
- Sample #5 - REBUS equilibrium calculation with DIF3D finite difference option with third core symmetry (no reprocessing).
- Sample #6 - REBUS equilibrium calculation with DIF3D finite difference option with third core symmetry (with reprocessing and one iteration between MC^2-3 and REBUS equilibrium).
- Sample #7 - GAMSOR calculation for gamma transport calculation.
- Sample #8 - REBUS once-through fuel depletion calculation with explicitly defined fuel management strategy and third core symmetry.
- Sample #9 - PERSENT perturbation calculations to process needed reactivity coefficients.
- Sample #10 - PERSENT sensitivity calculations to perform SA/UQ analyses on k-eff and reactivity coefficients.
- Sample #11 - Dakota coupling with PyARC to run SA/UQ and Optimization problems
- Sample #12 - PROTEUS-NODAL calculation
- Sample #13 - PROTEUS-NODAL calculation with molten salt fuel (MSR)
- Sample #14 - REBUS to ORIGEN-S calculation
- Sample #16 - DASSH for sub-channel thermal-hydraulics calculations
- Sample #17 - PyGriffin calculation for k-effective at BOL
- Sample #18 - PyMCSim (with OpenMC) calculation for k-effective at BOL

Those Sample problems are available in the "*PyARC/tutorial*" folder that also contains the AdditionalFiles folder where the user has access to different decay chains, lumped fission products, and covariance matrices. It also contains the inputs to the SFR-UAM benchmark problems [30][5].

3 Capabilities Integrated in PyARC

One of the benefits of the "black box" type of integration adopted with the PyARC module is that the user is shielded from the original input of the legacy codes. The associated challenge is that some of the options and code's capabilities may not be made available to the user through the "black box". The ARC integration focuses on the popular and important capabilities of each code to streamline most user's workflows. This section summarizes the status of the extended ARC codes integration within the NEAMS Workbench, while highlighting the work completed in FY-2022.

3.1 MC^2-3 [8]

The MC^2-3 code is developed within the NEAMS program for multi-group cross-section processing for both fast and thermal spectrum reactors. From the Workbench, one can generate cross-sections for pre-generated or user-defined energy-group structure with different scattering orders. The user can merge cross-sections to define lumped fission products used in REBUS-3. Neutron slowing down equation can be solved over a homogenized cell or over a heterogeneous geometry [31] based on 1D cylindrical or slab geometries.

For region-wise group condensation, two approaches were implemented within the Workbench. The first approach consists in generating neutron leakage files from the fuel regions that can be used as external sources in the non-fuel regions (for instance, reflector), as demonstrated in Sample #1. The second approach consists of using TWODANT [32] which is a S_n neutron transport equation solvers, for fine-group (1000 – 2000 groups) flux calculation using an equivalent 2D (RZ or XY) core model. The option to use PARTISN was removed due to a lack of users and difficulty to maintain it. This approach is demonstrated in Sample #3. In terms of output processing, the multi-group cross-sections generated in the ISOTXS file can be plotted automatically in the Workbench interface, as illustrated in Figure 3-1.

Upon completion, PyARC returns the TWODANT and MC^2-3 inputs and outputs (in the "*.inp*" and "*.out*" files) and the multi-group cross-section files:

- The "*.isotxs*" files are binary files that can be re-used by DIF3D/REBUS/PERSENT to avoid re-running the initial MC^2-3 calculation.
- The "*.isotxs.edit*" files are text files containing all the multi-group XS results that can be opened directly with the Workbench to plot automatically the cross-section using the "ISOTXS – ISOTOPE XS" processor (as shown in Figure 3-1).

Different "isotxs" files may be generated at different depletion steps, the following nomenclature logic is used:

- "*.isotxs_R_0*": Reference (un-perturbed), depletion time-step 0 (provided composition or beginning of equilibrium cycle);
- "*.isotxs_D_4*": "D" perturbation, depletion time-step 4.

Additional Notes:

- PyARC provides the option to give a lower threshold to the atom density in heterogeneous regions using the option "*min_dens_het_calc*". This option is especially useful for

simulating coolant void coefficient when using the heterogeneous treatment in MC^2-3 since its 1D transport solver provides convergence issues with very low-density regions.

- MC^2-3 is also applied for computing the DLAYXS, GAMISO and PMATRX required for PERSENT (delayed neutron fraction), PROTEUS (MSR) and GAMSOR calculations, as further discussed in Sections 3.4, 3.5, and 3.6.
- Automatic generation of RZ geometry for TWODANT and 1D geometries for MC^2-3 are available to facilitate the use of these options by reducing pre-processing time to the user and risks of mistakes. These methods are only available for hexagonal-z core geometries. Description of the methods developed and benchmark calculations is provided in Appendix A of [5]. Sample #3 provides an example to get started with these options.

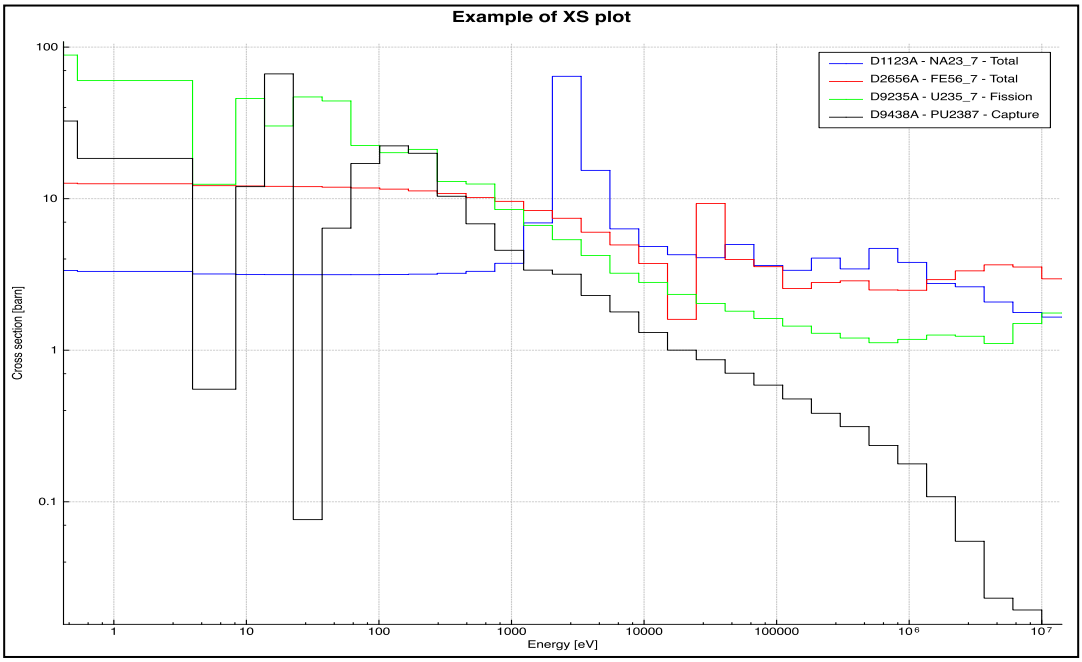

Figure 3-1. Example of multi-group XS plot automatically generated with the Workbench from ISOTXS edit file.

3.2 DIF3D [9]

The DIF3D code is a legacy ARC tool used for neutron and gamma flux calculations on various types of geometries, based on pre-generated multi-group cross-sections. The multi-group cross-sections can be generated using MC^2-3 calculations or a compatible set of previously calculated multi-group cross-sections.

The 2D/3D-Hexagonal and 2D/3D-Cartesian types of geometries are supported through the Workbench. The DIF3D code includes 3 neutron solvers (Nodal, Finite Difference, and VARIANT [33]) that were all enabled. This is illustrated with Samples #1 (Finite Difference) and #2 (VARIANT). Both neutron flux and gamma flux (discussed in Section 3.5) calculations are integrated into the Workbench.

PyARC returns the full DIF3D input and output (in the "*.inp*" and "*.out*" files). Post-processing of DIF3D output was implemented by printing the main information of interest to a user (e.g., the neutron flux in different core areas, integrated flux and power per assemblies) in the summary file ("*.summary*"). When opening this "*.summary*" file with the Workbench, the user can use the "*flux spectrum*" processor to automatically plot the neutron flux spectrum. Direct visualization of the power density, neutron flux, atom densities, etc., is enabled by opening the generated "*.vtk*" file with ParaView through the Workbench, as also illustrated in Figure 3-2.

As discussed in Section 2.1.3, new 2D visualization is available, through an external script run on the generated summary file, or directly through the PyARC workflow by using the input line: "*calculations/plot_2d = true*". An example of peak power density radial distribution is provided in Figure 3-3.

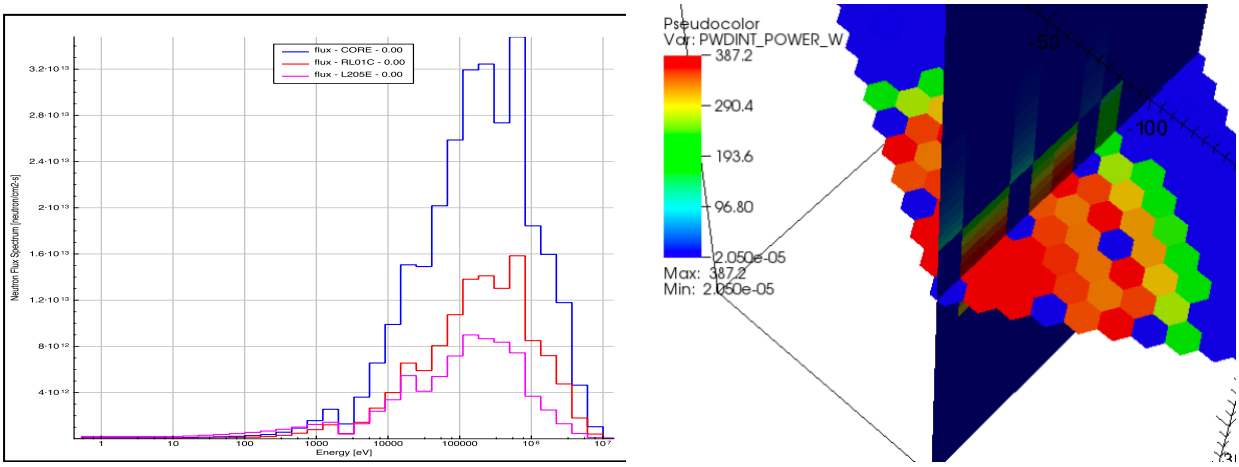

Figure 3-2. Example of visualizations available for DIF3D and REBUS-3: neutron flux spectrum (left) and power map (right) calculated and plotted with the Workbench.

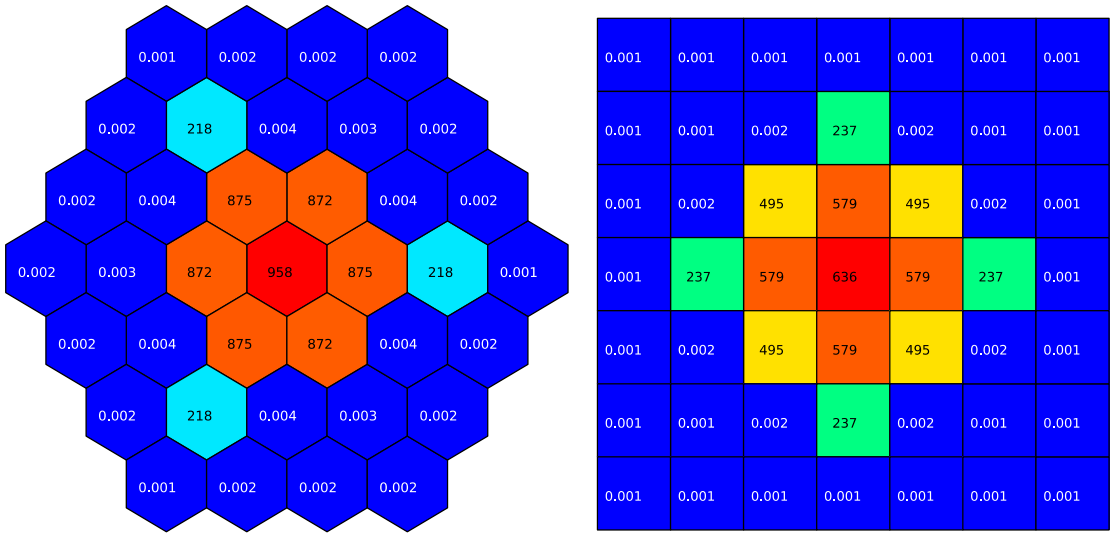

Figure 3-3. Example of 2D plot visualization of peak power density per assembly for hexagonal and Cartesian geometry.

3.3 REBUS-3 [10]

REBUS-3 is a legacy ARC code used for fuel cycle analysis using DIF3D solvers. It allows a wide range of fuel cycle modeling options such as assembly shuffling, enrichment or cycle length search at equilibrium state. In terms of post-processing, the same capabilities developed for DIF3D are made available with REBUS-3 at every time-step of the depletion calculation. In particular, the "*.rebus_X.vtk*" file is generated by REBUS-3 at time *X* days. Some specific information are also printed out in the "*.summary*" file such as the peak burnup and fast fluence, the optimized enrichment (computed for equilibrium calculation), etc.

PyARC integration of REBUS-3 allows its user specifying its own decay chain by directly providing an external text file containing the decay chain input from REBUS-3 (cards 09, 24, 25), which is being parsed in PyARC. Some examples of decay chains (and associated fission yields) are provided in AdditionalFiles.

The once-through depletion capability is illustrated in Sample #4. The original option to re-calculate the multi-group cross-sections at different time-steps of the depletion was implemented in the case of the once-through depletion, as illustrated in Figure 3-4. This capability can be especially relevant when modeling very high burnup fuel or a reactor with thermalized neutron flux.

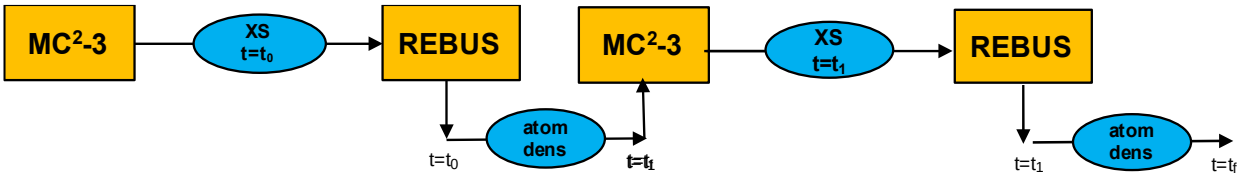

Figure 3-4. Multi-step depletion procedure implemented in PyARC.

The enrichment or cycle length search options at equilibrium state were also integrated, where the user can define reprocessing plants, external feeds, and fuel-cycle strategies, as illustrated in Samples #5 and #6. The original option was implemented to iterate between the multi-group cross-sections computed with MC^2-3 at beginning of equilibrium cycle, and the equilibrium search calculation performed with REBUS-3, as illustrated in Figure 3-5.

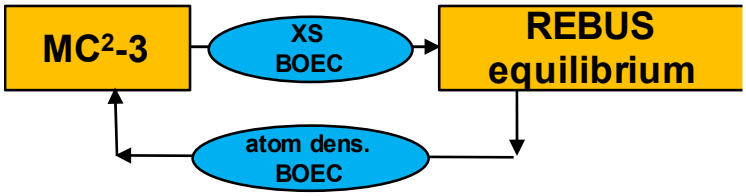

Figure 3-5. Equilibrium cross-section iteration procedure implemented in PyARC.

The explicit fuel management capability of REBUS is enabled using the "*once_through_shuffling*" option. This option allows defining different cycles and the paths for each assembly, allowing to discharge, move or reload assemblies. Sample #8 illustrates this capability and explains the input logic.

The REBUS-3 code relies on simplified decay chain tracking "only" ~200 isotopes. For higher fidelity depletion calculations that are typically needed for decay heat simulations, the ORIGEN-S code can be used, as discussed in Section 3.7.

3.4 PERSENT [11]

PERSENT is a perturbation theory code developed within the NEAMS program and based on the neutron transport equation in a 2D or 3D geometry. It allows calculating reactivity feedback coefficients, sensitivity coefficients [34], and nuclear data uncertainties. Perturbation and sensitivity calculations were implemented on eigenvalue, beta and lambda problems and can be automatically run at different depletion steps (computed with REBUS-3). The user can define which materials are perturbed with a change in density or in temperature or which surfaces are perturbed (only for direct DIF3D perturbations, as explained below). The cross-sections of the perturbed composition can be automatically re-calculated both for perturbation and for sensitivity calculations. The nuclear data uncertainties can be estimated on the eigenvalue, beta, lambda, and on the reactivity coefficients automatically by providing a covariance matrix (in a PERSENT-compatible file format).

Direct DIF3D calculations are enabled as an alternative to PERSENT perturbation theory calculations. This can be especially useful for double checking the perturbation theory result and for modeling geometric perturbations such as the axial and radial feedback coefficients or the control rod worth. Second, to optimize the workflow of PERSENT calculations, one added the option to perform preliminary un-perturbed DIF3D calculations to use generated flux files (adjoint and forward) in different PERSENT runs. Finally, every PERSENT or DIF3D calculations can be run in parallel on different CPUs.

Visualization of the perturbation results is enabled within the Workbench using ParaView [21]: for instance, "*.persent_**P**_ref.vtk*" is the vtk file generated by PERSENT for perturbation ***P***. For illustration purposes, Figure 3-6 shows the distribution of the sodium void worth calculated on an SFR design and plotted by ParaView within the Workbench.

Perturbation calculations can be run based on already perturbed geometry or compositions using the "*ref_is_pert_config*" option. This allows simulation of the voided Doppler coefficient, or of any reactivity coefficients with control rods inserted at critical location through a geometry perturbation. Samples #9 and #10 were developed to train users in correctly computing reactivity coefficients feedback as required for safety analyses (using SAS4A/SASSYS).

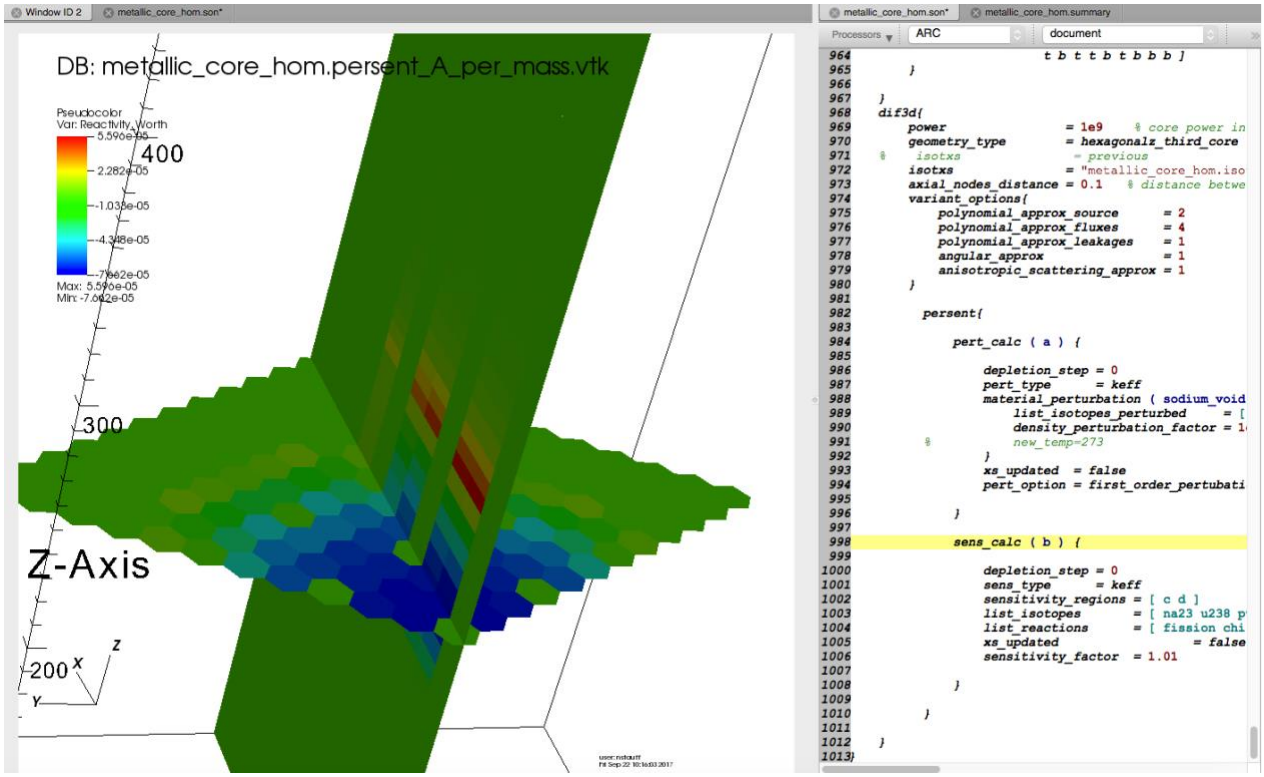

Figure 3-6. Sodium void worth distribution [kg^{-1}] calculated and plotted within the Workbench.

3.5 GAMSOR [12]

The GAMSOR code is a legacy code developed within the ARC suite to assist analysts in calculating gamma heating. GAMSOR computes the gamma flux through a sequence of MC^2-3 for neutron and gamma cross-section preparation, and DIF3D calculations to solve the neutron flux, the gamma flux, and then to combine the results for summary edition. This complex workflow is summarized in Figure 3-7.

The GAMSOR Workflow is fully integrated within the PyARC interface and Sample #7 provides training material. The GAMSOR user input proposed through PyARC only requires the energy-group structure of GAMMA calculation and the list of depletion time-steps on which to perform GAMMA calculations.

In terms of post-processing, similar capabilities as developed for DIF3D are made available, as illustrated in Figure 3-8. The user has access to summary tables with assembly-integrated neutron and gamma powers in the "*.summary*" file. Automatic 2D plotting of the power map is also available. The ParaView visualization tool can be used for 3D visualization of the neutron and gamma power. Finally, the region-wise neutron and gamma power levels are provided back to the user in the "*.zip/gamsor_table_*.out*" files.

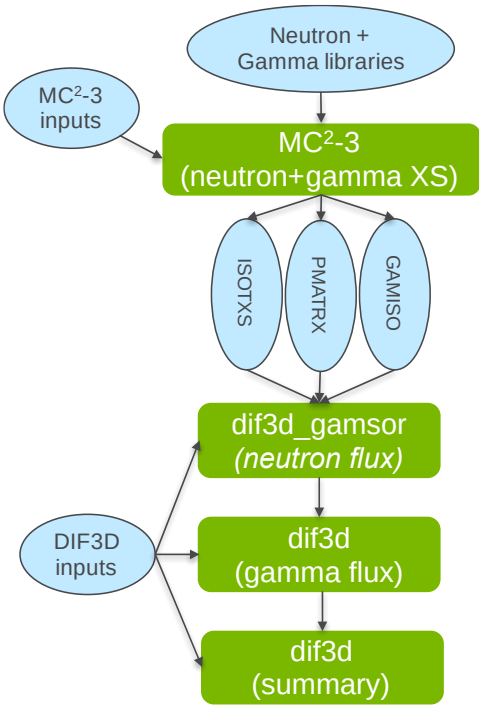

Figure 3-7. GAMSOR workflow implemented in PyARC.

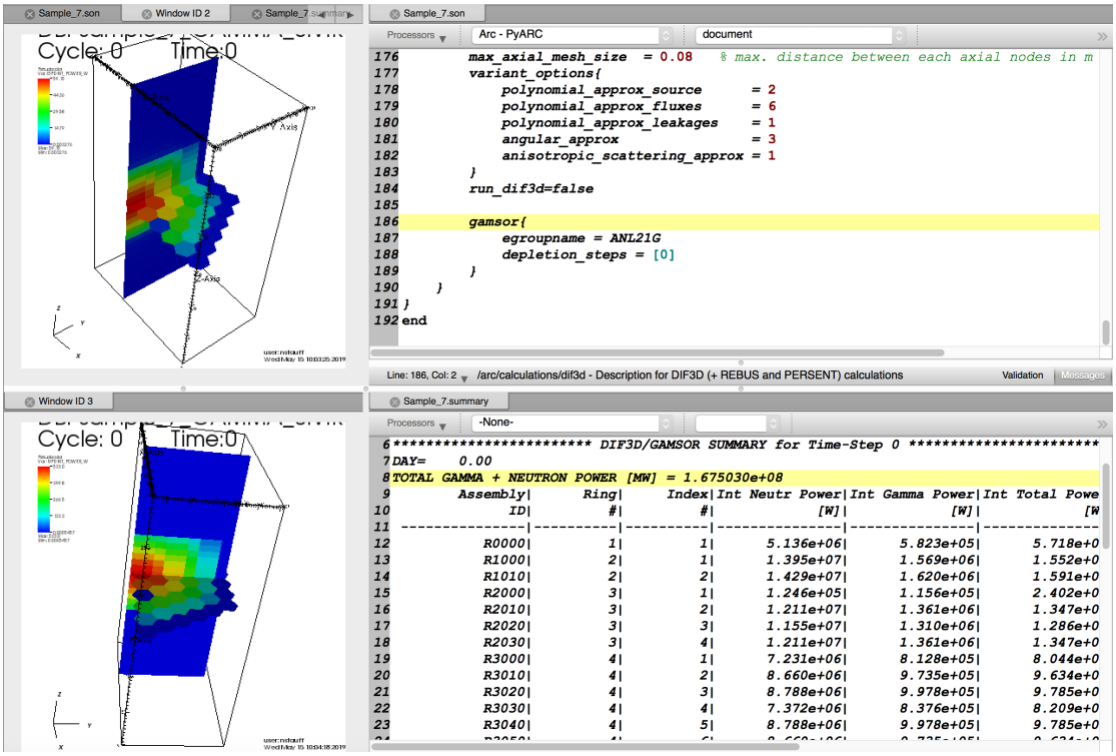

Figure 3-8. Example of enabled GAMSOR input and result visualization.

3.6 PROTEUS Nodal [13]

The PROTEUS code developed under NEAMS project is a high-fidelity deterministic neutron transport code based on unstructured finite element meshes, which solves the steady-state and transient neutron transport problem using the method of characteristics (MOC) or the discrete ordinate (SN) method as high-fidelity neutron transport solvers. Additionally, the nodal transport method (NODAL) option for structured geometries is available to provide a fast running solution option within the same framework so that a user can choose a level of solution fidelity and computational resource requirements depending on its need. The PROTEUS codes were integrated into the NEAMS Workbench interface to improve the usability by taking advantage of the PyARC framework. For the PROTEUS integrations, the extension of the PyARC module referred to its PyPROTEUS sub-module was developed for connecting the Workbench interfaces using the "black box" approach of code integration. Figure 3-9 illustrates how the Workbench interface connects with the PROTEUS codes through the PyPROTEUS/PyARC wrappers. Currently, only N*ODAL* is fully integrated for steady-state calculations. The integration supports all the features of the Workbench/PyARC framework (input generation, workflow management, post-processing). The *MOC* solver was partially integrated for steady-state and transient calculations. However, this MOC module was removed from deployed version of PyARC due to lack of maintenance and of user-base. *SN* solver has not been integrated.

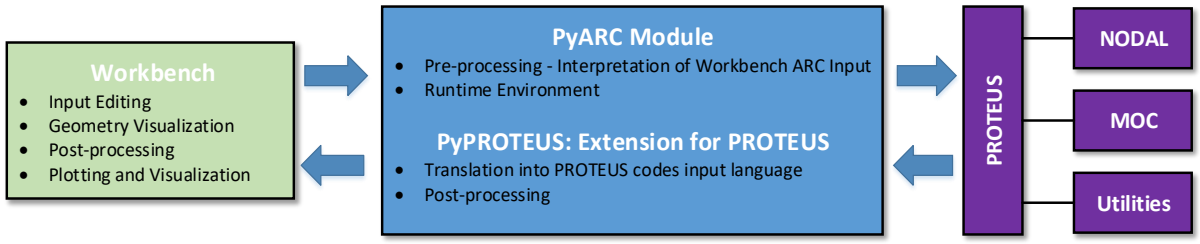

Figure 3-9. Structure of the PROTEUS integration in the PyARC and the Workbench.

The PROTEUS-NODAL code is a nodal transport solver based on homogenized assemblies that provides a conventional fidelity level in a consistent PROTEUS code framework. Two solver methodologies were implemented on that framework that constitute the nodal solver capabilities: P_N and Simplified P_N (SP_N). The P_N approach is the identical methodology used in VARIANT although the release version only handles diffusion theory on Cartesian and hexagonal grids. For the SP_N approach, a transverse integrated nodal methodology was built on the hexagonal grid model utilizing up to a SP_3 approximation. The PROTEUS-NODAL code has capabilities to solve steady-state and transient problems. Additionally, the flowing fuel modeling capability enables to model the impact on the neutron precursor distribution for a flowing fuel in molten salt reactor (MSR) analyses. Currently, only the steady-state and MSR analysis capabilities are fully integrated into the Workbench, while transient analysis capability should be implemented in the future.

The workflow for PROTEUS-NODAL calculations was built upon the existing sub-modules for DIF3D calculations and the implemented workflow is illustrated in Figure 3-10. The PyPROTEUS modules return the following input files for PROTEUS-NODAL execution:

- *Mesh:* defines geometrical dimensions, region configurations, and boundary conditions.

- *Assignment*: defines compositions and assigns them to the geometrical regions.
- *Driver*: defines the simulation parameters such as power level, convergence criteria, and iteration limits.

Along with these PROTEUS-specific input files, the PyARC module generates the cross sections and the optional delayed neutron parameters in the ISOTXS and DLAYXS file formats respectively. The PROTEUS-NODAL calculation is executed via the runtime environment of PyARC. Once the calculation is completed, the PROTEUS-NODAL code produces three basic types of outputs as:

- *Main Text-based Screen Output:* contains confirmation that the input was imported successfully, computing timing summaries, and eigenvalue iteration history results.
- *Detailed Summary Output*: contains full solution in the entire domain which is exported to an organized ASCII file for detailed analysis.
- *Visualization Output*: contains the solutions of primary variables such as flux and power in the VTK file format which is readable by ParaView (within the Workbench).

Similar to the DIF3D calculation capability, post-processing of PROTEUS-NODAL output was implemented by printing the main information of interest to a user in the summary file ("*.summary*"). When opening this "*.summary*" file with the Workbench, the user can use the "*flux spectrum*" processing to automatically plot the neutron flux spectrum. The direct visualization of the primary variables is enabled by opening the generated "*.vtk*" file with ParaView through the Workbench. The implemented post-processing capabilities are illustrated in Figure 3-11. A sample input #12 demonstrating the Nodal workflow was developed within the released tutorial.

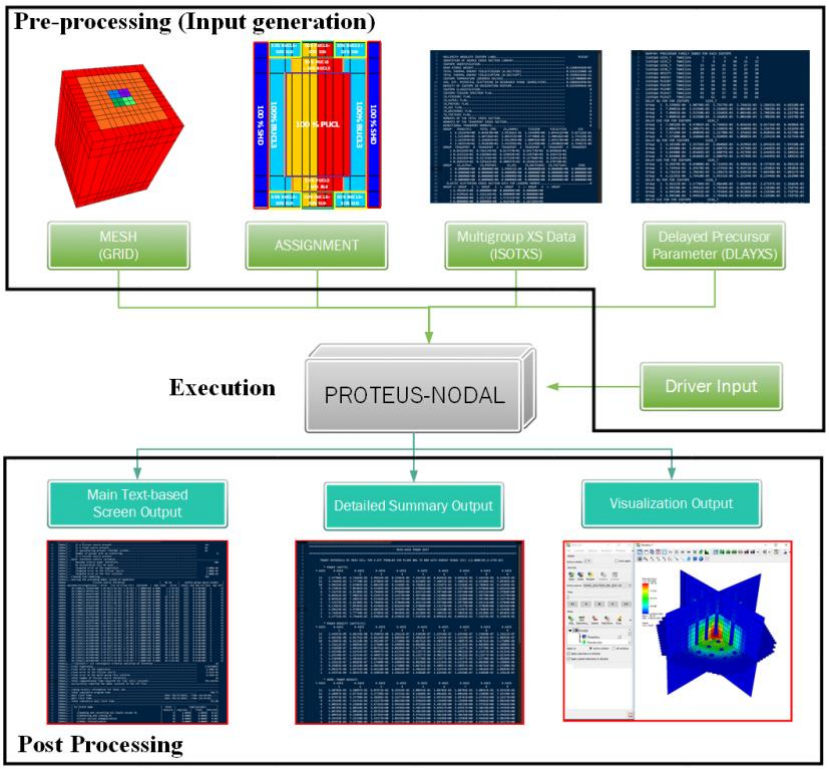

Figure 3-10. PROTEUS-NODAL workflow implemented in PyARC.

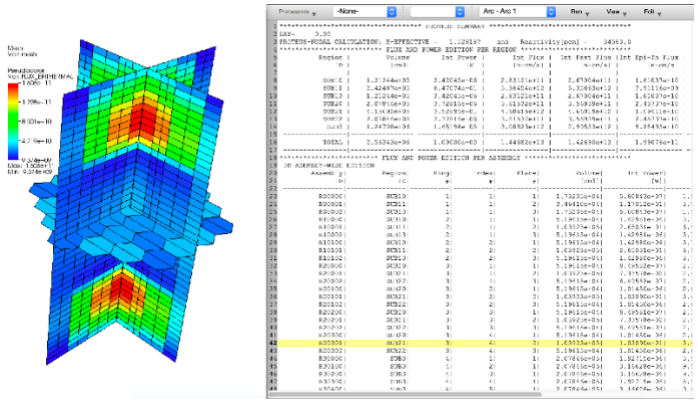

Figure 3-11. Example of post-processing for PROTEUS-NODAL: visualization of flux map (left) and assembly-wise summary table (right).

For enabling the MSR analysis capability within the Workbench interfaces, as illustrated in Figure 3-12, the additional pre-process logic was implemented to translate the Workbench input format of flowing fuel model description into the associated PROTEUS-NODAL input format. The DLAYXS file generation process was streamlined within the execution logic of PROTEUS-NODAL to provide delayed neutron precursor parameters. A sample input #13 demonstrating the MSR modeling was developed within the released tutorial.

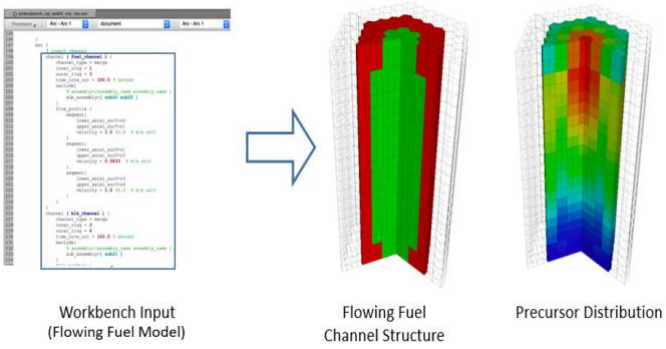

Figure 3-12. Example of MSR calculation within the Workbench.

3.7 ORIGEN-S [14]

The ORIGEN-S code deployed within the SCALE package is used by ARC users for detailed depletion and decay calculations. This is required to compute decay heat, detailed isotopic composition, radio-activity, neutron sources, etc. ORIGEN-S traces more than 1,300 nuclides by solving Bateman equations using pre-generated one-group neutron libraries. A coupling procedure was developed by ARC users to generate problem-dependent one-group cross sections and to reproduce the REBUS depletion simulation using ORIGEN-S. This procedure was integrated into PyARC within the PyREBORS.py function. Figure 3-13 shows the coupling procedures for depletion calculations using ORIGEN-S along with the REBUS-3 physics code.

In this PyREBORS procedure, problem-dependent effective one-group cross sections are obtained from the depletion calculations using the REBUS-3 code. However, since the physics code handles only about tens or hundreds of nuclides in the depletion calculation, the available one-group cross-sections are limited to the nuclides that are modeled by the ARC codes (limited to the ~200 isotopes available in the MC^2-3 library). The one-group cross sections are generated by using the COUPLE code [14] in the coupling-procedure with the ORIGEN-S code. It is noted that the SCALE code package has multi-group libraries with 238, 200, 49, or 40 group structures for thermal and fast systems. Thus, the one-group cross sections is generated by condensing those libraries using the problem-dependent neutron spectra obtained from the depletion calculations using REBUS-3.

The user input of the new "*rebus_to_origens*" block in the PyARC is shown in Figure 3-14 (extracted from Sample #14 in the tutorial). The ORIGEN-S calculation re-produces the once-through burnup simulation modeled with REBUS in the PyARC sub-assemblies selected by the user. The rebus calculation can be skipped by providing a pre-generated REBUS output using the "*rebus_output_file*" option. The one-group XS (capture, fission and (n,2n)) computed by REBUS-3 will be used by ORIGEN-S for the isotopes specified by "*list_isotope_XS_transfer*". The other 1-gp XS (for other actinides, reactions, fission products, etc.) are computed based on a detailed flux structure that can be specified with an external file "*detailed_flux.isotxs*", or by using the REBUS-generated flux spectrum (option by default) - the flux is automatically being linearly interpolated by PyARC to match the ORIGEN structure selected. The ORIGEN-S irradiation is then computed based on the initial power or on the flux at each step in every sub-assembly selected (using the

"*power_or_flux*" option). Following irradiation, decay calculations is completed using the cumulative steps specified in "*decay_cumul_steps*".

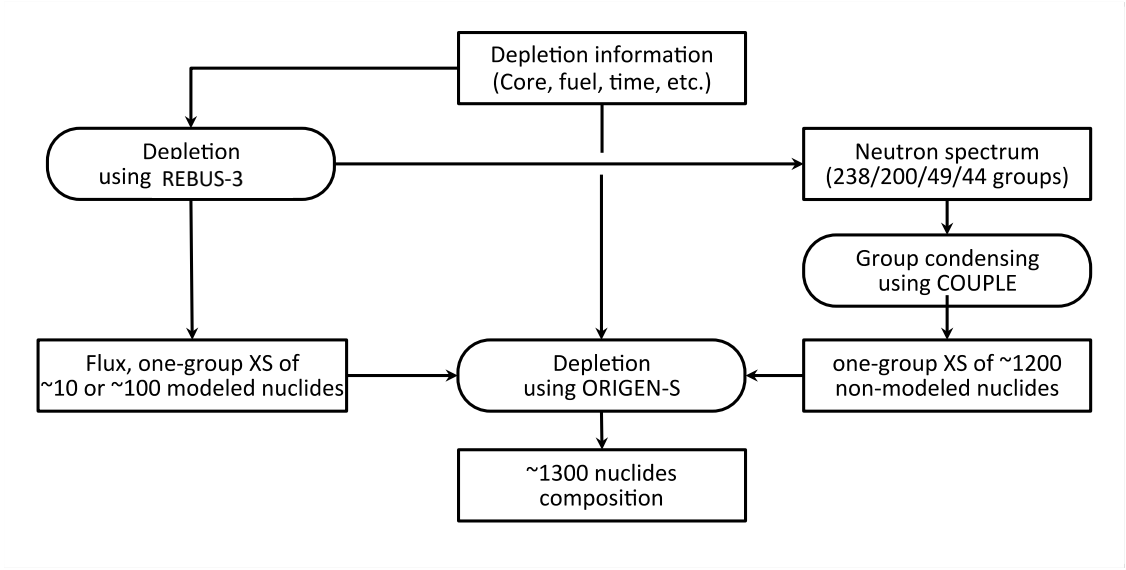

Figure 3-13. Workflow implemented of the REBUS-3 to ORIGEN-S coupling.

```
191        }
192    }
193    rebus_to_origens{
194        power_or_flux = flux
195        decay_cumul_steps = [ 0.1 1 10 100 1000 ] % cumulative
196        origen_base_lib = 44
197        list_subassembly = [ mid_fuel ]
198        list_isotope_XS_transfer = [u235 u238 pu239]
199         detailed_flux_file = "detailed_flux.isotxs"
200        link_to_scale = "/usr/scale6.1/cmds/batch6.1"
201        rebus_output_file = "rebus_sample14.out.isotxs"
202    }
```

```
1343 ************************ ORIGEN SUMMARY - ML05A ****************
1344  IRRADIATION - nuclide concentrations, grams
1345     DAYS | 1.00e-10 | 5.00e+00 | 1.00e+01 | 1.50e+01 | 2.00e+01 |
1346     u234 |1.440e-02 |1.440e-02 |1.440e-02 |1.441e-02 |1.441e-02 |
1347     u235 |5.492e+05 |5.492e+05 |5.492e+05 |5.492e+05 |5.492e+05 |
1348     u236 |1.452e-02 |1.452e-02 |1.452e-02 |1.452e-02 |1.452e-02 |
1349     u238 |2.716e+06 |2.716e+06 |2.716e+06 |2.716e+06 |2.716e+06 |
1350    np237 |1.458e-02 |1.458e-02 |1.458e-02 |1.458e-02 |1.458e-02 |
1351    pu236 |1.452e-02 |1.447e-02 |1.442e-02 |1.438e-02 |1.433e-02 |
1352    pu238 |1.464e-02 |1.495e-02 |1.525e-02 |1.554e-02 |1.583e-02 |
1353    pu239 |1.471e-02 |1.471e-02 |1.472e-02 |1.472e-02 |1.473e-02 |
1354    pu240 |1.477e-02 |1.478e-02 |1.478e-02 |1.479e-02 |1.480e-02 |
1355    pu241 |1.483e-02 |1.482e-02 |1.481e-02 |1.480e-02 |1.479e-02 |
1356    pu242 |1.489e-02 |1.489e-02 |1.489e-02 |1.489e-02 |1.489e-02 |
1357    am241 |1.483e-02 |1.484e-02 |1.485e-02 |1.486e-02 |1.487e-02 |
1358  am242m |1.489e-02 |1.489e-02 |1.489e-02 |1.489e-02 |1.489e-02 |
```

Figure 3-14. Example of coupled depletion PyARC input and summary output.

The ORIGEN input and output are provided back in the "**.zip/origen_XXXX.**", and the main results are extracted in the ".summary" file as shown in Figure 3-14. The results extracted are the nuclide concentration for all the actinides concentrations throughout irradiation and decay, together with their contribution to the total radioactivity and decay heat computed after discharge throughout decay simulation.

<u>Important Notes:</u>

- Only the ORIGEN-S version from the Scale 6.1.3 package was tested.
- The depletion implemented only represent the initial loading of heavy nuclei, while activation products are not tracked yet.
- The default branching ratio of Am-241 are representative of a fast neutron spectrum.

- The ORIGEN-S irradiation is only used to re-produce once-through depletion calculation in a sub-assembly – it is currently not suitable to represent equilibrium search and assembly shuffling problems.

This methodology coupling REBUS-3 to ORIGEN-S should be verified comparing the masses of the main heavy nuclides of the discharged fuel compositions provided by REBUS-3 to the detailed composition generated by ORIGEN-S. Future work should include addressing some of the limitations of the current implementation above-discussed.

3.8 DASSH [15]

New in V2.0.0 (with initial releases starting from V1.5.0)

The Ducted Assembly Steady-State Heat Transfer software (DASSH) is a full-core sub-channel thermal hydraulics code. It is integrated in PyARC to enable sub-channel thermal-hydraulic calculations for hexagonally gridded fast reactor cores with ducted assemblies. The DASSH integration in PyARC provides the users with the capabilities of DASSH in a much-facilitated workflow. For instance, PyARC executes the DASSH code directly following GAMSOR, and it transfers the proper binary files from one application to another. The PyARC-DASSH input only requires DASSH-specific options with minimum overlap with geometry specifications provided in the main PyARC geometry block.

With the multi-dimensional power distribution in the reactor from the DIF3D/GAMSOR calculations in PyARC, DASSH performs direct sub-channel thermal hydraulic calculations at any depletion step to calculate coolant, structure, and fuel temperatures within the reactor core by solving energy balance equations. DASSH offers multiple models for inter-assembly heat transfer, calculates the pin-by-pin temperature distributions in the core, and determines pressure drop as well as other important thermal-hydraulic parameters. Almost all of the currently available DASSH modeling capabilities have been integrated in PyARC, and upcoming DASSH features will be integrated as they are released. The DASSH plotting capabilities are enabled within PyARC, with some examples from the tutorial Sample #16 shown in in Figure 3-15.

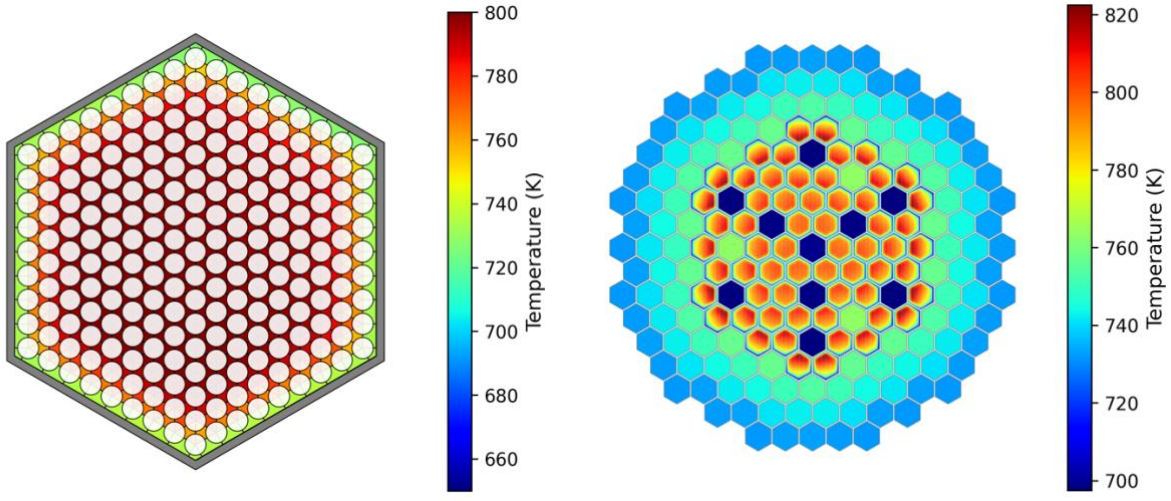

Figure 3-15. Examples of plots from DASSH for subchannel temperatures within 1 assembly (left) core-wide subchannel temperatures (right).

To execute DASSH with PyARC using Workbench (version 5.0.0 or later), DASSH needs to be accessible from the Workbench Python API. To accomplish this, a special installation and execution process has been implemented for DASSH that leverages the Workbench Conda distribution. A script called "dassh_install.sh" was added to PyARC to create a DASSH-specific virtual environment for DASSH; DASSH and its dependencies are installed within this environment as a Python package. Then, at runtime, the "dassh_run.sh" script is called to activate the virtual environment and run DASSH. This development offers a streamlined installation and execution process for DASSH.

3.9 Monte Carlo

New in V2.0.0

The PyMCSim module was developed for PyARC which adds the capability to run Monte Carlo (MC) eigenvalue and volume calculation simulations using the PyARC SON geometry input. This module is designed to be code-agnostic meaning it can be used for multiple MC codes without additional input from the user. The current implementation runs full eigenvalue and volume calculations with OpenMC [16], but the module is aimed at being compatible with the SCALE Shift code [35] in the future.

To use this module, the user supplies a reactor geometry input in the standard PyARC SON geometry input and then sets simulation parameters in the "*mcsim*" block of the calculations in the SON input. Simulation parameters that can be set include number of histories, source parameters, physics treatment, whether to run a volume calculation, parallelism parameters, and output information.

The PyMCSim module reads the standard PyARC SON geometry input and converts it to the appropriate material and geometric definitions using the MC code's Python API. The simulation parameters supplied in the *mcsim* code block are then used for defining runtime parameters, and a simulation using the designated MC code is launched by PyARC. Once the simulation is complete, the relevant output from the MC code is processed and added to the PyARC summary file.

3.9.1 Shift [35]

Due to delay in obtaining an RSICC license for SCALE/Shift in FY-2022, the focus for PyMCSim development was on OpenMC integration. Although Shift has yet to be integrated into PyMCSim, the framework has been developed such that it will be interchangeable with OpenMC when available.

3.9.2 OpenMC [16]

OpenMC is an open-source Monte Carlo tool that can be used for eigenvalue (k-effective) calculations, fixed source simulations, and stochastic volume calculations. OpenMC has been integrated into the PyMCSim module for eigenvalue and volume calculations. The module will first read the material information from the SON input. All types of material, blend, and lumped element definitions supported in the SON input are also supported as OpenMC material definitions. After the materials are defined in the OpenMC format, the module will generate the OpenMC geometry by generating surfaces, cells, and universes with the defined materials and

lattice structures. Models for both 2D and 3D hexagonal or square reactor/pin lattices can be generated. Verification of the generated OpenMC geometry is provided in Appendix B.

After the generation of the geometric model, additional desired physics options, source settings, and run settings are read from the SON input and applied. Three files are then exported that are required for running OpenMC: materials.xml (contains material definitions), geometry.xml (contains geometry definition and material assignments), and settings.xml (contains source, physics, and run settings). OpenMC is then invoked and the simulation is run. Upon successful completion of the simulation(s), the binary HDF5 output files from OpenMC are processed by PyARC and relevant simulation information is added to the summary output and available in the results of the user_object Python object. Results that are processed include:

- Eigenvalue calculation:
    - K-effective (combined)
    - K-collision
    - K-absorption
    - K-tracklength
    - Leakage
- Volume calculation:
    - Volume of each cell
    - Volume of each material

Because OpenMC is implemented into PyARC using its Python API, special setup and handling of the runtime environment is required if using PyARC from Workbench. A script called "setup_openmc.sh" has been added to PyARC which acts as a one-time installation of OpenMC into Workbench's base Conda environment (only available with Workbench-5.0.0 and later). This allows OpenMC to be available as a Python package.

A demonstration of the OpenMC capabilities is shown using the ABTR reactor model within the tutorial Sample #18. Figure 3-16 shows the relevant *mcsim* SON input block and a sample of the OpenMC simulation output in the messages window of Workbench.

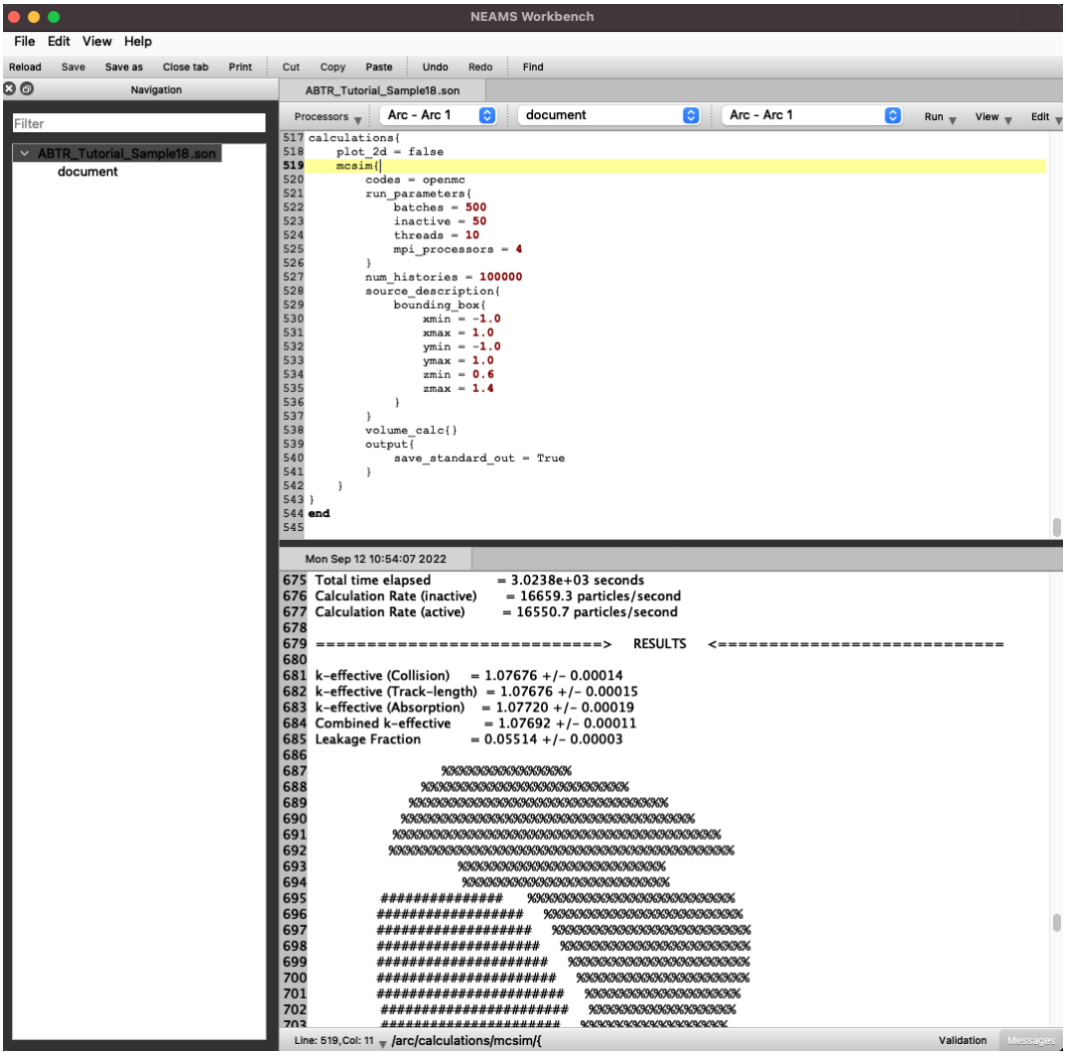

Figure 3-16. Image of Workbench showing the *mcsim* input block of the SON input and partial output from OpenMC in the messages window.

3.10 Additional Utility Scripts

Additional utility scripts are made available in the PyARC repository to support users with various reactor physics simulations. Some of those scripts can be run within or outside PyARC (similar to the 2D plotting script mentioned in Section 2.1.3).

3.10.1 CovMat Utility

This utility streamlines generation of covariance matrix of nuclear data uncertainties on reactivity coefficients, which is used for uncertainty propagation through transient simulations [35]. The workflow implemented in this utility is detailed in Appendix A of [6]. Online documentation is also available. This workflow is run completely outside of PyARC.

3.10.2 SafetyCodes Utility

New in V2.0.0

This utility script was developed to help users postprocess PyARC results into a format that is more directly useable by safety and transients analyses codes such as SAS4A [37] and SAM [38]. Documentation is available online for this script to describe the blocks of this utility script:

- *Decay heat curve fitting*

The objective of this script is to generate an exponential fit of a decay heat curve that is used for instance in transients analyses by the SAS4A code. Such decay heat curve is typically generated by ORIGEN-S (through the PyARC REBUS to ORIGEN-S procedure described in Section 3.7). Safety analyses codes will require exponential fitting of such decay heat curve $DH_{origen}(t)$ in the format:

$$DH_{fit}(\mathrm{t}) = \sum_i B_i * \mathrm{e}^{-L_i * \mathrm{t}}$$

The script developed intends to provide the optimum (B_i , L_i) that minimizes:

$$\sum_t \left[1 - \frac{DH_{fit}(t)}{DH_{origen}(t)}\right]^2$$

This script uses a brute force approach to solving this problem by using the scipy.minimize function using the Sequential Least Squares Programming (SLSQP) method to search for the set of (B_i, L_i) that minimizes the difference between the fitted and original decay heat curves. An iterative approach is used where 1 to 24 sets of (B_i, L_i) are researched, and the one providing best results is provided.

This decay heat fitting procedure is executed within PyARC directly following ORIGEN-S calculation. It can also be executed outside of PyARC by providing a decay heat curve to fit.

- *Reactivity coefficients summary*

This script simply postprocess the power data (from GAMSOR) and reactivity coefficients (from PERSENT) and sums them radially over all the assemblies within a channel. This is used to merge the detailed power and reactivity coefficient distribution in each thermal-hydraulic channel, which typically corresponds to a grouping of assemblies with similar power level and flow level.

This utility script is executed by PyARC following GAMSOR and PERSENT calculations if a user specifies channel distribution. It can also be executed outside of PyARC by providing appropriate output files.

4 PyGriffin

New in V2.0.0

4.1 Introduction to PyGriffin Module

The NEAMS program initiated development of PyGriffin in FY-2022 with the objective to streamline Griffin workflow to facilitate and improve user experience leveraging the Workbench user interface. PyGriffin is still under active development and some key features need to be developed for useful application (especially outside of PyARC). This section summarizes the capabilities envisioned by PyGriffin, and its development plan. The following list of benefits are planned for Griffin users:

- **PyGriffin to streamline Griffin workflow:**
  - multigroup cross-section generation with MC^2-3 or Shift (not yet available)
  - mesh generation using MOOSE mesh generators [22]
  - facilitate Griffin input generation with correct mapping between block ID of mesh (from MOOSE mesh generator) and material ID of cross section library (from MC2-3 or Shift)
  - Griffin execution
  - results post-processing
- **Workbench to support user experience:**
  - Assist SON and MOOSE input development (autocompletion, validation, etc.)
  - Visualize geometry through build-in Workbench visualization
  - Visualize mesh and results using Workbench built-in ParaView
  - Visualize Griffin postprocessed results and ISOXML cross-sections through built-in Workbench chart plotting (not yet available)
  - Manage remote launch execution

PyGriffin is a unique toolbox that handles the Griffin workflow logic. A user can interact with PyGriffin through two different pathways described below:

- Standalone PyGriffin - as described in Section 4.2. This path enables more flexibility for Griffin modeling on any type of geometry (modeled within MOOSE Mesh System). Future effort is required to fill out key missing capabilities required to make this path fully attractive to Griffin users.
- Through PyARC - as described in Section 4.3. This path is readily available within PyARC V2.0.0 to run and post-process Griffin simulations from PyARC model, as demonstrated in Tutorial #17.

4.2 Path #1: Standalone PyGriffin Analyses

This reference path enables the largest amount of modeling flexibility – any model that can be built through the MOOSE MeshGenerator will be supported. It is currently planned to be compatible only with cross-section generation from the Shift Monte Carlo code [35]. Figure 4-1 provides an example of PyGriffin input to show how a user would interact with PyGriffin. In this path, the Griffin user interacts with 2 input files:

- *(1)* PyGriffin "SON" input: to specify Griffin options and link to MOOSE mesh generator file. Additional options to provide pre-generated:
    - ISOXML or ISOTXS file with appropriate material mapping
    - Exodus mesh file
    - Griffin input
- *(2)* MOOSE MeshGenerator input: can be edited in Workbench – the user will specify a reactor geometry used by PyGriffin to automatically generate the Shift model (planned for FY-2023). Real-time geometry visualization through Workbench built-in tool *(3)* is proposed for future work.

Finally, the PyGriffin results can be visualized through Paraview, as illustrated in *(4)* of Figure 4-1.

To summarize, through Workbench/PyGriffin, the user will have the following steps facilitated or automated:

- Support MOOSE MeshGenerator input generation (through autocompletion features within Workbench)
- Automated generation and execution of Shift for cross-section tallying *(this is the key missing part as of end of FY-2022)*
- Automated execution of ISOXML and mesh files generation
- Support selection of Griffin options (through autocompletion features within Workbench) and input of correct mapping between mesh and cross-section regions
- Post-processing of Griffin results

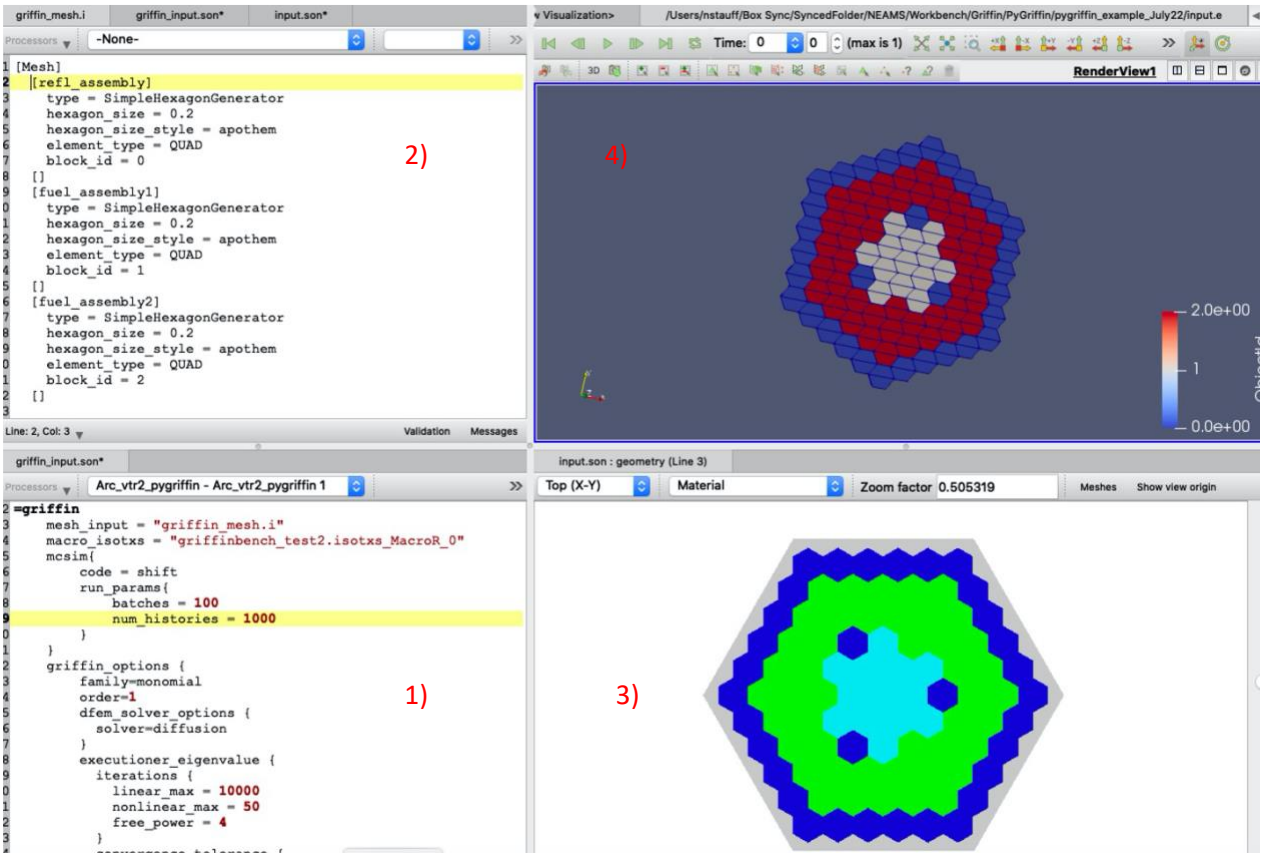

Figure 4-1. Illustration of **envisioned** standalone PyGriffin pathway with 1) PyGriffin SON input, 2) MOOSE meshgenerator input, 3) geometry visualization and 4) mesh/results visualization.

4.3 Path #2: PyARC/PyGriffin Connection

This is the Griffin implementation path through PyARC that currently enables straightforward use of PyGriffin/Griffin. This path is most applicable for Fast Reactor core modeling relying on cross-section generation through MC^2-3. An example of input and visualized geometry is shown in Figure 4-2. An Advanced Burner Test Reactor (ABTR)-based tutorial example that calls PyGriffin through PyARC can be found in Sample #17 of the PyARC tutorial.

Through PyARC, the user only needs to specify Griffin solver options within the PyARC "SON" input (1 of Figure 4-2), and the PyARC/PyGriffinConnect module handles the following:

- Run MC^2-3 and TWODANT for ISOTXS generation (leveraging existing workflow discussed in Section 3.1) – or skip this calculation if pre-generated ISOTXS is provided by the user.
- Generate MOOSE mesh generator input generation.
- Geometry visualization through Workbench and Mesh/results visualization through Paraview (2 and 3 of Figure 4-2, respectively).
- Execute PyGriffin module described in Section 4 that will handle the following:
  - ISOTXS to ISOXML conversion

- Griffin input generation
- Execute MOOSE to generate exodus mesh file
- Execute Griffin
- Post-process Griffin results

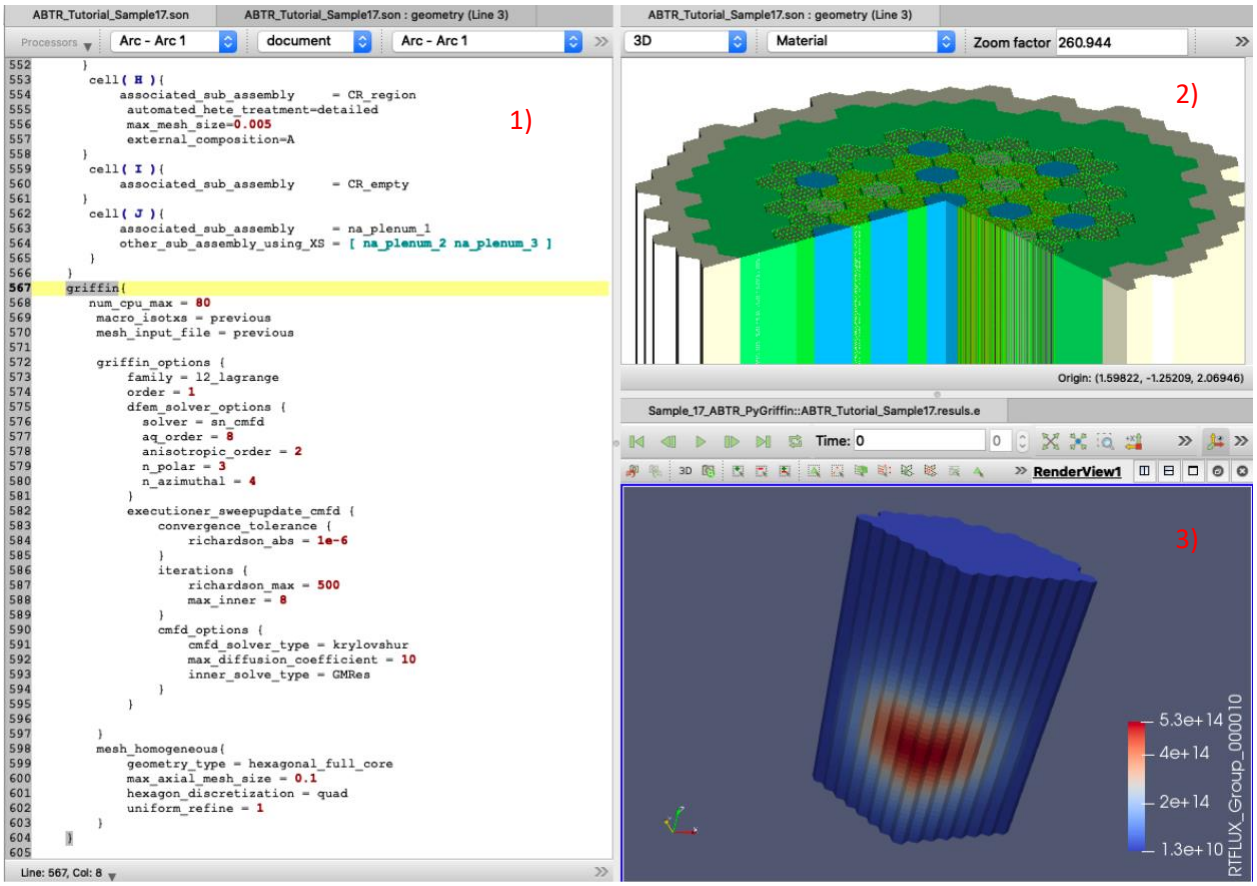

Figure 4-2. Illustration of PyARC/PyGriffin pathway with 1) PyARC SON input specifying Griffin solver options, 2) geometry visualization through Workbench, and 3) results (and mesh) visualization through ParaView.

4.4 Status and plan for future development

The status and plan for future development of PyGriffin is summarized in Table 4-1. At the end of FY-2022, the software infrastructure was developed, and initial capabilities were demonstrated. Path #2 was completely implemented, with ongoing testing and improvement planned moving forward. Path #1 is currently missing the key XS generation step, which currently limits the benefits envisioned in the longer term.

Table 4-1. capabilities planned for PyGriffin and implementation as of end of FY-2022.

Capabilities implemented	Path #1 – Standalone PyGriffin	Path #2 –PyGriffin through PyARC
Workbench features		
Workbench input generation	X	X
Workbench geometry visualization	*FY-2023+*	X (through PyARC)
Mesh and results visualization	X (through Paraview)	
ISOXML visualization	*End of FY-2022*	
Remote launch demonstration	Yes – on ANL clusters	
PyGriffin workflow		
Workflow for XS generation	*FY-2023+ (through Shift)*	X (through PyARC/MCC3)
Workflow for mesh generation	X	X (only for hexagonal geom)
Workflow for postprocessing	*Initiated - FY-2023+*	

Assuming continued support, the following tasks are planned for implementation in FY-2023 by the NEAMS Workbench team:

- Monte-Carlo (Shift) integration into PyGriffin workflow for cross-section generation
- Workbench built-in visualization for MOOSE mesh generation
- Continue integrate Griffin post-processing capabilities
- Add Workbench visualization of ISOXML (leveraging the ISOXML Python processing developed in PyARC utility)
- Setup PyARC/PyGriffin on NCRC and demonstrate remote launch
- Extend PyARC/Griffin to cartesian geometry (once Cartesian geometries supported by MOOSE meshgenerator)
- Development of training material and documentation

In the longer term, PyGriffin will be updated to support new solvers and new workflows. The PyGriffin approach may be generalized to facilitate use of other physics tools and MultiApp coupling with Griffin.

5 Conclusions and Future Work

This report details the status of the extended ARC and NEAMS codes capabilities integrated into the NEAMS Workbench. Integrating these codes into the Workbench benefits directly the advanced reactor community (within the DOE national laboratories, universities and companies) by:

- Providing a set of controlled, maintained, documented and validated scripts to generate codes' inputs, which promotes best practices, reduces the learning curve, and facilitates project collaboration.
- Improving the user experience: the Workbench interface provides assistance for building an input through auto-completion, real-time validation, document navigation, and geometry and results visualization.
- The PyARC and PyGriffin modules facilitates and automatizes complex calculations and workflows for reactor analysis. The Dakota/PyARC coupling in the Workbench was also demonstrated to enable mathematical optimization and sensitivity analysis/uncertainty quantification (SA/UQ) techniques with ARC neutronic simulations. It could be extended to PyGriffin as well (but this hasn't been attempted as of yet).
- Helping users transition to high-fidelity NEAMS codes such as Griffin.

In FY-2021 and FY-2022, effort focused on integrating the DASSH sub-channel thermal-hydraulic code, the Griffin high-fidelity deterministic code, and the OpenMC Monte Carlo code. Those capabilities were made available in version 2.0.0 of PyARC release (PyGriffin hasn't been released yet). Various minor improvements were completed to enable additional modeling options and to respond to user requests.

The ARC and NEAMS codes are currently used at ANL, Westinghouse, INL, and NCSU through the Workbench by nuclear engineers for LFR, MSR, micro-reactor, and SFR core design analyses [28], [39], [40], [41], [42]. Additional verification work and code-to-code comparison was recently completed under the ARDP-Natrium project [43] and under the DOE-NE ART Fast Reactor program [44].

Future efforts will focus on continuously adding new and existing modeling capabilities available with the ARC and NEAMS codes (especially Shift to both the PyARC and PyGriffin workflows), training new users and supporting them to continue building user experience.

REFERENCES

[1] B. T. Rearden, R. A. Lefebvre, "Objectives of the NEAMS Workbench," ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[2] B. T. Rearden, R. A. Lefebvre, A. B. Thompson, B. R. Langley, N.E. Stauff, "Introduction to the Nuclear Energy Advanced Modeling and Simulation Workbench," M&C 2017, Jiju Island, South Korea, April (2017).

[3] N. Stauff, N. Gaughan, and T. Kim, "ARC integration into the NEAMS Workbench," ANL/NE-17/31, September 30, 2017.

[4] N. Stauff, "Updated status of the ART neutronic fast reactor tools integration to the NEAMS Workbench," ANL/NEAMS-18/1, September 30, (2018).

[5] N. Stauff, P. Lartaud, Y. S. Jung, K. Zeng, J. Hou, "Status of the NEAMS and ARC neutronic fast reactor tools integration to the NEAMS Workbench," ANL/NEAMS-19/1, Sept. 30, (2019).

[6] N. Stauff "Status of the NEAMS and ARC neutronic fast reactor tools integration to the NEAMS Workbench," ANL/NEAMS-20/2, September 30, 2020.

[7] ARC 11.0: Code System for Analysis of Nuclear Reactors, Argonne National Laboratory (2014). Available from Available from Radiation Safety Information Computational Center as CCC-824.

[8] Changho Lee, Yeon Sang Jung, and Won Sik Yang, "MC^2-3: Multigroup Cross Section Generation Code for Fast Reactor Analysis," ANL/NE-11-41 Rev.3, August 31 (2018).

[9] K. L. Derstine, "DIF3D: A Code to Solve One-, Two-, and Three-Dimensional Finite Difference Diffusion Theory Problems," ANL-82-64, Argonne National Laboratory (1984).

[10] B. J. Toppel, "A User's Guide to the REBUS-3 Fuel Cycle Analysis Capability," ANL-83-2, Argonne National Laboratory (1983).

[11] M. A. Smith, C. Adams, W. S. Yang, E. E. Lewis, "VARI3D & PERSENT: Perturbation and Sensitivity Analysis," Argonne National Laboratory, ANL/NE-13/8 Rev. 3, Apr 30 (2020).

[12] M. A. Smith, C. H. Lee, and R. N. Hill, "GAMSOR: Gamma Source Preparation and DIF3D Flux Solution," ANL/NE-16/50 Rev. 1.0, June 28, 2017.

[13] Y. S. Jung, C. H. Lee, M. A. Smith, "PROTEUS-NODAL User Manual (Rev.0)," ANL/NE-18/4, Argonne National Laboratory, September 30 (2018).

[14] "Scale: A Comprehensive Modeling and Simulation Suite for Nuclear Safety Analysis and Design" ORNL/TM-2005/39 Version 6.1 (June 2011).

[15] Milos Atz, Micheal A. Smith, Florent Heidet. "DASSH software for ducted assembly thermal hydraulics calculations – overview and benchmark," Transactions of the American Nuclear Society 123 pp. 1673-1676 (2020).

[16] Paul K. Romano, Nicholas E. Horelik, Bryan R. Herman, Adam G. Nelson, Benoit Forget, and Kord Smith, "OpenMC: A State-of-the-Art Monte Carlo Code for Research and Development," Ann. Nucl. Energy, 82, 90--97 (2015).

[17] C. H. Lee et al., “Griffin Software Development Plan,” ANL/NSE-21/23, INL/EXT-21-63185, Argonne National Laboratory and Idaho National Laboratory (2021).

[18] R. A. Lefebvre, A. B. Thompson, B. R. Langley, B. T. Rearden, “NEAMS Workbench 1.0 Beta Status,” ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[19] Nicolas E. Stauff, Taek K. Kim, Robert A. Lefebvre, Brandon R. Langley, Bradley T. Rearden, “Integration of the Argonne Reactor Computation codes into the NEAMS Workbench,” ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[20] Robert A. Lefebvre, Brandon R. LANGLEY, and Jordan P. LEFEBVRE, “Workbench Analysis Sequence Processor", ORNL/TM-2017/619, UT-Battelle, LLC, Oak Ridge National Laboratory (2017).

[21] Kitware: ParaView Visualization Tool. https://www.paraview.org

[22] E Shemon et al. “MOOSE Framework Enhancements for Meshing Reactor Geometries,” proceedings of PHYSOR (2022).

[23] Nicolas E. Stauff, Paul K. Romano, Zhiee Jhia Ooi, Amanda L. Lund, Ling Zou, W. Neal Mann, Yinbin Miao, “WATTS Framework for Multidisciplinary Reactor Physics Analyses,” Proceedings of the ANS Winter (2022).

[24] Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.7 User’s Manual.

[25] Nicolas E. Stauff, Robert A. Lefebvre, Laura Swiler, Bradley T. Rearden, “Coupling of DAKOTA with the ARC suite of codes in the NEAMS Workbench for Uncertainty Quantification,” ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[26] Kaiyue Zeng, Nicolas E. Stauff, Jason Hou, T. K. Kim “Development of multi-objective core optimization framework and application to sodium-cooled fast test reactors,” Progress in Nuclear Energy, Vol 120, February (2020) 103184.

[27] K. Zeng, Nicolas Stauff, “Multi-criteria optimization of the Advanced Burner Test Reactor,” – Submitted to Progress in Nuclear Energy, (2019).

[28] T. K. Kim, N. Stauff, C. Stansbury, A. Levinsky, F. Franceschini, “Long Core Life Options for the Westinghouse LFR,” proceedings of Global 2019, Seattle, WA, Sept (2019).

[29] "Advanced Burner Test Reactor Preconceptual Design Report", ANL-ABR-1, September 5, 2006. https://publications.anl.gov/anlpubs/2008/12/63007.pdf

[30] Gerald Rimpault et al, “Objectives and Status of the OECD/NEA sub-group on Uncertainty Analysis in Best-Estimate Modelling (UAM) for Design, Operation and Safety Analysis of SFRs (SFR-UAM),” FR’17, Yekaterinburg, Russia.

[31] C. H. Lee, N. E. Stauff, “Improved Reactivity Worth Estimation of MC2-3/DIF3D in Fast Reactor Analysis,” Proceedings of ANS Sumer Meeting, paper 14201, San Antonio, Texas (2015).

[32] R. E. Alcouffe, F. W. Brinkley, D. R. Marr, and R. D. O'Dell, "User's Guide for TWODANT: A Code Package for Two-Dimensional, Diffusion-Accelerated, Neutral-Particle Transport," LA-10049-M, Los Alamos National Laboratory (1990).

[33] G. Palmiotti et al, "Variational nodal transport methods with anisotropic scattering," Nuclear Science and Engineering, Vol. 115, pp. 233-243 (1993).

[34] G. Aliberti and M. Smith, "PERSENT: need of a deterministic code for sensitivity analysis in 3D geometry and transport theory," Proceedings of PHYSOR2014, Kyoto, Japan (2014).

[35] W. A. Wieselquist, R. A. Lefebvre, and M. A. Jessee, Eds., SCALE Code System, ORNL/TM-2005/39, Version 6.2.4, Oak Ridge National Laboratory, Oak Ridge, TN (2020).

[36] Nicolas E. Stauff, K. Zeng, G. Zhang, G. Aliberti, J. Hu, T. Fanning, and T. K. Kim, "Uncertainty quantification of ABR transient safety analysis – nuclear data uncertainties," BEPU 2018, May 13-19, Lucca, Italy (2018).

[37] T.H. Fanning, A. J. Brunett, and T. Sumner, eds., The SAS4A/SASSYS-1 Safety Analysis Code System, Version 5, ANL/NE-16/19, Nuclear Engineering Division, Argonne National Laboratory, March 31, 2017.

[38] R. Hu, "An advanced one-dimensional finite element model for incompressible thermally expandable flow," Nuclear Technology, 190 (2015).

[39] Bo Feng and Nicolas Stauff, "High Power Density Annular Fuel in a Fast Test Reactor," ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[40] Nicolas E. Stauff, F. Heidet, "Assessment of Low Enriched Uranium Fueled Core Configurations for the Versatile Test Reactor," proceedings of ANS Annual 2019, Minneapolis, MN, June 9-13 (2019).

[41] Yinbin Miao, Nicolas Stauff, Aaron Oaks, Abdellatif M. Yacout, Taek K. Kim, "Fuel Performance Evaluation of Annular Metallic Fuels for an Advanced Fast Reactor Concept," Nuclear Engineering and Design, Vol. 352, (2019). https://doi.org/10.1016/j.nucengdes.2019.110157

[42] I. T. Usman, P. Lartaud, and N. E. Stauff, "Sensitivity Analysis and Uncertainty Quantification of FFTF Cycle 8C using the NEAMS Workbench," Submitted to ANS Winter Meeting (2019).

[43] Nicolas Stauff, Ting Fei, Mike Smith, "ARDP Natrium Neutronic Methodology: Argonne Neutronic Assessment of ABR-1000," ANL/NSE-22/31, May 31, (2022).

[44] S. Kumar, B. Feng, and T. Fei, "Fast Reactor Physics Model Verification Studies using ARC and PyARC Workflows," ANL/NSE-22/46, September (2022).

[45] N. E. Stauff, A. Abdelhameed, M. Atz, P. Shriwise, Y. S. Jung, R. Lefebvre, "PyARC – A User-Friendly Open-Source Reactor Physics Framework for Fast Reactor Design and Analysis," proceedings of PHYSOR (2022).

[46] M. B. Chadwick et al., "ENDF/B-VII.0: Next Generation Evaluated Nuclear Data Library for Nuclear Science and Technology," Nuclear Data Sheets 107, 2931 (2006).

[47] https://github.com/openmc-dev/plotter

Appendix A : Results Comparison for ABTR Tutorial Model

This section discusses the PyARC results obtained on the Advanced Burner Test Reactor (ABTR) [29] model available in the tutorial. The detail reference core configuration and geometrical dimensions for the core fueled with U-TRU-10%Zr is obtained from the ABTR pre-conceptual report. The ABTR is a pool-type sodium-cooled fast reactor and its radial core layout is displayed in Figure A-1. It displays 199 assemblies – 54 driver fuel assemblies, 78 reflector assemblies, 48 shield assemblies, 9 test assemblies for material test and fuel test purposes, and 10 primary and secondary control rods. The reactor core is radially divided into two enrichment zones: inner core region and outer core region composed of 24 and 30 driver fuel assemblies, respectively.

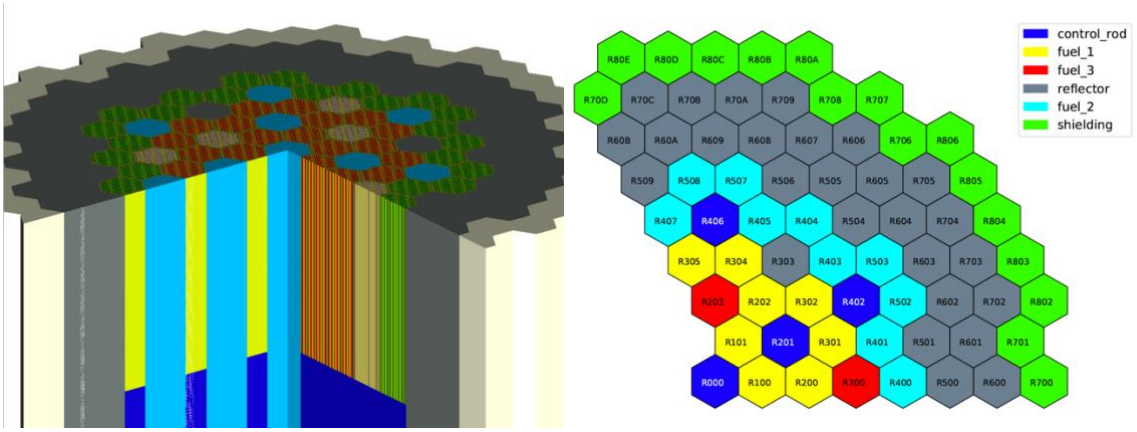

Figure A-1. Layout of ABTR model obtained through (left) PyARC 2D plotting utility script (right) Workbench visualization.

Code to code comparison was completed as part of the PyARC tutorial using the OpenMC Monte Carlo code [16] to provide a reference solution for the eigenvalue calculation on the fresh ABTR core described above. Initial comparison was published in [45], but was updated with OpenMC and Griffin solutions obtained directly within PyARC 2.0.0.

All the results are obtained using the ENDF/B-VII.0 nuclear data libraries [46]. OpenMC uses explicitly defined fuel and control rod regions, while the other regions were homogenized (similar to the geometry displayed in the left of Figure A-1). The results of this comparison are summarized in Table A-1 and links to the different input and output files stored in the PyARC tutorial are provided. Various deterministic approaches are available with PyARC to provide different levels of accuracy, which is especially important for small fast neutron reactors such as ABTR.

The MC^2-3 code is used to calculate the homogenized 33-group cross-sections for DIF3D with 2 different processes. In the “Low-fidelity XS”, the MC^2-3 code directly condenses cross sections into 33 energy groups, with a buckling search applied to the fuel regions, while the flux estimated in the fuel regions is used for cross-section condensation in non-fuel regions. This simplified XS generation process leads to ~1,300 pcm of added discrepancy (under-estimation) for the ABTR model when compared to the “high-fidelity XS” generation process. In the high-fidelity approach, the first step in MC^2-3 uses a fine 2082-energy-group structure and cross sections are condensed

into 1041 energy groups. It is followed by a flux calculation step using the 2-D Sn transport solver TWODANT for an approximated equivalent 2-D cylindrical core with a P3 scattering approximation and 1041 energy groups. During the third step, the cross sections are condensed into 33 groups using the flux spectrum obtained from TWODANT for each region. Heterogeneous cross-section treatment is applied in this second step to the driver fuel and control rod regions using MC^2-3 based on an equivalent 1-D model to account for geometrical self-shielding [31].

The whole-core flux calculation is performed with the DIF3D code using the Diffusion (finite difference solver) and the Transport (variational nodal transport solver VARIANT with the 3rd order angular flux and 1st order scattering approximations) options, and using the 33-energy group discretization. The Diffusion approximation leads again to ~800pcm of added discrepancy (under-estimation) on the ABTR model when compared with the Transport option.

The highest fidelity option considered with PyARC (High-fidelity XS and Transport solution with VARIANT) results in only 50 pcm of discrepancy when compared with the OpenMC solution obtained through PyARC. Such high level of agreement may conceal some error cancellation.

Preliminary Griffin results are obtained through PyARC/PyGriffinConnect using a low fidelity XS treatment (with homogeneous treatment), the MOOSE mesh generator system for mesh generation, and Griffin's DFEM-SN transport solver with CMFD acceleration. The mesh discretization used in the input mesh divided each homogenized hexagonal assembly meshes into 2 quadrilateral elements, and uniformly refines the resulting mesh by a factor of 2. Moreover, a Level-Symmetric quadrature with a quadrature order of 8, an anisotropic order of 2 is used with 3 polar angles and 4 azimuthal angles. About 900pcm of discrepancy with OpenMC solution is observed. Further discrepancies in the eigenvalue results are likely to be reduced by using a finer mesh discretization and higher order approximations to the transport solver, together with application of higher-fidelity cross-section treatment.

Table A-1. Eigenvalue comparison between different PyARC methods and integrated codes.

Code	Method	K-eff	Diff to Ref. [pcm]
PyARC/OpenMC	Monte Carlo	1.08064 +/- 0.00011	Ref.
PyARC/DIF3D	MC^2-3 - Low-fidelity XS DIF3D - Diffusion	1.05567	-2189
PyARC/DIF3D	MC^2-3 - Low-fidelity XS DIF3D - Transport	1.06459	-1395
PyARC/DIF3D	MC^2-3 - High-fidelity XS DIF3D - Transport	1.08003	-52
PyARC/PyGriffin	MC^2-3 - Low-fidelity XS Griffin – DFEM-SN transport with CMFD	1.07046	-880

Appendix B : Model Comparison between OpenMC and ARC

This section summarizes a verification exercise done on the PyARC-generated OpenMC geometry based on the ABTR model within the tutorial. The PyMCSim prerun process generates the material.xml, geometry.xml, and settings.xml files for OpenMC. The OpenMC model defined in these files can be viewed using the OpenMC Plotter tool [47]. Figures B-1 to 3 show the OpenMC model in the Plotter compared to the SON model that was generated from the Workbench viewer for each the x-y, x-z, and y-z viewing planes.

These comparisons indicate that the overall assembly configuration is consistent between the two model representations. To better assess the details in the assemblies, Figures B-4 to 6 show close ups of detailed regions in each plane. The black outlines on the OpenMC Plotter images indicate the presence of a cell boundary, while the outlines on the Workbench viewer represent surfaces. Despite the slight differences in which boundaries or surfaces are explicitly shown, the overall structure and material locations are identical between the two models.

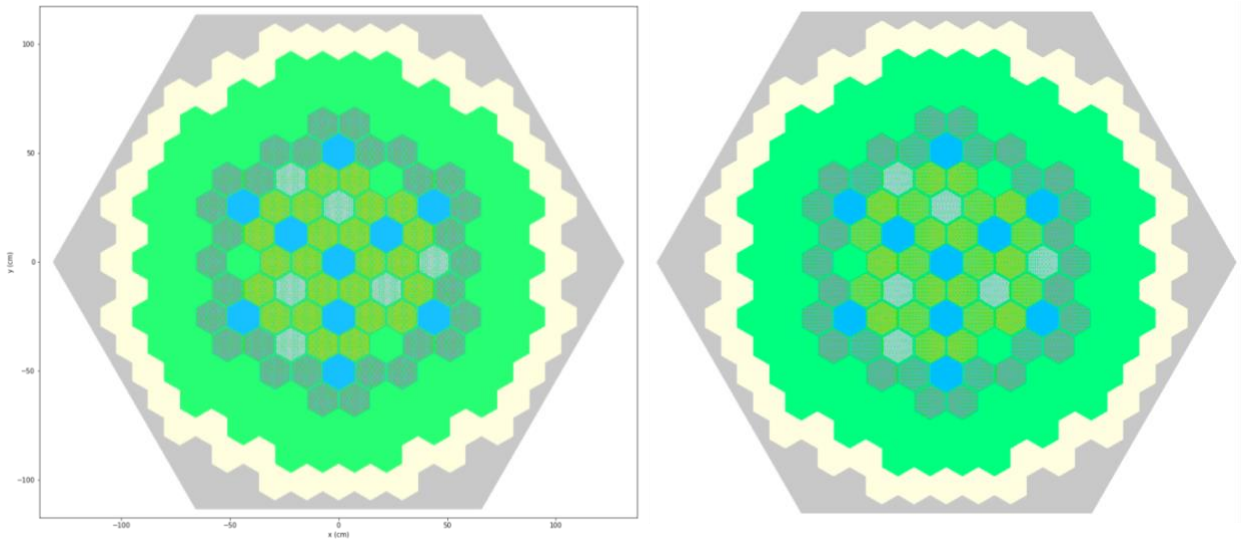

Figure B-1. Cross sections of the x-y plane at z=1.3 m. Left shows the OpenMC model from the XML files in the OpenMC Plotter; right is the SON model in the Workbench viewer.

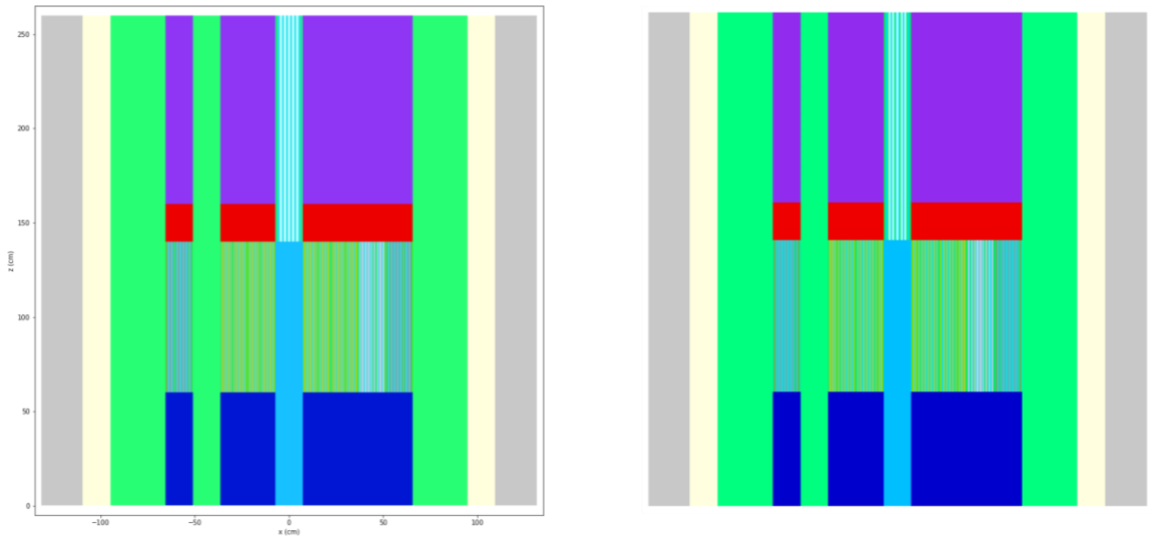

Figure B-2. Cross sections of the x-z plane at y=0 m. Left is the OpenMC model as viewed in the OpenMC Plotter; right is the SON model in the Workbench viewer.

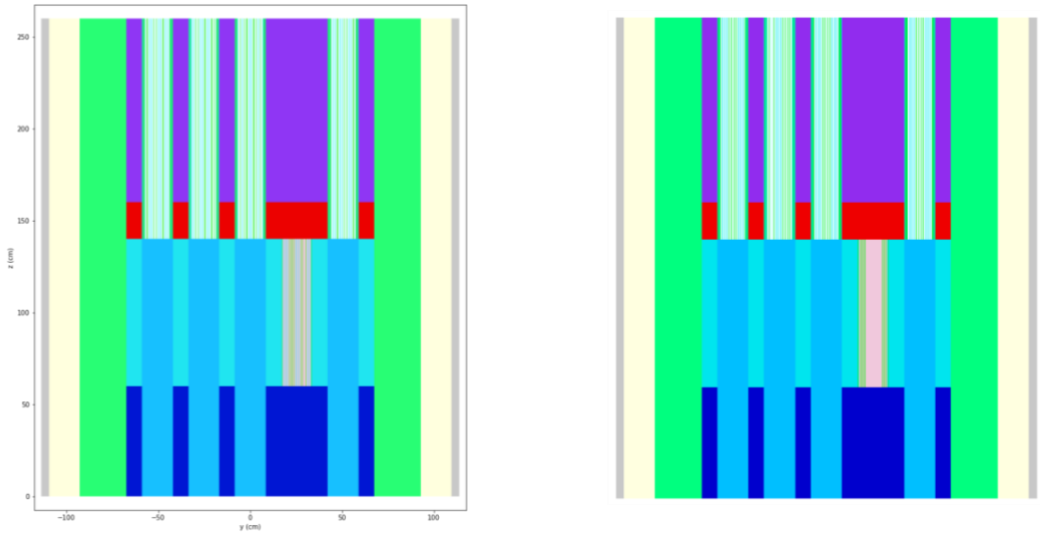

Figure B-3. Cross sections of the y-z plane at x=0 m. Left is the OpenMC model as viewed in the OpenMC Plotter; right is the SON model in the Workbench viewer. Differences in resolution during rendering account for the visual discrepancies.

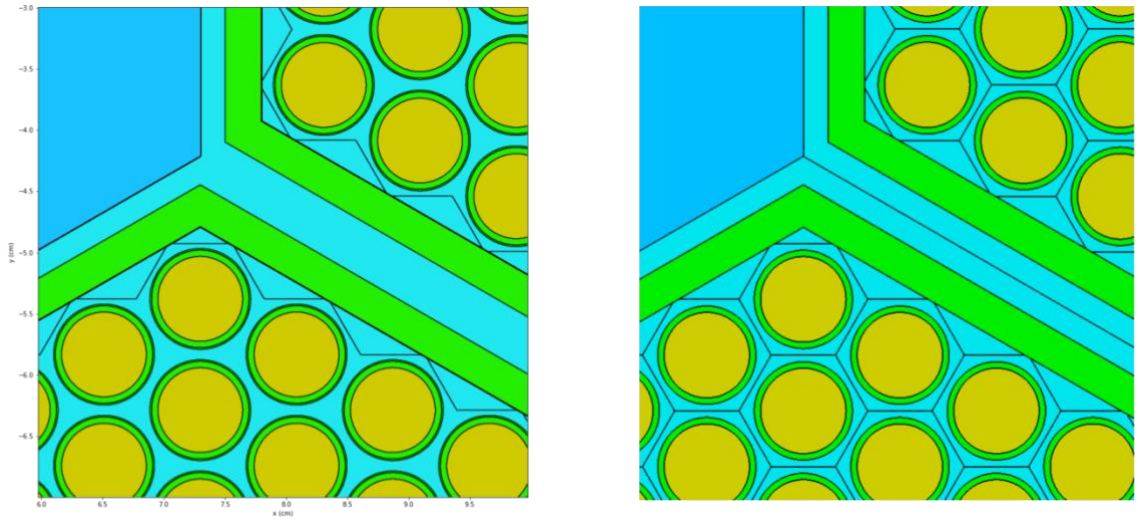

Figure B-4. Close up of the lattice structure of pins in the fuel assemblies in both the OpenMC model as viewed by the OpenMC plotter (left) and the SON model in Workbench (right). The outlines indicate cell boundaries in OpenMC and surfaces in Workbench. Note that the OpenMC model has an additional surface in the cladding layer to account for how the wire wrap on a pin is homogenized and added as an additional cell around the cladding (which additional surface is not shown in the Workbench plot).

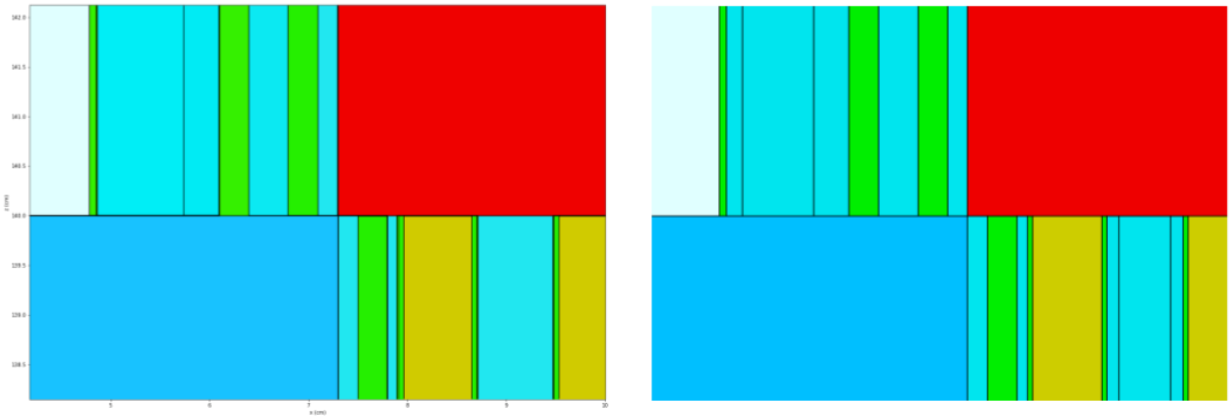

Figure B-5. A close up view of the x-z plane where the fuel and control assemblies are present for both the OpenMC model as viewed by the OpenMC plotter (left) and the SON model in Workbench (right). The outlines indicate cell boundaries in OpenMC and surfaces in Workbench.

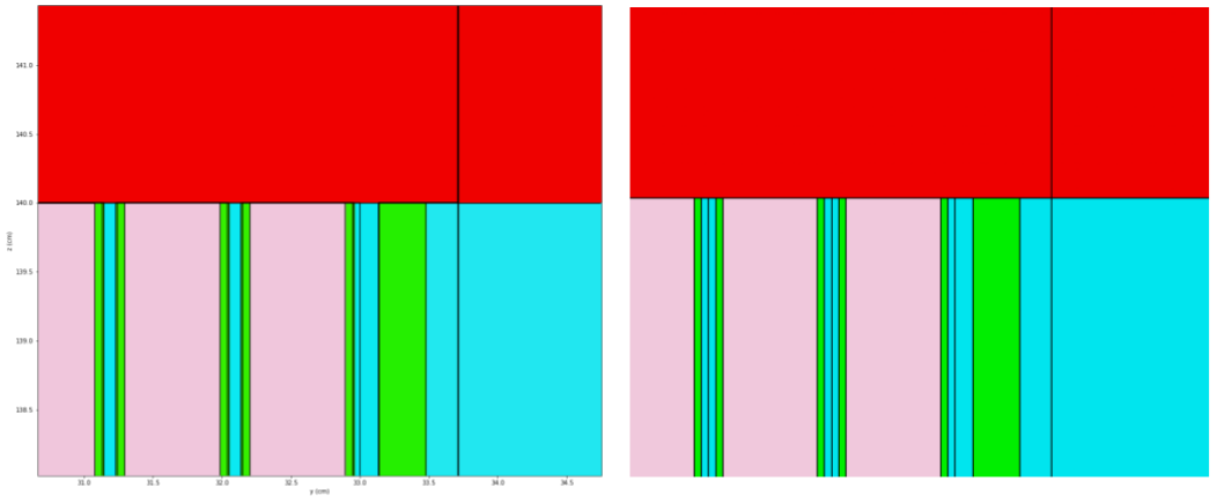

Figure B-6. A close up view of the y-z plane where the gas plenum and a fuel assembly are present for both the OpenMC model as viewed by the OpenMC plotter (left) and the SON model in Workbench (right). The outlines indicate cell boundaries in OpenMC and surfaces in Workbench.

Nuclear Science and Engineering Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 208
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC