ORNL/TM-2020/1742

HFIRCON Version 1.0.5 User Guide

S.C. Wilson
S.M. Mosher
C.R. Daily
D. Chandler

October 2020

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2020/1742

Nuclear Energy and Fuel Cycle Division

HFIRCON Version 1.0.5 User Guide

Stephen C. Wilson
Scott W. Mosher
Charles R. Daily
David Chandler

Date Published: October 2020

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-BATTELLE, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The authors acknowledge the support for the work provided by NASA's Science Mission Directorate, the US Department of Energy's (DOE's) Office of Nuclear Infrastructure Programs, and the DOE Isotope Program managed by the Office of Science for isotope R&D and production.

Special thanks are due to Dr. Ben Betzler of the Oak Ridge National Laboratory Nuclear Energy and Fuel Cycle Division(NEFCD)/Reactor Physics Group for his provision of detailed Shift results for comparison with HFIRCON (High Flux Isotope Reactor Controller) in the low-enriched uranium silicide validation case; Kara Godsey of the NEFCD/Radiation Transport Group for her assistance in running numerous verification and validation (V&V) test suite problems on a variety of the NEFCD computing clusters; Victor Bautista of the Isotopes and Fuel Cycle Division/Nuclear and Radiochemistry group and Dr. Zain Karriem of the Isotopes and Fuel Cycle Division/Medical, Industrial, and Research Isotopes group for their review of this document; and Hailey Green of the NEFCD/Radiation Transport group, Dr. Arzu Alpan of the NEFCD/Radiation Transport group, and Victor Bautista for their reviews of the numerous software quality assurance V&V test suite jobs.

ABSTRACT

The High Flux Isotope Reactor (HFIR) Controller (HFIRCON) code is a collection of python routines and C plugins that automate the workflow for fuel and single- or multicycle target depletion analyses for HFIR at Oak Ridge National Laboratory (ORNL). This code calls the LAVAMINT (LAVA Model Interrogator) parallel (MCNP) Monte Carlo N-Particle model interrogator to stochastically calculate cell volumes and bounding boxes, the ADVANTG (Automatic Variance Reduction Generation) code package for all variance reduction and source biasing calculations, the ORNL-Transformative Neutronics/MCNP5 transport solver for all transport solutions, and the MSX_DEPLETE module to perform all depletion calculations via the ORIGEN (Oak Ridge Isotope Generation) application programming interface. It also performs a robust set of postprocessing functions to automatically provide summaries of several key metrics that are common to a wide variety of typical HFIR design and safety-basis analyses.

1. INTRODUCTION

The High Flux Isotope Reactor (HFIR) Controller (HFIRCON) code is a collection of python routines and C plugins that automate the workflow for fuel and single- or multicycle target depletion analyses for HFIR at Oak Ridge National Laboratory (ORNL). This code calls the LAVA Model Interrogator (LAVAMINT) parallel Monte Carlo N-Particle (MCNP) model interrogator to stochastically calculate cell volumes and bounding boxes, the Automatic Variance Reduction Generation (ADVANTG) code package for all variance reduction and source biasing calculations, the ORNL-Transformative Neutronics (TN)/MCNP5 transport solver for all transport solutions, and the MSX_DEPLETE module to perform all depletion calculations via the Oak Ridge Isotope Generation (ORIGEN) application programming interface (API). It also performs a robust set of postprocessing functions to automatically provide summaries of several key metrics that are common to a wide variety of typical HFIR design and safety-basis analyses.

HFIRCON was initially developed to support the Plutonium Supply Project [1]. Much of the functionality in HFIRCON was achieved by leveraging existing tools that were previously developed to enable high-fidelity neutronics analyses of the ITER fusion reactor complex and site [2–4]. The mesh-based activation functionality of the fusion neutronics toolkit was enhanced to provide the cell-based, multicycle depletion capabilities required to support a large portion of the HFIR mission space, including isotope production, materials testing, and core design analyses. The postprocessing functionality is robust but coded to be easily extensible as new user features are requested. In addition to the human-readable text output summaries generated by HFIRCON, all raw data calculated by HFIRCON are stored in Hierarchical Data Format 5 (HDF5) binary files that can easily be viewed with a variety of commercially available third-party viewers. The structure of these files is well-documented for users who desire to create their own unique, innovative, or case-specific post processing tool(s).

2. CODE LOCATION AND EXECUTION

The HFIRCON launcher script sets appropriate environment variables, writes a file containing all the appropriate Portable Batch System (PBS) job scheduler commands, and submits this file to be queued for execution. The HFIRCON program is currently available on five separate computational clusters. These include cades, apollo, romulus, remus, and lise.

On the cades cluster, the launcher script is located at:

`/software/user_tools/current/cades-nsed-hfircon/PROD/V1.0.5/bin/hfircon`

Users must be members of the `cades-nsed` and `cades-nsed-hfircon` groups to access the code on the cades cluster.

On the apollo, romulus, remus, and lise clusters, the launcher script is located at:

`/projects/hfircon/PROD/V1.0.5/bin/hfircon`

Users must be members of the HFIRCON group on these clusters to access the code.

To become a member of either the cades-HFIRCON or HFIRCON groups, one must possess current Radiation Safety Information Computational Center (RSICC) licenses for MCNP5-1.60, SCALE6.2.3, and ADVANTG.

HFIRCON may be executed in two command-line modes, depending on user preference. The first is the `execute` mode. This will attempt to execute the code interactively according to the contents of the `controller_input.json` file, which must be present in the execution directory. This is generally not recommended except for during development and testing because users must manually ensure their interactive job parameters (e.g., number of nodes) match the contents of the relevant job controls in `controller_input.json`.

The second mode is the `launch` mode. This will parse the `controller_input.json` file, generate a PBS input according to its contents, and submit the PBS job to be queued for execution as soon as the requested resources are available. Executing in batch mode with the launch command is preferred.

An example execution line on the cades cluster is:

```
/software/user_tools/current/cades-nsed-
hfircon/PROD/V1.0.5/bin/hfircon launch
```

On any of the other supported clusters, the execution line would be:

```
/projects/hfircon/PROD/V1.0.5/bin/hfircon launch
```

A valid controller_input.json file must be present in the directory in which users execute this command. Other input files (Section 4.1) should exist in the user-specified *input_location* directory and must be appropriately configured for ORNL-TN/MCNP execution, as described in Section 4.1.2.

The HFIRCON launcher script should be executed on a login node on cades or the head node on any other cluster. Users are not to execute any long-running or resource-intensive jobs on either a cades login node or the head node of any other cluster. The only acceptable tasks to perform on one of these special nodes are simple system commands, such as listing the contents of a directory, moving or renaming a file, editing a file, or submitting a job to the job scheduler. Submitting jobs to the scheduler must be done from the head node for users to have control over actions such as easily deleting the job from the queue once it has started. Because the HFIRCON launcher script only creates a small file with a few job execution directives and submits this job to the job scheduler, it is acceptable and appropriate to invoke this script from a cluster's login or head node.

3. SOLVER CODES AND METHODS

A summary of the codes and utility modules contained in HFIRCON is provided in Sections 3.1–3.4. Additional, previously unpublished information about the classes, functions, methods, and updates to these codes is provided in APPENDIX A through APPENDIX D. Many of the capabilities described in these appendices were developed for previous ITER analyses [2] with additional capabilities added as needed to facilitate the HFIRCON-specific multicycle depletion requirements. The documentation in these appendices is taken verbatim from internal documents provided by the original HFIRCON developers and is primarily intended for current and future developers looking to extend HFIRCON's capabilities.

3.1 MCNP5-1.60/ORNL-TN

MCNP is a general-purpose Monte Carlo (MC) code that is suitable for neutron and photon transport. It includes the capability to calculate eigenvalues for critical systems. It accepts as input geometry cells defined the intersection and/or union of first- and second-degree surfaces, material data defined by

isotopic fraction and atom density, and radiation sources defined as distributions of various quantities. It can provide output in the form of tallied integral quantities that are useful to nuclear analyses. The X-5 Monte Carlo Team [5] provides complete documentation of the MCNP5-1.60 software in their document.

HFIRCON uses the MCNP5-1.60 solver to perform radiation transport simulations. HFIRCON automatically sets up MCNP inputs, executes MCNP in a parallel framework, and collects and rebuilds the output after execution. HFIRCON uses MCNP's flexible tallies to solve for:

- k-eigenvalues,
- fission rates by isotope in each depletion cell,
- capture rates by isotope in each depletion cell,
- neutron fluxes by depletion cell on a user-defined energy structure,
- neutron heating rates in each target cell, and
- gamma heating rates in each target cell.

Unfortunately, MCNP's native implementation (i.e., "stock" MCNP) is not optimally configured to perform these calculations within a larger depletion framework. This was not the original purpose of the code and was not part of the developers' scope. There are also several bugs in MCNP5-1.60 that must be addressed to use it as the transport solver for a depletion implementation.

The ORNL-TN upgrade to MCNP is implemented as a stand-alone patch to the RSICC-distributed source code for MCNP5-1.60. It provides several performance improvements and bug fixes that were developed for HFIRCON.

3.1.1 ORNL-TN Performance Improvements

ORNL-TN development was originally initiated due to the exceptional requirements imposed by ITER MCNP models [2]. Its development was extended to facilitate the automation of HFIR depletion as implemented in HFIRCON.

Some of the significant performance improvements in ORNL-TN include the following:

- **Fast geometry initialization:** Discards $O(N^2)$ algorithms on geometry initialization in favor of O(N) algorithms [3].
- **Python interface to MCNP data structures:** Facilitates the addition of depletion tallies and updates to cell number densities and compositions at each depletion step without resorting to American Standard Code for Information Interchange (ASCII) parsing or deck generation.
- **Source plugin functionality:** Allows compiled shared object libraries to be used with source generation and sampling functionality to extend the native source sampling capability of MCNP. Developed specifically to allow arbitrarily large numbers of cell rejection sources for end-of-cycle (EOC)decay gamma transport simulation.
- **Support for binary-format weight windows (wwinp) files:** Eliminates the ASCII parse on initialization for large weight window files.
- **Option to suppress binary runtpe output:** Eliminates writing large binary runtpe files on problem completion.

3.1.2 ORNL-TN Bug Fixes

Several bugs in MCNP5-1.60 were found and fixed for HFIRCON via the ORNL-TN upgrade. Some of the more significant are described as follows.

- Index overflow when using weight-window parameter sets with more than 2^{31}-1 elements.
- Out-of-bound writes to cell-neighbor array in threaded executions with geometry universes.
- Infinite loop caused by nonpositive distance to surface calculated due to roundoff error.
- Zero-weight particle bug caused by gap in resonance scattering data.
- CPU time calculation in multithreaded gfortran builds.

The ACECAS subroutine did not properly handle some data in "law44" format. This affects gamma production results and is believed to be the cause of several unexplained code failures encountered by HFIR analysts in the past.

The RONGE subroutine logic caused code crashes due to the inadvertent promotion of double-precision variables to quad-precision.

MCNP5-1.60 applies the wrong volume normalization for repeatedly filled universe cells. ORNL-TN does not fix this directly due to the possibility of parent cell truncation (i.e., simply multiplying by the number of repeats might be incorrect). HFIRCON addresses this issue via its calls to LAVAMINT, which will always integrate over the true geometry cell configuration as it is defined in the MCNP input.

3.2 ORIGEN/MSX_DEPLETE

The ORIGEN software [6] provides a high-performance solution for point burnup and depletion via its Chebyshev Rational Approximation solver [7]. The MSX suite of utilities is a set of codes developed to provide a parallel interface to ORIGEN via the ORIGEN API and to provide interface functionality, as needed. Much of MSX was developed to perform mesh-based activation calculations for ITER applications [4]. The MSX_DEPLETE module was developed specifically for HFIRCON; it provides a parallel interface to the ORIGEN API to perform cell-based depletion and decay calculations using neutron fluxes, fission cross sections, and capture cross sections tallied in MCNP/ORNL-TN. The MSX_DEPLETE output includes updated number densities, gamma emission spectra, and source strengths for every depletion cell, as well as useful physics data for HFIRCON postprocessing tasks.

HFIRCON automatically sets up and calls MSX_DEPLETE, which calls the ORIGEN API in parallel, at appropriate times during the depletion process to facilitate an explicit Euler depletion scheme. Higher order schemes are currently unavailable via HFIRCON. Complete documentation of the underlying mechanics of the ORIGEN solver is available [8].

3.3 ADVANTG

The ADVANTG code generates variance reduction parameters (i.e., space- and energy-dependent weight windows) for MCNP simulations [9]. It automates the Consistent Adjoint-Driven Importance Sampling (CADIS) and Forward-Weighted CADIS (FW-CADIS) techniques [10].

HFIRCON optionally sets up and calls ADVANTG to generate weight windows for target tallies. These are used during k-eigenvalue simulations to calculate depletion and energy deposition tallies in targets while maintaining the stability of fuel fission source convergence.

Mosher et al. [9] provides a complete description of the ADVANTG code. ADVANTG was updated during HFIRCON development to support binary weight window file writes; this functionality is not available via the version of ADVANTG documented in Mosher et al. [9].

3.4 LAVAMINT

The Lava low-level geometry kernel is an MCNP geometry emulator written in C that was developed for ADVANTG. The LAVAMINT code was developed using the Lava library to perform several useful tasks commonly required for activation and depletion calculations. Originally, LAVAMINT was developed for ITER MCNP model integration; it was extended and optimized to perform similar calculations for HFIR.

Stand-alone documentation for LAVAMINT does not currently exist. However, it has been used extensively in ITER applications [2] and has been tested as part of the validated Multi-Step (MS)-CADIS analysis process [4].

LAVAMINT performs the following tasks when called via HFIRCON:

1. **Find all geometry cells**. LAVAMINT scores a single ray in every cell in its initial sweep of the problem extent. These initial scored rays are used to seed subsequent calculation tasks.

   Then, for every geometry cell:
2. **Generate a vector of scored rays**. LAVAMINT uses the initial scored ray across each cell to generate additional rays across the user-defined problem extent. As additional rays are scored this vector grows; subsequent rays are seeded randomly from the existing list of scored rays. This guarantees that every new ray launched for a particular cell will score in that cell, provided that the problem extent is defined appropriately.
3. **Calculate a bounding box**. The maximum and minimum x, y, and z locations in the vector of scored rays define a Cartesian box around the current geometry cell.
4. **Calculate cell volume**. Using the bounding box, stochastically calculate the volume of the cell tally by tracing rays from the bounding planes comprising the box.

Tasks 2, 3, and 4 are iterated until a user-defined volume tolerance is met. If bounding box cell truncation is detected, then LAVAMINT resets the volume calculation.

HFIRCON sets up and launches LAVAMINT automatically at the beginning of every cycle. LAVAMINT is message passing interface (MPI)/Open Multi-Processing (OpenMP) parallel; parallel decomposition is performed by geometry cell. As the number of processors increases, runtime generally decreases linearly if there is a sufficiently large number of cells and their volume calculation cost is approximately equal. The LAVAMINT stochastic cell volumes are used for all volume-averaged tallies, and the LAVAMINT bounding boxes are used to define EOC activation gamma sources.

LAVAMINT will converge volumes of simple cells very quickly. If the user-defined volume tolerance is low enough, then it will converge the volumes of complex cells accurately, including disjoint cells, but will be more computationally expensive.

Figure 1 is a point-cloud plot of the vector of scored rays on a single explicit inner fuel element in a HFIR geometry definition. LAVAMINT successfully finds all 171 separate, disjoint instances of the filled universe.

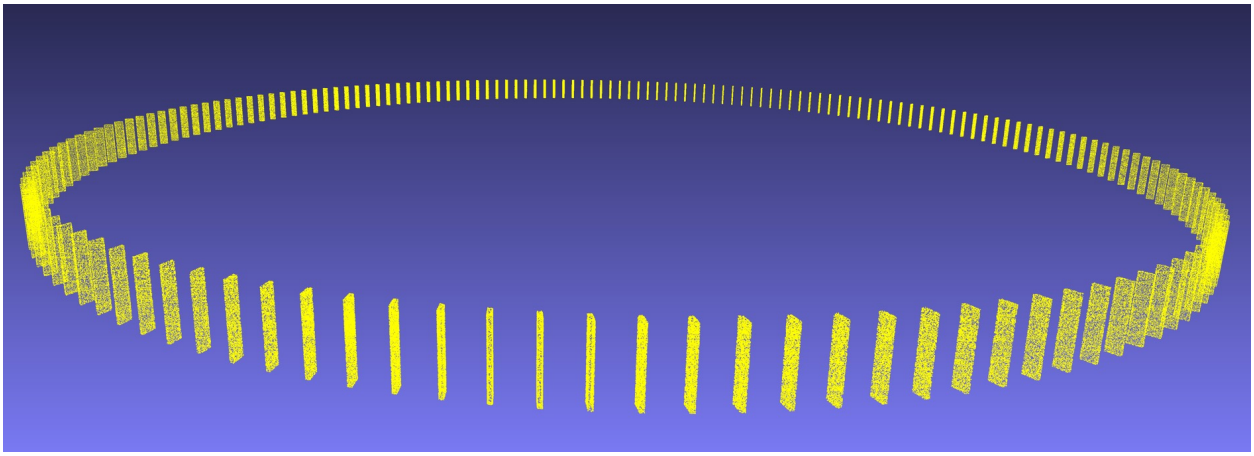

Figure 1. Point-cloud on HFIR inner fuel element surfaces generated with LAVAMINT.

3.5 BINARY DISTRIBUTION

All HFIRCON dependencies are built statically and installed together. Nuclear data for use with HFIRCON is pointed to during the installation process. If the code installer takes the appropriate steps to control the installation (e.g., group access, no write permissions outside the owner), then this facilitates configuration control on the cluster platform.

On the ORNL cades moderate cluster, access is controlled via the cades-nsed-hfircon group. On the Nuclear Energy and Fuel Cycle Division (NEFCD) apollo, romulus, and remus clusters and the Research Reactors Division (RRD) lise cluster, access is controlled via the HFIRCON group.

Users will not run the HFIRCON installer, but some information on the process is included here for completeness. The HFIRCON binary installer is a self-extracting .sh script; the person installing HFIRCON on a new cluster or in a new location simply executes the script. It will prompt for the MCNP cross section data location and the installation directory. Once it is installed, the rc file in the top-level bin/directory can be used to set up the environment, which is necessary if executing with “execute“ instead of “launch.”

Figure 2 shows the high-level architecture of the HFIRCON installation. All these items are included in the binary distribution and cannot be changed by anyone except the developer who owns the HFIRCON installation directory.

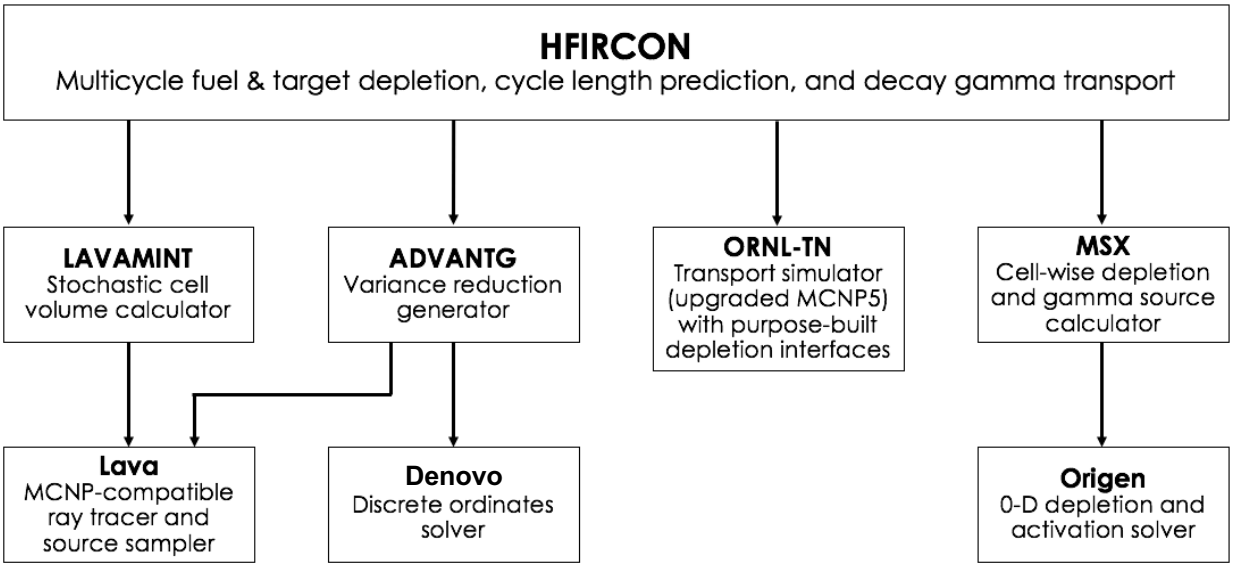

Figure 2. High-level architecture of the HFIRCON code.

4. HFIRCON EXECUTION

HFIRCON may be executed in three different modes. In order of complexity and cost, the modes are "volume calculation," "fuel depletion," and "target depletion." A description of the solver steps and data flow for each is provided in this section.

Many controls are available for users to tailor HFIRCON executions to their needs. These are described in detail in Section 4.1.1.1 Users should read the descriptions of each control in Section 4.1.1.1 to obtain insight into the optimal controls for their problem type.

4.1 INPUTS

HFIRCON requires three input files:

1. **controller_input.json**. This JavaScript object notation (JSON)-formatted file contains all the HFIRCON user controls. The controls are described in Section 4.1.1.1.
2. **Template MCNP input file**. A viable MCNP input file, preferably containing a HFIR model. HFIRCON imposes certain requirements on the MCNP input file (Section 4.1.2).
3. **Stand-alone MCNP SDEF card**. Users must provide a valid MCNP SDEF card in a separate file to initialize rod search and/or fission source convergence calculations.

4.1.1 controller_input.json

The `controller_input.json` controls all the tasks that HFIRCON performs. This section describes each input. A useful understanding of the code's workflow can be obtained by inspecting the input functionality.

The file is formatted as a .json file, which allows rapid and robust input parsing in Python while remaining human-readable. Examples of valid controller_input.json files for HFIRCON are available in the validation problem working directories detailed in Section 7.

It is strongly recommended that new users start with a correctly configured input when setting up a new problem.

The input checking in HFIRCON is relatively robust. A significant effort is made to catch invalid or incompatible inputs, and checks are made for the existence of additional files required for HFIRCON execution. If the controller_input.json file is incorrectly formatted (e.g., a list missing a bracket), then a JSON parse error will be thrown with the line number of the error.

4.1.1.1 Controls

Files and Resources

analysis_name: This controls the name of the PBS file and the submitted job. It will also control the names of the extracted output files. It is a good idea to make this descriptive. HFIRCON will always append the current cycle number to the analysis name (e.g., analysis_name "job" will be modified to "job_cycle_2" when executing cycle number 2). This prevents output files from subsequent cycles from overwriting previous cycle outputs.

num_nodes: The number of compute nodes on which the user wants to execute. HFIRCON is designed to scale reasonably well up to the entire cluster. Users are administratively limited to eight nodes on the smaller apollo, romulus, remus, and lise clusters. Attempts to run larger jobs on any of these clusters will result in the job being killed by the system administrator. On cades, there are no administrative limits. Users could request the entire cluster. There are currently 155 nodes accessible to users in the Nuclear Science and Engineering Directorate (NSED) administrative group (some non-NSED collaborators from RRD are also in this group); however, there are typically a handful of nodes down for maintenance at any given time, and due to the shared nature of the cluster, no job requiring this many nodes will ever move from a "pending" to "active" status in the job queue. Entries of 20–40 may move through the queue quickly and generally provide overnight turnaround for many typical models. However, during times of peak usage, it could take up to a few days to start a 20 node job and up to a week or more for a 40 node job.

procs_per_node: The number of cores on each compute node the user wants to use. This varies by cluster, but it is strongly recommended to use all available CPUs on a node. This will ensure that the job scheduler does not start job(s) from any other users on the nodes that the user wants to use. On cades, romulus, and remus, there are 32 CPUs per node. There are effectively 48 CPUs per node on apollo and 56 CPUs per node on lise.

mem_per_proc: The amount of memory required for each processor. This entry should not be greater than 3,900 mb on cades, romulus, remus, or apollo, except for very special use cases. On lise, this value should be no larger than 3,400 mb.

hours_per_job: Wall hours for the PBS job created and submitted by HFIRCON when executed with the "launch" command line option. This has no effect when executing with the "execute" command line option. It defaults to 1 h, and users should set this value intelligently based on test problem feedback. On cades, there is a 2 day job time limit. The romulus, remus, apollo, and lise clusters have much more lenient overall job time limits (i.e., up to 30 days), but jobs that are predicted to run for more than 4–5 days should be limited to 1week and run as a series of restarts.

platform: This indicates the high-performance computing platform on which the user wants to execute. The currently supported platforms are cades, apollo, romulus, remus, and lise. Other platforms may be supported in the future. The lise platform is an RRD resource, not an NEFCD resource, and accounts on lise will be restricted to personnel performing HFIR-related calculations.

reservation: This entry is only valid on the "cades" platform. Jobs requiring many nodes and/or CPUs and that must be run immediately to support a critical milestone might require special treatment to move from a *pending* to *running* status by the job scheduler. A reservation must be justified and requested by a program manager or group leader. If granted, a reservation will push a job to the top of the list for execution, and that job will start as soon as the requested resources become available, even if other jobs have been waiting in the queue before submission of the job(s) submitted under a valid reservation.

Reservations are granted to a specific user for a specific amount of time. The form of the entry in the "reservation" field is "system.XXX," where *XXX* is the unique reservation number assigned to a user's jobs. All jobs submitted by the approved user, subject to a two-job-per-user limit, will start immediately or as soon as the requested resources become available for the duration of the reservation. Users should ensure that all jobs submitted under a reservation are ready to run (i.e., short test jobs have been run to ensure valid inputs). Users should also ensure that the queue is filled with jobs during the time the reservation is valid. If all the user's jobs intended for execution during the time that the reservation is valid complete before the reservation expires, then the user should notify the cades system administrators via email (cades-help@cades.ornl.gov) that the user no longer requires the reservation to avoid having numerous nodes on the cluster remain idle for the duration of the reservation. This abuse of system resources will negatively impact all other cades users with jobs in the queue and might result in the user having a difficult time obtaining future reservations.

mcnp_inp_name: The name of the template MCNP input deck on which HFIRCON will operate. See the next paragraph for special requirements imposed on the template input deck. HFIRCON checks for the existence of *mcnp_inp_name* at *input_location* during the initial input parse.

working_dir: The absolute directory location in which users want HFIRCON to construct its cycle-dependent execution directories. This directory must exist and have user read/write/execute permission before launching HFIRCON. On cades, this directory should be in a subdirectory of the high-performance Lustre drive (`/lustre/hydra/`) to which the user has write permission. No equivalent file system exists on the romulus, remus, apollo, or lise platforms. A final slash is always appended to this entry by HFIRCON. There are some additional requirements for the working_dir entry due to how HFIRCON sets the ORNL-TN execution mode. Specifically, the string cannot contain the following forbidden substrings: "mc_exec," "dg_exec," "volume," or "rod_search."

input_location: The absolute directory location of other input files required for HFIRCON execution. Currently, this set consists of (1) the MCNP template deck and (2) an optional SDEF card suitable for starting rod search calculations. A final slash is always appended to this entry by HFIRCON.

output_location: The absolute directory location in which extracted output from the cycle depletion calculation will be written. A final slash is always appended to this entry by HFIRCON.

user_name: The three-character user ID of the person launching HFIRCON. This is currently used only in constructing the PBS input file and is optional. The value of *user_name* is set automatically in the execution shell via "whoami," if possible.

sdef_card_name: The name of a file containing a valid SDEF card representing a beginning-of-cycle (BOC) HFIR fission source distribution. It is only used for an initial source guess for rod search calculations.

Execution:

ICE_transform: The single integer entry controlling the number of the inner control element (CE) transformation.

OCE_transform: The single integer entry controlling the number of the outer CE transformation.

cycle_number: The number of the cycle being executed within the context of a single set of multicycle depletions. This number must always be zero for fresh fuel and fresh targets. Numbers greater than zero

will assume fresh fuel at the start of the cycle (e.g., obtain it from the base deck) but depleted targets obtain target number densities from previous cycle runs.

initial_decay_time: The number of days of shutdown time between the previous cycle and the current cycle. This is only used for cycle numbers greater than zero.

depletion_step_times: A list of times (in days) controlling the length of the depletion steps. It must always start at 0.0, and values must increase monotonically. The stability of the predictor method implemented in HFIRCON is conditional on the step size. Steps that are too large might become unstable or result in inaccurate cycle-length predictions.

advantg_steps: A list of integers explicitly indicating at which steps an ADVANTG run should be executed. An empty list indicates that ADVANTG should never be run in this cycle. ADVANTG will attempt to optimize for fuel neutron flux, fission rates, and capture rates in mode "fuel depletion" or fuel neutron flux, fission rates, and capture rates and target flux, fission rates, capture rates, and heating (neutron and gamma) in mode "target depletion." For depletion time steps in which an ADVANTG run is not requested, HFIRCON will create a symbolic link to the ADVANTG-generated weight windows (wwinp) file in the previous time step. For jobs in which ADVANTG runs are not requested for multiple, contiguous depletion steps, each time step in that chain will contain a symbolic link to the previous time steps wwinp file until a link is established to an actual ADVANTG-generated wwinp file. On the romulus and remus computing clusters, operating system constraints limit how long this "chain" of links can be. If the path to the most recent ADVANTG-generated wwinp file becomes too long, then the user will see a fatal error in the ORNL-TN job output files stating that the "wwinp file does not exist," even though it does. For jobs run on the romulus or remus clusters, it is recommended that users request an ADVANTG calculation be performed at least every five time steps (e.g., "advantg_steps": [0, 5, 10, …]). More frequent invocation of ADVANTG is acceptable but might not be needed during the middle portion of a cycle in which CE motion is minimal.

eoc_decay_steps: An optional list of user-defined times (in days) after cycle completion at which to calculate decay. Additional outputs will be produced with "AS_#" appended to the filenames (indicating after-shutdown and the number of the decay step) if this list is provided. The eoc_decay_steps are always decayed from the output of the last successful msx_deplete execution.

mctal_write: Boolean control instructing HFIRCON to call MCNP5/ORNL-TN with mctal writes after the main MC executions in each step. The default is "False".

fixed_rod_positions: An optional list of user-defined CE locations. If provided, the rod search calculations are converted to source convergence calculations and rod location is set to the user-defined value in each step. This list must contain one entry for every depletion step, as defined in *depletion_step_times*.

start_reaction_materials: An integer indicating the number at which reaction rate tally materials can start. Generally, this is an optional entry. HFIRCON will set this automatically by parsing the *mcnp_inp_name* file and setting *start_reaction_materials* to max (cell_numbers, material_numbers) + 1. If the automatic setting fails, then HFIRCON will use the user-provided value.

restart: Boolean value (True or False) indicating whether to attempt to restart this cycle calculation. Restarts are available for any cycle calculation that has progressed past its volume calculation, if one is being executed. If the value of *restart* is "False" and the cycle directory is found to exist already, then it will be completely removed to start the calculation.

execution_mode: A string indicating how to execute HFIRCON. Setting this to “fuel_depletion” will set up only the tallies and solutions required to perform fuel depletion. Setting it to “target_depletion” will set up tallies and solutions required to perform fuel and target depletion, as well as target heating. Setting it to “volume_calculation” will create the cycle directory structure but will only perform the stochastic volume calculation step, which will generate a VOL card and an ADVANTG mesh.

volume_calc: Boolean indicating whether to perform a stochastic volume calculation. This is recommended unless all volumes are already known and set in the input file. Currently, HFIRCON sets this value to “True” internally for all available execution modes since an EOC fuel activation gamma transport calculation is always performed that requires volumes for all possible source cells. For some jobs for which the volume calculation might take a long time (e.g., models with the HFIR explicit core definition), it is preferable to do the volume calculation in a separate “volume only” job and then perform the transport/depletion calculations as a restart job with the value of “*volume_calc*” set to “False”. It might also be advisable for multicycle jobs to only run the volume calculation on the first cycle and then copy the volume directory from the “cycle_0” directory to the other “cycle_#” directories if this volume calculation will take considerable time to complete.

volume_tol: A floating point value setting the tolerance of the stochastic volume calculation. It is recommended to be 0.01 or perhaps 0.005; very small values (e.g., 0.0001) will take a prohibitive amount of wall time for most input decks.

mesh_tols: A list of two floating point values providing coarse control over the ADVANTG mesh generator. The first value is the minimum tolerance, which sets a limit on the minimum distance between consecutive *x*, *y*, or *z* planes. The second value is the maximum tolerance, which sets a limit on the maximum distance between consecutive *x*, *y*, or *z* planes. Values of 0.2 and 10.0 appear to provide stable, useful mesh definitions for ADVANTG. Smaller values of the minimum tolerance are probably supportable, perhaps down to 0.1 or 0.08 cm, but values smaller than that will result in very large wwinp files that will have a prohibitive memory impact. Larger values will decrease ADVANTG execution time at the expense of deterministic flux resolution; this might be appropriate for some problems.

problem_bounds: A list of six floating point values setting the bounds of the ADVANTG mesh. In order, they are: x_min, x_max, y_min, y_max, z_min, and z_max.

vol_problem_bounds: An optional list of six floating point values setting the problem bounds for the LAVAMINT volume calculation. Internally, HFIRCON will default this list to *problem_bounds* unless it is specified by the user. If it is present in controller_input.json, then the bounds for the LAVAMINT volume calculation can be different from the ADVANTG mesh problem bounds. This feature is available to allow users to place objects, typically targets, in the geometry but outside the weight window bounds that should be tracked through a cycle duration but should not be irradiated.

rod_pos_cycles: An integer value setting the number of histories performed in each rod position transport cycle. HFIRCON starts NP jobs per node with 1/NP*num_nodes CPUs bound to each job, where *NP* is the number of nonuniform memory access (NUMA) pools on a given platform (currently four on romulus and remus and two on cades, apollo, and lise). Fifty cycles are performed in each rod search MC execution, with 10 discarded. Therefore, the total number of histories executed in each step is equal to: **(num_nodes)*(NP)*(50)*rod_pos_cycles**. Generally, a value of 5.0e4 is suitable for a broad set of problems being run on clusters with two NUMA pools; values as small as 2.0e4 are stable, whereas more might be required for very accurate cycle length calculations. For jobs run on romulus or remus, the value of *rod_pos_cycles* can be reduced by a factor of 2 relative to jobs run on the cades, apollo, or lise cluster if the same total number of histories is desired.

mc_exec_cycles: An integer value setting the number of histories performed in each cycle of the main in-step transport solutions. The number of cycles is hard coded to 30 for every calculation, which is the minimum allowed by MCNP5. HFIRCON starts NP jobs per node with 1/NP*num_nodes CPUs bound to each job, where *NP* is the number of NUMA pools on a given platform (currently four on romulus and remus and two on cades, apollo, and lise). Therefore, the total number of histories executed in each step is equal to: **(num_nodes)*(NP)*(30)*mc_exec_cycles**. Generally, a value of 5.0e4 is suitable for a broad set of problems being run on clusters with two NUMA pools; more will be required for particularly onerous calculations. For jobs run on romulus or remus, the value of *mc_exec_cycles* can be reduced by a factor of 2 relative to jobs run on the cades, apollo, or lise cluster if the same total number of histories is desired.

dg_exec_nps: An integer value setting the number of histories performed in each EOC fuel activation gamma transport calculation. A value of 1e6 is suitable for a broad range of problems.

reactor_power: A floating point value setting the reactor power in megawatts. Used for normalizing tallies.

fission_energy: A floating point value setting the energy per fission. Used for normalizing tallies.

simple_gamma_phys: Boolean control that determines the gamma physics treatment. The default value for this is “True”, which turns off thick target bremsstrahlung. MCNP generates thick target bremsstrahlung tables for every material in the problem; for depletion problems with many unique material numbers, this can result in a significant run time penalty. Only change to “False” if necessary for the application.

lavamint_override: Boolean control that instructs HFIRCON to override analytically calculated volumes with LAVAMINT stochastic volumes. This is typically unnecessary unless users are depleting cells replicated in multiple instances of a universe (e.g., filled multiple times, as is the case with HFIR models using the explicit core representation). If this is the case, users must change this value to “True” or the MCNP fluxes in those will be normalized to the incorrect volume. Figure 1 represents a small portion of a HFIR explicit core model in which setting *lavamint_override* to True is required.

fuel_mts: A list containing a single string entry indicating the special material treatment for all fuel cells (defaults to [“lwtr.10t”]). MCNP5, on which ORNL-TN is based, does not handle continuous thermal scattering laws (suffix .2xt) correctly. The input checker will allow continuous thermal scattering mt libraries such as [“lwtr.20t”], but these generally should not be used with MCNP5.

target_mts: A list containing a single string entry indicating the special material treatment for all target cells (defaults to [““]).

fuel_nlib: A single string entry indicating the suffix of the appropriate neutron data library for fuel cells (e.g., “80c”).

fuel_plib: A single string entry indicating the suffix of the appropriate photon data library for fuel cells (e.g., “04p”).

target_nlib: A single string entry indicating the suffix of the appropriate neutron data library for target cells (e.g., “80c”).

target_plib: A single string entry indicating the suffix of the appropriate photon data library for target cells (e.g., “04p”).

fuel_cell_list: A list of integer cell numbers of depletable fuel regions. Used in execution modes "fuel_depletion" and "target_depletion."

target_cell_list: A list of integer cell numbers of depletable target regions. Used in execution mode "target_depletion." Currently must be present in any HFIRCON execution, even "fuel_depletion" executions.

target_fission_rate_zaids: An optional list of strings indicating the zaids of reaction rate materials for the target fission rate tallies. It will default to all ENDF-B/VII.1 cross sections with nonzero fission cross sections if no entry is found.

target_capture_rate_zaids: An optional list of strings indicating the zaids of reaction rate materials for the target capture rate tallies. It will default to the entire ENDF-B/VII.1 library if no entry is found.

fuel_fission_rate_zaids: An optional list of strings indicating the zaids of reaction rate materials for the fuel fission rate tallies. It will default to all ENDF-B/VII.1 cross sections with nonzero fission cross sections if no entry is found.

fuel_capture_rate_zaids: An optional list of strings indicating the zaids of reaction rate materials for the target capture rate tallies. It will default to the entire ENDF-B/VII.1 library if no entry is found.

flux_group_structure: A list of monotonically increasing floating point values setting the energy group structure of the neutron flux tallies used for depletion. Values must be entered in mega-electronvolts (MeV). It will default to the SCALE 252-group neutron structure if no entry is found.

gamma_source_group_structure: A list of monotonically decreasing floating point values setting the energy group bounds for the EOC fuel activation gamma source. Values must be entered in MeV. It will default to a 189-group gamma structure if no entry is found.

CEmax: Maximum allowable displacement of the HFIR CEs (fully withdrawn). The default is 68.58 cm.

CEmin: Minimum allowable displacement of the HFIR CEs (fully inserted). The default is 43.688 cm.

eoc_ce_position: CE position at which the volume calculation will execute. This must correspond to the expected EOC CE position to enable the decay gamma transport calculation. The default is 68.58 cm.

forced_cycle: An optional input to force HFIRCON to continue execution during a rod search in which it cannot find a position yielding k_{eff} = 1.0. It is useful for forcing cycle lengths of a specific duration for a calculation. If specified, this control will consist of a single floating-point value that must be between *CEmin* and *CEmax,* inclusive. The execution of all the time steps in the "depletion_time_steps" control will be performed with the CE position being set to the value of the *forced_cycle* control for any time steps for which a position yielding k_{eff} = 1.0 cannot be found.

Postprocessing:

inventory_isos: A list of all isotopes to be included in the *analysis_name_cycle_#_inventory_masses.txt* and *analysis_name_cycle_#_inventory_masses_AS.txt* files generated by the output extractor. In the initial implementation of HFIRCON, these files contained only the isotopic masses by time for eight hard-coded isotopes of interest in the production of $^{238}PuO_2$ from $^{237}NpO_2$ targets. Versions 1.0.2 and later of HFIRCON allow users to specify a list of arbitrary length containing the MCNP ZAIDS for all isotopes of

interest in all cells listed in the *target_cell_list* entry. It defaults to the Pu-specific list of [93237, 93238, 94236, 94238, 94239, 94240, 94241, 94242].

fuel_output_map: A list of fuel cell output organization instructions for the HFIRCON output extractor. It is formatted as a list of lists. Each entry in fuel_cell_list has a single list corresponding to it in the fuel_output_map. The first entry is the fuel cell number, the second is a tag indicating which region the cell contributes to, the third is a radial location in the fuel region, and the fourth is an axial location in the fuel region. Figure 3 provides an example.

```
"fuel_output_map": [ [2101,"IFE",1,1],
                     [2102,"IFE",2,1],
                     [2103,"IFE",3,1],
                     [2104,"IFE",4,1],
                     [2105,"IFE",5,1],
                     [2106,"IFE",6,1],
                     [2107,"IFE",7,1],
                     [2108,"IFE",8,1],
                     [2111,"IFE",1,2],
                     [2112,"IFE",2,2],
                     [2113,"IFE",3,2],
```

Figure 3. Partial listing of fuel_output_map control.

This partial list shows the general organization of the control. Each cell listed in the *fuel_cell_list* list must have a corresponding entry in the *fuel_output_map* list.

target_output_groups: A list of target cell output organization instructions for the HFIRCON output extractor. This control is formatted as a list of lists, with the interior list length being arbitrary. The first entry in each interior list is a target cell number, with each subsequent entry indicating a summation group. Figure 4 provides an example.

```
"target_output_groups": [ [57111," Pellet_1"," All_Pellets"," Assembly"],
                              .
                              .
                              .
                          [57190," Pellet_1"," All_Pellets"," Assembly"],
                          [57191," Pellet_2"," All_Pellets"," Assembly"],
                              .
                              .
                              .
                          [57270," Pellet_2"," All_Pellets"," Assembly"],
                          [57271," Pellet_3"," All_Pellets"," Assembly"],
                              .
                              .
                              .
                          [57350," Pellet_3"," All_Pellets"," Assembly"],
                          [57351," Pellet_4"," All_Pellets"," Assembly"],
                              .
                              .
                              .
                          [57430," Pellet_4"," All_Pellets"," Assembly"],
                          [ 3114," ICE_Gray_Q1"," CEs"," Assembly"],
                          [ 3115," ICE_Black_Q1"," CEs"," Assembly"],
                          [ 3116," ICE_Gray_Q2"," CEs"," Assembly"],
                          [ 3117," ICE_Black_Q2"," CEs"," Assembly"],
                          [ 3118," OCE_Gray_Q3"," CEs"," Assembly"],
                          [ 3119," OCE_Black_Q3"," CEs"," Assembly"],
                          [ 3120," OCE_Gray_Q4"," CEs"," Assembly"],
                          [ 3121," OCE_Black_Q4"," CEs"," Assembly"],
                          [  5615," Upper_SS_Spring"," Structure"," Assembly"],
                          [  5616," Zirc_Clad"," Structure"," Assembly"],
                          [  5622," Moly_Spacer"," Structure"," Assembly"],
```

Figure 4. Partial listing of target_output_groups control.

For brevity, the list presented in Figure 4 has been modified to indicate where numerous lines have been removed, as indicated by three consecutive lines containing only a single ".". The removed lines are identical to the lines immediately preceding and following the lines containing on the single ".", except for the first entry (cell number), which increases by one for each removed line. Therefore, Figure 4 indicates that cells 57111 through 57190 belong to a group labeled "Pellet_1." The results from these pellets will also contribute to the groups labeled "All_Pellets" and "Assembly."

Similarly, cells 57191 through 57270 will contribute to the groups labeled "Pellet_2", "All_Pellets", and "Assembly", whereas cells 57271 through 57350 contribute to group "Pellet_3" and the groups providing summations over all four pellets and the entire assembly. By extension, cells 57351 through 57430 contribute to the "Pellet_4", All_Pellets" and "Assembly" groups. The remaining cells depicted in Figure 4 contribute to individual component results (e.g., cell 3114 represents results for the portion of the gray region of the inner CE (ICE) that is located within the first quadrant (0–90°) of the model) and multiple summation groups (e.g., cell 3114 contributing to a summation over all cells comprising the CEs) and a summation over all depleted cells in the overall assembly.

All summation groups are used in reporting the isotopic activities (in units of Curies) found in the "*analysis_name*_cycle_*cycle_num*_target_activity.txt," "*analysis_name*_cycle_*cycle_num*_target_activity_AS.txt," "*analysis_name*_cycle_*cycle_num*_group_activity_EOC.txt," and "*analysis_name*_cycle_*cycle_num*_group_activity_AS_#.txt" files, where "#" is an integer from one to the number of postirradiation time steps listed in the *eoc_decay_steps* variable. The capability to sum the isotopic activities into multiple groups allows for a single calculation to support shipping, disassembly, processing, and disposal tasks for a variety of subsets of a particular experiment as it passes from HFIR to various hot cells or glove boxes for processing of the various components and eventually to multiple storage, recycle, or disposal sites.

The first summation group is used to provide a cell description in several of the "target_depletion" mode ASCII output files not listed in the preceding paragraph.

Each cell listed in the *target_cell_list* list must have a corresponding entry in the *target_output_groups* list.

mcnp_file_steps: A list of integer time steps at which MCNP input files are to be written. By default, HFIRCON keeps all cell and material definitions in memory during a depletion calculation. This list allows users to print a valid MCNP input file that contains cell and material definitions for one or more specific time steps during the depletion calculations. The MCNP input files created via this option, combined with the "srctp_converged" source file for the corresponding time step, can then be used to run numerous fixed-source parameter studies.

4.1.2 MCNP Input Deck Requirements

There are several requirements that the template MCNP input deck referenced in the controller_input.json file must meet. HFIRCON attempts to check for these when it performs its input check.

- ICE transform must be present in the input deck and contained on a single line.
- OCE transform must be present in the input deck and contained on a single line.

- The “mt” card for all fuel and target materials that specify an associated thermal moderator treatment must be constrained to a single line if the controller_input.json file contains a non-blank entry for the *mcnp_file_steps* control.
- All solver controls can be present in the input deck, except for SDEF, but they should each be contained on a single line.
- An SDEF card should not be included in the input deck.
- No VOL or cell-dependent vol= cards should be included in the input deck. HFIRCON will fail in its volume calculation if these are present in the deck.
- Tally numbers 4, 14, 24, 34, 44, 54, 66, and 76 are reserved for fuel flux, target flux, fuel capture rates, target capture rates, fuel fission rates, target fission rates, target neutron heating, and target gamma heating, respectively. Do not use these tally numbers.

ORNL-TN uses a cell-to-material map that is automatic after the first depletion step. This is simply material # = cell #. The template input does not require a unique material for every cell, but it will define a unique material via its Python interface after the first step. Therefore, users must not define materials in a way that will interfere with this internal mapping convention. For example, consider the case in which there are two depletion cells, “1000” and “10000”, and a user assigns both of them material “1” in the base input. Material numbers “1000” or “10000” should not be used for other cells because the material assigned to cell “1000” will be material “1000” after the first depletion step, and cell “10000” will be assigned material “10000”.

ORNL-TN will define tallies using reserved material numbers controlled by “start_reaction_materials” and the lists of reaction rate zaids. Other materials cannot be used that will exist in this range. HFIRCON should set the start_reaction_materials parameter automatically.

HFIRCON input checking has been built to attempt to catch all of these potential issues before computationally expensive solver executions.

The user-provided sdef_card file must simply contain a valid MCNP source entry to initialize the HFIRCON fission source calculation. This can be a space- and energy-dependent SDEF card, a simpler SDEF source, or simply a ksrc point entry. The sdef_card will only be used to initialize rod search or fission source convergence calculations; the main MC simulations will use converged fission sources from previous simulations.

4.2 INITIALIZATION

In all execution modes, HFIRCON performs some common initialization tasks.

First, it parses the controller_input.json file containing all HFIRCON controls. If there is a syntax error in the controller_input.json file (e.g., a missing comma between controls, a skipped entry in a list), then the user will see only a JSON parsing error. At the bottom of these JSON error messages is typically a line number and character number pointing to the exact location of the syntax error.

Next, HFIRCON checks the input. HFIRCON attempts to perform a rigorous check of all user-provided input. The gifted user could find a way around these checks to force the code to perform unusually, but significant effort has been made to avoid launching code execution with any unviable control settings. HFIRCON input checking parses the entire input file and provides feedback on all the errors it can find with messages indicating how to resolve them.

If the command line control is `launch`, then no PBS file will be submitted if HFIRCON finds a user input error. If the command line control is `execute`, then the execution will stop after the input check. If HFIRCON finds no user input errors, then it will proceed to launch the relevant calculations per the information in the MCNP template file, the initial source in `sdef_card`, and the controls in controller_input.json.

If the HFIRCON input checker passes, then the cycle directory structure will be created in the user-defined *working_dir*. Section 5 describes the contents of the cycle-dependent working directory and its subdirectories after a successful HFIRCON depletion calculation.

4.3 VOLUME CALCULATION

HFIRCON may be executed to perform only the stochastic volume calculation step. This is selected by setting the *execution_mode* control to "volume_calculation."

HFIRCON automatically performs a stochastic volume calculation in all the other execution modes unless the *volume_calc* control is set to "False," so running a stand-alone volume calculation is unnecessary unless the user only desires the outputs of the volume calculation (Section 4.8). It might also be useful if the user has a particularly expensive volume calculation and desires to decompose the problem (Section 4.6).

In this mode, HFIRCON will set up the initial cycle execution directory (as always), but the step_ directories will remain empty. Only the volume directory will be modified.

HFIRCON will enter the volume directory and perform the following tasks.

1. Copy the MCNP template input into the volume directory.
2. Copy an ORNL-TN config file with appropriate execution controls into the volume directory.
3. Set the ICE_transform and OCE_transform in the local copy of the MCNP template to fully withdrawn locations.
    a. *Note: this is done to facilitate conservative EOC decay gamma transport simulations. Fully withdrawn CEs minimize the CE shielding of the fuel activation gamma sources.*
4. Execute ORNL-TN to generate a valid runtpe file for LAVAMINT.
5. Call parallel LAVAMINT using all available processors.
    a. LAVAMINT is executed in stochastic volume calculation mode and will calculate the volumes of all the cells in the input. One MPI process will be launched per socket (i.e., 2 MPI processes with 16 threads each will be launched on nodes with 32 CPU contexts on 2 sockets).
6. Generate ADVANTG spatial mesh.
    a. HFIRCON parses LAVAMINT text output and uses relevant user controls (see *mesh_tols* in Section 4.1.1.1) to generate a useful ADVANTG mesh.
    b. *Note: very small settings in the first entry of mesh_tols (i.e., the minimum distance between consecutive Cartesian planes) can result in very large weight window files and a prohibitive memory impact. Exercise caution when setting this below ~0.2 cm for typical HFIR problems.*
7. Generate the ratio.txt file.
    a. This file contains a list of the ratios of LAVAMINT stochastic volumes to MCNP analytical volumes (per MCNPs "Table 60" output). It may be used as an inline

verification of the LAVAMINT stochastic volume calculation and should always be inspected.

8. Generate the volcard.txt file.
    a. This file contains a complete VOL card suitable for use with the MCNP input file provided by the user.
    b. *Note: cell volumes not calculated either by MCNP or LAVAMINT (e.g., complex cells outside the user-defined problem bounds) will be set to 0.0. This is done to prevent bad volume-averaged tally data from passing into the downstream calculations*.

These volume calculation steps are performed once per depletion cycle unless overridden by setting the *volume_calc* control to "False". HFIRCON requires a completed volume calculation to exist for each cycle, but it is possible to simply copy the volume directory from one cycle directory to any other cycle directories when multicycle calculations are performed for a given geometry.

For many problems, this will not be very valuable because the volume calculations are generally inexpensive and require no more than a few minutes to perform. However, for some complex geometries, such as any model containing the HFIR explicit core description (540 individual plates, each split into several small radial and axial zones for each of the cladding, filler, and fuel meat zones) the volume calculation can take from several hours on the cades cluster up to a few days on the oldest, smallest, and slowest clusters where HFIRCON is currently available (i.e., romulus and remus). Repeating this calculation for each cycle is inadvisable. In these cases, it is best to run an initial cycle_0 HFIRCON job in "volume_calculation" mode with the *restart* control set to "False" and the *volume_calc* control set to "True". Once the volume calculation has completed, the remainder of the cycle_0 calculation can be run as one or more restart jobs with the *restart* control set to "True" and the *volume_calc* control set to "False." If results for cycles beyond the initial cycle_0 calculation are needed, then the user could create the necessary cycle_# directories, where "#" is an integer value from one to the total number of desired cycles; copy the volume subdirectory from the cycle_0 directory to the various cycle_# directories; and submit the jobs for each of the cycle_# calculations as restart jobs with the *restart* control set to "True" and the *volume_calc* control set to "False".

4.4 FUEL DEPLETION

The HFIRCON "fuel_depletion" execution mode is intended for fuel depletion analyses and is essentially a subset of the "target_depletion" capability.

Users define fuel depletion regions on a geometry cell basis in the controller_input.json file via the *fuel_cell_list* control. All the geometry cells in this list will be treated as fuel depletion regions. Fuel depletion regions are distinct from target regions in that their number densities are reset to the base input number densities on every cycle initialization. Therefore, although HFIRCON can be run over multiple cycles in the fuel_depletion execution mode, doing so will simply repeat the first cycle calculation. A fuel_depletion calculation is most useful for cycle length calculations in which targets are known to not contribute to the fission source or for new HFIR designs (e.g., low-enriched uranium [LEU] fuel designs).

When HFIRCON begins a fuel depletion calculation, it first performs a volume calculation. The steps are identical to those performed in a stand-alone volume calculation (Section 4.3).

Once the volume calculation is complete, HFIRCON performs the following tasks at every user-defined time step.

1. Changes the current directory to the relevant step directory (step_#, where "#" is the current depletion step).
2. Copies relevant files to the step_# working directory (e.g., ORNL-TN config files).
3. Generate tallies suitable for the fuel_depletion execution mode.
    a. Multigroup neutron fluxes, fission rates, and capture rates for all the geometry cells in the *fuel_cell_list* control.
4. If the *fixed_rod_positions* control list is present in the controller_input.json file, then set the OCE_transform and ICE_transform to the user-provided value and perform a fission source convergence simulation (a "mode n" transport calculation with no tallies).
5. If the *fixed_rod_positions* control list is not present in the controller_input.json file, then perform a rod search.
    a. HFIRCON calculates a range of possible CE positions based on the current time step. HFIRCON will launch MC simulations with CE positions set evenly across this range. The number of simulations is equal to the number of sockets in the PBS job; 2**num_nodes* on the cades, apollo, and lise clusters; *4*num_nodes* on the romulus or remus clusters. Upon completion, HFIRCON interpolates the k_{eff} eigenvalue results against the simulated CE positions and searches for $k_{eff} = 1.0$. The CE position for this step is set to the interpolated value.
        i. *Note: if no* $k_{eff} = 1.0$ *position can be found and the* forced_cycle *control is not present, HFIRCON will assume the cycle is complete and terminate execution*.
6. If the current depletion step is present in the *advantg_steps* control list, then perform an ADVANTG calculation.
    a. *Note: generating ADVANTG variance reduction parameters is typically unnecessary for HFIR fuel_depletion analyses. However, HFIRCON will generate and use these parameters, if instructed.*
7. If the current depletion step is not present in the *advantg_steps* control list, then search for Variance Reduction (VR) parameters generated in previous depletion steps.
    a. If found, then link the most recently generated ADVANTG outputs to the current directory.
    b. If not found, then proceed without variance reduction parameters.
8. Execute main MC simulations for this step.
    a. MC simulations are performed in "asynchronous parallel" mode, much like the rod search simulations, except these main MC simulations all use the same CE position. Several threaded simulations equal to the number of available sockets in the parallel job will be launched. Their output is rebuilt after all simulations are complete. Variance reduction parameters (weight windows) are used if they are found in the current step directory. The initial fission source is set by using the rod search simulation output closest to the interpolated $k_{eff} = 1.0$ value.
        i. Note: asynchronous parallel execution will generally improve performance due to certain implementation choices made in the MPI version of MCNP. It should also improve fission source convergence by reducing fission source correlation.
9. Postprocess the MC simulation output.
10. Execute msx_deplete.
    a. HFIRCON calls msx_deplete in parallel, using all available processors. The msx_deplete execution is MPI parallel by cell. The flux normalization is calculated by using the number of neutrons per fission tallied in the MC simulations, the recoverable energy per fission set by the user via the *fission_energy* control, and the reactor power set via the *reactor_power* control.

<u>After the final step calculation has been completed:</u>

1. Generate decay gamma sources in the EOC reactor state.
    a. HFIRCON uses LAVAMINT and the automated interface to ORIGEN via MSX to generate decay gamma sources at EOC. All depleted cells are treated as gamma sources in this step.
2. Do NOT execute EOC decay gamma transport.
    a. Only the "target_depletion" mode executes the EOC decay gamma transport step (Section 4.5) because HFIRCON does not collect heating edits for fuel cells. The EOC decay gamma transport step is performed exclusively to determine decay gamma contributions to heating rates.
3. For all times in "eoc_decay_steps," decay the EOC number densities to the user-defined entry.
4. Postprocess all output and create various ASCII text files from step dependent HDF5 files. The conditions defining which text files are created are described in Section 4.8.3.

4.5 TARGET DEPLETION

The "target_depletion" execution mode is HFIRCON's most general execution mode. In this execution mode all "fuel" and "target" cells will be depleted throughout the current cycle. At the end of each cycle, all "fuel" cell materials will be reset to their BOC compositions while all "target" cell material compositions will be decayed from the end of the current cycle to the beginning of the next cycle.

Like in the "fuel_depletion" mode, the user defines fuel depletion regions on a geometry cell basis in the controller_input.json file via the *fuel_cell_list* control. All the geometry cells in this list will be treated as fuel depletion regions. Fuel depletion regions are distinct from target regions in that their number densities are reset to the base input number densities on every cycle initialization.

Contrary to the "fuel_depletion" mode, the "target_depletion" mode uses an additional list in the controller_input.json file called *target_cell_list*. All the geometry cells in this list are treated as target depletion regions. The primary difference between fuel and target cells is how the cycle-to-cycle transition is treated: in cycles greater than zero, target cell number densities are acquired from the previous cycle end point, decayed according to the value specified on the *initial_decay_time* control, and used as the step 0 number densities for the target cells in the new cycle execution.

When HFIRCON begins a target depletion calculation, it first performs a volume calculation. The steps are identical to those performed in a stand-alone volume calculation (Section 4.3).

Once the volume calculation is complete, HFIRCON performs the following tasks.

<u>For every time step defined by the user:</u>

1. Changes its current directory to the relevant step directory (step_#, where # is the current depletion step).
2. Copies relevant files to the step_# working directory (e.g., ORNL-TN config files).
3. Generate tallies suitable for the target_depletion execution mode.
    a. Multigroup neutron fluxes, fission rates, and capture rates for all the geometry cells in the *fuel_cell_list* control.
    b. Multigroup neutron fluxes, fission rates, capture rates, gamma heating rates, and neutron heating rates for all the geometry cells in the *target_cell_list* control.

4. If the *fixed_rod_positions* control list is present in the controller_input.json file, then set the OCE_transform and ICE_transform to the user-provided value and perform a fission source convergence simulation (a "mode n" transport calculation with no tallies).
5. If the *fixed_rod_positions* control list is not present in the controller_input.json file, then perform a rod search.
    a. HFIRCON calculates a range of possible CE positions based on the current time step. HFIRCON will launch MC simulations with CE positions set evenly across this range. The number of simulations is equal to the number of sockets in the PBS job; 2**num_nodes* on the cades, apollo, and lise clusters; and *4*num_nodes* on the romulus or remus clusters. Upon completion, HFIRCON interpolates the k_{eff} eigenvalue results against the simulated CE positions and searches for $k_{eff} = 1.0$. The CE position for this step is set to the interpolated value.
        i. *Note: if no* $k_{eff} = 1.0$ *position can be found and the* forced_cycle *control is not present, then HFIRCON will assume the cycle is complete and terminate execution*.
6. If the current depletion step is present in the *advantg_steps* control list, then perform an ADVANTG calculation.
    a. *Note: exercise caution with the ADVANTG executions. It is rare to need more than a few ADVANTG executions per cycle, if any, and modifying the mesh controls can result in exceeding hardware memory limits.*
7. If the current depletion step is not present in the *advantg_steps* control list, then search for VR parameters generated in previous depletion steps.
    a. If found, then link the most recently generated ADVANTG outputs to the current directory.
    b. If not found, then proceed without variance reduction parameters.
8. Execute the main MC simulations for this step.
    a. MC simulations are performed in "asynchronous parallel" mode, much like the rod search simulations, except these main MC simulations use the same CE position. Several threaded simulations equal to the number of available sockets in the parallel job will be launched. Their output is rebuilt after all simulations complete. Variance reduction parameters (weight windows) are used if they are found in the current step directory. The initial fission source is set by using the rod search simulation output closest to the interpolated $k_{eff} = 1.0$ value.
        i. Note: asynchronous parallel execution will generally improve performance due to certain implementation choices made in the MPI version of MCNP. It should also improve fission source convergence by reducing fission source correlation.
9. Postprocess the MC simulation output.
10. Execute msx_deplete.
    a. HFIRCON calls msx_deplete in parallel, using all available processors. The msx_deplete code is MPI parallel by cell. The flux normalization is calculated by using the number of neutrons per fission tallied in the MC simulations, the recoverable energy per fission set by the user via the *fission_energy* control, and the reactor power set via the *reactor_power* control

<u>After the final step calculation has been completed:</u>

1. Generate decay gamma sources in the EOC reactor state.
    a. HFIRCON uses LAVAMINT and the automated interface to ORIGEN via MSX to generate decay gamma sources at EOC. All depleted cells are treated as gamma sources in this step.
2. Execute EOC decay gamma transport.

a. A custom source plugin to ORNL-TN is used to perform a gamma transport calculation in which decay gamma heating is tallied for every geometry cell in the *fuel_cell_list* control and the *target_cell_list* control.

3. For all times in "eoc_decay_steps," decay the EOC number densities to the user-defined entry.
4. Postprocess all output and create various ASCII text files from step-dependent HDF5 files. The conditions defining which text files are created are described in Section 4.8.3.

4.6 RESTARTING HFIRCON

HFIRCON was designed to be restarted in the event of a failed simulation. The only control required to restart a HFIRCON calculation is *restart* in the controller_input.json file. If set to "True", then HFIRCON will search for an existing cycle directory corresponding to the cycle number specified in the controller_input.json file. If it is not found, then HFIRCON will immediately stop execution and inform the user that no restart is possible. If the current cycle directory is found, then HFIRCON will search the execution subdirectories to find the most recent completed step. It will delete any subsequent directories, including those with partially completed calculations, and restart execution at the latest incomplete depletion step.

Some circumstances will modify the way the restart works. In the case in which a CE position search is performed, if $k_{eff} = 1.0$ cannot be found at a particular step, then a file named rod_search_completed.out will be written in the step directory. HFIRCON will search for this file and, if found, will consider the cycle complete. Likewise, if the CE position in which $k_{eff} = 1.0$ is past the CEmax value, which defaults to 68.58 cm but can be overridden with the *CEmax* control, then the same file will be written with the same effect.

HFIRCON "volume_calculation" mode executions cannot currently be restarted. However, it might be useful to decouple a very expensive volume calculation from the remainder of either a "fuel_depletion" or "target_depletion" calculation. This can be accomplished by first executing a "volume_calculation" to completion and then executing a subsequent "fuel_depletion" or "target_depletion" calculation with the *restart* variable set to "True" and the *volume_calc* variable set to "False".

It is strongly recommended not to alter the restart capability in HFIRCON by modifying the directory structure. As an example, consider the case in which an insufficient number of depletion steps and MC histories were used in a cycle length calculation. Users might want to delete the last six depletion steps, then modify the controller_input.json file to use more histories and more depletion steps. However, if users make a mistake and render the controller_input.json "depletion_step_times" list inconsistent with the HFIRCON execution directory (e.g., they delete seven completed step directories, then delete eight entries in the "depletion_step_times" list and add 12 new ones), HFIRCON might not detect it and will attempt to complete the cycle calculation. The final results will be incorrect.

Instead of trying to modify an existing execution directory to obtain better convergence, rerun the simulation with new controls.

4.7 GENERAL USER GUIDANCE

This section provides some guidelines to help HFIRCON users quickly obtain useful results.

HFIRCON was designed to solve several different HFIR depletion problem types. The following is a representative list:

- HFIR cycle length calculation,
- single-cycle target depletion within the flux trap,
- single-cycle target depletion in an outer vertical experiment facility (VXF), and
- multicycle target depletion in multiple VXFs.

Each of these are best solved with different control parameters. For example, a cycle length calculation might not require any variance reduction (i.e., "advantg_steps" should be left empty) but could require many depletion steps with more histories per step.

Single-cycle target depletion within the flux trap will almost certainly not require variance reduction or finely resolved depletion steps. Fixed rod positions might be appropriate.

Multicycle target depletion across multiple outer VXF locations might require variance reduction with finer resolution than that available with the default settings but might best be approached with a homogenized fuel model and fewer depletion steps per calculation using fixed rod positions. Many MC histories might still be required in each depletion step to converge results.

In short, HFIRCON users should examine the problem behavior and adjust the control parameters appropriately. Do not expect HFIRCON to function well as a black box.

Do not change "fuel_cell_list" and "target_cell_list" entries from cycle to cycle in a multicycle run. The physical locations of the cells can be changed between cycles. If this occurs, LAVAMINT will correctly find a new bounding box for each cell at the beginning of the new cycle, but changing the contents of the cell lists between cycles will cause undesirable behavior.

Do not modify the contents of directories or files generated by HFIRCON. These are postprocessed after all calculations for a cycle are completed, and HFIRCON expects to find certain outputs in certain locations. Moving the contents of the directories around will likely result in unusual postprocessing or restart behavior.

HFIRCON exhibits an effective limit of ~1.5e4 depletion regions on cades hardware (two processes executing 16 threads on each 128 GB compute node). This limitation is due to additional tally and edit data structure issues in MCNP that have not been addressed in ORNL-TN. The development team hopes these issues will be resolved in new MCNP releases in the near future.

4.8 OUTPUTS

HFIRCON generates several useful outputs during and after execution. This section is subdivided into output files generated immediately after code execution within a step directory and output files generated after HFIRCON completes a cycle calculation.

This section discusses the content of the output files of interest. For more information on the structure of the cycle execution directories, see Section 5.

4.8.1 Volume Outputs

Output files of note from the volume calculations are ratio.txt and volcard.txt. The ratio.txt file is essentially an inline verification of the stochastic LAVAMINT volume calculation vs. available MCNP analytical volumes. For example, the first few lines of a ratio.txt file might look like:

```
cell number      s.volume   a.volume   ratio
86021            27.27479   27.28      0.999809017595
4104             702143.1   702478.0   0.999523259091
4105             702266.0   702478.0   0.99969821119
5701             2479.398   2480.86    0.999410688229
20501            5.066205   5.06968    0.999314552398
20502            9.589335   9.59914    0.998978554329
20503            9.789608   9.79706    0.999239363646
20504            34.16281   34.0863    1.0022445968
20505            36.3371    36.2854    1.00142481549
```

These columns contain the cell number, stochastic volume, analytical volume, and ratio of analytical to stochastic volumes. The ratio.txt file should be checked for unusual behavior if using a new or heavily modified MCNP input.

If using a HFIR deck containing the explicit core model, then the LAVAMINT stochastic volume could be very different from the MCNP analytical volume for inner and outer fuel element cells. This is because MCNP5-1.60, upon which ORNL-TN is built, generates analytical volumes for only a single instance of a repeated cell. In these cases, users will want to use the *lavamint_override* control set to "True" to generate correct fuel volumes. This does not apply to homogenized HFIR fuel inputs.

The volcard.txt file contains a correctly formatted volume card for the entire MCNP input. It is concatenated to the end of any analysis input that requires volumes (i.e., any analysis that accumulates volume-averaged tallies). The volcard.txt will use MCNP analytical volumes if available, unless the "lavamint_override" control is set. Any cell volume that cannot be generated via MCNP or LAVAMINT is set to zero. HFIRCON also assumes that users will not be depleting zones that contain a void or fill gas. Generally, users should not enter any cell containing such low-density materials into the *fuel_cell_list* or *target_cell_list* controls. If any cell in either of these lists has a total density of less than 1e-04 atoms/barn-cm (roughly twice the density of dry air), then HFIRCON will remove this cell from the list of cells for which a volume is to be calculated.

The advantg_mesh.txt file contains an ADVANTG spatial mesh constructed by using the "mesh_tol" controls, LAVAMINT bounding boxes, and the "problem_bounds" control. The resulting grid contains as many Cartesian planes as possible in the set of all the LAVAMINT bounding boxes and the problem boundaries, but still meets the minimum mesh separation tolerance. Additional planes may be added due to the maximum mesh separation tolerance; if two planes are too far apart, then another plane is added that bisects them. The resulting mesh is typically well-posed for ADVANTG-driven deterministic calculations. Most usefully, it requires almost no user effort to generate.

4.8.2 Step Outputs

Within the step directories (e.g., inside /cycle_#/step_#/), the most useful output files are located in the msx_depl_exec subdirectory. The output_rebuilt.h5 file contains the stochastically recombined output from all the independent MC executions performed in the main transport calculation within the step. It also contains the full number densities from the previously depleted step (unless on step 0).

The output_depleted.h5 file contains the depleted number densities for fuel cells and, if applicable, target cells. These are depleted to the end of the current step.

These HDF5 files may be viewed with the HDFView utility, which can be found online. Groups and datasets are logically named within the HDF5 files. For example, Figure 5 provides a screenshot of an output_rebuilt.h5 file with the target neutron flux tally data opened in the viewer:

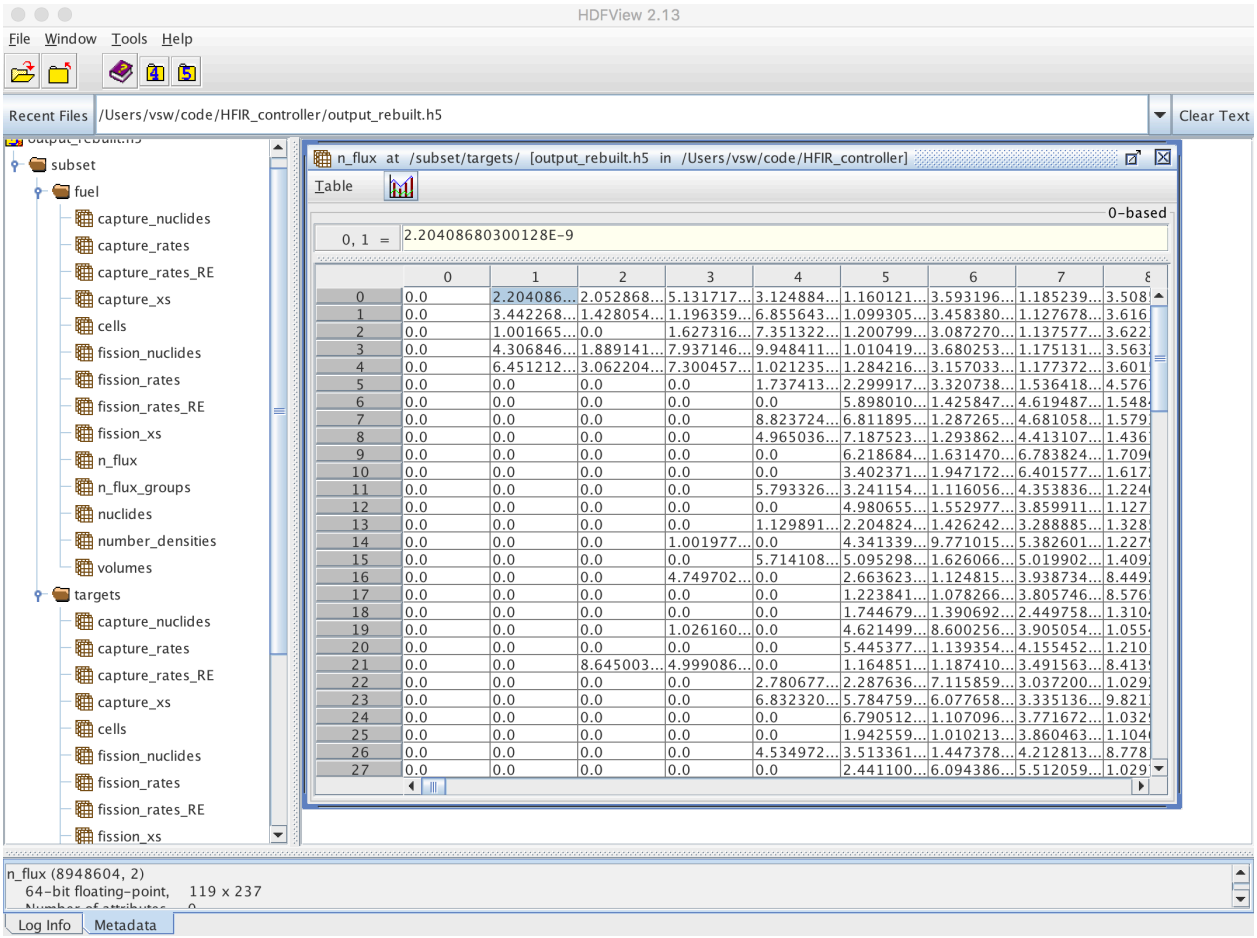

Figure 5. Screenshot of output_rebuilt.h5.

The datasets within the fuel and target groups contain all the tallied data, cell numbers, ORIGEN nuclides, number densities, and volume HFIRCON needs to pass to MSX to perform its depletion calculation.

The output_depleted.h5 file contains the output from that MSX depletion calculation, as shown in Figure 6.

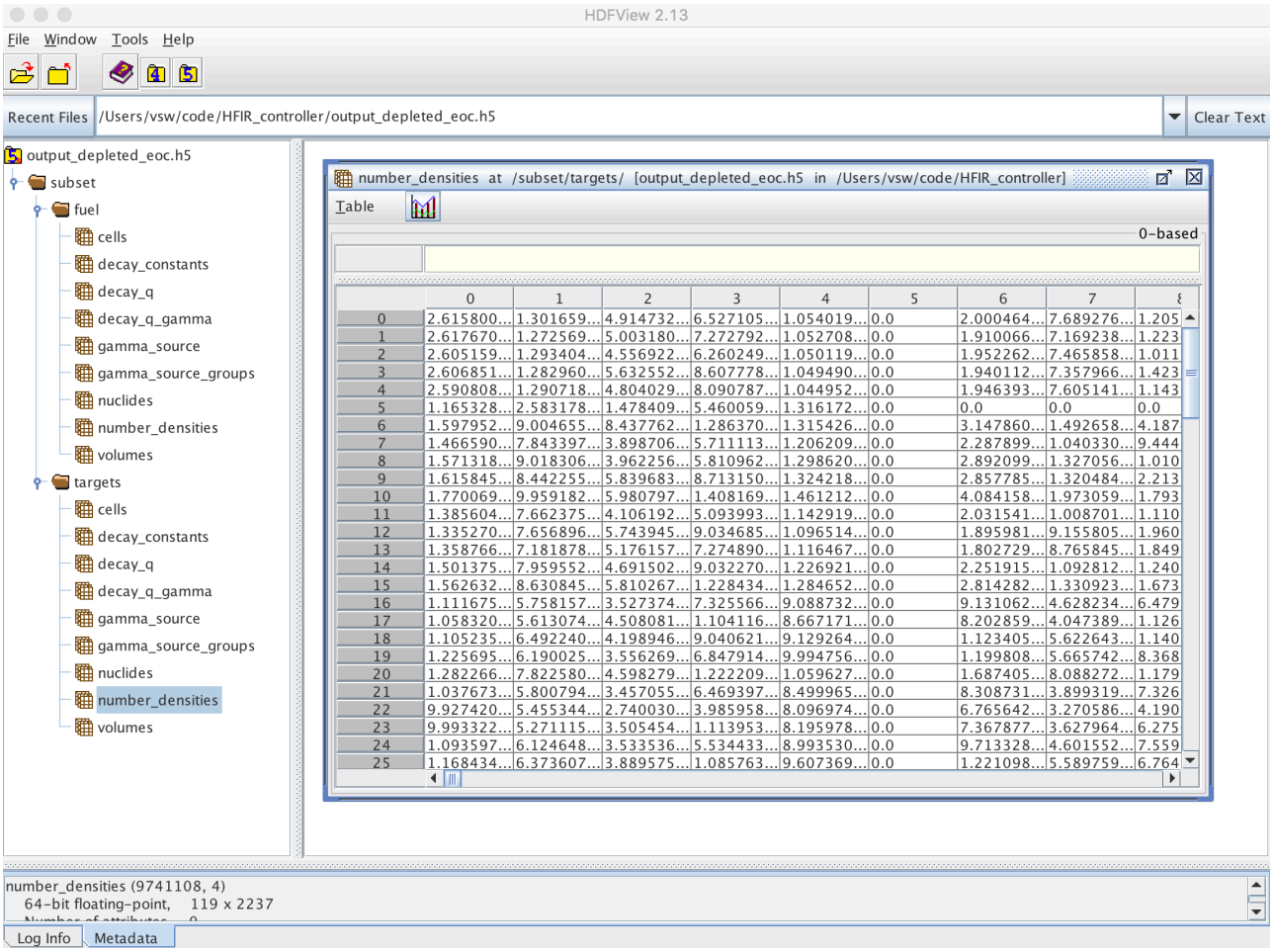

Figure 6. Screenshot of output_depleted.h5.

In **Figure 6**, the "number density" array is open in the HDFViewer, and relevant datasets are seen in the navigation pane on the left. For the "number_densities" array, there is one row of data for each cell listed in the "cells" array. Each row of data contains one column of data for each nuclide listed in the "nuclides" array. The order of the data in the "number_densities" array is consistent with the order in which the cells and nuclides are listed in the "cells" and "nuclides" arrays. Some data are replicated between the output_rebuilt.h5 and output_depleted.h5 files.

In msx_depl_exec directories past step_0, a file named output_depleted_previous.h5 is present. This is a link to the previous step's output_depleted.h5 file. It is linked to the current step to provide the starting number densities (using the full ORIGEN nuclide set) to the MSX depletion calculation. The transport nuclide set (i.e., ENDF) is only used for depletion at time step 0.

Other .h5 files of note are the decay gamma output_rebuilt.h5 and the after-shutdown output_decayed.h5 files.

The EOC decay gamma transport calculation occurs at the end of the final depletion time step for a given cycle. The step_#/decay_gamma_exec/decay_gamma_output/output_rebuilt.h5 file contains the rebuilt tally data from independent gamma transport calculations executed in the step_#/decay_gamma_exec/ directory. Its structure is very similar to the more general output_rebuilt.h5 located in msx_depl_exec. If the user requests additional times after EOC for decay using the "eoc_decay_steps" control, then extra directories immediately below the cycle_#/ directory will appear. They are named "decay_step_#" and contain an output file named "output_decayed.h5." Its format is very similarly to that of output_depleted.h5; it has number densities decayed for a user-defined number of days past EOC.

The user generally does not need to worry about parsing the individual .h5 files located in the msx_depl_exec directories inside the step_# execution directories unless an edit beyond the current capabilities of HFIRCON's automated output parser is required. However, all depletion data are contained within a HFIRCON cycle execution directory inside these .h5 files. No tally or depletion data are deleted during execution.

4.8.3 Cycle Outputs

HFIRCON attempts to accumulate useful information from the step-dependent HDF5 files. The ASCII output files created by HFRICON are a function of the execution mode, the *analysis_name* control and, if present, the *eoc_decay_times* and *mcnp_file_steps* controls.

Table I. Conditions for triggering HFIRCON output files. lists all the ASCII output files that might be generated and the conditions that would trigger their generation. In this table, the *analysis_name* control has been set to "Name" and the cycle 0 calculations have just been completed.

Table I. Conditions for triggering HFIRCON output files.

Example File Name	Conditions
Name_cycle_0_ce_positions.txt	fuel_depletion or target_depletion
Name_cycle_0_EOC_target_g_heating.txt	target_depletion
Name_cycle_0_fuel_depletion.txt	fuel_depletion or target_depletion
Name_cycle_0_fuel_fission_rates.txt	fuel_depletion or target_depletion
Name_cycle_0_fuel_poisons_actinides.txt	fuel_depletion or target_depletion
Name_cycle_0_group_activity_AS_#.txt	target_depletion AND *eoc_decay_times*. # will range from 0 to NT-1, where NT is the total number of entries in the *eoc_decay_times* control.
Name_cycle_0_group_activity_EOC.txt	target_depletion
Name_cycle_0_step_#_MCNP.inp	Fuel_depletion or target_depletion AND *mcnp_file_steps*. A separate step specific MCNP file, with the "#" in the file name replaced by a unique entry from the *mcnp_file_steps* control, will be created for each entry in this control.
Name_cycle_0_target_activity_AS.txt	target_depletion AND *eoc_decay_times*. Results for all times in the *eoc_decay_times* control are included in this single file.
Name_cycle_0_target_activity.txt	target_depletion
Name_cycle_0_target_decay_gamma_power_AS.txt	target_depletion AND *eoc_decay_times*. Results for all times in the *eoc_decay_times* control are included in this single file.
Name_cycle_0_target_decay_gamma_power.txt	target_depletion
Name_cycle_0_target_decay_power_AS.txt	target_depletion AND *eoc_decay_times*. Results for all times in the *eoc_decay_times* control are included in this single file.
Name_cycle_0_target_decay_power.txt	target_depletion
Name_cycle_0_target_fission_rates.txt	target_depletion
Name_cycle_0_target_g_heating.txt	target_depletion
Name_cycle_0_target_helium_moles_AS.txt	target_depletion AND *eoc_decay_times*. Results for all times in the *eoc_decay_times* control are included in this single file.
Name_cycle_0_target_helium_moles.txt	target_depletion
Name_cycle_0_target_inventory_masses_AS.txt	target_depletion AND *eoc_decay_times*. Results for all times in the *eoc_decay_times* control are included in this single file.
Name_cycle_0_target_inventory_masses.txt	target_depletion
Name_cycle_0_target_krypton_moles_AS.txt	target_depletion AND *eoc_decay_times*. Results for all times in the *eoc_decay_times* control are included in this single file.
Name_cycle_0_target_krypton_moles.txt	target_depletion
Name_cycle_0_target_n_heating.txt	target_depletion
Name_cycle_0_target_xenon_moles_AS.txt	target_depletion AND *eoc_decay_times*. Results for all times in the *eoc_decay_times* control are included in this single file.
Name_cycle_0_target_xenon_moles.txt	target_depletion

The contents of each of these ASCII output files are as follows.

Name_cycle_0_ce_positions.txt: Contains the CE positions by time step.

Name_cycle_0_EOC_target_g_heating.txt: Contains the EOC target prompt gamma heating by cell number.

Name_cycle_0_fuel_depletion.txt: Contains the EOC ^{235}U depletion distribution (%) by radial and axial fuel location according to the "fuel_output_map" control.

Name_cycle_0_fuel_fission_rates.txt: Contains the fission rate distribution by time step and radial and axial fuel location according to the "fuel_output_map" control.

Name_cycle_0_fuel_poisons_actinides.txt: Contains the core poison and actinide inventory (g) by time step, "fuel_output_map" group, and nuclide of interest. As of HFIRCON v1.0.5, the nuclides of interest that are reported in this file are ^{10}B, ^{135}Xe, ^{149}Sm, $^{234\text{-}239}$U, and $^{238–242}$Pu.

Name_cycle_0_group_activity_AS_0.txt: Contains the activity (Ci) by "target_output_groups" and by ORIGEN nuclide at the first "after shutdown" decay step. *Name_cycle_0_group_activity_AS_1.txt* would correspond to the second after-shutdown decay step, and so on. The same naming convention holds for other files written for after-shutdown quantities.

Name_cycle_0_group_activity_EOC.txt: Contains the activity (Ci) by "target_output_groups" and by ORIGEN nuclide at EOC.

Name_cycle_0_step_#_MCNP.inp: Contains a step specific MCNP input file with the total number density on all cell definition cards for cells in the *fuel_cell_list* and *target_cell_list* controls updated to be consistent with the values at the end of depletion step #. The corresponding material definition cards for all materials contained within these cells are also updated to include every isotope existing at a concentration of at least 1e-10 atoms/barn-cm at the end of depletion step #, regardless of whether or not that isotope existed in the original step 0 material definition.

Name_cycle_0_target_activity_AS.txt: Contains the activity (Ci) by target cell and time step after shutdown. Only contains quantities decayed at zero power after EOC.

Name_cycle_0_target_activity.txt: Contains the activity (Ci) by target cell and depletion time steps. Only contains quantities calculated during the cycle depletion.

Name_cycle_0_target_decay_gamma_power_AS.txt: Contains the heat generation rate (W) in all target cells for decay gammas only at time steps after shutdown. Only contains quantities decayed at zero power after EOC.

Name_cycle_0_target_decay_gamma_power.txt: Contains the heat generation rate (W) in all target cells for decay gammas only at depletion time steps. Only contains quantities calculated during the cycle depletion.

Name_cycle_0_target_decay_power_AS.txt: Contains decay heat generation rates (W) in all target cells for all sources (gamma, alpha, and beta) at time steps after shutdown. Only contains quantities decayed at zero power after EOC.

Name_cycle_0_target_decay_power.txt: Contains decay heat generation rates (W) in all target cells for all sources (gamma, alpha, and beta) at depletion time steps. Only contains quantities calculated during the cycle depletion.

Name_cycle_0_target_fission_rates.txt: Contains fission reaction rates in all target cells at depletion time steps.

Name_cycle_0_target_g_heating.txt: Contains prompt gamma heat generation rates (W/g) by target cell at all depletion time steps.

Name_cycle_0_target_helium_moles_AS.txt: Contains moles of He in all target cells at time steps after shutdown. Each time step contains four columns corresponding to a He isotope in the ORIGEN library (2003, 2004, 2005, and 2006).

Name_cycle_0_target_helium_moles.txt: Contains moles of He in all target cells at depletion time steps. Each time step contains four columns corresponding to a He isotope in the ORIGEN library (2003, 2004, 2005, and 2006).

Name_cycle_0_target_inventory_masses_AS.txt: Contains the isotopic inventory (g) by target cell number and user-defined isotopes for every time step after shutdown. If no user-defined list of isotopes of interest is defined via the *inventory_isos* control, isotopic inventories for ^{237,238}Np, ^{236}Pu, and $^{238–242}$Pu will be provided.

Name_cycle_0_target_inventory_masses.txt: Contains the isotopic inventory (g) by target cell number and user-defined isotopes for every depletion time step. If no user-defined list of isotopes of interest is defined via the *inventory_isos* control, isotopic inventories for ^{237,238}Np, ^{236}Pu, and $^{238–242}$Pu will be provided.

Name_cycle_0_target_krypton_moles_AS.txt: Contains moles of Kr in all target cells at time steps after shutdown. Each time step contains several columns corresponding to a Kr isotope in the ORIGEN library (36076–36100).

Name_cycle_0_target_krypton_moles.txt: Contains moles of Kr in all target cells at depletion time steps. Each time step contains several columns corresponding to a Kr isotope in the ORIGEN library (36076–36100).

Name_cycle_0_target_n_heating.txt: Contains prompt neutron heat generation rates (W/g) by target cell at all depletion time steps.

Name_cycle_0_target_xenon_moles_AS.txt: Contains moles of Xe in all target cells at time steps after shutdown. Each time step contains several columns corresponding to a Xe isotope in the ORIGEN library (54122–54147).

Name_cycle_0_target_xenon_moles.txt: Contains moles of Xe in all target cells at depletion time steps. Each time step contains several columns corresponding to a Xe isotope in the ORIGEN library (54122–54147).

5. CYCLE DIRECTORY STRUCTURE

An example of a cycle directory listing in shown in **Figure 7**.

```
[vsw@mod-condo-c153 cycle_0]$ ls -lh
total 9.6M
-rwxr-xr-x   1 vsw cades-nsed-mcnp6  485 Mar 19 19:04 config_noruntpe
-rwxr-xr-x   1 vsw cades-nsed-mcnp6  485 Mar 19 19:04 config_runtpe
drwxr-sr-x   2 vsw cades-nsed-mcnp6  33K May 16 06:17 decay_step_0
-rwxr-xr-x   1 vsw cades-nsed-mcnp6 3.3M Mar 19 19:04 homheu.rev17.repre.v1.inp.HFIRCON
-rwxr-xr-x   1 vsw cades-nsed-mcnp6   94 Mar 19 19:04 sdef_card_homheu_rev17
drwxr-sr-x 167 vsw cades-nsed-mcnp6  57K Mar 19 20:13 step_0
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 19 21:03 step_1
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 05:08 step_10
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 06:00 step_11
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 06:53 step_12
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 07:45 step_13
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 08:37 step_14
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 09:30 step_15
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 10:23 step_16
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 11:16 step_17
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 12:09 step_18
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 13:02 step_19
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 19 21:53 step_2
drwxr-sr-x 167 vsw cades-nsed-mcnp6  57K Mar 20 14:22 step_20
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 15:16 step_21
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 16:10 step_22
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 17:03 step_23
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 17:57 step_24
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 18:51 step_25
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 19:45 step_26
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 20:39 step_27
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 21:34 step_28
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 22:28 step_29
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 19 22:44 step_3
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 23:22 step_30
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 21 00:16 step_31
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 21 01:11 step_32
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 21 02:06 step_33
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 21 03:00 step_34
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 21 03:55 step_35
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 21 04:49 step_36
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 21 05:44 step_37
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 21 06:38 step_38
drwxr-sr-x 167 vsw cades-nsed-mcnp6  57K May 16 01:40 step_39
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 19 23:35 step_4
drwxr-sr-x 167 vsw cades-nsed-mcnp6  57K Mar 20 00:51 step_5
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 01:42 step_6
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 02:33 step_7
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 03:25 step_8
drwxr-sr-x 166 vsw cades-nsed-mcnp6  57K Mar 20 04:17 step_9
drwxr-sr-x   2 vsw cades-nsed-mcnp6 281K May 16 05:29 volume
```

Figure 7. Cycle execution directory listing.

The config files contain ORNL-TN configuration data suitable for the various modes in which it is executed, either writing a runtpe or skipping the runtpe write. The day0_fuel_ltwtr input is the template MCNP input deck, which is copied from the input directory (*input_location* control) to the cycle directory (as is the sdef_card optionally provided by the user). The volume directory contains the output of the stochastic volume calculation requested by the user. The step_# directories each contain the complete output of the individual depletion step calculations. The decay_step_0 directory in this case contains the output of a single after-shutdown decay calculation.

5.1 VOLUME DIRECTORY CONENTS

The content of the aforementioned volume directory should appear as shown in **Figure 8**.

```
[vsw@mod-condo-login02 volume]$ ls -lh
total 2.5G
-rw-r--r-- 1 vsw cades-nsed-mcnp6 8.7K Apr 26 17:09 advantg_mesh.txt
-rw-r--r-- 1 vsw cades-nsed-mcnp6  195 Apr 26 17:02 complements.1001.out
-rw-r--r-- 1 vsw cades-nsed-mcnp6  197 Apr 26 17:02 complements.1002.out
-rw-r--r-- 1 vsw cades-nsed-mcnp6  194 Apr 26 17:03 complements.1211.out
-rw-r--r-- 1 vsw cades-nsed-mcnp6  194 Apr 26 17:03 complements.1212.out
-rw-r--r-- 1 vsw cades-nsed-mcnp6  194 Apr 26 17:04 complements.1213.out
-rw-r--r-- 1 vsw cades-nsed-mcnp6  194 Apr 26 17:03 complements.1214.out
-rw-r--r-- 1 vsw cades-nsed-mcnp6  195 Apr 26 17:05 complements.1280.out
                                    ...
-rw-r--r-- 1 vsw cades-nsed-mcnp6  194 Apr 26 17:02 complements.954.out
-rw-r--r-- 1 vsw cades-nsed-mcnp6  196 Apr 26 17:02 complements.955.out
-rw-r--r-- 1 vsw cades-nsed-mcnp6  196 Apr 26 17:04 complements.9997.out
-rwxr-xr-x 1 vsw cades-nsed-mcnp6  431 Apr 26 17:01 config
-rwxr-xr-x 1 vsw cades-nsed-mcnp6  54K Apr 26 17:01 controller_input.json
-rw-r--r-- 1 vsw cades-nsed-mcnp6   69 Apr 26 17:01 detcomp.inp
-rw-r--r-- 1 vsw cades-nsed-mcnp6 281K Apr 26 17:01 initial_locations.out
-rwxr-xr-x 1 vsw cades-nsed-mcnp6 1.1M Apr 26 17:01 input
-rw-r--r-- 1 vsw cades-nsed-mcnp6 157K Apr 26 17:09 list.txt
-rw-r--r-- 1 vsw cades-nsed-mcnp6  39K Apr 26 17:01 output.h5
-rw-r--r-- 1 vsw cades-nsed-mcnp6  49K Apr 26 17:09 ratio.txt
-rw-r--r-- 1 vsw cades-nsed-mcnp6 256M Apr 26 17:01 runtpe
-rw-r--r-- 1 vsw cades-nsed-mcnp6 2.9M Apr 26 17:01 trash
-rw-r--r-- 1 vsw cades-nsed-mcnp6  34K Apr 26 17:09 volcard.txt
```

Figure 8. Volume calculation directory listing.

The complements.* files contain the cell-dependent output of the parallel stochastic volume calculation. They are likely uninteresting to users unless users want to inspect the standard deviation of the cell-dependent volume calculation or look at the bounding boxes calculated for each cell.

Those complements.* files are parsed to produce the following useful output. As discussed in Section 4.8.1, the ratio.txt file contains the ratio of the stochastic volume calculation to the analytic volume calculated by MCNP for every cell in which this is possible. It serves as a good estimate of the accuracy of the stochastic volume calculation. The volcard.txt file is a correctly formatted VOL card containing the volumes for every cell in the problem. The stochastic volumes are only used for cells where necessary; if MCNP can calculate a cell volume analytically, then the volume card uses it. Finally, the advantg_mesh.txt file contains a Cartesian grid definition in ADVANTG format that is generated from the cell-dependent bounding boxes produced by the stochastic volume calculation. This advantg_mesh.txt file is used for the ADVANTG mesh definition if it exists. Other files do not contain interesting output.

5.2 STEP DIRECTORY STRUCTURE AND CONTENTS

A complete listing of a depletion step execution directory should appear as shown in **Figure 9**.

```
advantg.inp        mc_exec_28  mc_exec_54  mc_exec_9             rod_search_exec_1_30  rod_search_exec_1_57
advantg.mcnp       mc_exec_29  mc_exec_55  mc_outputs            rod_search_exec_1_31  rod_search_exec_1_58
advantg_tallies    mc_exec_3   mc_exec_56  msx_depl_exec         rod_search_exec_1_32  rod_search_exec_1_59
decay_gamma_exec   mc_exec_30  mc_exec_57  rod_position.txt      rod_search_exec_1_33  rod_search_exec_1_6
launch_mc.bash     mc_exec_31  mc_exec_58  rod_search_decks_1    rod_search_exec_1_34  rod_search_exec_1_60
launch_rs_1.bash   mc_exec_32  mc_exec_59  rod_search_exec_1_0   rod_search_exec_1_35  rod_search_exec_1_61
machinefile        mc_exec_33  mc_exec_6   rod_search_exec_1_1   rod_search_exec_1_36  rod_search_exec_1_62
mc_decks           mc_exec_34  mc_exec_60  rod_search_exec_1_10  rod_search_exec_1_37  rod_search_exec_1_63
mc_exec_0          mc_exec_35  mc_exec_61  rod_search_exec_1_11  rod_search_exec_1_38  rod_search_exec_1_64
mc_exec_1          mc_exec_36  mc_exec_62  rod_search_exec_1_12  rod_search_exec_1_39  rod_search_exec_1_65
mc_exec_10         mc_exec_37  mc_exec_63  rod_search_exec_1_13  rod_search_exec_1_4   rod_search_exec_1_66
mc_exec_11         mc_exec_38  mc_exec_64  rod_search_exec_1_14  rod_search_exec_1_40  rod_search_exec_1_67
mc_exec_12         mc_exec_39  mc_exec_65  rod_search_exec_1_15  rod_search_exec_1_41  rod_search_exec_1_68
mc_exec_13         mc_exec_4   mc_exec_66  rod_search_exec_1_16  rod_search_exec_1_42  rod_search_exec_1_69
mc_exec_14         mc_exec_40  mc_exec_67  rod_search_exec_1_17  rod_search_exec_1_43  rod_search_exec_1_7
mc_exec_15         mc_exec_41  mc_exec_68  rod_search_exec_1_18  rod_search_exec_1_44  rod_search_exec_1_70
mc_exec_16         mc_exec_42  mc_exec_69  rod_search_exec_1_19  rod_search_exec_1_45  rod_search_exec_1_71
mc_exec_17         mc_exec_43  mc_exec_7   rod_search_exec_1_2   rod_search_exec_1_46  rod_search_exec_1_72
mc_exec_18         mc_exec_44  mc_exec_70  rod_search_exec_1_20  rod_search_exec_1_47  rod_search_exec_1_73
mc_exec_19         mc_exec_45  mc_exec_71  rod_search_exec_1_21  rod_search_exec_1_48  rod_search_exec_1_74
mc_exec_2          mc_exec_46  mc_exec_72  rod_search_exec_1_22  rod_search_exec_1_49  rod_search_exec_1_75
mc_exec_20         mc_exec_47  mc_exec_73  rod_search_exec_1_23  rod_search_exec_1_5   rod_search_exec_1_76
mc_exec_21         mc_exec_48  mc_exec_74  rod_search_exec_1_24  rod_search_exec_1_50  rod_search_exec_1_77
mc_exec_22         mc_exec_49  mc_exec_75  rod_search_exec_1_25  rod_search_exec_1_51  rod_search_exec_1_78
mc_exec_23         mc_exec_5   mc_exec_76  rod_search_exec_1_26  rod_search_exec_1_52  rod_search_exec_1_79
mc_exec_24         mc_exec_50  mc_exec_77  rod_search_exec_1_27  rod_search_exec_1_53  rod_search_exec_1_8
mc_exec_25         mc_exec_51  mc_exec_78  rod_search_exec_1_28  rod_search_exec_1_54  rod_search_exec_1_9
mc_exec_26         mc_exec_52  mc_exec_79  rod_search_exec_1_29  rod_search_exec_1_55  sdef_card_ADVANTG
mc_exec_27         mc_exec_53  mc_exec_8   rod_search_exec_1_3   rod_search_exec_1_56  srctp_converged
```

Figure 9. Step execution directory listing.

Almost all the directories and files in these step_ directories are named descriptively. For example, the MCNP input decks for the rod search in this directory are contained in rod_search_decks_1. The execution directories for the rod search are named rod_search_exec_#, and so on.

Two things are of special interest in this directory. First, the rod_position.txt file contains the final calculated critical rod position as interpolated from a spread of rod search executions. If a fixed_rod_positions entry is provided, then it will contain the user-defined position with an interpolated k_{eff} value.

Second, the mc_outputs directory contains all the output files of the main depletion transport solution. Writing and rebuilding a mctal file is optional, and the final MCTALMRG data will not be parsed by the output extractor. A listing of a complete mc_outputs directory is shown in **Figure 10**.

```
out_0   out_38  out_67        output_24.h5  output_53.h5
out_1   out_39  out_68        output_25.h5  output_54.h5
out_10  out_4   out_69        output_26.h5  output_55.h5
out_11  out_40  out_7         output_27.h5  output_56.h5
out_12  out_41  out_70        output_28.h5  output_57.h5
out_13  out_42  out_71        output_29.h5  output_58.h5
out_14  out_43  out_72        output_2.h5   output_59.h5
out_15  out_44  out_73        output_30.h5  output_5.h5
out_16  out_45  out_74        output_31.h5  output_60.h5
out_17  out_46  out_75        output_32.h5  output_61.h5
out_18  out_47  out_76        output_33.h5  output_62.h5
out_19  out_48  out_77        output_34.h5  output_63.h5
out_2   out_49  out_78        output_35.h5  output_64.h5
out_20  out_5   out_79        output_36.h5  output_65.h5
out_21  out_50  out_8         output_37.h5  output_66.h5
out_22  out_51  out_9         output_38.h5  output_67.h5
out_23  out_52  output_0.h5   output_39.h5  output_68.h5
out_24  out_53  output_10.h5  output_3.h5   output_69.h5
out_25  out_54  output_11.h5  output_40.h5  output_6.h5
out_26  out_55  output_12.h5  output_41.h5  output_70.h5
out_27  out_56  output_13.h5  output_42.h5  output_71.h5
out_28  out_57  output_14.h5  output_43.h5  output_72.h5
out_29  out_58  output_15.h5  output_44.h5  output_73.h5
out_3   out_59  output_16.h5  output_45.h5  output_74.h5
out_30  out_6   output_17.h5  output_46.h5  output_75.h5
out_31  out_60  output_18.h5  output_47.h5  output_76.h5
out_32  out_61  output_19.h5  output_48.h5  output_77.h5
out_33  out_62  output_1.h5   output_49.h5  output_78.h5
out_34  out_63  output_20.h5  output_4.h5   output_79.h5
out_35  out_64  output_21.h5  output_50.h5  output_7.h5
out_36  out_65  output_22.h5  output_51.h5  output_8.h5
out_37  out_66  output_23.h5  output_52.h5  output_9.h5
```

Figure 10. mc_outputs directory listing.

This HFIRCON execution was run with 40 nodes, resulting in 80 mc_exec simulations.

msx_depl_exec is the execution directory for msx_deplete. In it are the rebuilt .h5 outputs from all the separate mc_exec simulations (output_rebuilt.h5), as well as the depleted output from msx_deplete (output_depleted.h5). This file contains the number densities for all depletable regions defined in the problem. A listing of a complete msx_depl_exec directory is shown in **Figure 11**.

```
input.msx     output_20.h5  output_34.h5  output_48.h5  output_61.h5  output_75.h5
machinefile   output_21.h5  output_35.h5  output_49.h5  output_62.h5  output_76.h5
nperfis.txt   output_22.h5  output_36.h5  output_4.h5   output_63.h5  output_77.h5
output_0.h5   output_23.h5  output_37.h5  output_50.h5  output_64.h5  output_78.h5
output_10.h5  output_24.h5  output_38.h5  output_51.h5  output_65.h5  output_79.h5
output_11.h5  output_25.h5  output_39.h5  output_52.h5  output_66.h5  output_7.h5
output_12.h5  output_26.h5  output_3.h5   output_53.h5  output_67.h5  output_8.h5
output_13.h5  output_27.h5  output_40.h5  output_54.h5  output_68.h5  output_9.h5
output_14.h5  output_28.h5  output_41.h5  output_55.h5  output_69.h5  output_depleted.h5
output_15.h5  output_29.h5  output_42.h5  output_56.h5  output_6.h5   output_depleted_previous.h5
output_16.h5  output_2.h5   output_43.h5  output_57.h5  output_70.h5  output_rebuilt.h5
output_17.h5  output_30.h5  output_44.h5  output_58.h5  output_71.h5  source_ascii.txt
output_18.h5  output_31.h5  output_45.h5  output_59.h5  output_72.h5  source_binary
output_19.h5  output_32.h5  output_46.h5  output_5.h5   output_73.h5  total_decay_gamma_source.txt
output_1.h5   output_33.h5  output_47.h5  output_60.h5  output_74.h5
```

Figure 11. msx_depl_exec directory listing.

The nperfis.txt contains the average tallied neutrons per fission from every ORNL-TN execution. This is used in the final tally normalization.

5.3 DECAY STEP DIRECTORY STRUCTURE

Finally, the decay_step_0 directory in the cycle execution directory contains only what is necessary to decay the EOC number densities to the first time specified in the “eoc_decay_times” control. Figure 12 provides an example.

```
input.msx           output_depleted.h5
machinefile         output_rebuilt.h5
output_decayed.h5
```

Figure 12. decay_step directory listing.

The output_decayed.h5 file is the item of interest in this directory; its contents are discussed in Section 4.8.2.

6. NUCLEAR DATA

The location of the nuclear data used with HFIRCON is specified during code installation by the installer. HFIRCON is specifically designed to use ENDF/B-VII.0 and ENDF/B-VII.1 cross sections (e.g., .7xc and .8xc zaid extensions). A hybrid library based on ENDF/B-VII.1 but supplemented with data from the JEFF3.1.2, JENDL4.0u, CENDL3.1, and TENDL-2013 data repositories as necessary to ensure that all isotopes contain reasonable gamma production data (GPD) [11]. HFIRCON will not allow any nuclear data other than those from ENDF/B-VII.0, ENDF/B-VII.1, or GPD to be used for transport.

At ORNL, HFIRCON nuclear data are located at:

```
/software/user_tools/current/cades-nsed-hfircon/DATA/ on cades;
/projects/hfircon/DATA/ on apollo, romulus, remus, and lise.
```

For ENDF/B-VII.1 data, nuclides that were erroneously distributed (e.g., H-1 without capture gamma yields) cannot be referenced in error. The corrected data libraries (e.g., 1001.90c) are referenced by the original zaid extension (e.g., 1001.80c actually points to the 1001.90c data).

For transmutation and decay calculations in ORIGEN via the MSX utility, all data are production SCALE data. Complete SCALE data are included in the HFIRCON installation. The SCALE data are located at:

```
/software/user_tools/current/cades-nsed-hfircon/PROD/V1.0.5/data/scale/ on
cades;
/projects/hfircon/PROD/V1.0.5/data/scale/ on apollo, romulus, remus, and
lise.
```

Multigroup cross section libraries for ADVANTG (used for variance reduction only) are also included in the HFIRCON installation. On cades, the ADVANTG data are located at:

```
/software/user_tools/current/cades-nsed-hfircon/PROD/V1.0.5/data/advantg/ on
cades;
/projects/hfircon/PROD/V1.0.5/data/advantg/ on apollo, romulus, remus, and
lise.
```

All the standard ADVANTG multigroup libraries are included, but HFIRCON automatically selects the 27n19g_gpd library. This library is similar to the 27n19g library, but contains gamma production for all isotopes in the library. Currently, users cannot change this selection.

7. VERIFICATION AND VALIDATION TEST PROBLEMS

The HFIRCON code package was developed to seamlessly integrate multiple commercial off-the-shelf (COTS) nuclear analysis tools (e.g., ADVANTG) or local modifications thereof, such as ORNL-TN, which provides local enhancements to MCNP5-1.60, and the ORIGEN C++ library, which is compiled from the SCALE6.2.3 source code with the API option enabled. It also automatically generates several tabular, human-readable output files containing results (e.g., prompt and delayed heating rates, activation product concentrations, fission densities) that are common to a wide variety of HFIR safety-basis neutronics calculations and provides binary HDF5-formatted files with additional computed values (e.g., cell-based flux, spectrum, volume, mass) from which other design products can be extracted or generated on an as-needed, case-specific basis.

The COTS components, all HFIRCON postprocessing and "glue" code, and all required code dependencies (e.g., OpenMP, MPI, Python) are compiled into static libraries. Creating a static, "virtual" environment is typically not necessary on computing platforms for which a stable operating environment is maintained in support of safety-basis calculations (e.g., the RRD computing cluster, lise). However, on research-oriented computing clusters where the default behavior is to upgrade to the latest and greatest tools with some regularity (e.g., the cades cluster), the creation of a static, "virtual" environment is essential to the performance of safety-basis calculations.

The HFIRCON verification used three separate strategies. First, when available, the developer test suites were run in their entirety and results were compared with those provided by the developer. The second set of verification test problems consists of specific models that were shown to produce undesirable outcomes with previous versions of HFIRCON. Once a new version of the code is developed to resolve the issue(s), the model(s) are added to the verification test suite to confirm the correct behavior of the current version and any future versions of HFIRCON. Finally, a third set of problems was created to further test the functionality of the LAVAMINT utility.

Developer test suites are available for ADVANTG and MCNP5-1.60. All the jobs in these developer test suites were executed with the HFIRCON versions of either ADVANTG or ORNL-TN in stand-alone mode. Results from all these author-run cases compared favorably with the developer-supplied results and provide verification of proper installation of these HFIRCON modules; the results are summarized as follows.

The ADVANTG test suite consists of 11 separate cases covering variations of the three models described in Mosher et al. [9]. These jobs were run with the version of ADVANTG (v3.1) embedded in the HFIRCON package. The author-generated and developer-supplied results files were compared with the Linux "diff" utility. For the developer test suite jobs, ADVANTG produces 11 small (typically a few kilobytes) "inp" files, each containing a set of source biasing parameters for use in a subsequent MCNP job. All 11 of the author-generated "inp" files contained results identical to those supplied by the developer. For the developer test suite jobs, ADVANTG produces 11 large (typically a few dozen megabytes to several hundred megabytes) "wwinp" files, each containing a set of weight windows for use in a subsequent MCNP job. Ten of the 11 of the author-generated "wwinp" files contained results identical to those supplied by the developer. One of the 11 author-generated wwinp files contained a single weight window value (in a 614 MB file) that differed from the developer-supplied value in the last printed digit (i.e., 8.94286e-17 vice 8.94285e-17).

ORNL-TN is a modification of the MCNP5-1.60 code package. The verification and validation package (V&V) supplied with MCNP5-1.60 was modified to use the ORNL-TN executable embedded in HFIRCON. The MCNP5-1.60 V&V package consists of six test suites: Regression,

VALIDATION_CRITICALITY, VERIFICATION_KEFF, VALIDATION_SHIELDING, KOBAYASHI, and POINT_KINETICS. Each of these V&V test suites contains anywhere from 6 to well over 100 individual model inputs. The test suites and the individual problems within each suite are described in detail in Brown et al. [12]. Each of the six test suites was run with the HFIRCON version of ORNL-TN and produced results that were identical to or within the statistical uncertainty of the developer-supplied MCNP5-1.60 results.

The MCNP5 Regression test suite consists of 61 different jobs, each producing some or all of the following file types: Output, MCTAL, WWOUT, PTRAC, and MESH Tally. All 61 cases produced identical results between the developer-provided MCTAL, WWOUT, PTRAC, and MESH Tally files and the corresponding author-generated files. Fifty-nine of the 61 cases produced identical results between the developer-provided and author-generated Output files, whereas two of the 61 Output files each contained a single number that differed due to roundoff error.

The developer-supplied MCNP5 VALIDATION_CRITICALITY test suite consists of calculated and measured results for 31 criticality experiments. The developer-supplied calculational results were generated with different input nuclear data from the author-generated results and a code executable that was compiled with a variety of compilers, none of which were the same as those used to compile HFIRCON. Despite these differences, the author-generated calculational results for 26 of the 31 cases matched the developer-supplied results identically. For the remaining five cases, the author-generated calculational results were within the calculational uncertainty of the developer-supplied results for one of the columns of results presented in Table IX of Brown et al. [12]. This table presents results that were calculated with either Intel-10 or pgi-7/9 compiled versions of MCNP5-1.60. The five cases in which the author-generated results and the developer-supplied results are also the same five cases in which the developer-supplied results differ between the two MCNP5-1.60 executables that were compiled with different compilers by the developer.

The developer-supplied MCNP5 VERIFICATION_KEFF test suite contains 75 analytic benchmark models. Results for only 10 of these models were documented in Table X of Brown et al. [12]. For these 10 models, the developer-supplied and author-generated results matched identically.

The developer-supplied VALIDATION_SHIELDING test suite consists of eight pulsed sphere models, five fusion shielding models, and six photon shielding models. These 19 models produce 95 different output files. For each of these 95 files, the developer-supplied and author-generate files are compared with the Linux “diff” utility. For 68 of the 95 files, the results are identical between the developer-supplied and author-generated files. For the remaining 27 files, the differences between the developer-supplied and author-generated files can be attributed to the fact that the developer-supplied results were run in multi-threaded mode, whereas the author-generated results were all run with a single thread. This will cause some blocks of lines in the various developer-supplied output files to be printed in a different order than the corresponding blocks of lines in the author-generated output files. Additionally, some results will be slightly different due to roundoff errors.

The developer-supplied KOBAYASHI test suite consists of three separate models, each with a streaming path (void region) in a solid material. For each of these three models, two separate cases are run. In one case the solid material is a pure absorber and in the other case, the solid material consists of a 50/50 mix of a pure absorber and pure scatterer. Results are tallied at up to 22 different point detectors in each model. For all tally locations in all models, regardless of the composition of the nonvoid material, the results were identical between the author-generated results and the developer-supplied results presented in Tables XIa, XIc, and XId of Brown et al. [12].

The developer-supplied POINT_KINETICS test suite consists of 16 criticality models for which neutron generation times and delayed neutron precursor parameters are calculated. Of the 16 calculated values presented in Table XII of Brown et al. [12], the author-generated and developer-supplied results are identical for six, are within one standard deviation of each other for another three, are between one and two standard deviations for another four, are between two and three standard deviations for another two, and are different by more than three standard deviations for two cases. For the two cases that differed by more than three standard deviations, the calculated relative error was significant (up to ~44%), indicating a lack of convergence in both the developer-supplied and author-generated calculational results. The developer-supplied results from Brown et al. [12] were generated with a newer version of MCNP5 (v1.61) than the version of MCNP5 (v1.60) upon which ORNL-TN is based.

Based on the good agreement between the large number and wide variety of calculation results calculated with the HFIRCON ORNL-TN module and the developer-supplied MCNP5 code, proper installation of the ORNL-TN module in the HFIRCON package is considered to be verified.

The HFIR-specific verification test suit currently consists of two models. The first contains an irradiation experiment consisting of several very thin foils of Ni enriched in the isotope ^{62}Ni and wrapped in a very thin Al sheath. The second contains an irradiation experiment consisting of radium carbonate, which is used in the production of ^{229}Th.

The ^{62}Ni model exposed the potential for an infinite loop to occur in the volume calculation phase of HFIRCON. A patch to prevent this behavior was developed and implemented in HFIRCON v1.0.5. The ^{62}Ni verification model is run in volume calculation mode to ensure that the small cell infinite loop patch functions properly.

The radium carbonate model exposed some gaps in the existing ORIGEN nuclear data libraries. Multiple production paths exist for the creation of ^{229}Th from ^{226}Ra. Some of these pathways highlighted the effect of gaps in the ORIGEN nuclear data libraries. The primary indicator was the production of ^{228}Ra in quantities that were roughly 10 orders of magnitude lower than expected. This was due to the lack of a ^{227}Ra capture cross section in the ORIGEN reaction resource. This and a few other data gaps were addressed with the creation of updated ORIGEN reaction and decay resources by the SCALE development team. In all, six new items were made available for ORIGEN use. These include capture reactions for ^{227}Ra, ^{229}Ra, ^{228}Ac, and ^{229}Ac and decay data for ^{229}Ra and ^{229}Ac. The ORIGEN library additions were made available in HFIRCON v1.0.4. The radium carbonate verification model is run in target depletion mode to ensure that intermediary isotopes produced in the creation of ^{229}Th are generated in the right order of magnitude (e.g., ^{228}Ra production is in the millicurie vice picocurie range).

Additional verification of the LAVAMINT module consisted of comparing HFIRCON stochastic, MCNP stochastic, and ORNL-TN analytic volumes for two HFIR-specific models. Both models have been used in previous detailed MCNP heat generation calculations that have been reviewed and documented in an internal report. These MCNP heat generation calculations required a known volume for all cells for which results were desired; however, for many cells in these models, MCNP was not able to provide an analytic volume. Therefore, the MCNP models were modified to generate stochastically calculated volumes for all cells of interest. These volumes were then inserted into the MCNP heat generation input files. The relative error on the stochastically calculated volumes ranged from as low as 0.01% to nearly 0.5%.

HFIRCON performs a stochastic volume calculation for all nonvoid cells in an initial ORNL-TN input file and automatically inserts these values into the modified ORNL-TN input file used for any requested follow-on transport calculations. Also, as described in Section 4.3, HFIRCON generates a file, named *ratio.txt*, containing the LAVAMINT stochastic volumes, and the ORNL-TN analytic volumes and stochastic/analytic volume ratios for all cells in the model for which ORNL-TN can provide an analytic

volume. HFIRCON allows users to define a target relative error for the stochastically calculated volumes. For the LAVAMINT verification jobs, stochastically calculated volumes for all nonvoid cells in the model were converged to a relative error of <0.3%.

For the BOC heat generation models, analytic volumes were calculated for 2,766 cells, and stochastic volumes were calculated for 4,124 cells. For the EOC models, analytic volumes were calculated for 1,972 cells, and stochastic volumes were calculated for 2,279 cells. Ratios of the various stochastic and analytic volumes ranged from 0.989 to 1.016. The lowest/highest ratios occur when comparing the HFIRCON and MCNP stochastically calculated volumes. The MCNP stochastic/HFIRCON stochastic volume ratios, HFIRCON stochastic/MCNP analytic volume ratios, and MCNP stochastic/MCNP analytic volume ratios for the BOC heat generation model are shown in **Figure 13**, Figure 14, and **Figure 15**. The analogous ratios for the EOC model are presented in **Figure 16**, Figure 17, and **Figure 18**. Because the stochastic volumes are converged to a few tenths of a percent relative error, most of the stochastic/analytic ratios should fall within roughly ±1% of 1.0. For example, the HFIRCON volumes are all converged to a 1σ relative error of roughly 0.3%. Therefore, roughly 68% of the stochastic volumes should be within ±0.3% of the true (analytic) volume. Roughly 95 and 99.7% of the stochastic volumes should be within ±2σ (0.6%) and 3σ (0.9%), respectively. This is the behavior seen in **Figure 14**, **Figure 15**, **Figure 17**, and **Figure 18**. Also, as expected, the MCNP stochastic volume/HFIRCON stochastic volume ratios exhibit a slightly wider spread, as seen in **Figure 13** and **Figure 16**. The results shown in **Figure 13** through **Figure 18** were generated on the RRD *lise.ornl.gov* computing cluster. Similar results were obtained on the apollo, romulus, remus, and cades computing cluster.

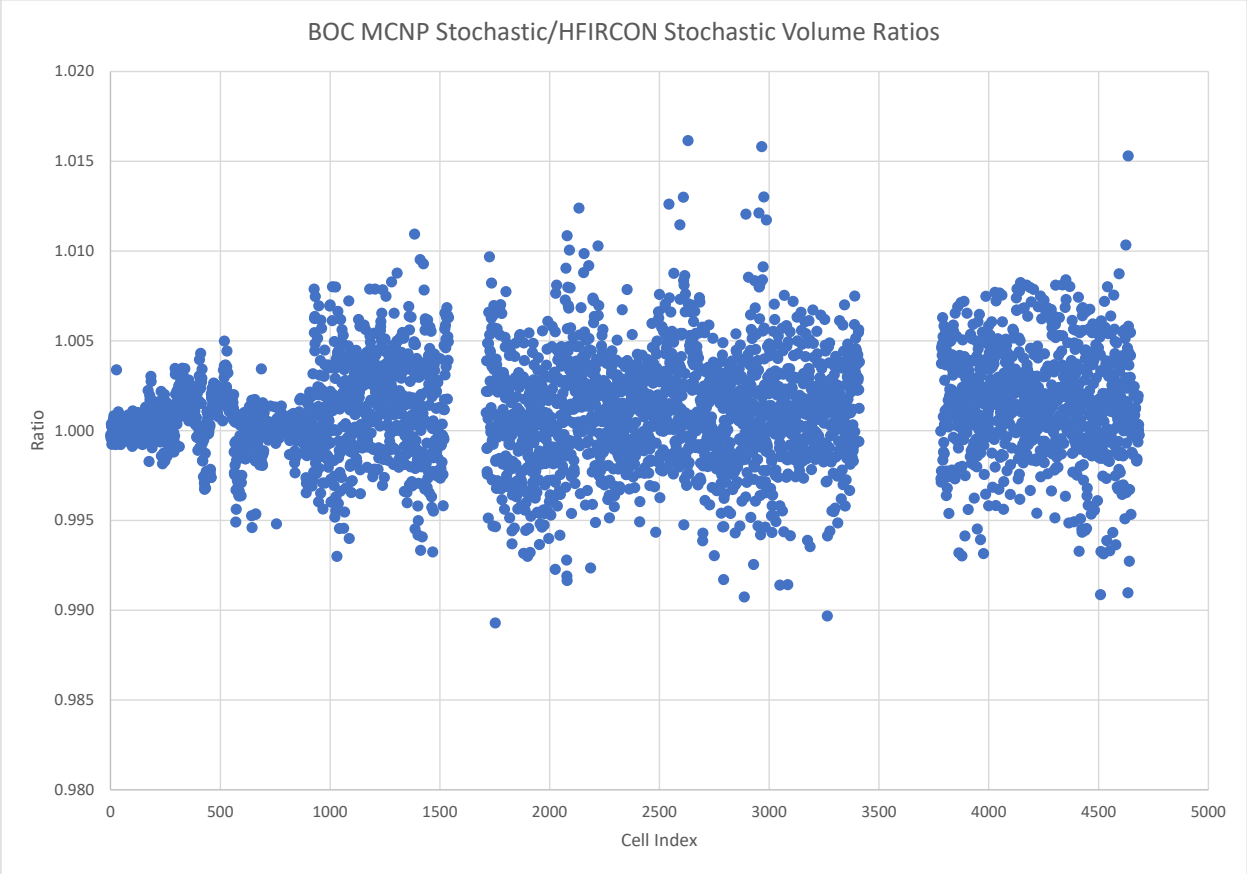

Figure 13. MCNP stochastic volume/HFIRCON stochastic volume ratios for the BOC heat generation model.

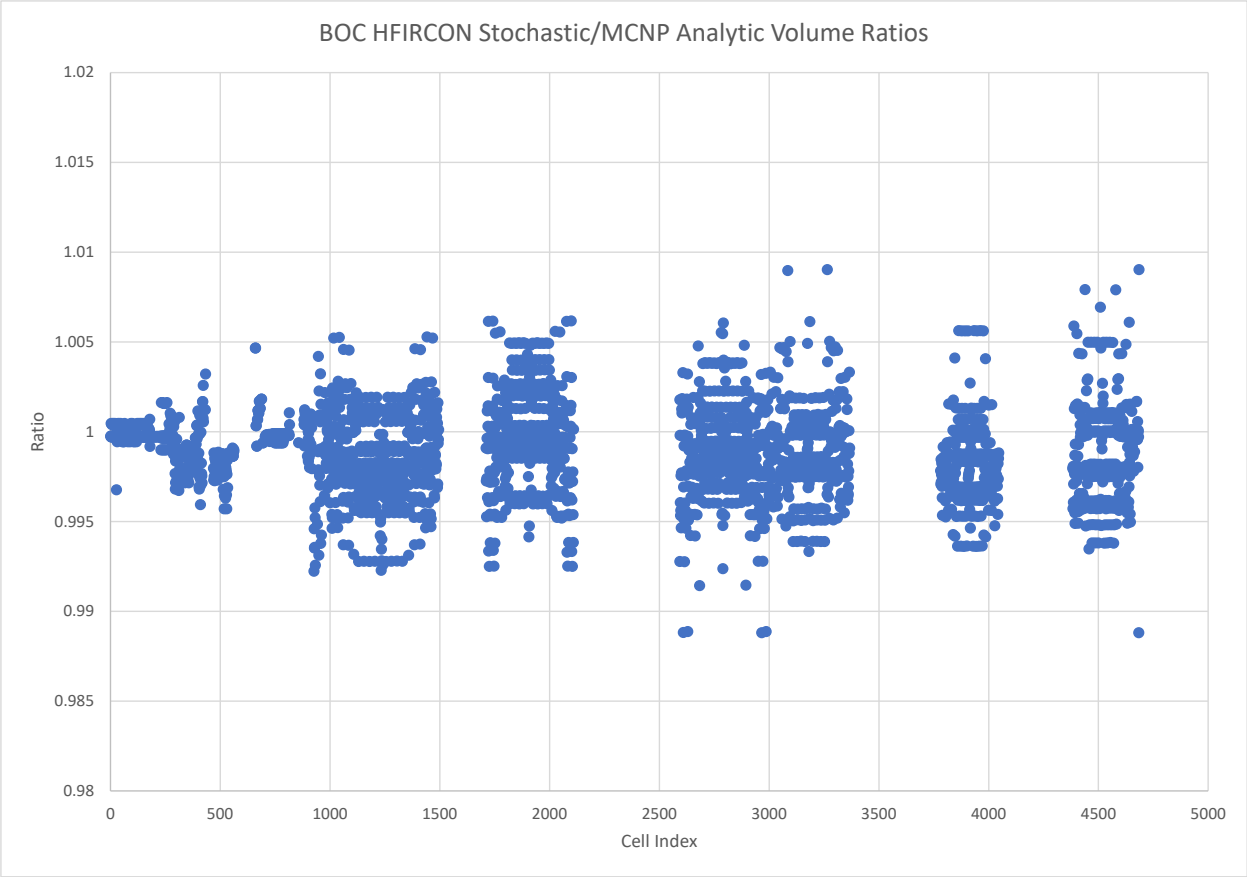

Figure 14. HFIRCON stochastic volume/MCNP analytic volume ratios for the BOC heat generation model.

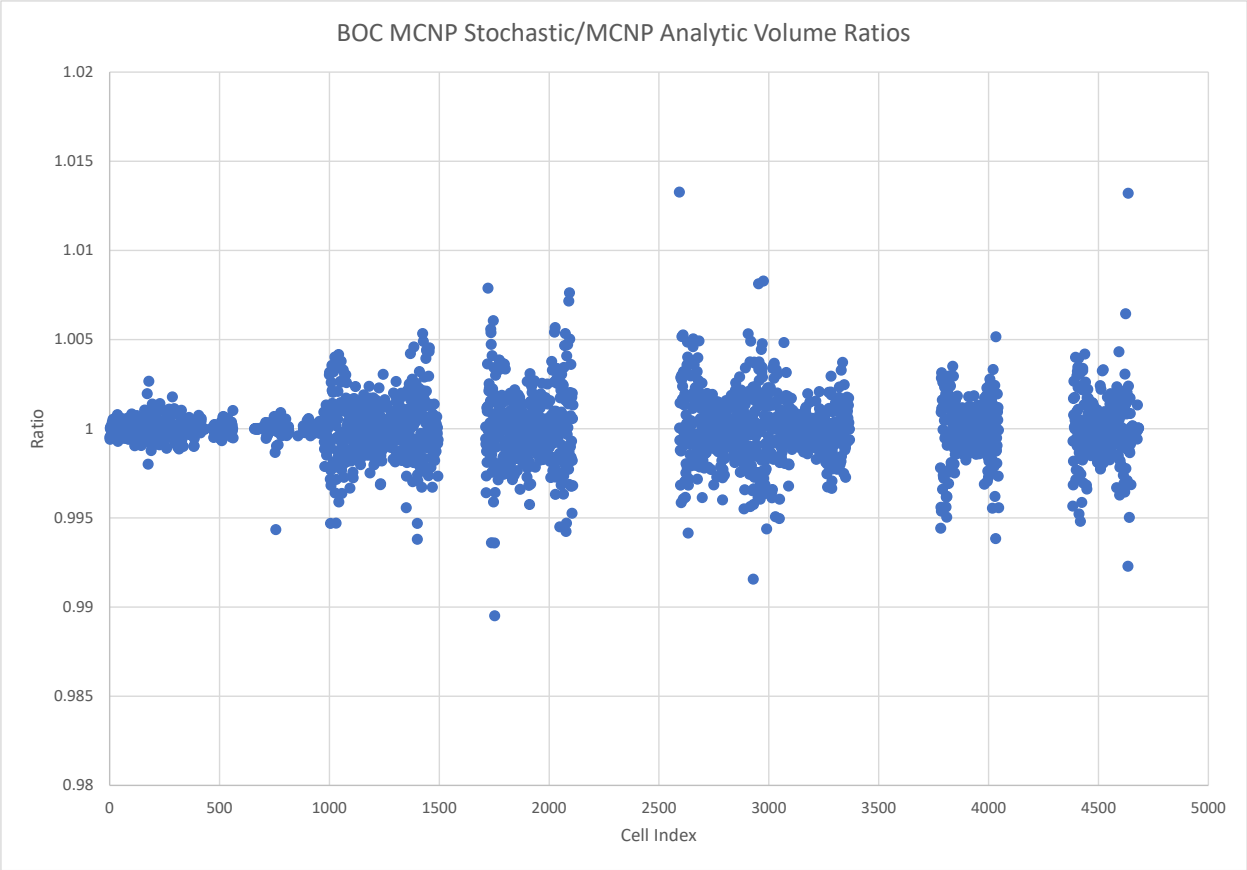

Figure 15. MCNP stochastic volume/MCNP analytic volume ratios for the BOC heat generation model.

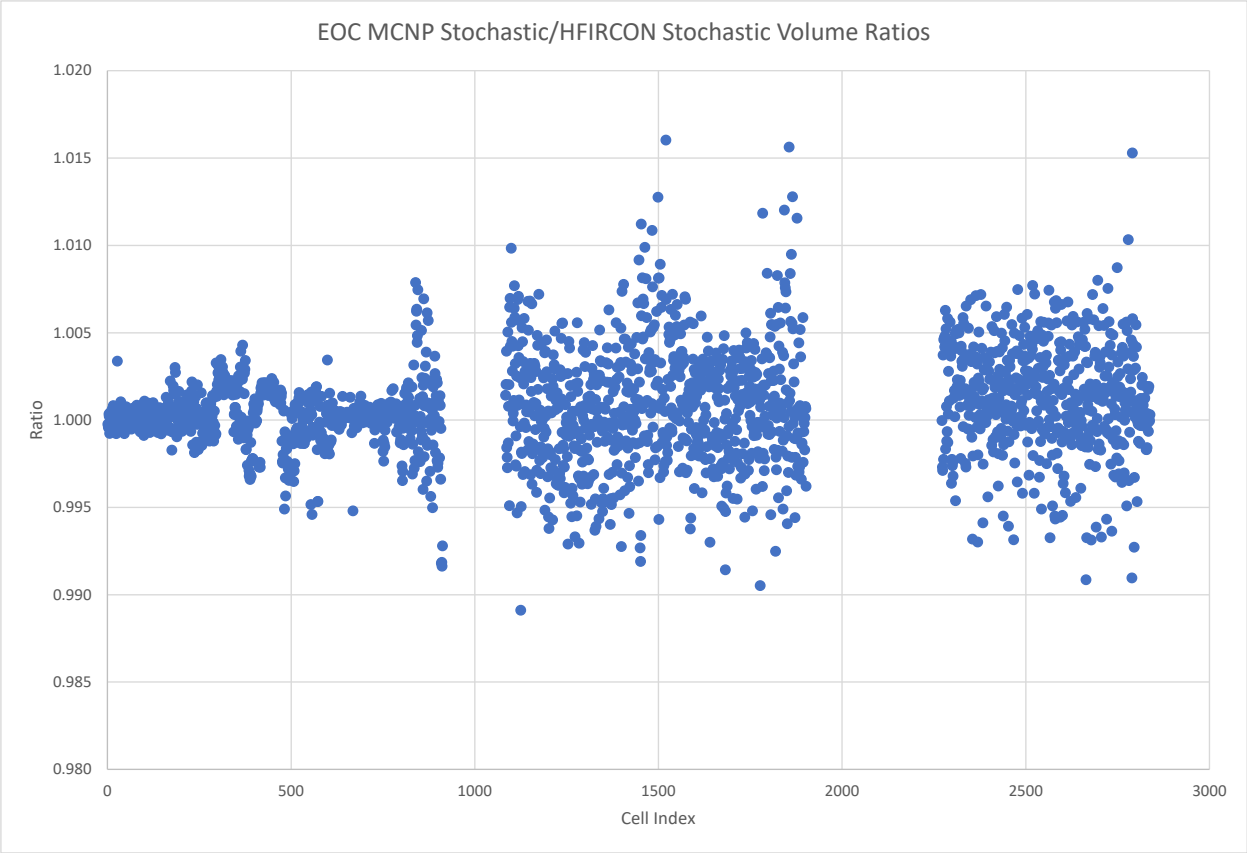

Figure 16. MCNP stochastic volume/HFIRCON stochastic volume ratios for the EOC heat generation model.

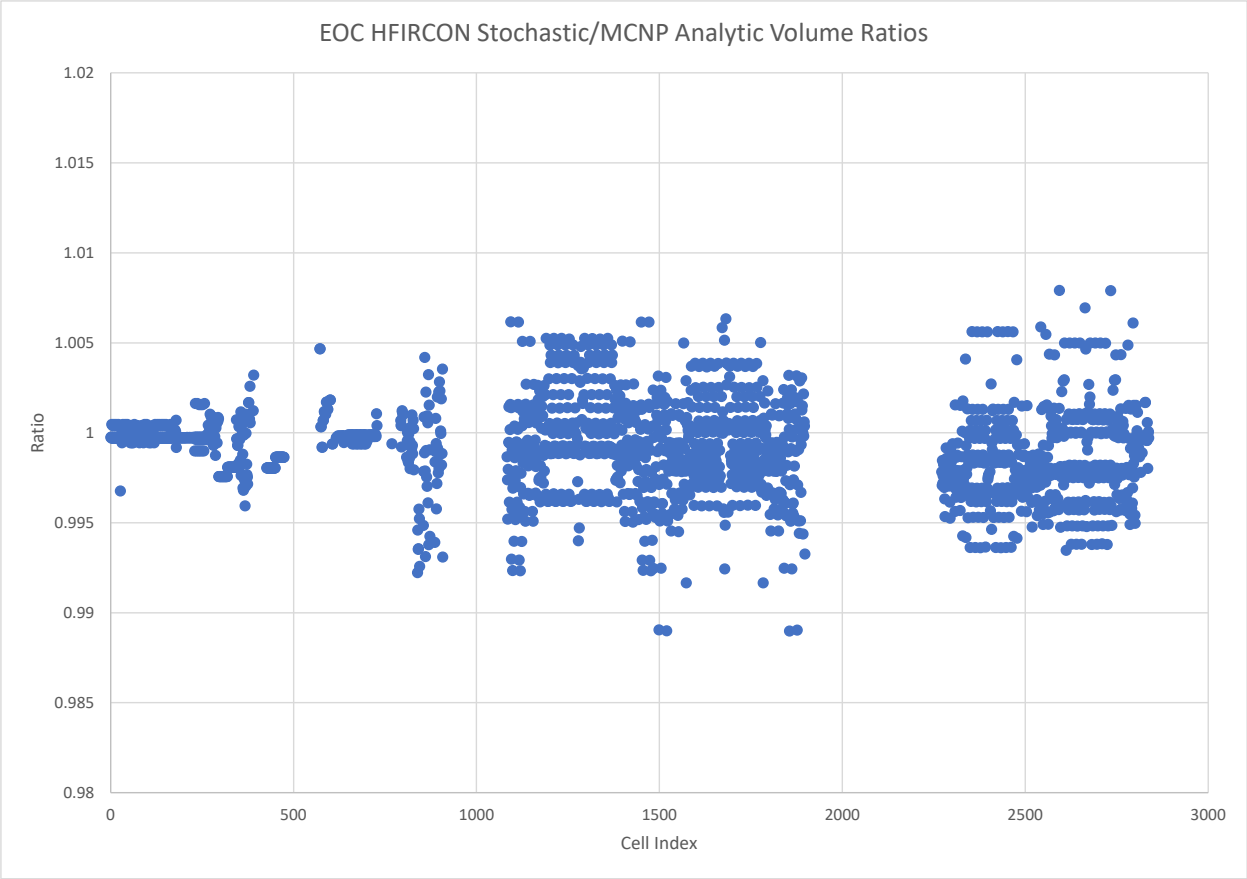

Figure 17. HFIRCON stochastic volume/MCNP analytic volume ratios for the EOC heat generation model.

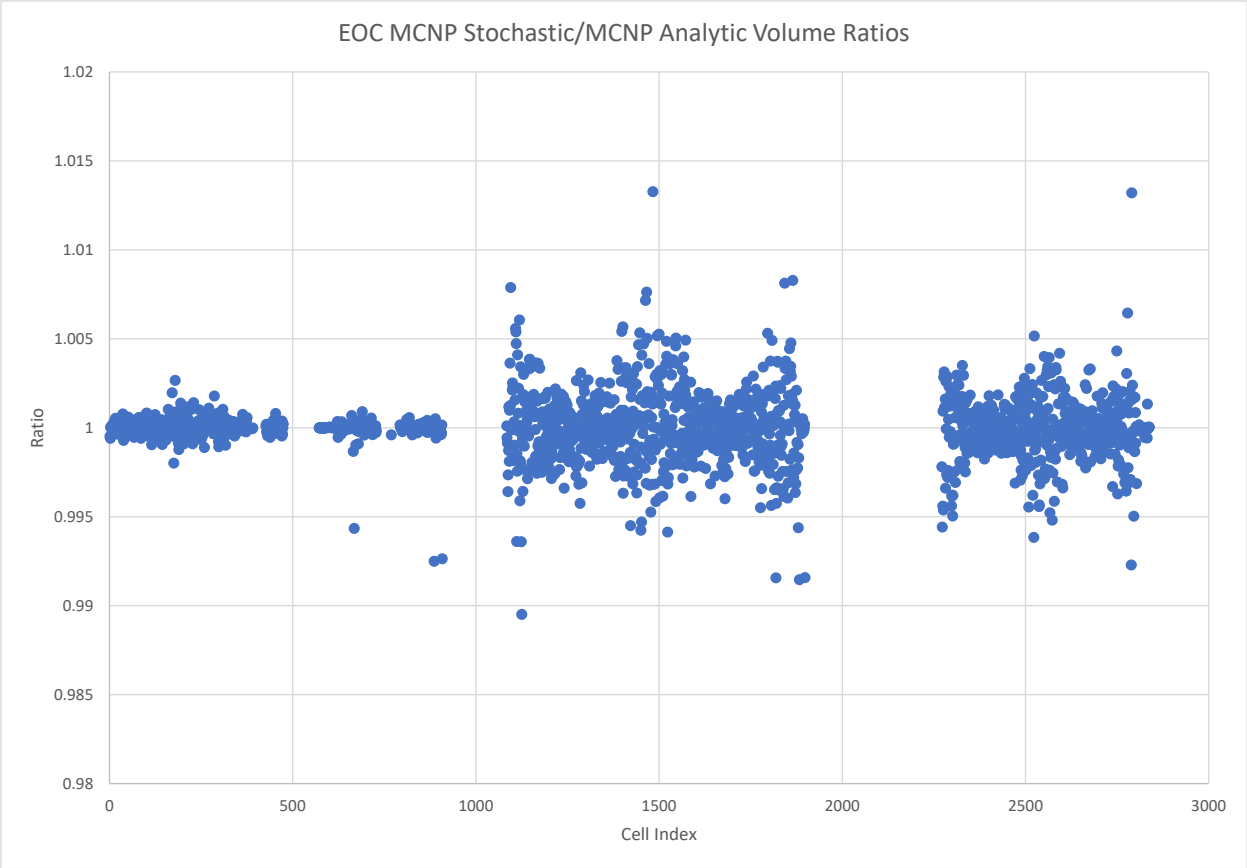

Figure 18. MCNP stochastic volume/MCNP analytic volume ratios for the BOC heat generation model.

Validation of HFIRCON is achieved by comparison with high-quality calculations performed with other depletion methods. A subset of the results calculated with these alternated methods have been compared to available measurements. The VESTA code [13] has been used to deplete high-enriched uranium (HEU) and LEU versions of HFIR models, and these calculations have been reviewed and documented in internal reports. Additionally, the Shift MC code depletion capability has been built and developed to replicate VESTA results and provides an additional basis for comparison.

The depletion methods in HFIRCON differ from those in VESTA and Shift. VESTA and Shift both tally neutron fluxes on a very fine (44,000 group) structure and rely on ORIGEN to generate one-group cross sections based on this fine-group flux. The use of such a fine-group flux representation is intended to accurately represent the effects of self-shielding in the one-group cross sections. The HFIRCON approach employs reaction rate tallies for fission rate and capture rate in all depletion cells in the transport solution, which allows it to provide appropriately self-shielded one-group fission and capture cross sections directly to ORIGEN.

The complete V&V suites were run on all platforms on which HFIRCON is currently installed. The small differences in some of the results generated on the various platforms are due to the manner in which the ORNL-TN calculations are parallelized. As mentioned in Section 4.1.1, HFIRCON runs multiple instances of ORNL-TN, each of which uses multiple threads. The number of independent ORNL-TN jobs run on a given node is equal to the number of NUMA pools available on that node. The total number of threads assigned to each independent ORNL-TN job on a node is equal to the total number of CPUs available on that node divided by the number of NUMA pools on that node. For a computing platform with 32 CPUs and two NUMA pools available on each node, there will be two independent ORNL-TN jobs, each using 16 threads, started on each node allocated to the HFIRCON job. Each of these jobs will be assigned a different starting random seed. After all ORNL-TN jobs have completed a user-defined number of particle histories, the results are statistically combined. Therefore, identical models run on two different platforms with a different number of CPUs and/or NUMA pools per node will yield results that are similar (statistically equivalent) but not identical to each other. This behavior can be noted in several of the tables and figures in Sections 7.1–7.4.

The following validation problems have been executed for HFIRCON:

7.1 HEU SIMPLIFIED FUEL MODEL WITH NPO_2/AL (CERMET) TARGETS

Source: C-HFIR-2015-013 internal report
Description: This HFIRCON validation problem is a minor modification of the HEU simplified model prepared for C-HFIR-2015-013.
Comparisons: CE positions; fuel poisons and actinides in the inner fuel element (IFE), outer fuel element (OFE), and combined over the entire fuel element (FE); ^{188}W and ^{252}Cf target transmutation; NpO_2/Al ^{238}Pu production in the inner small VXF (ISVXF), outer small VXF (OSVXF), and large VXF (LVXF) locations.

Location of outputs: On the cades cluster:

```
/lustre/hydra/cades-nsed/proj-shared/hfircon_V-and-V/V1.0.5/
VALIDATION/HEU/SIMPLIFIED/
```

Location of outputs: On the lise, apollo, romulus, and remus clusters[1]:

`/projects/hfircon/V-and-V/V1.0.5/VALIDATION/HEU/SIMPLIFIED/`

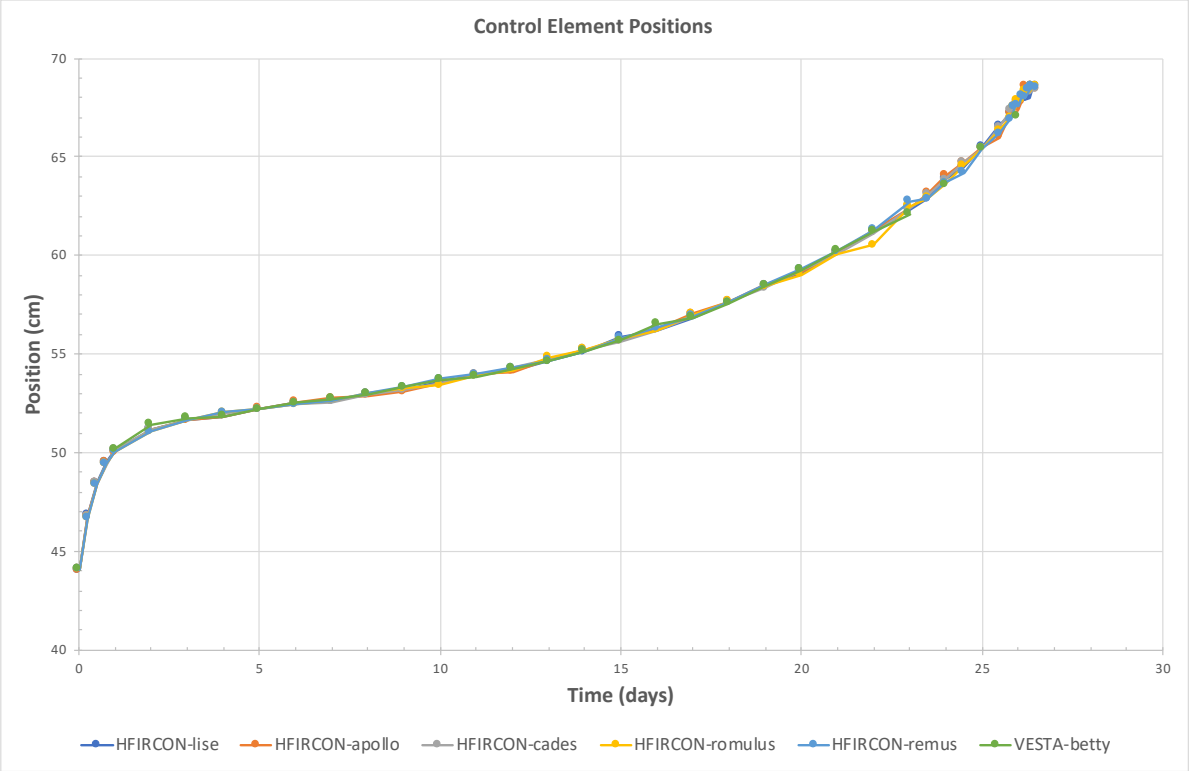

Figure 19. CE position vs. time, HFIRCON vs. VESTA simplified inputs.

[1]The /projects directory on these clusters is a limited resource. Subsequent to completion of the test jobs, the entire SIMPLIFIED subdirectory was moved to an archive maintained by the developer.

Table II. HFIRCON simplified fuel poisons and actinides.

VESTA Simplified Model (betty)						
Isotope	BOC			EOC		
	IFE (g)	OFE (g)	FE (g)	IFE (g)	OFE (g)	FE (g)
B-10	2.709	0.030	2.739	0.167	0.005	0.171
Xe-135	0.000	0.000	0.000	0.012	0.040	0.052
Sm-149	0.000	0.000	0.000	0.090	0.276	0.366
U-234	28.685	75.173	103.858	24.209	66.408	90.617
U-235	2607.749	6833.881	9441.630	1603.211	5033.021	6636.232
U-236	10.431	27.336	37.767	181.807	350.738	532.546
U-237	0.000	0.000	0.000	2.389	3.974	6.363
U-238	151.149	396.101	547.250	144.566	381.820	526.386
U-239	0.000	0.000	0.000	0.006	0.013	0.019
Pu-238	0.000	0.000	0.000	0.139	0.193	0.332
Pu-239	0.000	0.000	0.000	3.457	8.505	11.962
Pu-240	0.000	0.000	0.000	0.567	1.016	1.583
Pu-241	0.000	0.000	0.000	0.280	0.438	0.719
Pu-242	0.000	0.000	0.000	0.032	0.032	0.064

Table III. HFIRCON simplified fuel poisons and actinides.

HFIRCON Simplified Model (lise)						
Isotope	BOC			EOC		
	IFE (g)	OFE (g)	FE (g)	IFE (g)	OFE (g)	FE (g)
B-10	2.709	0.030	2.739	0.165	0.005	0.170
Xe-135	0.000	0.000	0.000	0.012	0.039	0.050
Sm-149	0.000	0.000	0.000	0.093	0.281	0.374
U-234	28.686	75.174	103.860	24.187	66.366	90.554
U-235	2607.811	6834.038	9441.850	1598.938	5023.255	6622.194
U-236	10.431	27.336	37.767	182.502	352.383	534.884
U-237	0.000	0.000	0.000	2.404	4.002	6.405
U-238	151.152	396.110	547.262	144.554	381.781	526.335
U-239	0.000	0.000	0.000	0.006	0.013	0.019
Pu-238	0.000	0.000	0.000	0.140	0.195	0.336
Pu-239	0.000	0.000	0.000	3.457	8.513	11.970
Pu-240	0.000	0.000	0.000	0.569	1.021	1.590
Pu-241	0.000	0.000	0.000	0.282	0.442	0.723
Pu-242	0.000	0.000	0.000	0.032	0.033	0.065

Table IV. Ratios of HFIRCON* to VESTA simplified model fuel poisons and actinides.

HFIRCON (lise) / VESTA (betty)						
Isotope	BOC			EOC		
	IFE (g)	OFE (g)	FE (g)	IFE (g)	OFE (g)	FE (g)
B-10	1.000	1.000	1.000	0.989	0.990	0.989
Xe-135	1.000	1.000	1.000	0.982	0.975	0.977
Sm-149	1.000	1.000	1.000	1.028	1.017	1.020
U-234	1.000	1.000	1.000	0.999	0.999	0.999
U-235	1.000	1.000	1.000	0.997	0.998	0.998
U-236	1.000	1.000	1.000	1.004	1.005	1.004
U-237	1.000	1.000	1.000	1.006	1.007	1.007
U-238	1.000	1.000	1.000	1.000	1.000	1.000
U-239	1.000	1.000	1.000	1.004	1.006	1.005
Pu-238	1.000	1.000	1.000	1.009	1.012	1.011
Pu-239	1.000	1.000	1.000	1.000	1.001	1.001
Pu-240	1.000	1.000	1.000	1.003	1.005	1.004
Pu-241	1.000	1.000	1.000	1.004	1.007	1.006
Pu-242	1.000	1.000	1.000	1.008	1.028	1.018

*HFIRCON results generated on the RRD lise cluster. HFIRCON results from other clusters were nearly identical to each other and to the lise result. The minimum and maximum X/lise ratio for any value in Table IV, where X = {cades, apollo, romulus, or remus}, were 0.998 and 1.003, respectively, with most of these ratios being 1.000.

Table V. NpO_2/Al Target ^{238}Pu production, VESTA vs. HFIRCON* simplified model.

Model	HEU Simplified					
Code	VESTA (Betty)			HFIRCON (remus)		
Experimental Facility	ISVXF	OSVXF	LVXF	ISVXF	OSVXF	LVXF
Np-237 (relative)	0.942	0.963	0.977	0.941	0.963	0.977
Pu-238 (grams)	54.544	21.912	37.625	54.810	21.999	37.777
Conversion (%)	5.020%	3.361%	2.126%	5.044%	3.221%	2.135%
Quality (%)	92.765%	95.109%	96.941%	92.375%	94.805%	96.626%
236Pu (ppm)	1.450	0.987	0.978	1.263	0.863	0.846

*HFIRCON results generated on the RRD lise cluster. HFIRCON results from other clusters were nearly identical to each other and to the lise result. The minimum and maximum X/lise ratio for any value in Table V, where X = {cades, apollo, romulus, or remus}, were 0.969 and 1.002, respectively, with most of these ratios being 1.000.

Table VI. Other target isotope production, VESTA vs. HFIRCON* simplified model.

	VESTA (betty)	HFIRCON (lise)
Isotope	mg	mg
W-188	11.109	8.720
Re-188	2.828	2.840
Bk-249	38.593	38.607
Cf-252	37.139	37.112

*HFIRCON results generated on the RRD lise cluster. HFIRCON results from other clusters were nearly identical to each other and to the lise result. All X/lise ratios, where X = {cades, apollo, romulus, or remus}, were between 0.983 and 1.006, respectively.

7.2 HEU EXPLICIT FUEL MODEL WITH NPO_2/AL (CERMET) TARGETS

Source: C-HFIR-2015-013 internal report
Description: This HFIRCON validation problem is a minor modification of the HEU explicit model prepared for C-HFIR-2015-013.
Comparisons: CE positions.

Location of outputs: On the cades cluster:

```
/lustre/hydra/cades-nsed/proj-shared/hfircon_V-and-V/V1.0.5/
VALIDATION/HEU/EXPLICIT/
```

Location of outputs: On the lise, apollo, romulus, and remus clusters[2]:

```
/projects/hfircon/V-and-V/V1.0.5/VALIDATION/HEU/EXPLICIT/
```

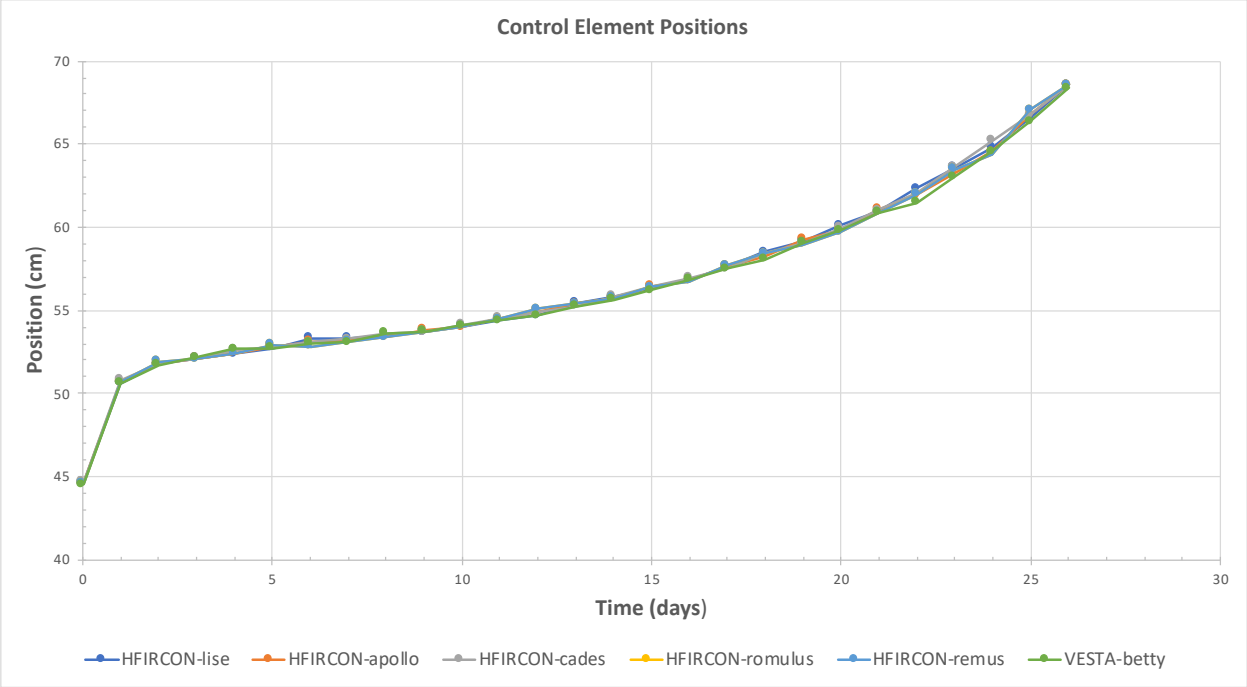

Figure 20. CE position vs. time, HFIRCON and VESTA explicit model.

[2]The /projects directory on these clusters is a limited resource. Subsequent to completion of the test jobs, the entire EXPLICIT subdirectory was moved to an archive maintained by the developer.

7.3 HEU SIMPLIFIED FUEL MODEL WITH MULTI-CYCLE NPO_2 TARGETS

Source: C-HFIR-2017-009 internal report
Description: This HFIRCON validation problem is a slight variation of the Phase-1 NpO_2 test capsule model. It is run for four cycles to test the multicycle depletion capability of HFIRCON.
Comparisons: Peak prompt neutron and gamma heating (W/g); peak $\alpha+\beta$ decay heat (W/g); temporal variation of total ^{238}Pu mass and fission gas moles (He, Xe, and Kr) over four irradiation cycles.

Location of outputs: On the cades cluster:

```
/lustre/hydra/cades-nsed/proj-shared/hfircon_V-and-V/V1.0.5/
VALIDATION/MULTI_CYCLE/NpO2_PHASE-1/
```

Location of outputs: On the lise, apollo, romulus, and remus clusters[3]:

```
/projects/hfircon/V-and-V/V1.0.5/VALIDATION/MULTI_CYCLE/NpO2_PHASE-1/
```

Table VII. Peak EOC prompt neutron heating rate (W/g) in the four NpO_2 Phase-1 test capsule pellets.

	Peak Prompt Neutron Heating Rate (W/g) for Phase-1 Test Capsule					
	HFIRCON					Python
	apollo	cades	lise	remus	romulus	betty
EOC 0						
Pellet_1	269.27	271.43	271.90	273.79	273.79	277.06
Pellet_2	277.62	279.71	272.67	279.23	279.23	286.98
Pellet_3	279.09	284.25	284.30	287.05	287.05	291.28
Pellet_4	284.47	284.64	288.31	281.03	281.03	293.09
EOC 1						
Pellet_1	359.69	366.65	369.66	364.00	364.00	370.22
Pellet_2	371.53	374.44	379.62	375.49	375.49	379.16
Pellet_3	383.56	375.64	370.79	380.50	380.50	386.43
Pellet_4	381.69	379.59	382.63	382.75	382.75	388.97
EOC 2						
Pellet_1	466.76	467.11	461.31	466.19	466.19	475.39
Pellet_2	471.67	476.77	464.49	471.03	471.03	485.23
Pellet_3	484.65	486.32	481.53	488.53	488.53	494.50
Pellet_4	496.24	489.19	502.36	488.12	488.12	499.97
EOC 3						
Pellet_1	554.96	563.87	560.90	561.44	561.44	578.17
Pellet_2	572.24	575.26	576.53	581.49	581.49	590.69
Pellet_3	577.89	598.34	588.72	594.01	594.01	601.87
Pellet_4	595.61	593.62	600.26	597.87	597.87	608.38

[3]The /projects directory on these clusters is a limited resource. Subsequent to completion of the test jobs, the entire SIMPLIFIED subdirectory was moved to an archive maintained by the developer.

Table VIII. Peak EOC prompt photon heating rate (W/g) in the four NpO_2 Phase-1 test capsule pellets.

	Peak Photon Heating Rate (W/g) for Phase-1 Test Capsule					
	HFIRCON					Python
	apollo	cades	lise	remus	romulus	betty
EOC 0						
Pellet_1	12.30	12.34	13.33	13.48	13.48	13.03
Pellet_2	12.25	12.40	12.89	13.41	13.41	13.43
Pellet_3	12.79	12.93	12.64	13.22	13.22	13.32
Pellet_4	12.27	13.09	12.68	13.06	13.06	13.71
EOC 1						
Pellet_1	13.45	12.27	12.89	12.53	12.53	12.93
Pellet_2	12.77	12.63	11.57	13.22	13.22	13.12
Pellet_3	13.41	12.98	12.94	12.20	12.20	13.10
Pellet_4	12.35	12.90	13.03	13.04	13.04	13.64
EOC 2						
Pellet_1	13.91	13.04	12.38	12.25	12.25	12.92
Pellet_2	11.61	13.75	13.07	11.90	11.90	13.23
Pellet_3	12.43	13.27	12.68	12.81	12.81	12.96
Pellet_4	13.22	12.89	12.48	12.10	12.10	13.54
EOC 3						
Pellet_1	12.43	12.40	12.32	12.16	12.16	12.89
Pellet_2	12.91	12.42	12.67	12.85	12.85	13.25
Pellet_3	13.71	13.15	12.19	12.90	12.90	13.10
Pellet_4	12.39	12.62	12.41	12.60	12.60	13.53

Table IX. Peak EOC α+β decay heating rate (W/g) in the four NpO_2 Phase-1 test capsule pellets.

	Alpha+Beta Decay Heating Rate (W/g) for Phase-1 Test Capsule					
	HFIRCON					Python
	apollo	cades	lise	remus	romulus	betty
EOC 0						
Pellet_1	12.24	12.23	12.25	12.34	12.34	12.41
Pellet_2	12.81	12.71	12.25	12.61	12.61	12.80
Pellet_3	12.63	12.63	12.71	12.63	12.63	12.98
Pellet_4	12.84	12.65	12.42	12.93	12.93	12.96
EOC 1						
Pellet_1	13.98	14.30	14.54	14.11	14.11	14.43
Pellet_2	14.88	14.62	15.20	14.45	14.45	14.87
Pellet_3	14.80	14.89	14.59	14.73	14.73	15.20
Pellet_4	14.82	14.77	15.14	14.67	14.67	15.18
EOC 2						
Pellet_1	16.41	16.64	16.71	17.18	17.18	17.07
Pellet_2	17.10	17.10	17.27	17.51	17.51	17.49
Pellet_3	17.47	17.53	17.67	17.90	17.90	17.91
Pellet_4	17.59	17.76	17.91	17.40	17.40	18.02
EOC 3						
Pellet_1	19.54	18.98	19.16	19.29	19.29	19.71
Pellet_2	19.57	19.74	19.85	19.96	19.96	20.17
Pellet_3	20.03	20.27	20.44	19.89	19.89	20.68
Pellet_4	20.55	20.24	20.64	20.39	20.39	20.93

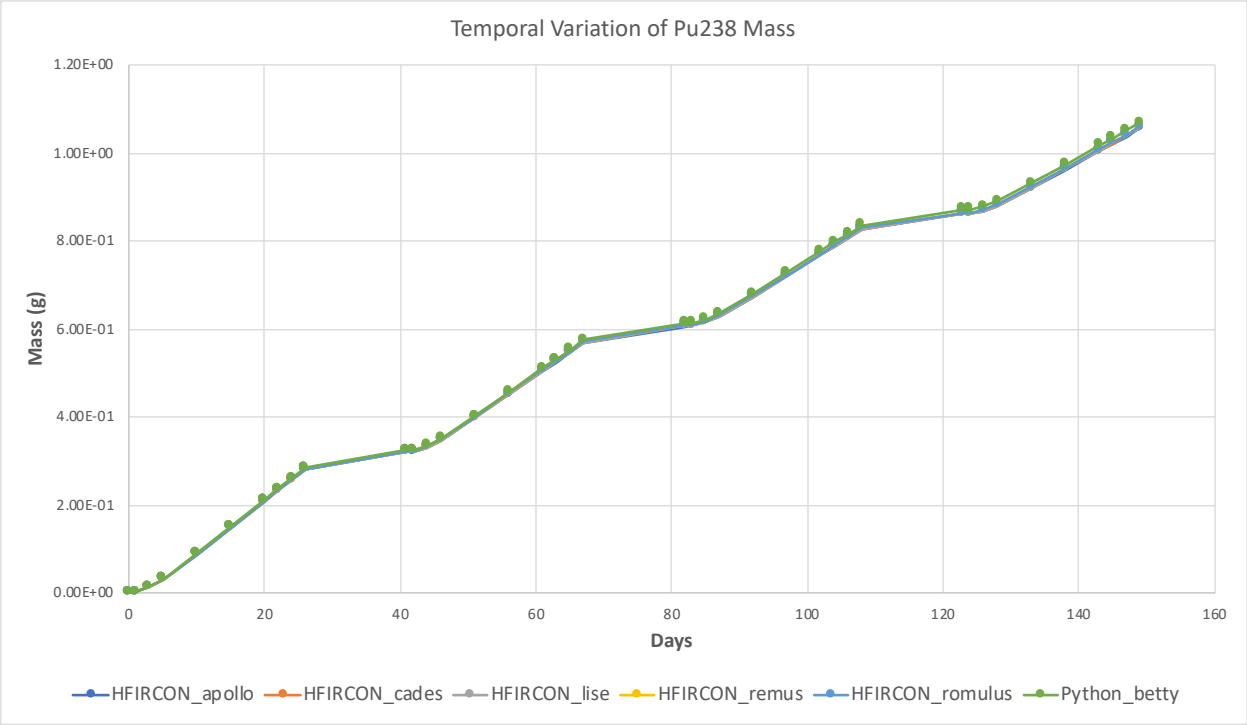

Figure 21. Temporal variation of total ^{238}Pu mass in the four NpO_2 Phase-1 test capsule pellets.

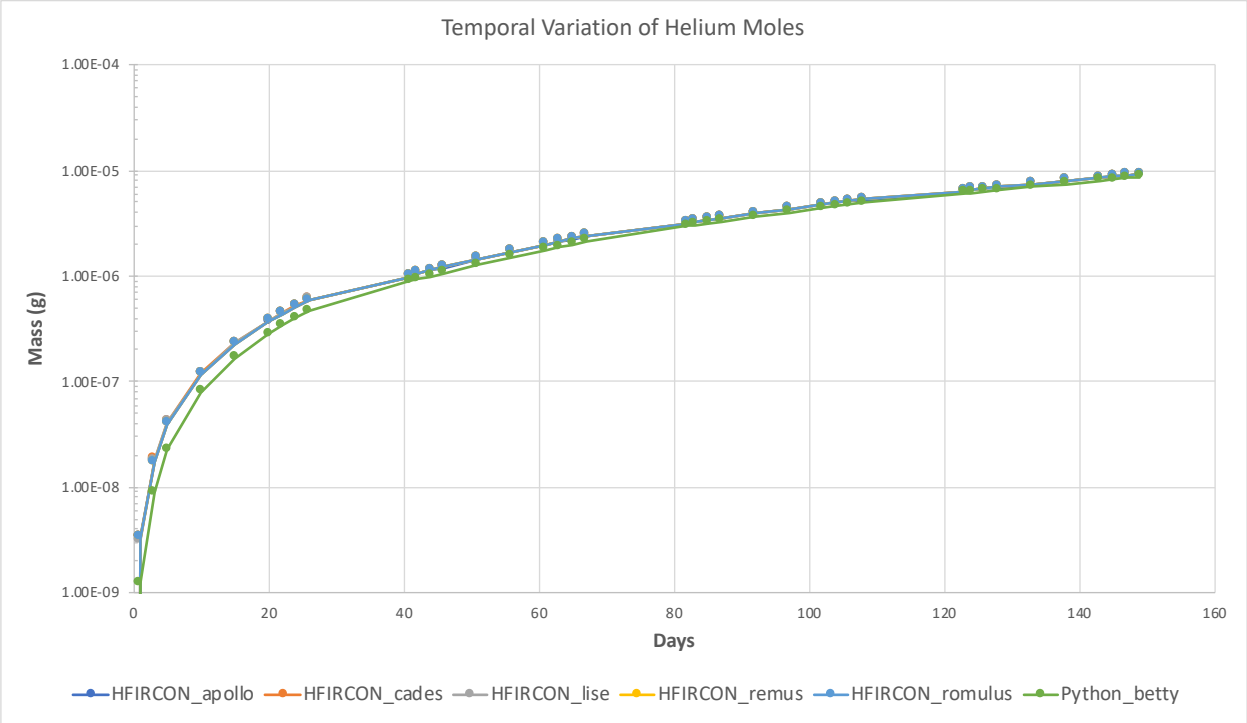

Figure 22. Temporal variation of total moles of He in the four NpO_2 Phase-1 test capsule pellets.

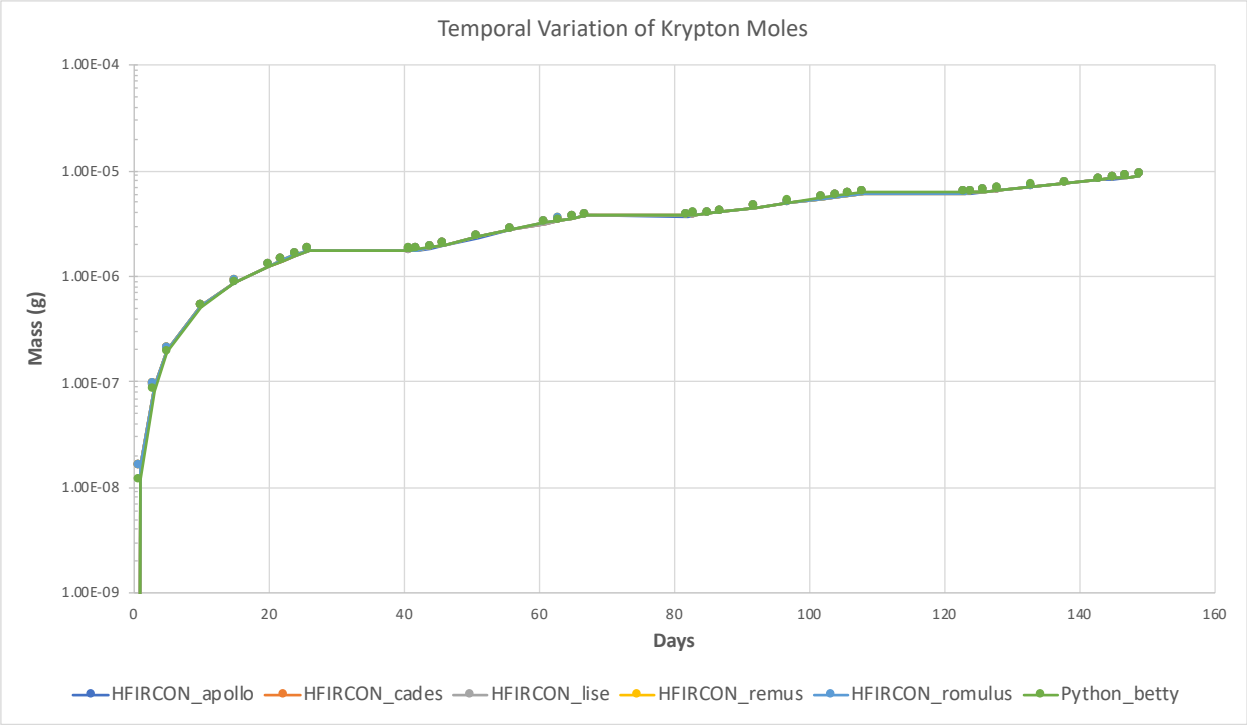

Figure 23. Temporal variation of total moles of Kr in the four NpO_2 Phase-1 test capsule pellets.

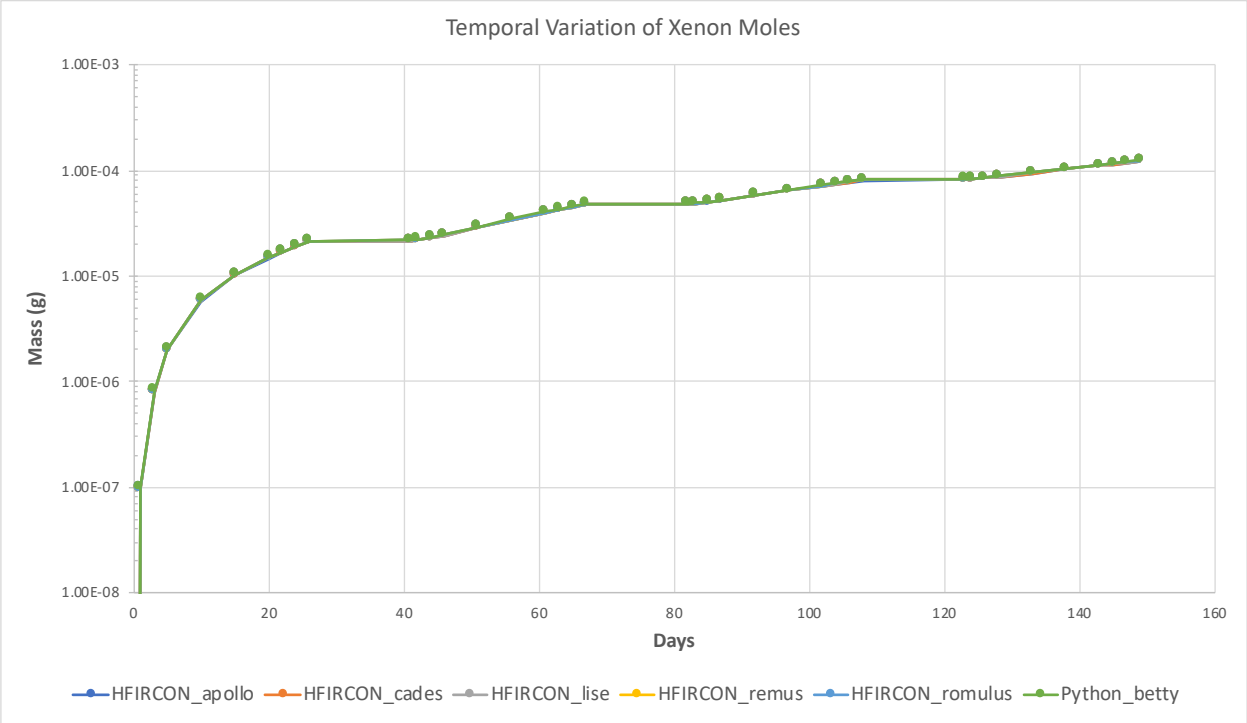

Figure 24. Temporal variation of total moles of Xe in the four NpO_2 Phase-1 test capsule pellets.

7.4 LEU SILICIDE EXPLICIT FUEL MODEL WITH NPO_2/AL (CERMET) TARGETS

Source: Shift, VESTA, and HFIRCON outputs
Description: This HFIRCON validation problem is a minor modification of a LEU silicide fuel analysis. It is compared directly with VESTA and Shift physics results.
Comparisons: CE positions, fuel poisons, and actinides

Location of outputs: On the cades cluster:

```
/lustre/hydra/cades-nsed/proj-shared/hfircon_V-and-V/V1.0.5/
VALIDATION/LEU/
```

Location of outputs: On the lise, apollo, romulus, and remus clusters[4]:

```
/projects/hfircon/V-and-V/V1.0.5/VALIDATION/LEU/
```

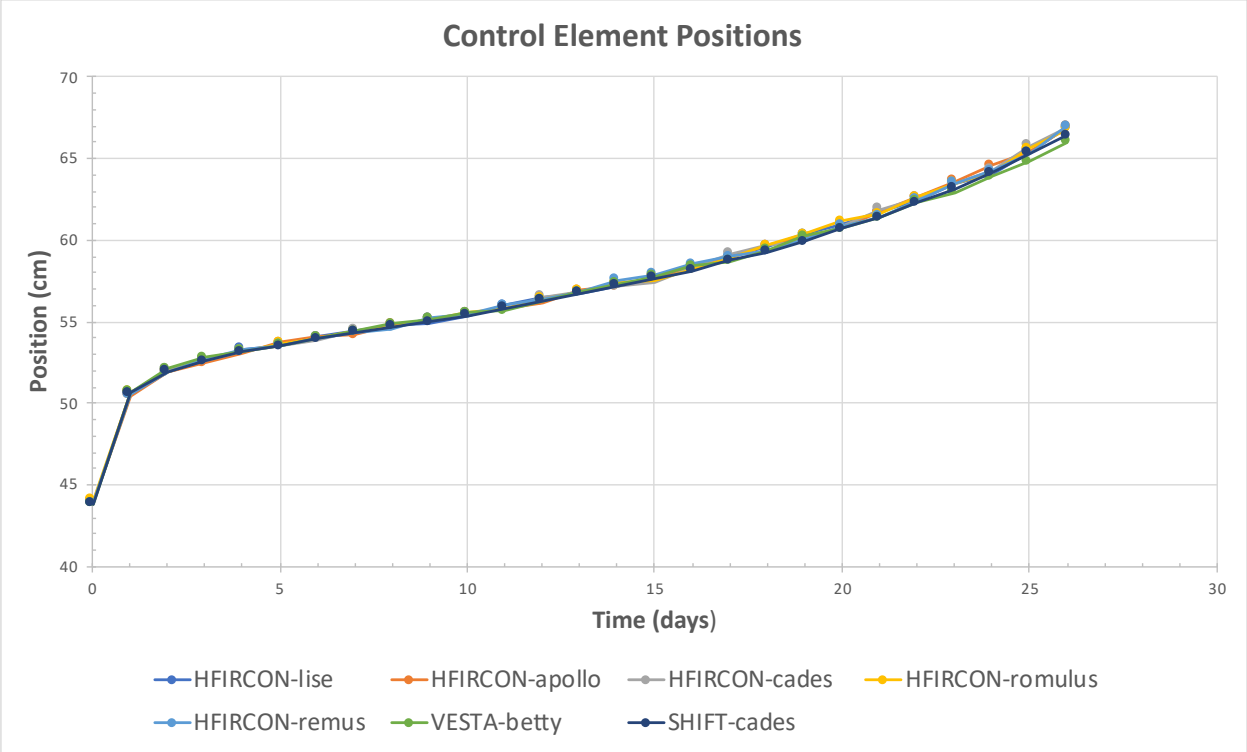

Figure 25. CE positions vs. time for the HFIRCON, SHIFT, and VESTA LEU silicide explicit model.

[4]The /projects directory on these clusters is a limited resource. Subsequent to completion of the test jobs, the entire LEU subdirectory was moved to a 4 TB archive mounted on the developer's laptop (MAC11545).

Table X. HFIRCON LEU silicide fuel poisons and actinides (g) vs. depletion time.

HFIRCON* Fuel Plate Poisons and Actinides (grams)									
Day	B10_filler	U235_IFE	Pu239_IFE	Xe135_IFE	Sm149_IFE	U235_OFE	Pu239_OFE	Xe135_OFE	Sm149_OFE
0.0	2.20E+00	4.08E+03	0.00E+00	0.00E+00	0.00E+00	9.87E+03	0.00E+00	0.00E+00	0.00E+00
1.0	2.02E+00	4.03E+03	7.71E-01	3.05E-02	2.19E-02	9.79E+03	1.30E+00	7.71E-02	3.72E-02
2.0	1.87E+00	3.98E+03	2.82E+00	3.32E-02	6.39E-02	9.72E+03	4.84E+00	8.33E-02	1.19E-01
3.0	1.73E+00	3.94E+03	5.75E+00	3.30E-02	1.01E-01	9.64E+03	1.00E+01	8.33E-02	2.06E-01
4.0	1.60E+00	3.89E+03	9.28E+00	3.26E-02	1.30E-01	9.57E+03	1.64E+01	8.29E-02	2.82E-01
5.0	1.47E+00	3.85E+03	1.32E+01	3.22E-02	1.52E-01	9.49E+03	2.35E+01	8.23E-02	3.44E-01
6.0	1.36E+00	3.80E+03	1.74E+01	3.19E-02	1.66E-01	9.42E+03	3.12E+01	8.17E-02	3.93E-01
7.0	1.26E+00	3.76E+03	2.17E+01	3.16E-02	1.77E-01	9.35E+03	3.93E+01	8.12E-02	4.29E-01
8.0	1.16E+00	3.72E+03	2.61E+01	3.12E-02	1.84E-01	9.27E+03	4.75E+01	8.06E-02	4.56E-01
9.0	1.08E+00	3.67E+03	3.05E+01	3.09E-02	1.88E-01	9.20E+03	5.58E+01	8.01E-02	4.76E-01
10.0	9.94E-01	3.63E+03	3.49E+01	3.06E-02	1.91E-01	9.12E+03	6.42E+01	7.95E-02	4.90E-01
11.0	9.19E-01	3.59E+03	3.92E+01	3.03E-02	1.93E-01	9.05E+03	7.25E+01	7.89E-02	4.99E-01
12.0	8.50E-01	3.54E+03	4.34E+01	3.00E-02	1.94E-01	8.97E+03	8.07E+01	7.83E-02	5.06E-01
13.0	7.87E-01	3.50E+03	4.75E+01	2.96E-02	1.95E-01	8.90E+03	8.89E+01	7.77E-02	5.10E-01
14.0	7.28E-01	3.46E+03	5.15E+01	2.93E-02	1.95E-01	8.83E+03	9.69E+01	7.72E-02	5.12E-01
15.0	6.74E-01	3.42E+03	5.54E+01	2.90E-02	1.95E-01	8.75E+03	1.05E+02	7.66E-02	5.14E-01
16.0	6.24E-01	3.37E+03	5.91E+01	2.87E-02	1.95E-01	8.68E+03	1.12E+02	7.61E-02	5.15E-01
17.0	5.78E-01	3.33E+03	6.28E+01	2.84E-02	1.95E-01	8.61E+03	1.20E+02	7.54E-02	5.14E-01
18.0	5.36E-01	3.29E+03	6.63E+01	2.81E-02	1.95E-01	8.53E+03	1.27E+02	7.47E-02	5.12E-01
19.0	4.97E-01	3.25E+03	6.97E+01	2.78E-02	1.94E-01	8.46E+03	1.34E+02	7.42E-02	5.12E-01
20.0	4.61E-01	3.21E+03	7.30E+01	2.75E-02	1.94E-01	8.38E+03	1.42E+02	7.36E-02	5.11E-01
21.0	4.27E-01	3.17E+03	7.62E+01	2.72E-02	1.94E-01	8.31E+03	1.48E+02	7.29E-02	5.09E-01
22.0	3.96E-01	3.13E+03	7.92E+01	2.69E-02	1.93E-01	8.24E+03	1.55E+02	7.23E-02	5.08E-01
23.0	3.68E-01	3.09E+03	8.21E+01	2.66E-02	1.93E-01	8.16E+03	1.62E+02	7.17E-02	5.06E-01
24.0	3.41E-01	3.05E+03	8.50E+01	2.63E-02	1.93E-01	8.09E+03	1.68E+02	7.10E-02	5.05E-01
25.0	3.17E-01	3.01E+03	8.77E+01	2.60E-02	1.92E-01	8.02E+03	1.74E+02	7.04E-02	5.03E-01
25.5	3.05E-01	2.99E+03	8.90E+01	2.58E-02	1.92E-01	7.98E+03	1.77E+02	6.97E-02	5.02E-01
26.0	2.94E-01	2.97E+03	9.03E+01	2.57E-02	1.92E-01	7.94E+03	1.80E+02	6.94E-02	5.00E-01
26.5	2.84E-01	2.95E+03	9.16E+01	2.55E-02	1.92E-01	7.90E+03	1.83E+02	6.92E-02	4.99E-01

*Results were generated on cades. Results generated on other clusters (apollo, lise, remus, and romulus) were all within 1% of the cades results, with most within a few tenths of a percent of the cades results.

Table XI. SHIFT LEU silicide fuel poisons and actinides (g) vs. depletion time.

SHIFT Fuel Plate Poisons and Actinides (grams)									
Day	B10_filler	U235_IFE	Pu239_IFE	Xe135_IFE	Sm149_IFE	U235_OFE	Pu239_OFE	Se135_OFE	Sm149_OFE
0.0	2.20E+00	4.07E+03	0.00E+00	0.00E+00	0.00E+00	9.88E+03	0.00E+00	0.00E+00	0.00E+00
1.0	2.02E+00	4.03E+03	7.69E-01	3.05E-02	2.19E-02	9.80E+03	1.30E+00	7.72E-02	3.71E-02
2.0	1.87E+00	3.98E+03	2.81E+00	3.32E-02	6.39E-02	9.73E+03	4.83E+00	8.34E-02	1.19E-01
3.0	1.73E+00	3.94E+03	5.73E+00	3.30E-02	1.01E-01	9.65E+03	1.00E+01	8.35E-02	2.06E-01
4.0	1.60E+00	3.89E+03	9.25E+00	3.26E-02	1.31E-01	9.58E+03	1.63E+01	8.30E-02	2.83E-01
5.0	1.47E+00	3.85E+03	1.32E+01	3.22E-02	1.52E-01	9.51E+03	2.35E+01	8.24E-02	3.45E-01
6.0	1.36E+00	3.80E+03	1.73E+01	3.19E-02	1.67E-01	9.43E+03	3.12E+01	8.19E-02	3.94E-01
7.0	1.26E+00	3.76E+03	2.17E+01	3.16E-02	1.77E-01	9.36E+03	3.92E+01	8.13E-02	4.30E-01
8.0	1.17E+00	3.72E+03	2.60E+01	3.12E-02	1.84E-01	9.28E+03	4.74E+01	8.07E-02	4.58E-01
9.0	1.08E+00	3.67E+03	3.04E+01	3.09E-02	1.89E-01	9.21E+03	5.57E+01	8.02E-02	4.77E-01
10.0	9.97E-01	3.63E+03	3.48E+01	3.06E-02	1.92E-01	9.14E+03	6.40E+01	7.96E-02	4.92E-01
11.0	9.22E-01	3.59E+03	3.90E+01	3.03E-02	1.94E-01	9.06E+03	7.23E+01	7.90E-02	5.01E-01
12.0	8.54E-01	3.54E+03	4.32E+01	3.00E-02	1.95E-01	8.99E+03	8.05E+01	7.85E-02	5.08E-01
13.0	7.90E-01	3.50E+03	4.74E+01	2.97E-02	1.96E-01	8.92E+03	8.86E+01	7.79E-02	5.12E-01
14.0	7.32E-01	3.46E+03	5.13E+01	2.94E-02	1.96E-01	8.84E+03	9.66E+01	7.73E-02	5.15E-01
15.0	6.78E-01	3.42E+03	5.52E+01	2.90E-02	1.96E-01	8.77E+03	1.04E+02	7.68E-02	5.16E-01
16.0	6.28E-01	3.38E+03	5.90E+01	2.87E-02	1.96E-01	8.70E+03	1.12E+02	7.62E-02	5.16E-01
17.0	5.82E-01	3.34E+03	6.26E+01	2.84E-02	1.95E-01	8.62E+03	1.20E+02	7.56E-02	5.16E-01
18.0	5.40E-01	3.29E+03	6.61E+01	2.81E-02	1.95E-01	8.55E+03	1.27E+02	7.50E-02	5.16E-01
19.0	5.01E-01	3.25E+03	6.95E+01	2.78E-02	1.95E-01	8.48E+03	1.34E+02	7.44E-02	5.15E-01
20.0	4.64E-01	3.21E+03	7.28E+01	2.75E-02	1.94E-01	8.41E+03	1.41E+02	7.38E-02	5.14E-01
21.0	4.31E-01	3.17E+03	7.60E+01	2.72E-02	1.94E-01	8.33E+03	1.48E+02	7.31E-02	5.12E-01
22.0	4.00E-01	3.13E+03	7.90E+01	2.69E-02	1.94E-01	8.26E+03	1.55E+02	7.25E-02	5.11E-01
23.0	3.71E-01	3.09E+03	8.19E+01	2.66E-02	1.94E-01	8.19E+03	1.61E+02	7.19E-02	5.09E-01
24.0	3.45E-01	3.06E+03	8.48E+01	2.63E-02	1.93E-01	8.11E+03	1.67E+02	7.13E-02	5.07E-01
25.0	3.20E-01	3.02E+03	8.75E+01	2.60E-02	1.93E-01	8.04E+03	1.74E+02	7.06E-02	5.06E-01
26.0	2.98E-01	2.98E+03	9.01E+01	2.57E-02	1.93E-01	7.97E+03	1.80E+02	7.00E-02	5.04E-01
27.0	2.77E-01	2.94E+03	9.26E+01	2.55E-02	1.92E-01	7.90E+03	1.85E+02	6.93E-02	5.02E-01

Table XII. VESTA LEU silicide fuel poisons and actinides (g) vs. depletion time.

VESTA Fuel Plate Poisons and Actinides (grams)									
Day	B10_filler	U235_IFE	Pu239_IFE	Xe135_IFE	Sm149_IFE	U235_OFE	Pu239_OFE	Se135_OFE	Sm149_OFE
0.0	2.20E+00	4.08E+03	0.00E+00	0.00E+00	0.00E+00	9.88E+03	0.00E+00	0.00E+00	0.00E+00
1.0	2.03E+00	4.03E+03	7.87E-01	3.08E-02	2.32E-02	9.81E+03	1.33E+00	7.84E-02	3.98E-02
2.0	1.87E+00	3.98E+03	2.85E+00	3.35E-02	6.56E-02	9.73E+03	4.89E+00	8.44E-02	1.23E-01
3.0	1.73E+00	3.94E+03	5.78E+00	3.32E-02	1.03E-01	9.66E+03	1.01E+01	8.44E-02	2.10E-01
4.0	1.60E+00	3.89E+03	9.30E+00	3.29E-02	1.32E-01	9.58E+03	1.64E+01	8.39E-02	2.86E-01
5.0	1.48E+00	3.85E+03	1.32E+01	3.25E-02	1.53E-01	9.51E+03	2.36E+01	8.33E-02	3.48E-01
6.0	1.37E+00	3.81E+03	1.74E+01	3.22E-02	1.67E-01	9.43E+03	3.13E+01	8.28E-02	3.96E-01
7.0	1.26E+00	3.76E+03	2.17E+01	3.18E-02	1.78E-01	9.36E+03	3.93E+01	8.22E-02	4.32E-01
8.0	1.17E+00	3.72E+03	2.61E+01	3.15E-02	1.84E-01	9.29E+03	4.75E+01	8.16E-02	4.59E-01
9.0	1.08E+00	3.67E+03	3.05E+01	3.12E-02	1.89E-01	9.21E+03	5.59E+01	8.10E-02	4.78E-01
10.0	9.99E-01	3.63E+03	3.48E+01	3.09E-02	1.92E-01	9.14E+03	6.42E+01	8.05E-02	4.92E-01
11.0	9.24E-01	3.59E+03	3.91E+01	3.05E-02	1.93E-01	9.06E+03	7.25E+01	7.99E-02	5.01E-01
12.0	8.55E-01	3.55E+03	4.33E+01	3.02E-02	1.94E-01	8.99E+03	8.07E+01	7.94E-02	5.08E-01
13.0	7.92E-01	3.50E+03	4.74E+01	2.99E-02	1.95E-01	8.92E+03	8.88E+01	7.88E-02	5.12E-01
14.0	7.33E-01	3.46E+03	5.14E+01	2.96E-02	1.95E-01	8.84E+03	9.68E+01	7.82E-02	5.14E-01
15.0	6.80E-01	3.42E+03	5.53E+01	2.93E-02	1.95E-01	8.77E+03	1.05E+02	7.75E-02	5.14E-01
16.0	6.30E-01	3.38E+03	5.91E+01	2.90E-02	1.94E-01	8.70E+03	1.12E+02	7.70E-02	5.14E-01
17.0	5.84E-01	3.34E+03	6.27E+01	2.87E-02	1.94E-01	8.63E+03	1.20E+02	7.63E-02	5.14E-01
18.0	5.41E-01	3.30E+03	6.62E+01	2.84E-02	1.93E-01	8.55E+03	1.27E+02	7.57E-02	5.13E-01
19.0	5.02E-01	3.26E+03	6.96E+01	2.80E-02	1.93E-01	8.48E+03	1.34E+02	7.51E-02	5.11E-01
20.0	4.66E-01	3.22E+03	7.29E+01	2.77E-02	1.92E-01	8.41E+03	1.41E+02	7.45E-02	5.09E-01
21.0	4.32E-01	3.18E+03	7.61E+01	2.74E-02	1.91E-01	8.33E+03	1.48E+02	7.39E-02	5.08E-01
22.0	4.01E-01	3.14E+03	7.91E+01	2.71E-02	1.91E-01	8.26E+03	1.55E+02	7.32E-02	5.06E-01
23.0	3.73E-01	3.10E+03	8.20E+01	2.68E-02	1.90E-01	8.19E+03	1.61E+02	7.26E-02	5.04E-01
24.0	3.46E-01	3.06E+03	8.49E+01	2.65E-02	1.90E-01	8.11E+03	1.68E+02	7.20E-02	5.02E-01
25.0	3.21E-01	3.02E+03	8.76E+01	2.62E-02	1.89E-01	8.04E+03	1.74E+02	7.13E-02	5.00E-01
26.0	2.98E-01	2.98E+03	9.02E+01	2.59E-02	1.89E-01	7.97E+03	1.80E+02	7.07E-02	4.98E-01
27.0	2.77E-01	2.94E+03	9.27E+01	2.57E-02	1.88E-01	7.90E+03	1.86E+02	7.00E-02	4.95E-01

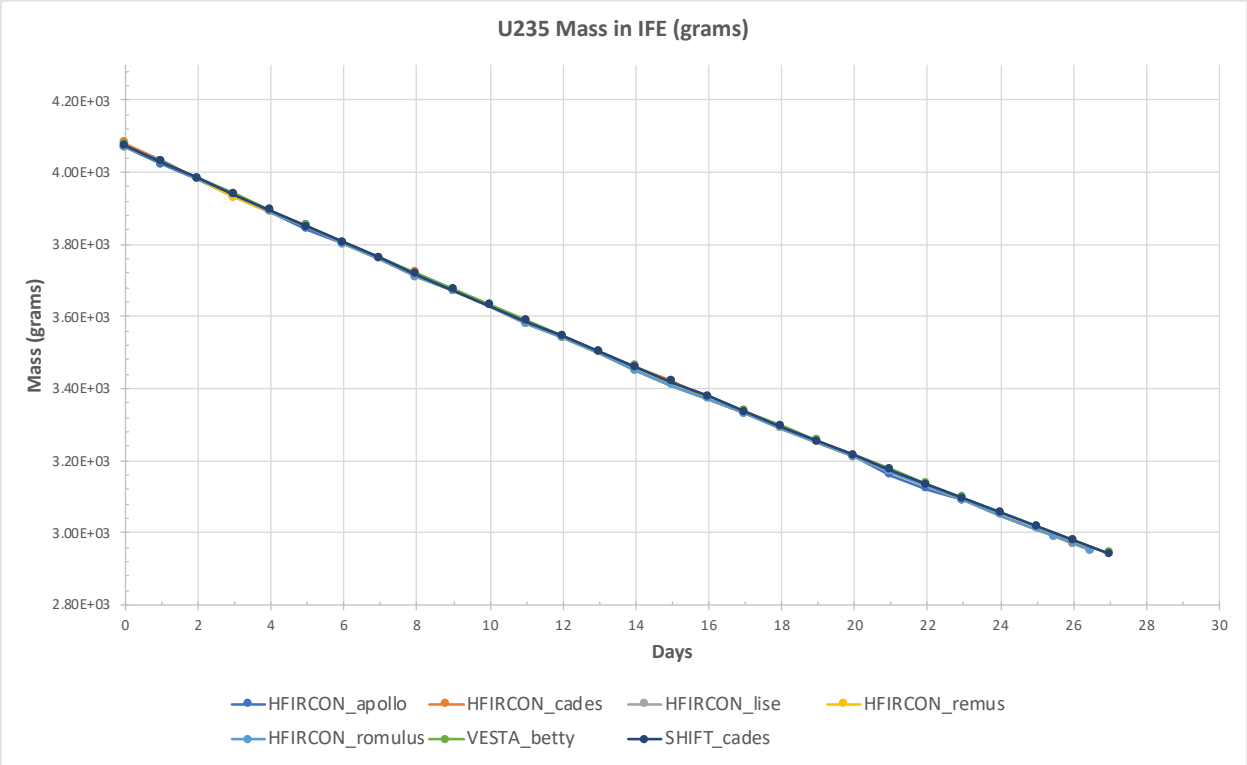

Figure 26. Inner fuel element ^{235}U mass by day for HFIRCON vs. SHIFT and VESTA.

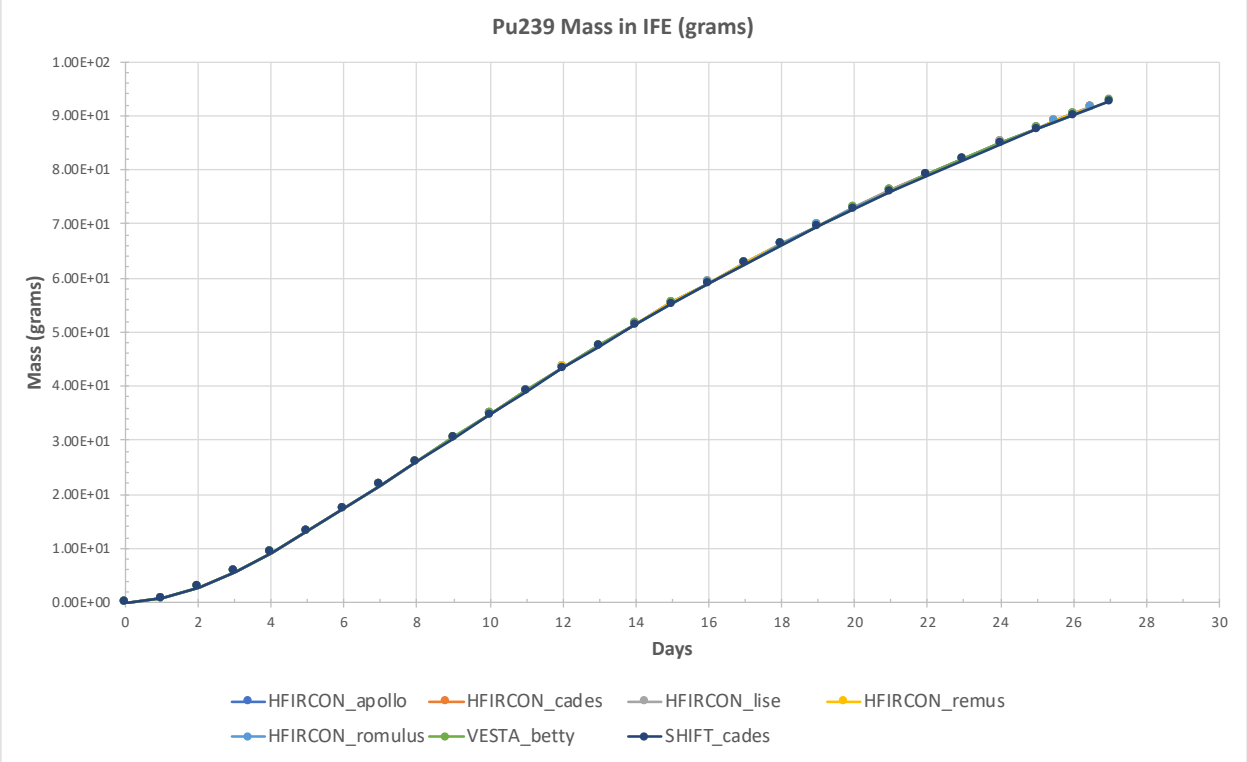

Figure 27. Inner fuel element ^{239}Pu mass by day for HFIRCON vs. SHIFT and VESTA.

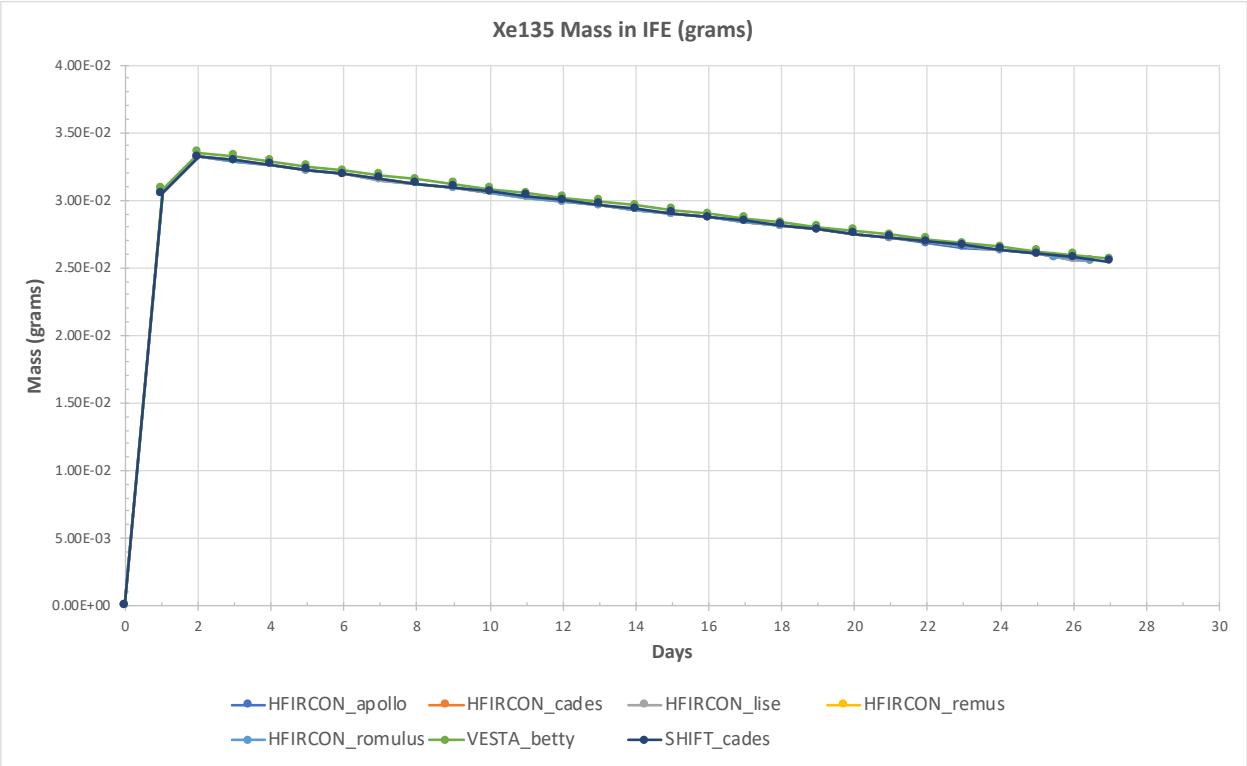

Figure 28. Inner fuel element ^{135}Xe mass by day for HFIRCON vs. SHIFT and VESTA.

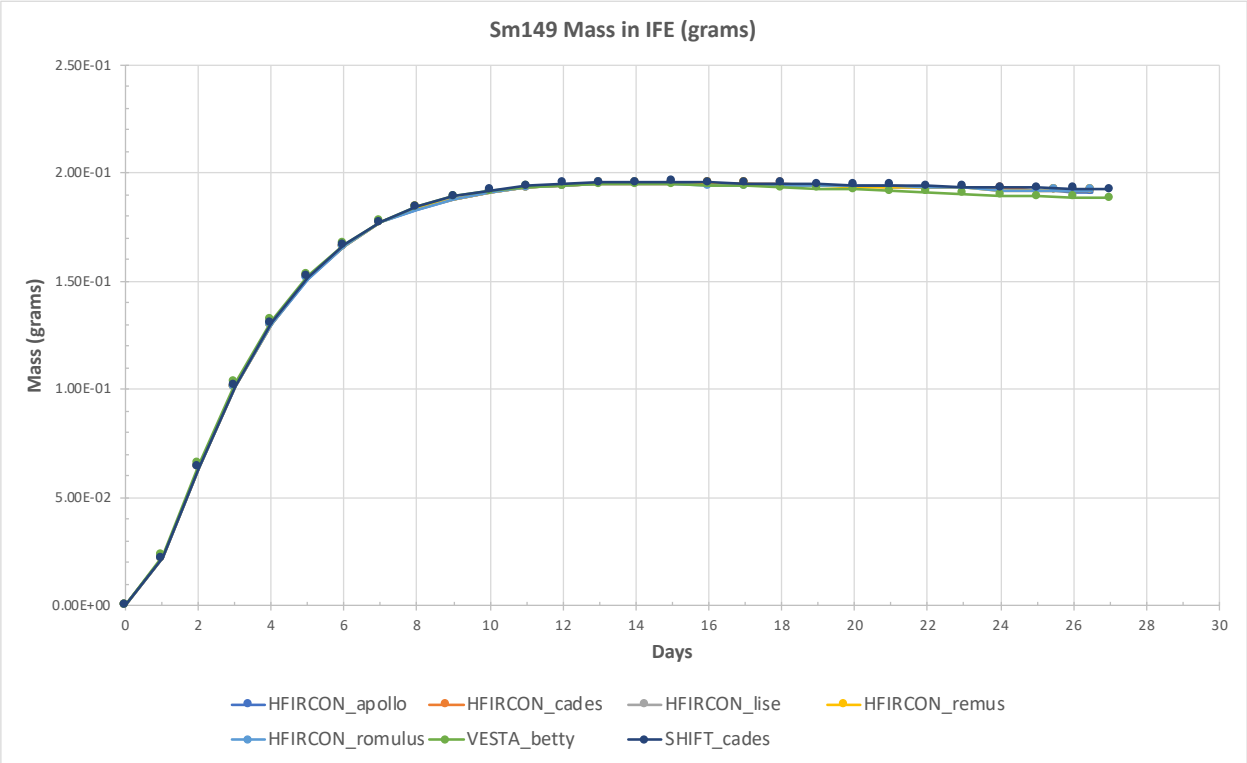

Figure 29. Inner fuel element ^{149}Sm mass by day for HFIRCON vs. SHIFT and VESTA.

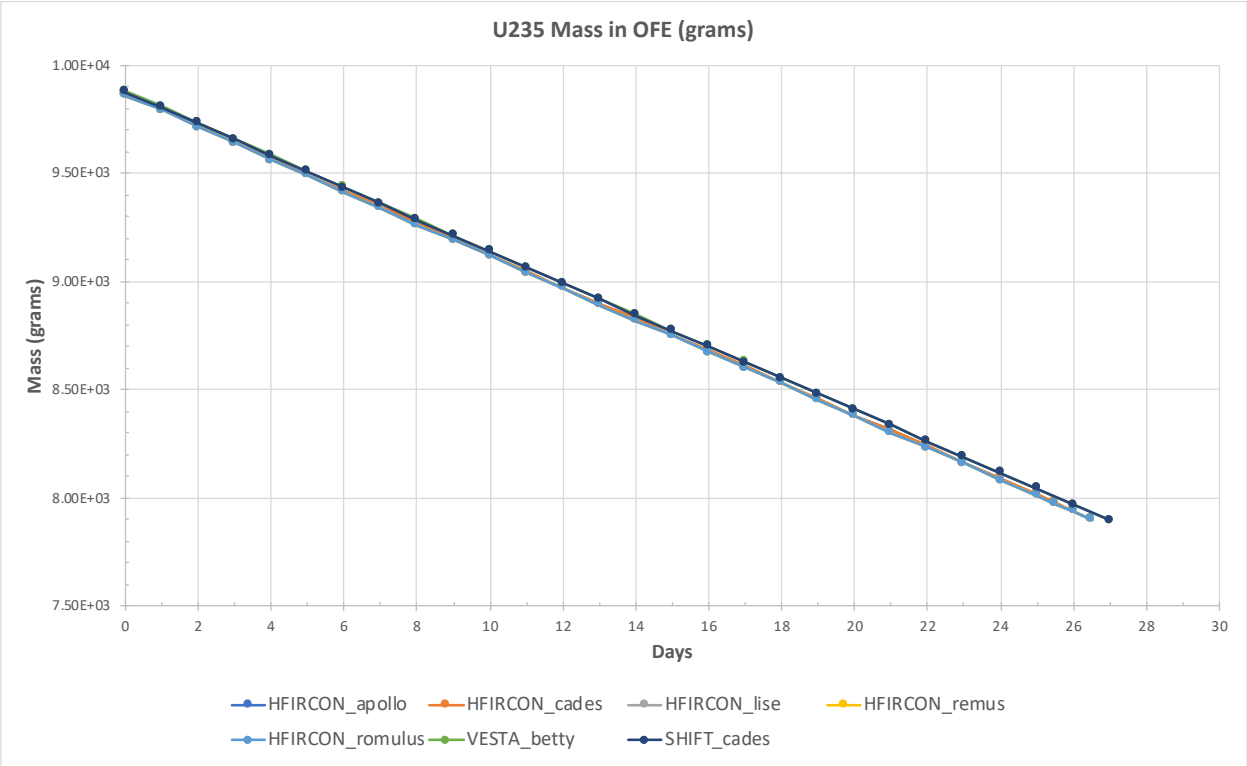

Figure 30. Outer fuel element ^{235}U mass by day for HFIRCON vs. SHIFT and VESTA.

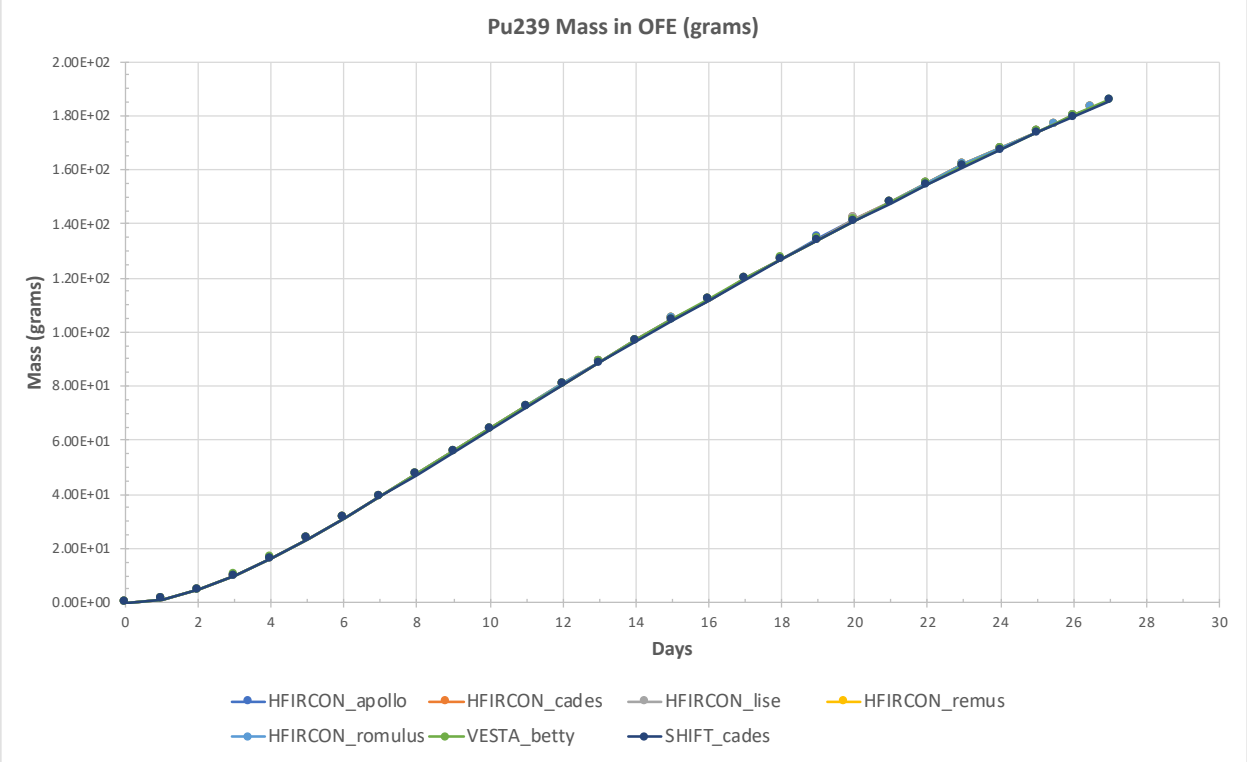

Figure 31. Outer fuel element ^{239}Pu mass by day for HFIRCON vs. SHIFT and VESTA.

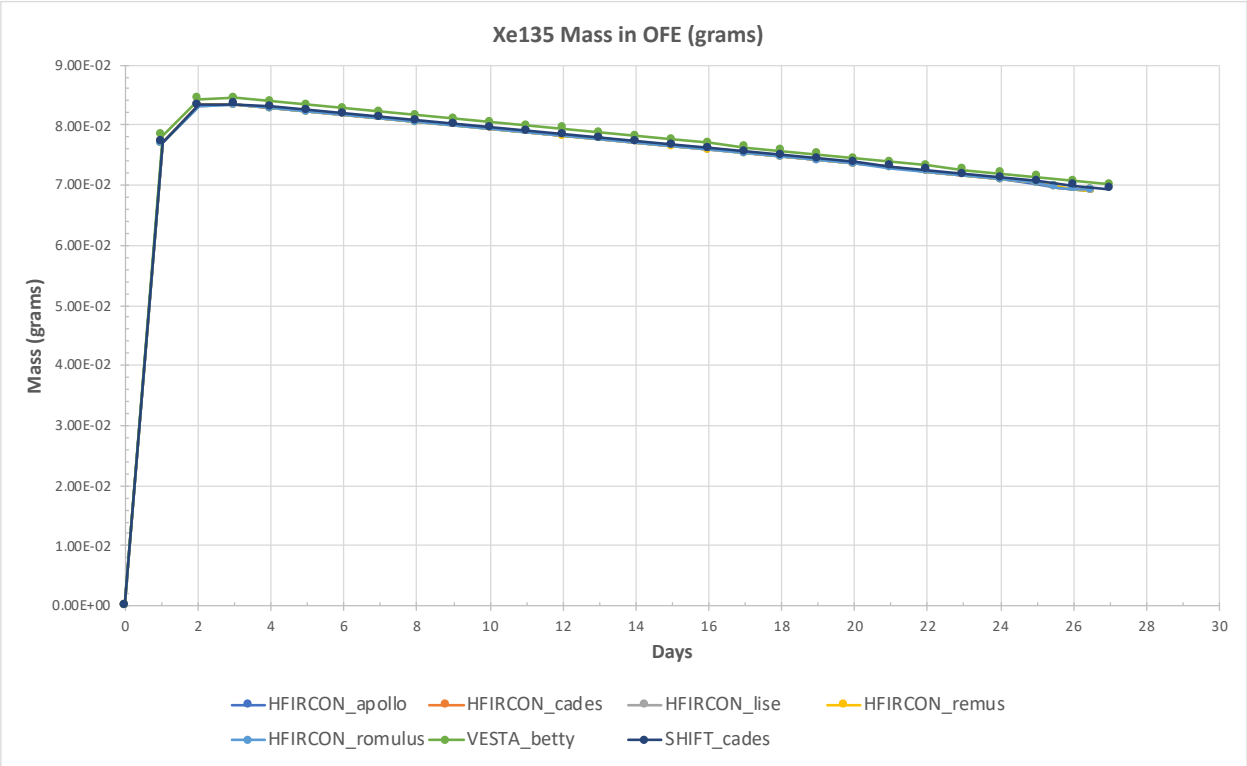

Figure 32. Outer fuel element ^{135}Xe mass by day for HFIRCON vs. SHIFT and VESTA.

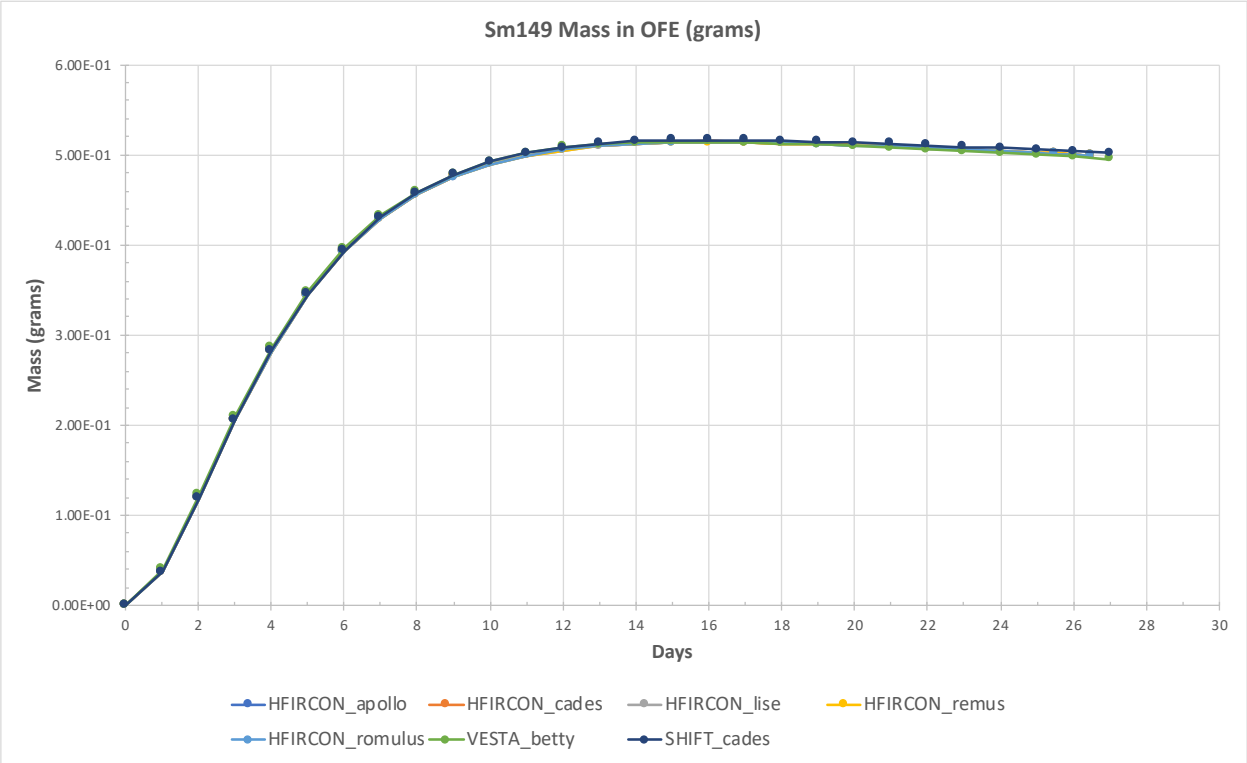

Figure 33. Outer fuel element ^{149}Sm mass by day for HFIRCON vs. SHIFT and VESTA.

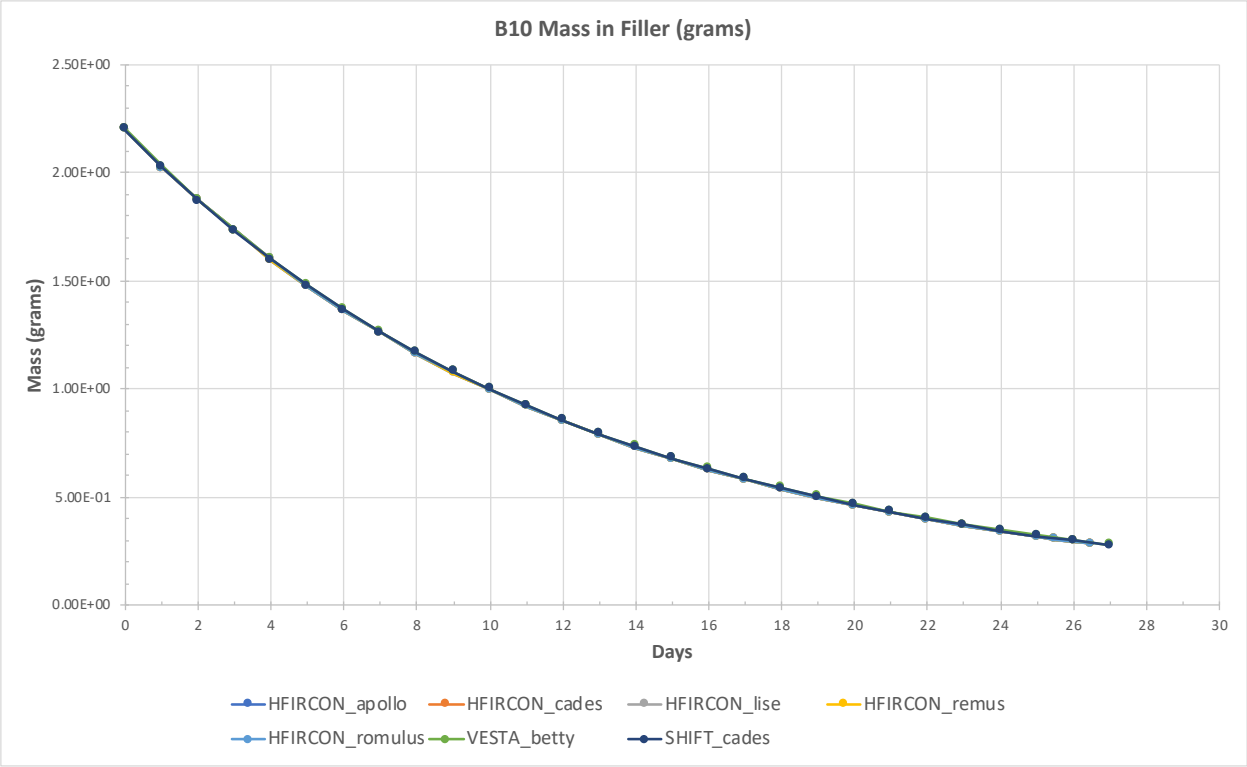

Figure 34. Inner fuel element ^{10}B mass by day for HFIRCON vs. SHIFT and VESTA.

8. REFERENCES

1. R. M. Wham et al., "The Plutonium-238 Supply Project," *The 19th Pacific Basin Nuclear Conference*, Vancouver, British Columbia, Canada, 2014.
2. J. Yang, S. C. Wilson, S. W. Mosher, G. R. Radulescu, "Integration of the Full Tokamak Reference Model with the Complex Model for ITER Neutronics Analysis," *Fusion Science and Technology* 74, no. 4 (2018): 277–287.
3. S. W. Mosher, S. C. Wilson, "Algorithmic Improvements to MCNP5 for High-Resolution Fusion Neutronics Analyses," *Fusion Science and Technology* 74, no. 4 (2018): 263–276.
4. S. C. Wilson et al., "Validation of the MS-CADIS Method for Full-Scale Shutdown Dose Rate Analysis," *Fusion Science and Technology* 74, no. 4 (2018): 288–302.
5. X-5 Monte Carlo Team, *MCNP – A General Monte Carlo N-Particle Transport Code, version 5, vol. 1, Overview and Theory*, LA-UR-03-1987, 2008.
6. I. C. Gauld, G. Radulescu, G. Ilas, B. D. Murphy, M. L. Williams, D. Wiarda, "Isotopic Depletion and Decay Methods and Analysis Capabilities in SCALE," *Nucl Technol* 174 (2011): 169.
7. M. Pusa and J. Leppanen, "Computing the Matrix Exponential in Burnup Calculations," *Nucl. Sci. Eng.*164 (2010): 140–150.
8. *SCALE: A Comprehensive Modeling and Simulation Suite for Nuclear Safety Analysis and Design*, ORNL/TM-2005/39, version 6.1, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 2011.
9. S. W. Mosher, S. R. Johnson, A. M. Bevill, A. M. Ibrahim, C. R. Daily, T. M. Evans, J. C. Wagner, J. O. Johnson, and R. E. Grove, *ADVANTG – An Automated Variance Reduction Parameter Generator*, ORNL/TM-2013/416, Rev. 1, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 2015.
10. J. C. Wagner, D. E. Peplow, S. W. Mosher, and T. M. Evans, "Review of Hybrid (Deterministic/Monte Carlo) Radiation Transport Methods, Codes, and Applications at Oak Ridge National Laboratory," *Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo (SNA + MC2010*), Tokyo, Japan, October 17–21, 2010.
11. C. R. Daily and D. Chandler, "Development and Testing of Nuclear Data Libraries for Improved Energy Deposition Modeling," *18th Topical Meeting of the Radiation Protection & Shielding Division of ANS*, Knoxville, Tennessee, September 14–18, 2014.
12. F. Brown, B. Kiedrowski, J. Bull, M. Gonzales, N. Gibson, *Verification of MCNP5-1.60*, LA-UR-10-05611, Los Alamos National Laboratory, Los Alamos, New Mexico, 2010.
13. W. Haeck, *VESTA User's Manual*, Version 2.0.0, IRSN Report DSU/SEC/T/2008-331, Indice A, France (2009).

APPENDIX A. ORNL-TN Modifications to MCNP5-1.60

APPENDIX A. ORNL-TN MODIFICATIONS TO MCNP5-1.60

Oak Ridge National Laboratory Transformative Neutronics (ORNL-TN) Upgrade to MCNP5-1.60

June 1, 2019

Scott W. Mosher and Stephen C. Wilson

A-1. INTRODUCTION

ORNL-TN [A-1] is a modified version of MCNP5-1.60 [A-2] that was developed to improve the scalability and performance of the software. Geometry initialization algorithms that exhibit poor performance on very detailed models were replaced with highly scalable algorithms. Additionally, a shared-memory mesh tally algorithm was developed, implemented, and tested that drastically reduces the memory required to generate high-resolution mesh tally results in multithreaded simulations. The implementation provides comparable run time performance to unmodified MCNP5 by employing batch statistics instead of history-based statistics.

ORNL-TN also provides a Python interface to a limited set of I/O facilities that, for example, allow additional materials and tallies to be defined during problem initialization and allow tally results to be accessed after transport is complete. The Python interface is described in Mosher and Wilson [A-3].

A-2. INPUT PARAMETERS

ORNL-TN reads additional input parameters from an optional configuration file supplied by the user. The name of the configuration file is specified in the same as for any other MCNP input file. The default name is `config`, and this can be overridden using the `config=<name>` option on the MCNP execution line. If no configuration file is provided, ORNL-TN will execute in the same way as unmodified MCNP5, except for the bug fixes that are listed in Section A-3.

The configuration file is a plain-text formatted file. Input parameters are specified using a lowercase keyword followed by a Boolean value, string, or an integer on the same line. Strings containing whitespace must be enclosed in double quotes (“”). Boolean literals may be specified as `true`, `false`, `yes`, or `no`. Comment lines begin with the `#` character and extend to the end of the line.

A description of all parameters that can be specified in the configuration file is given as follows.

Geometry Initialization Options

`igeom_fast_init` *`bool`* `(default: no)`
Enables or disables fast geometry initialization. Enabling this option will noticeably reduce problem start-up time for models with more than ~10^4 cells and surfaces.

`igeom_volume_calculations` *`bool`* `(default: yes)`
Enables or disables cell volume and tally surface area calculations. In simulations in which only mesh tallies are defined, disabling the cell volume calculations can noticeably reduce start-up times for models with more than ~10^4 cells.

Shared-Memory Mesh Tally Options and Parameters

`fmesh_batch_statistics` *`bool`* `(default: no)`
Enables or disables the shared-memory mesh tally implementation.

The shared-memory mesh tally algorithm reduces the memory consumed by mesh tally data by a factor approximately equal to the number threads. For performance reasons, mesh tally relative errors are calculated using batch statistics. It is strongly recommended to use at least 100 batches to obtain valid confidence intervals. Conventional history-based statistics are used to estimated relative errors for all other tally types (cell, surface, and point-detector).

If this option is enabled, then the MCNP input value specified on the NPS card is overridden as the product of `fmesh_num_batches` and `fmesh_num_per_batch`. If this option is disabled, then the remaining options listed in this section have no effect. **NOTE**: The shared-memory mesh tally algorithm was implemented for multithreaded simulation only. It cannot be used in MPI-parallel runs.

- `fmesh_num_batches` *`integer`* `(default: 1000)`
  Number of batches used to estimate shared-memory mesh tally relative errors.
- `fmesh_num_per_batch` *`integer`* `(default: 1000)`
  Number of particles per batch to be used to estimate shared-memory mesh tally relative errors.
- `fmesh_disable_re` *`bool`* `(default: no)`
  Disable or enable relative error calculations. This option is provided for the case in which relative errors will be estimated using the `merge_meshtal_hdf5` utility based on the results of multiple independent simulations. Disabling the relative error calculations reduces the in-memory size of mesh tally data by a factor of three. In this case, for efficiency it is recommended (but not required) to set `fmesh_num_batches` to 1 and to set `fmesh_num_per_batch` to the total number of particles to be simulated per run.
- `fmesh_hdf5_output` *`bool`* `(default: no)`
  Enable or disable the writing of mesh tally results to an HDF5 format file. For mesh tallies with more than ~10^7 space-energy bins, writing the results in binary HDF5 format significantly reduces I/O overhead and disk usage relative to the writing the results conventional plain-text output format.

Parameters that control the operation of the shared-memory tally algorithm are listed as follows. The default values should be appropriate for most problems. The performance of the shared-memory tally routines is only weakly dependent on the value of these parameter. If mesh tally scores are generated extremely rapidly during the simulation (e.g., due to a very fine tally mesh), then the number of buffers and number of scores per buffer might need to be increased above the default value. Similarly, the sleep interval might need to be decreased below the default value.

- `fmesh_num_buffers` *`integer`* `(default: 10)`
  Number of score buffers per transport thread.
- `fmesh_buffer_size` *`integer`* `(default: 100000)`
  Capacity of each score buffer, in number of scores.
- `fmesh_sleep_interval` *`integer`* `(default: 100)`
  Amount of time a thread should suspend execution, in milliseconds, when waiting for useful work.

Output Options

`output_runtpe` *`bool`* `(default: yes)`
Disable or enable writing the runtpe file. In cases with large number of mesh tally bins in which restarts are not needed, disabling the runtpe output significantly reduces I/O overhead and disk usage.

Python Scripting Options

`python_script_filename` *`string`* `(default: none)`
File name of a user-provided Python script. Mosher and Wilson [A-3] provide further details.

Source Plugin Options

`source_plugin_filename` *`string`* `(default: none)`
Source plugin shared-object library file name. On UNIX/Linux platforms, shared object libraries have a `.so` file extension.

`source_plugin_argfile` *`string`* `(default: none)`
File name of an optional source plugin argument file. Arguments read from this file are passed, as an array of strings, to the plugin initialization function.

A-3. LIST OF FIXED BUGS

- Fixed index overflow when using weight-window parameter sets with more than 2^{31}-1 elements
- Fixed out-of-bound writes to cell-neighbor array
- Fixed infinite loop caused by nonpositive distance to surface calculated due to roundoff error
- Fixed improper handling of Law 44 data in acecas
- Fixed zero-weight particle bug caused by gap in resonance scattering data
- Fixed CPU time calculation in multithreaded gfortran builds

A-4. REFERENCES

[A-1] S. W. Mosher and S. C. Wilson, "Algorithmic Improvements to MCNP5 for High-Resolution Fusion Neutronics Analyses," *Fusion Science and Technology*, 74, no. 4 (2018): 263–276. https://doi.org/10.1080/15361055.2018.1496691.

[A-2] X-5 MONTE CARLO TEAM, *MCNP—A General Monte Carlo N-Particle Transport Code, Version 5*, LA-UR-03-1987, Los Alamos National Laboratory, Los Alamos, New Mexico, 2008.

[A-3] S. W. Mosher and S. C. Wilson, "ORNL-TN Python Interface," October 15, 2018.

APPENDIX B. ORNL-TN Python Interface

APPENDIX B. ORNL-TN PYTHON INTERFACE

ORNL-TN Python Interface

October 15, 2018

Scott W. Mosher and Stephen C. Wilson

B-1. INTRODUCTION

ORNL-TN is a modified version of MCNP5-1.60 that was first developed to improve the scalability and performance of the software. It also provides a Python interface to a limited set of I/O facilities that, for example, allow additional materials and tallies to be defined during problem initialization and allow tally results to be accessed after transport is complete. The scope of the current set of functionality was determined by the requirement to enable the use of ORNL-TN with MSX to perform cell-wise material depletion over an arbitrary number of steps. This document describes the constants, objects, functions, and classes defined by the ORNL-TN Python interface.

The file name of a user-provided Python script can be specified using the `python_script_filename` keyword in the ORNL-TN `config` file. Within the user script, the Python API is accessed by importing the `ornltn` module; for example:

```
import ornltn
```

Interface entities can then be accessed as attributes of the `ornltn` module; for example:

```
print ornltn.GetTallyIds()
```

B-2. EXECUTION SEQUENCE POINTS

ORNL-TN executes the user-supplied Python script early in the MCNP run before the problem input file has been read and processed. Any code in the script that is defined outside a function or class will be executed at this time. Some of the API functions provide access to data that are only available after problem initialization or after the transport simulation is completed. These interfaces must be accessed by callback functions that are called by ORNL-TN at specific points in the execution sequence.

A typical MCNP run executes three high-level code sections:

- `imcn`: Problem initialization
- `xact`: Cross section I/O and processing
- `mcrun`: Monte Carlo transport

Because data are initialized and processed at different times, the interface defines several sequence points that correspond to the beginning and end of two of these sections: `IMCN_START`, `IMCN_END`, `MCRUN_START`, and `MCRUN_END`. User-defined functions can be registered to be called automatically when a given sequence point is reached using the `AddCallback()` function; for example:

```
ornltn.AddCallback(my_function, IMCN_END)
```

In this case, ORNL-TN will call `my_function` after problem initialization is complete but before cross section processing begins. The function will be called with no arguments, and any value returned by the function will be discarded.

Section B-6, “API Classes” describes several input card classes that must be constructed and added to the problem specification at `IMCN_START`. API functions that return cell ids, material ids, tally ids, or material compositions must not be called before `IMCN_END` is reached. Functions that return tally mean values must not be called until `MCRUN_END` is reached.

B-3. API CONSTANTS

`NEUTRON`
`PHOTON`

Particle type identifiers. For example, these can be used when constructing an `F4card` object.

`IMCN_START`
`IMCN_END`
`MCRUN_START`
`MCRUN_END`

Execution sequence point identifiers. These constants are used with the `AddCallback()` function to specify when the given function should be called.

B-4. API OBJECTS

`OrigenNuclides`

A listing of Origen nuclide identifiers stored as a tuple of tuples. For the i-th nuclide, `OrigenNuclides[i]` is the tuple `(S, I, ZZZ, AAA, SIZZZAAA)`, where:

- `S` is 1 for light nuclide, 2 for actinide, or 3 for fission product;
- `I` is the metastable state index (0 for stable isotopes);
- `ZZZ` is the nuclide proton number;
- `AAA` is the nuclide mass number; and
- `SIZZZAAA` is an eight-digit concatenation of the previously listed values.

B-5. API FUNCTIONS

`AddCallback(func, loc)`
Registers the callback function `func` to be called at the execution sequence point `loc`. The given function must be callable without any arguments. Any value returned from the call to `func` will be discarded.

`GetCellIds()`, `GetMaterialIds()`, and `GetTallyIds()`
Return a list of the cell, material, or tally IDs defined in the problem.

`GetCellAtomDensity(id)`
Returns the total atom density in atoms/b-cm associated with the given cell ID.

`GetCellMaterial(id)`
Returns the material ID associated with the given cell ID.

`GetMaterial(id)`
Returns the tuple (`zaids`, `fractions`) for the given material ID, where `zaids` is a list of ZZZAAA numbers (e.g., `92235`) and `fractions` is a corresponding list of atom densities in atoms/b-cm.

`GetMaterialMap()`
Returns a list containing the material ID associated with each cell.

`GetTallyDims(id)`
Returns a tuple of the bin dimensions of the tally with given ID. All non-mesh tallies have eight bin dimensions:

1. cell, surface, or detector bins;
2. all vs. flagged or all vs. direct;
3. user bins;
4. segment bins;
5. multiplier bins;
6. cosine bins;
7. energy bins; and
8. time bins.

For example, the dimensions returned for an F4 tally with 237 energy bins and no multipliers, segments, flagging, or other bins would be: `(1, 1, 1, 1, 1, 1, 237, 1)`.

`GetTallyMeans(id, icell=(0,1), imult=(0,1), ierg=(0,1))`
Returns a list of tally mean values corresponding to the given lower and upper index bounds for the cell (or surface or detector), multiplier, and energy bin dimensions. Indices start from zero and exclude the upper bound value (Python or C-style indexing). For access to all eight tally bin dimensions, use the `GetTallyMeansEx()` function.

`GetTallyMeansEx(id, ilo, ihi)`
Returns a list of tally mean values corresponding to the eight-dimensional lower and upper index bounds given by `ilo` and `ihi`, respectively. Indices start from zero and exclude the upper bound value (Python or C-style indexing).

For example, the call:

```
GetTallyMeansEx(id, (0,0,0,0,0,0,0,0), (1,1,1,1,1,1,237,1))
```

is equivalent to:

```
GetTallyMeans(id, ierg=(0,237))
```

`GetTallyRelErrors(id, icell=(0,1), imult=(0,1), ierg=(0,1))`
Like `GetTallyMeans()`, except that it returns tally relative errors.

`GetTallyRelErrorsEx(id, ilo, ihi)`
Like `GetTallyMeansEx()`, except that it returns tally relative errors.

`SetCellDensity(cell_id, density)`
Sets the total atom density in atoms/b-cm (if `density` > 0) or the mass density in g/cm^3 (if `density` < 0) associated with the given cell ID, overriding the value given on the corresponding cell card.

`SetCellMaterial(cell_id, matl_id)`
Sets the material ID associated with the given cell ID, overriding the value given on the corresponding cell card.

`SetDebugFlag(flag)`
Enable or disable extra interface checks. Extra checks are performed by default but can be disabled to improve performance, if needed.

B-6. API CLASSES

`class Ecard`
Tally energy (`E`) card.

`ECard(id, energies)`
Constructs an `ECard` object with the given ID and energy bins. The `id` parameter must be greater than zero and correspond to a tally ID or be equal to zero, in which case it applies to all tallies unless overridden. The `energies` parameter must be a sequence (e.g., list or tuple) of nonnegative energy values in units of MeV in increasing order.

`.AddToMCNP()`
Add the card to MCNP's data structures as if it had been read from the input file.

`.Validate()`
Check the card parameters and report any errors. This function is called automatically by `AddToMCNP()` when extra interface checking is enabled.

`.id`
The card ID value.

`.energies`
List of bin energies.

`class F4Card`
Cell-average flux tally (`F4`) card.

`F4Card(id, particle, cells)`
Constructs an `F4Card` object with the given ID, particle type, and cell bins. The `id` parameter must be greater than zero, and `id % 10` must be equal to 4. The `cells` parameter must be a sequence of at least one cell ID.

`.AddToMCNP()`
Add the card to MCNP's data structures as if it had been read from the input file.

`.Validate()`
Check the card parameters and report any errors. This function is called automatically by `AddToMCNP()` when extra interface checking is enabled.

`.id`
The card ID value.

`.particle`
Particle type identifier.

`.cells`
List of tally cells.

`class FMCard`
Tally multiplier (`FM`) card.

`FMCard(id)`
Constructs an `FMCard` object with the given ID. The `id` parameter must be greater than zero and correspond to a tally ID. No bins are defined initially. Multiplier sets can be added using the `AddMultiplierSet()` method.

`.AddMultiplierSet(C, m, MT)`
Add a multiplier set with multiplicative constant `C`, material ID `m`, and ENDF or special reaction number `MT`. Each multiplier set created this way generates a single additional multiplier bin.

`.AddToMCNP()`
Add the card to MCNP's data structures as if it had been read from the input file.

`.Validate()`
Check the card parameters and report any errors. This function is called automatically by `AddToMCNP()` when extra interface checking is enabled.

`.id`
The card ID value.

`.bins`
List of multiplier sets.

`class MCard`
Material (`M`) card.

`MCard(id, nuclides, fractions, nlib=None, plib=None, pnlib=None)`
Constructs an `MCard` object with the given ID, nuclides, and fractions. The `id` parameter must be greater than zero. The `nuclides` parameter must be a sequence of nuclide identifiers (see next paragraph). The `fractions` parameter must be a corresponding sequence of weight (if negative) or atom (if positive) fractions.

Nuclide identifiers can be specified either as a ZZZAAA value (e.g., `92235`) or as a tuple consisting of a ZZZAAA value and a library identifier string of the form "`nnX`," where `nn` is a two-digit number, and `X` is `c`, `p`, or `u`, (e.g., (`92235, "70c"`)).

Library identifiers for all nuclides can alternatively be specified by setting any of the optional `nlib`, `plib`, and `pnlib` arguments to a string in "`nnX`" form.

`.AddToMCNP()`
Add the card to MCNP's data structures as if it had been read from the input file.

`.Validate()`
Check the card parameters and report any errors. This function is called automatically by `AddToMCNP()` when extra interface checking is enabled.

`.id`
The card ID value.

`.nuclides`
List of nuclide identifiers.

`.fractions`
List of weight or atom fractions.

`.nlib`
`.plib`
`.pnlib`
Neutron, photon, and photoneutron library identifier strings or `None`.

`class MTCard`
S(α, β) material (`MT`) card.

`MTCard(id, mtlib1[, mtlib2, ...])`
Constructs an `MTCard` object with the given ID and S(α, β) table identifier string(s) (e.g., `"lwtr.10t"`). At least one `mtlib` parameter must be given.

`.AddToMCNP()`
Add the card to MCNP's data structures as if it had been read from the input file.

`.id`
The card ID value.

`.table_names`
List of S(α, β) table identifier strings.

APPENDIX C. New Features in ADVANTG-3.2

APPENDIX C. NEW FEATURES IN ADVANTG-3.2

New Features in ADVANTG-3.2

June 1, 2019

Scott W. Mosher, Seth R. Johnson, and Aaron M. Bevill

C-1. INTRODUCTION

The ADVANTG variance reduction generator [C-1] has been upgraded with new capabilities. The v3.2 release of the software adds the capability to correctly estimate spatial biased probabilities for SDEF sources that use rejection from multiple geometry cells. Tally support has been extended to include cylindrical mesh tallies at any orientation. Reflective boundary conditions for discrete ordinates calculations can now be specified in the ADVANTG input file. In Denovo, the algorithm used to calculate uncollided fluxes has been improved to provide much better accuracy in the vicinity of the point source.

The v3.2 release also includes significant performance improvements. A thread-parallel ray-tracing and source-sampling capability has been added to reduce the time needed to set up the deterministic calculations. An algorithm for thinning the collection of adjoint source spectra generated in global FW-CADIS calculations has been implemented to reduce memory usage in the Denovo calculations. The interface between ADVANTG and Denovo has been completely reworked. A new option was added to output wwinp files in a more compact binary format for use with the ORNL-TN upgrade [2] to MCNP5-1.60 [3] included in this distribution.

C-2. MULTICELL REJECTION SUPPORT

The ADVANTG software provides the capability to generate space- and energy-dependent biased source probabilities. In the MC simulations, these parameters ensure that source particles are preferentially sampled in important regions of the phase space. Complications arise when cell rejection is used with spatial source biasing. Whenever biased probabilities are employed, the particle weight must be adjusted by the ratio of the unbiased to the biased probability to preserve tally mean values. The fundamental issue with using rejection and biasing is that the weight adjustments applied for the biasing do not account for the rejection process. This is because the information needed to apply the proper weight adjustments (i.e., rejection probabilities) is typically not known *a priori*.

When rejection from a single geometry cell is used with spatial source biasing, the expected source particle weight is generally no longer equal to one. Because this affects the normalization of all tally results, a correction must be applied to the starting particle weight to obtain unbiased results. The previous release of ADVANTG v3.0.3 includes the capability to estimate this correction and its associated statistical uncertainty. The number of samples used to estimate the correction (`mcnp_num_wgt_samples`) can be set to reduce the uncertainty to an acceptable level. In the MCNP simulation, the correction is applied by modifying the WGT parameter on the SDEF card.

When rejection from multiple cells is used with spatial source biasing, the expected weight of a source particle is no longer equal to one (as before), and the fraction of source particles sampled from spatial bins is generally incorrect. To obtain unbiased tally results, corrections must be applied to both the starting particle weight and the unbiased probability of sampling a cell. An algorithm for estimating these corrections [C-4] has been developed, tested, and implemented in the v3.2 release. When rejection from multiple cells is used, ADVANTG will automatically estimate and output the corrections as a modified WGT parameter on the SDEF card and modified cell SP cards (i.e., modified unbiased probabilities of sampling the source cell). As in the previous version, the number of samples used to estimate these corrections can be modified to reduce uncertainties to an acceptable level by setting the value of `mcnp_num_wgt_samples`.

C-3. CYLINDRICAL MESH TALLY SUPPORT

Support for generating variance reduction parameters for cylindrical mesh tallies was added in the v3.2 release. The IDs (i.e., FMESH numbers) of cylindrical mesh tallies may now be listed on the `mcnp_tallies` input card.

MCNP constrains the definition of the cylindrical tally mesh so that the radial mesh must have a lower limit of zero and the theta mesh must have lower and upper limits of zero and one (in revolutions), respectively. ADVANTG v3.2 adds several input options, which are described as follows, to allow the modified cylindrical region of interest to be defined.

`mcnp_tally_min_radius`
`mcnp_tally_max_radius`

`int  real >= 0 [int  real >= 0 ...]`
`int  real >= 0 [int  real >= 0 ...]`

Specifies the minimum and maximum radii (cm) to use when constructing adjoint sources for cylindrical mesh tallies. Arguments are pairs of cylindrical mesh tally IDs and radial coordinates.

For example:
mcnp_tally_min_radius 4 25.0 14 500.0
mcnp_tally_max_radius 4 75.0 14 1,000.0

Sets the minimum radii of mesh tallies 4 and 14 to 25 and 500 cm, respectively, and the maximum radii to 75 and 1,000 cm.

`mcnp_tally_min_theta`
`mcnp_tally_max_theta`

`int  real >= 0 [int  real >= 0 ...]`
`int  real >= 0 [int  real >= 0 ...]`

Specifies the minimum and maximum theta values (in revolutions) to use when constructing adjoint sources for cylindrical mesh tallies. Arguments are pairs of cylindrical mesh tally IDs and theta coordinates.

For example:
mcnp_tally_min_theta 4 0.5 14 0.25
mcnp_tally_max_theta 4 1 14 0.75

Sets the minimum theta coordinate of mesh tallies 4 and 14 to 0.5 and 0.25 revolutions, respectively, and the maximum theta values to 1 and 0.75 revolutions.

C-4. BOUNDARY CONDITION OPTIONS FOR DENOVO

The v3.2 release adds an input option to set reflective boundary conditions for the Denovo discrete ordinates calculations. This was not an option in the previous release, and vacuum conditions were applied to all boundaries.

`denovo_reflect` **`int(6)`**

Specifies external boundary conditions for the discrete ordinates calculation in the following order: -*x*, +*x*, -*y*, +*y*, -*z*, and +*z*. A value of 1 specifies a specular reflective (symmetry) boundary condition, whereas 0 specifies a vacuum boundary.

Example:

```
denovo_reflect  1 0  1 0  1 0
```

The example sets reflective boundary conditions on the -*x*, -*y*, and -*z* boundaries and vacuum boundaries on the +*x*, +*y*, and +*z* boundaries.

C-5. MULTITHREADING

Thread-parallel algorithms for the ray-tracing and source-sampling steps were developed, implemented, and tested in the v3.2 release to take advantage of multicore architectures. Multithreading is activated by using the `-t` or `--threads` option on the command line. By default, a single thread of execution is used. For example, the command lines:

```
$ advantg --threads=8 INPUT_FILE $ advantg -t8 INPUT_FILE
```

equivalently specify the use of eight threads for the compute-intensive mapping operations that ADVANTG performs. There is generally no benefit to allocating more threads than the number of processing cores available on the host platform.

C-6. BINARY WWINP OUTPUT

For very large weight-window parameter sets, the time for ADVANTG to write and for MCNP to read the text-format wwinp file can be large, particularly on networked file systems. For this reason, an option was added to write the weight-window file in a compact binary format. This format is supported only by the ORNL-TN upgrade to MCNP5-1.60. Binary wwinp output is activated using the following input card:

`mcnp_binary_wwinp` **`bool`**

If true, ADVANTG will write the wwinp file in a binary format for use by ORNL-TN. If false, it will write the wwinp file in the standard text-based format.

C-7. REFERENCES

[C-1] S. W. Mosher, S. R. Johnson, A. M. Bevill, A. M. Ibrahim, C. R. Daily, T. M. Evans, J. C. Wagner, J. O. Johnson, and R. E. Grove, *ADVANTG—An Automated Variance Reduction Parameter Generator*, ORNL/TM-2013/416 Rev. 1, Oak Ridge National Laboratory, 2015. https://doi.org/10.2172/1210162.

[C-2] S. W. Mosher and S. C. Wilson, "Algorithmic Improvements to MCNP5 for High-Resolution Fusion Neutronics Analyses," *Fusion Science and Technology* 74,no. 4 (2018): 263–276. https://doi.org/10.1080/15361055.2018.1496691.

[C-3] X-5 MONTE CARLO TEAM, *MCNP—A General Monte Carlo N-Particle Transport Code, Version 5*, LA-UR-03-1987, Los Alamos National Laboratory, Los Alamos, New Mexico, 2008.

[C-4] S. W. Mosher and A. M. Bevill, "Estimating Biased Source Probabilities over Arbitrary Bins," *20th Annual Topical Meeting of the American Nuclear Society Radiation Protection and Shielding Division*, Santa Fe, New Mexico, August 26–31, 2018.

APPENDIX D. MSX—Utilities for Deterministic, Monte Carlo, and Hybrid Activation Analysis

APPENDIX D. MSX—UTILITIES FOR DETERMINISTIC, MONTE CARLO, AND HYBRID ACTIVATION ANALYSIS

MSX—Utilities for Deterministic, Monte Carlo, and Hybrid Activation Analysis

June 1, 2019

Scott W. Mosher and Stephen C. Wilson

D-1. INTRODUCTION

MSX [D-1] is a suite of utilities that implement the MS-CADIS method [D-2] for accelerating hybrid deterministic/MC simulations of post-activation biological dose rate analysis. The suite was developed to work with the ADVANTG [D-3] and MCNP [D-4] codes. MSX uses the ORIGEN solver [D-5] from the SCALE code system [D-6] to perform activation calculations.

The functionality that MSX provides is implemented by several executables:

- `msx_load`: Loads input data from ADVANTG run directories and MCNP `meshtal` files.
- `msx_load_rzt`: Generates geometry data and loads mesh tally data on cylindrical meshes.
- `msx_activate`: Performs voxel-by-voxel activation calculations using ORIGEN.
- `msx_post`: Postprocesses activated number densities.
- `msx_gen_kernel`: Generates linearized activation kernels.
- `msx_apply_kernel`: Applies activation kernels to fluxes.
- msx_deplete: Performs cell-by-cell depletion calculations using ORIGEN and ORNL-TN.

The details of each executable are described in the following sections.

D-2. MSX_LOAD

The `msx_load` utility gathers data from an ADVANTG run directory, and optionally an MCNP `meshtal` file, and writes the data to an HDF5 file for use by `msx_activate` and `msx_apply_kernel`.

To execute `msx_load` from the command line:

```
$ msx_load -h|--help
$ msx_load SCRIPT
```

where *`SCRIPT`* is the file name of a Python script that uses the `msx_load` API to direct the gathering of data. The script can contain any valid Python constructs (e.g., if branches, user-defined functions).

The `msx_load` API consists of the functions listed as follows. The functions direct which data should be loaded for writing to the HDF5 file. The data are written once the script is terminated.

`SetAdvantgRunDirectory(`*`path`*`)`

> Sets the path of an existing directory where ADVANTG has been previously executed. This path is used to locate spatial mesh, material composition, and deterministic flux data.

`SetMeshtalFile(`*`file name`*`)`

> Sets the file name of an existing MCNP `meshtal` file. This path is used to locate Monte Carlo flux data. Two formats of mesh tally files are supported: the plaintext format generated by unmodified versions of MCNP and the HDF5-format files generated by the ORNL-TN patch to MCNP5.

`SetHDF5File(`*`file name`*`)`

Sets the file name of the HDF5 output file. If this function is not called, then the default file name (`msx.h5`) is used.

`LoadAdvantgMatls([`*`include_only`*`])`

Loads spatial mesh, material compositions, material mixing table, and material map from ADVANTG output files. The `SetAdvantgRunDirectory()` function should be called first to set the location of the data.

The optional `include_only` argument can be used to specify a nuclide by its ZA identifier (e.g., `27059` for ^{59}Co). If given, the material specification will include only this nuclide at its original locations and densities. This option is intended to support the analysis of the linearized kernel approximation and is not intended for production use.

`LoadAdvantgFlux(`*`mode, particle`*`)`

Loads the discrete ordinates scalar flux field for the given transport mode and particle from ADVANTG output files. The`SetAdvantgRunDirectory()` function should be called first to set the location of the data.

The `mode` argument can be one of two string literals: `"fwd"` to load forward-mode fluxes or `"adj"` to load adjoint-mode fluxes. Similarly, `particle` can be either `"n"` for neutron fluxes or `"p"` for photon fluxes.

`LoadAdvantgAdjFlux(`*`particle`*`)`

Equivalent to LoadAdvantgFlux("adj", particle).

`LoadAdvantgFwdFlux(`*`particle`*`)`

Equivalent to LoadAdvantgFlux("fwd", particle).

`LoadMeshtalFlux(`*`tally_id`*`)`

Loads the space- and energy-dependent MC scalar flux field from an MCNP5 `meshtal` file. The `SetMeshtalFile()` function should be called first to set the location of the data. The `tally_id` argument is an integer specifying the mesh tally number (e.g., `4`).

D-3. MSX_LOAD_RZT

The `msx_load_rzt` utility is the counterpart of `msx_load` when performing calculations on cylindrical (rr, zz, $\theta\theta$) geometry meshes. However, instead of gathering discretized geometry data from an ADVANTG run directory, `msx_load_rzt` maps an MCNP geometry model onto a cylindrical mesh using ray-tracing. Discretized geometry and material composition information can be combined with a multigroup neutron flux distribution from an ASCII-format cylindrical MCNP mesh tally (`meshtal`) file to generate an HDF5 input file for use with `msx_activate`.

To execute `msx_load_rzt` from the command line:

```
$ msx_load_rzt [OPTIONS] INPUTFILE
```

where *`INPUTFILE`* is the name of a plaintext input file in `libconfig` format (see http://www.hyperrealm.com/libconfig/libconfig_manual.html#Configuration-Files for details). The following command line options are available:

`-h|--help` Print help and exit
`-t N|--num-threads=N` Number of execution threads when ray tracing

The `msx_load_rzt` input options are described as follows:

`runtpe_input = string`

File name of a runtpe file generated by MCNP5-1.60. The file can be generated from an MCNP input file using the command `mcnp5 ix i=INPUT`

For example: `runtpe_input = "./stlwtr/runtpe"`

`meshtal_input = string`

Optional file name of an ASCII-format mesh tally file generated by MCNP. The tally mesh **must be** identical to the mesh specified in the `msx_load_rzt` input file.

Example: `meshtal_input = "./stlwtr/meshtal"`

`meshtal_id = int`

Optional tally number of a cylindrical geometry neutron flux mesh tally listed in the `meshtal_input` file.

Example: `meshtal_id = 4`

`hdf5_output = string`

File name of the output HDF5 file. If mesh tally fluxes are provided, then the resulting output file can be used as input to `msx_activate` for performing activation calculations.

Example: `hdf5_output = "stlwtr.h5"`

`silo_output = string`

Optional file name of the output Silo file. If given, then the geometry and flux data are written to a Silo-format file for visualization using the VisIt software.

Example: `silo_output = "stlwtr.silo"`

`mesh_origin = (float, float, float)`

Optional location of the bottom center of the cylindrical mesh in the global rectangular coordinate system. This parameter has the same meaning as the ORIGIN keyword of the MCNP FMESH card.

Default: `(0, 0, 0)`

Example: `mesh_origin = (0, 0, -100)`

`mesh_axis = (float, float, float)`

Optional unit vector in the direction of the axis of the cylindrical mesh in the global rectangular coordinate system. This parameter has the same meaning as the AXS keyword of the MCNP FMESH card.

Default: `(0, 0, 1)`

Example: `mesh_axis = (0, 1, 0)`

`mesh_vec = (float, float, float)`

Optional unit vector that, along with the axis vector, defines the plane where $\theta\theta = 0$ in the global rectangular coordinate system. This parameter has the same meaning as the VEC keyword of the MCNP FMESH card.

Default: `(1, 0, 0)`

Example: `mesh_vec = (0, 0, 1)`

`mesh_r = (float, float[, ...])`

List of radial mesh coordinates in centimeters given in increasing order. This list of parameters has the same meaning as the IMESH keyword of the MCNP FMESH card.

Example: `mesh_r = (0, 10, 20, 50)`

`mesh_z = (float, float[, ...])`

List of axial mesh coordinates in centimeters given in increasing order. Each coordinate specifies a distance from `mesh_origin` along `mesh_axis`. This list of parameters has the same meaning as the JMESH keyword of the MCNP FMESH card.

Example: `mesh_z = (0, 100, 200, 300)`

`mesh_t = (`*`float, float`*`[, ...])`

List of azimuthal mesh coordinates in scaled units, given in increasing order. For convenience, each coordinate is scaled by the value of `mesh_t_factor` (see example). The final coordinate values specify angles in radians from the $\theta\theta = 0$ plane. This list of parameters has the same meaning as the KMESH keyword of the MCNP FMESH card.

Example: mesh_t = (0, 0.25, 0.5, 0.75, 1)
mesh_t_factor = 6.28318530717959

`mesh_t_factor =` *`float`*

Azimuthal mesh conversion factor to radians. This factor is applied to each value given in the `mesh_t` list.

Default: `1`

Example: mesh_t = (0, 90, 180, 270, 360) # degrees
mesh_t_factor = 0.0174532925199433

`min_rays =` *`int`*

Optional minimum number of rays to trace when estimating material volume fractions within each cylindrical mesh cell.

If only a single material is found within the mesh cell after `min_rays` have been traced, then the cell is assumed to homogeneously contain the material. Otherwise, tracing continues until `max_rays` is reached (see example).

Default: `50`

Example: `min_rays = 75`

`max_rays =` *`int`*

Optional maximum number of rays to trace when estimating material volume fractions within each cylindrical mesh cell.

Default: `1000`

Example: `max_rays = 10000`

`max_lost =` *`int`*

Optional maximum number of lost rays before aborting the program. Default: `100`

Example: `max_lost = 250`

D-4. MSX_ACTIVATE

The `msx_activate` utility performs voxel-by-voxel activation calculations and writes end-of-scenario number densities to an HDF5 file for subsequent use by `msx_post`. The activation calculations can be performed in parallel on an arbitrary number of processors.

To execute `msx_activate` from the command line:

```
$ msx_activate [OPTIONS] INPUTFILE
$ mpiexec -np N msx_activate [OPTIONS] INPUTFILE
```

where *INPUTFILE* is the name of a plaintext input file in libconfig format. The following command line options are available:

-h|--help	Print help and exit.
-q|--silent	Run silently.
--debug	Print debug log files.
--limit=*N*	Maximum number of ORIGEN calculations per process.

The `msx_activate` input options are described as follows.

`hdf5_input = string`

File name of an HDF5 file generated by `msx_load` or `msx_load_rzt`.

Example: `hdf5_input = "../msx_load/stlwtr.h5"`

`hdf5_output = string`

File name of the output HDF5 file. The end-of-scenario number densities will be written to this file for subsequent use by `msx_post`.

Example: `hdf5_output = "stlwtr_nd.h5"`

`min_number_density = float`

Output threshold in atoms/(barn·cm). No data will be output for nuclides that have number density values that are everywhere less than this value.

IMPORTANT NOTE: This setting directly affects the data available to `msx_post`. Setting the threshold too high might, for example, eliminate important contributions to gamma source fields.

Default: `1e-20`

Example: `min_number_density = 1e-15`

`output_nuclides = (`*`int`*`[, ...])`

Optional list of nuclides (as ZA numbers) to output. If this list is given, then all nuclides that are not listed will be omitted from output. The `min_number_density` threshold is still applied to the nuclides given in this list.

IMPORTANT NOTE: This setting directly affects the data available to `msx_post`. Limiting the number of output isotopes may result in very inaccurate estimates of per-voxel mass.

Example: `output_nuclides = (1003, 6014)`

`time_step_factor = `*`float`*

Time step conversion factor to seconds. This factor is applied to each value given in the `time_steps` list.

Default: `1`

Example: `time_step_factor = 86400 # time_steps given in days`

`time_steps = (`*`float`*`[, ...])`

Duration of each time step in the irradiation scenario. If `time_step_factor` is not given, then the values should be provided in units of seconds.

Example: `time_steps = (3920, 400, 3920, 400)`

`flux_factor = `*`float`*

Scaling factor to be applied to each value given in the `flux_step_factors` list.

Default: `1`

Example: `flux_factor = 100`

`flux_step_factors = (`*`float`*`[, ...])`

Flux multiplier for each time step in the irradiation scenario. The number of values given must be the same as in the `time_steps` list. The values are scaled by the `flux_factor` value.

Example: `flux_step_factors = (0, 1e11, 0, 1e11)`

D-5. MSX_POST

The `msx_post` utility reads number density data from an HDF5 file generated by `msx_activate` and performs user-directed postprocessing of the data to calculate, for example, gamma sources, activities, and masses.

To execute `msx_post` from the command line:

```
$ msx_post -h|--help
$ msx_post SCRIPT
```

where *`SCRIPT`* is the file name of a Python script that uses the `msx_post` API to direct the processing of data. The script may contain any valid Python constructs (e.g., if branches, user-defined functions).

The `msx_post` API consists of the classes and functions listed as follows.

`class HDF5File`

An HDF5 file wrapper for reading and writing field data.

`HDF5File(`*`filename`*`[, `*`mode`*`])`
Constructs an `HDF5File` object with the given *filename*. If the optional `mode` argument is given, then it must be either "`r`" or "`w`" for read- or write-mode access, respectively.

`.filename`
A string containing the file name.

`.mode`
A string containing the file access mode.

`.nuclides`

If opened for read access, then a tuple of nuclides for which data are available; otherwise, `None`.

`.mesh`
`Mesh` object for all fields in the file or `None`.

`class Mesh`

The attributes of a 3D rectangular or cylindrical structured spatial grid, depending on which type of mesh is stored in the HDF5 file. Objects of this class are constructed as attributes of `HDF5File` objects.

`.nx`, `.ny`, `.nz`
For rectangular meshes, the number of intervals in the x, y, and z dimensions, respectively.

`.nr`, `.nz`, `.nt`
For cylindrical meshes, the number of intervals in the r, z, and θ dimensions.

`.nxr`, `.nyz`, `.nzt`
The number of intervals in the x, y, and z or r, z, and θ dimensions.

`.num_voxels`
Number of mesh voxels.

.x, .y, .z
For rectangular meshes, tuple of x, y, and z dimension mesh boundaries in centimeters.

`.r`, `.z`, `.t`
For cylindrical meshes, tuple of r, z, and $\theta\theta$ dimension mesh boundaries in centimeters or radians.

`.xr`, `.yz`, `.zt`
Tuple of x, y, and z or r, z, and $\theta\theta$ dimension mesh boundaries in centimeters or radians.

`.origin`
For cylindrical meshes, tuple of the coordinates of the mesh origin in centimeters.

`.axis`
For cylindrical meshes, tuple of axis unit vector components.

.vec
For cylindrical meshes, tuple of $\theta = 0$ reference unit vector components.

`class Groups`

Multigroup energy boundaries.

`Groups(`*`bounds`*`[, `*`factor`*`])`
Constructs a `Groups` object with the given sequence of energy group boundaries. The boundaries must be listed in order of decreasing energy. The optional `factor` argument is a conversion factor to MeV (default = `1`).

`.num_groups`
The number of energy groups.

`.bounds`
A tuple containing the energy group boundaries, in MeV, in order of decreasing energy.

`class Field`

Base class for space- and possibly energy-dependent scalar fields. Objects of this class are not constructed directly. Derived classes inherit the attributes and methods of this base class.

`.mesh`
A `Mesh` object describing the spatial mesh over which the field is defined.

`.groups`
For energy-dependent fields, a `Groups` object describing the energy groups over which the field is defined. For energy-independent fields, this attribute is `None`.

`scale(`*`factor`*`[, `*`new_units`*`])`
Scale the field values by the given numeric factor. If `new_units` is given, then the `units` attribute is replaced by the given string.

`(i, j, k)`
For energy-dependent fields, returns the field value at the given x, y, z or r, z, θ mesh indices.

`(i, j, k, g)`
For energy-dependent fields, returns the field value at the given x, y, z or r, z, θ mesh indices and given energy group index.

`class ActivityField(Field)`

Field of radionuclide activities.

`ActivityField(`*`h5file`*`[, `*`nuclides`*`])`
Reads and constructs an `ActivityField` object given an `HDF5 File` object. The activities are calculated in units of Becquerel (Bq). By default, a total activity will be calculated by summing the activities of all nuclides for which data are available in the HDF5 file. The nuclides over which the summation is performed can be specified by providing a sequence of one or more ZA numbers as the optional `nuclides` argument.

`class GammaSourceField(Field)`

Field of gamma emission source intensities.

`GammaSourceField(`*`h5file, groups`*`[, `*`nuclides`*`])`
Reads and constructs a `GammaSourceField` object given an `HDF5File` object and a `Groups` object containing photon energy group boundaries. The gamma sources are calculated in units of gammas/(cm^3·s). By default, a total source will be calculated by summing the emission sources associated with all nuclides for which data are available in the HDF5 file. The nuclides over which the summation is performed can be specified by providing a sequence of one or more ZA numbers as the optional `nuclides` argument.

`write_nagss()`
Write the gamma source fields to input files for NAGSS.

`class MassField(Field)`

Field of material masses.

`MassField(`*`h5file`*`[, `*`nuclides`*`])`
Reads and constructs a `MassField` object given an `HDF5File` object. The masses are calculated in grams (g). By default, a total mass will be calculated by summing the masses of all nuclides for which data are available in the HDF5 file. The nuclides over which the summation is performed can be specified by providing a sequence of one or more ZA numbers as the optional `nuclides` argument.

`class NumberDensityField(Field)`

Field of number densities.

`NumberDensityField(`*`h5file`*`[, `*`nuclides`*`])`
Reads and constructs a `NumberDensityField` object given an `HDF5File` object. The number densities are read and stored in units of atoms/(b-cm). By default, a total number density will be calculated by summing the number densities of all nuclides for which data are available in the HDF5 file. The nuclides over which the summation is performed can be specified by providing a sequence of one or more ZA numbers as the optional `nuclides` argument.

`class VolumeField(Field)`

Field of voxel volumes.

`VolumeField(`*`h5file`*`)`
Constructs a `VolumeField` object given an `HDF5File` object. The volumes are calculated in cubic centimeters (cm^3).

`class SiloFile`

A Silo file wrapper for writing field data for subsequent visualization using VisIt.

`SiloFile(`*`filename`*`)`
Constructs a `SiloFile` object with the given filename.

`.filename`
A string containing the file name.

`add_field(`*`field, varname`*`)`
Write the data contained in the given `Field` object to the Silo file with the given variable name string. Variable names in Silo might contain only alphanumeric characters and underscores and must begin with a letter.

`DivideFields(`*`numerator, denominator, units`*`)`

Construct a new field as the ratio *`numerator/denominator`* and set the `units` attribute to the given value. The field value will be zero in any element where `denominator` is zero. The *numerator* and *denominator* fields must both be either energy-independent or energy-dependent.

`MultiplyFields(`*`first, second, units`*`)`

Construct a new field as the product *`first * second`* and set the `units` attribute to the given value. The *first* and *second* fields must both be either energy-independent or energy-dependent.

`SumFields(`*`field1`*`[, `*`field2, ...`*`])`

Construct a new field as the sum *`field1`* + *`field2`* + ... All given fields must be either energy-independent or energy-dependent.

D-6. MSX_GEN_KERNEL

The `msx_gen_kernel` utility generates linearized activation kernels for the 173 nuclides in the FENDL3 multigroup library that produce gamma emissions as a result of exposure to neutrons. The generated kernels are calculated based on a user-provided activation scenario. The calculations can be performed in parallel on an arbitrary number of processors.

To execute `msx_gen_kernel` from the command line:

```
$ msx_gen_kernel [OPTIONS] INPUTFILE
$ mpiexec -np N msx_gen_kernel [OPTIONS] INPUTFILE
```

where *`INPUTFILE`* is the name of a plaintext input file in `libconfig` format. The following command line options are available:

`-h|--help`	Print help and exit.
`-q|--silent`	Run silently.

The `msx_gen_kernel` input options are described as follows.

`kernel_output = string`

File name of the kernel output file. The kernel matrices will be written to this file for subsequent use by `msx_apply_kernel`.

Default: "`msx_kernel.dat`"

Example: `kernel_output = "sa2_kernels.dat"`

`neutron_group_boundaries = (float, float[, ...])`

List of neutron energy group boundaries in eV. The bounds must be listed in order of decreasing energy.

`gamma_group_boundaries = (float, float[, ...])`

List of gamma emission group boundaries in MeV. The bounds must be listed in order of decreasing energy.

`time_step_factor = float`

Time step conversion factor to seconds. This factor is applied to each value given in the `time_steps` list.

Default: `1`

Example: `time_step_factor = 86400 # time_steps given in days`

`time_steps = (`*`float`*`[, ...])`

Duration of each time step in the irradiation scenario. If `time_step_factor` is not given, then the values must be provided in units of seconds.

Example: `time_steps = (3920, 400, 3920, 400)`

`flux_factor = `*`float`*

Scaling factor to be applied to each value given in the `flux_step_factors` list.

Default: `1`

Example: `flux_factor = 100`

`flux_step_factors = (`*`float`*`[, ...])`

Flux multiplier for each time step in the irradiation scenario. The number of values given must be the same as in the `time_steps` list. The values are scaled by the `flux_factor` value.

Example: `flux_step_factors = (0, 1e11, 0, 1e11)`

D-7. MSX_APPLY_KERNEL

The `msx_apply_kernel` utility uses precomputed linearized activation kernels to approximately calculate forward gamma emission sources or MS-CADIS adjoint neutron sources. The calculations can be performed in parallel on an arbitrary number of processors.

To execute `msx_apply_kernel` from the command line:

```
$ msx_apply_kernel [OPTIONS] INPUTFILE
$ mpiexec -np N msx_apply_kernel [OPTIONS] INPUTFILE
```

where *`INPUTFILE`* is the name of a plaintext input file in `libconfig` format. The following command line options are available:

`-h|--help` Print help and exit.
`-q|--silent` Run silently.

The `msx_apply_kernel` input options are described as follows.

`hdf5_input = `*`string`*

File name of an HDF5 file generated by `msx_load`.

Example: `hdf5_input = "../msx_load/stlwtr.h5"`

`kernel_file = string`

File name of a kernel file generated by `msx_gen_kernel`.

Example: `kernel_file = "../msx_gen_kernel/msx_kernel.dat"`

`denovo_hdf5_input = string`

File name of a Denovo HDF5 input file generated by ADVANTG. This file will be used to generate a new HDF5 input for Denovo containing the generated source distribution.

Example: `denovo_hdf5_input = "../../adj_solution/denovo_adjoint_input.h5"`

`denovo_hdf5_output = string`

File name of a new Denovo HDF5 input file that will contain the generated source distribution.

Default: `denovo-adjoint.inp.h5` (if `transpose = true`)
`denovo-forward.inp.h5` (if `transpose = false`)

Example: `denovo_hdf5_output = "../../adj_solution/denovo_adjoint_input_msx.h5"`

`transpose = bool`

If `true`, then transpose the kernel matrices and calculate MS-CADIS adjoint neutron sources. Otherwise, calculate forward gamma emission sources.

Default: `false`

`time_step_factor = float`

Time step conversion factor to seconds. This factor is applied to each value given in the `time_steps` list.

Default: `1`

Example: `time_step_factor = 86400 # time_steps given in days`

`time_steps = (float[, ...])`

Duration of each time step in the irradiation scenario. If `time_step_factor` is not given, then the values should be provided in units of seconds.

Example: `time_steps = (3920, 400, 3920, 400)`

`flux_factor = float`

Scaling factor to be applied to each value given in the `flux_step_factors` list.

Default: `1`

Example: `flux_factor = 100`

`flux_step_factors = (float[, ...])`

Flux multiplier for each time step in the irradiation scenario. The number of values given must be the same as in the `time_steps` list. The values are scaled by the `flux_factor` value.

Example: `flux_step_factors = (0, 1e11, 0, 1e11)`

D-8. MSX_DEPLETE

The `msx_deplete` utility performs cell-by-cell depletion calculations and writes end-of-scenario number densities to an HDF5 file for subsequent use by ORNL-TN. The depletion calculations can be performed in parallel on an arbitrary number of processors.

To execute `msx_deplete` from the command line:

$ `msx_deplete` [OPTIONS] *INPUTFILE*
$ `mpiexec` -np *N* msx_deplete [OPTIONS] *INPUTFILE*

where *`INPUTFILE`* is the name of a plaintext input file in `libconfig` format (see http://www.hyperrealm.com/libconfig/libconfig_manual.html#Configuration-Files for details). The following command line options are available:

-help |	Help and exit.
-q | --silent	Run silently.
--debug	Print debug log files.

The `msx_deplete` input options are described as follows.

`hdf5_input = string`

File name of an HDF5 file generated by ORNL-TN.

Example: `hdf5_input = "hfir-day0.h5"`

`hdf5_output = string`

File name of the output HDF5 file. The end-of-scenario number densities will be written to this file for subsequent use by ORNL-TN. The number densities of all isotopes that are defined in the ORIGEN library are written to the file.

Example: `hdf5_output = "hfir-day1.h5"`

`time_step_factor = float`

Time step conversion factor to seconds. This factor is applied to each value given in the `time_steps` list.

Default: `1`

Example: `time_step_factor = 86400 # time_steps given in days`

`time_steps = (float[, ...])`

Duration of each time step in the depletion/decay scenario. If `time_step_factor` is not given, then the values should be provided in units of seconds.

Example: `time_steps = (3920, 400, 3920, 400)`

`flux_factor = float`

Scaling factor to be applied to each value given in the `flux_step_factors` list.

Default: `1`

Example: `flux_factor = 100`

`flux_step_factors = (float[, ...])`

Flux multiplier for each time step in the depletion scenario. The number of values given must be the same as in the `time_steps` list. The values are scaled by the `flux_factor` value.

Example: `flux_step_factors = (0, 1e11, 0, 1e11)`

`gamma_group_boundaries = (float, float[, ...])`

List of gamma emission group boundaries in MeV. The bounds must be listed in order of decreasing energy.

D-9. REFERENCES

[D-1] S. C. Wilson, S. W. Mosher, K. E. Royston, C. R. Daily, and A. M. Ibrahim, "Validation of the MS- CADIS Method for Full-Scale Shutdown Dose Rate Analysis," *Fusion Science and Technology* 74, no. 4 (2018): 288–302 (2018). http://dx.doi.org/10.1080/15361055.2018.1483687.

[D-2] A. M. Ibrahim, D. E. Peplow, R. E. Grove, J. L. Peterson, and S. R. Johnson, "The Multi-Step CADIS Method for Shutdown Dose Rate Calculations and Uncertainty Propagation," *Nuclear Technology* 192, no. 3, (2015) 286–298. https://doi.org/10.13182/NT15-1.

[D-3] S. W. Mosher, S. R. Johnson, A. M. Bevill, A. M. Ibrahim, C. R. Daily, T. M. Evans, J. C. Wagner, J. O. Johnson, and R. E. Grove, *ADVANTG—An Automated Variance Reduction Parameter Generator*, ORNL/TM-2013/416 Rev. 1, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 2015. https://doi.org/10.2172/1210162.

[D-4] X-5 MONTE CARLO TEAM, *MCNP—A General Monte Carlo N-Particle Transport Code, Version 5*, LA-UR-03-1987, Los Alamos National Laboratory, Los Alamos, New Mexico, 2008.

[D-5] I. C. Gauld, G. Radulescu, G. Ilas, B. D. Murphy, M. L. Williams, and D. Wiarda, "Isotopic Depletion and Decay Methods and Analysis Capabilities in SCALE," *Nuclear Technology* 174, no. 2 (2011),169–195. https://doi.org/10.13182/NT11-3.

[D-6] S. M. Bowman, "SCALE 6: Comprehensive Nuclear Safety Analysis Code System," *Nuclear Technology* 174, no. 2 (2011) 126–148. https://doi.org/10.13182/NT10-163.