HxF User Manual: A Pebble Bed Reactor Simulation Tool

Ludovic Jantzen

April 7, 2025

Contents

1 Introduction

This document serves as a user manual for the Pebble Bed Reactor (PBR) simulation tool, HxF. The primary purpose of this tool is to perform high-fidelity simulations of PBRs, providing pebble-wise results. Unlike traditional zone-averaged approaches, HxF computes the neutron flux, power, and isotopic concentrations for each individual pebble within the reactor. Additionally, it depletes each pebble separately and manages pebble circulation.

The tool incorporates two motion models:

- Discrete Motion (DM): Simulates pebble movement by swapping compositions on a static lattice structure. This approach is particularly suited for simple geometries, such as the gFHR.
- Discrete Element Method (DEM): Incorporates continuum mechanics equations to compute the forces acting on each pebble, accounting for small deformations. This method is not included within HxF itself; instead, externally generated pebble positions must be provided as input.

HxF offers additional features, such as the ability to account for different pebble types and implement various discard criteria based on either the number of passes or the accumulated burnup.

This tool is particularly useful for determining the equilibrium state of a reactor while obtaining detailed pebble-wise lifecycle data.

The tool relies mainly on the MC code Serpent 2 [1] and especially a modified version to handle pebble geometry construction and specific domain decomposition for PBR. Moreover, the dynamic interaction with Serpent is fostered thanks to the Cerberus/Kraken Python framework. See more here: `https://serpent.vtt.fi/kraken/index.php?title=Main_Page`. These tools help to dynamically interact with Serpent and handle Serpent input variables as Python variables.

2 Installation and Dependencies

To use this tool, ensure you have the following dependencies installed:

- Python
- Required Python libraries:
  - pandas
  - numpy
  - matplotlib
  - Cerberus

The Cerberus library can be found on the cluster storage under the path:

`/global/home/groups/co_nuclear/HxF_tools/Cerberus_HxF`

The first step will be to clone the given GitHub repository. Clone the repository:

```
git clone https://github.com/ludojantzen/HxF.git
```

It's recommended to clone it in a location where the simulation will have enough memory to store results (such as `scratch` space for example)

3 Repository Structure

This section describes the organization of the repository, outlining the key directories and files that compose the simulation framework for the Pebble Bed Nuclear Reactor. The repository is structured to facilitate ease of use, modification, and extension of the simulation capabilities.

The HxF repository is organized as follows:

```
HxF/
 Input_scripts/
 Models/
 Source/
 Tools/
 Utils/
 HxF.py
 README.md
 config/
 launch_HxF
```

- `Input_scripts/`: Contains input scripts for simulations. It takes the form of a Python script providing the main simulation parameters such as the motion mode, the pebble types, and the discarding threshold. This will be decsribed in the section
- `Models/`: Stores various reactor models used in simulations, gFHR, HTR-10, small PBR.
- `Source/`: Includes the source code of the HxF tool. These files should be left as they are because they have been built to simulate and plot general PBR-related stuff.
- `Tools/`: Provides auxiliary tools and utilities. Useless for basic applications. Contains some tools for detection and ML purposes.
- `Utils/`: Contains utility scripts and functions. Really helpful for permanent environment setup and LSURM environment creation and variable exportation (Serpent environment, module loading). Please check the section 5.
- `HxF.py`: The main Python script to run the HxF tool. Please check section 8
- `README.md`: Documentation and overview of the HxF project.
- `config`: configuration file. It helps to configure the SLURM job (resource, computational time, partition... see section 9
- `launch_HxF`: Script to launch HxF simulations. Submit the job for you with all the needed info: Input scripts, environment... see section 10

Additionally, HxF while running will create a directory **Cases/** and a directory **Logs/**. These two directories will contain the outputs and the log files from Serpent and the cluster,

4 Code Structure and Simulation Loop

The HxF tool operates in a sequential loop to simulate the evolution of a Pebble Bed Reactor (PBR). Usually, the user will set a total number of iterations. In addition, the iteration corresponds to an actual time step. Each iteration consists of the following steps:

1. **Neutron Transport:** Compute the flux and power distribution for each pebble in a static bed.
2. **Burnup Calculation:** Deplete each pebble individually based on the previously computed neutron flux. The burnup step duration matches the motion step duration.
3. **Pebble Motion:** Move the pebbles according to the selected motion model (Discrete Motion or Discrete Element Method).
4. **Pebble Handling:** After the motion, the tool manages the discard, discharge, and reinsertion of pebbles. New pebbles are reinserted with random radial positions. You can also apply an artificial decay time corresponding to a potential

For instance, let say that your time step is 3 days between two iterations. It means that your burnup step will be 3 days and that your sequential motion step will correspond to three days of continuous motion.

This iterative process continues until you reach the predefined number of simulated steps. Some features propose to stop the simulator based on certain criteria such as the variation in the multiplication factor. The overall flexibility of the Python tool allows for you to define your own convergence criteria as long as it's related to a Serpent output value.

Generally, in figure 1, you can find the overall workflow for HxF. A thermal coupling block interacting with the transport calculation is pointed out and discussed in the Appendix B.

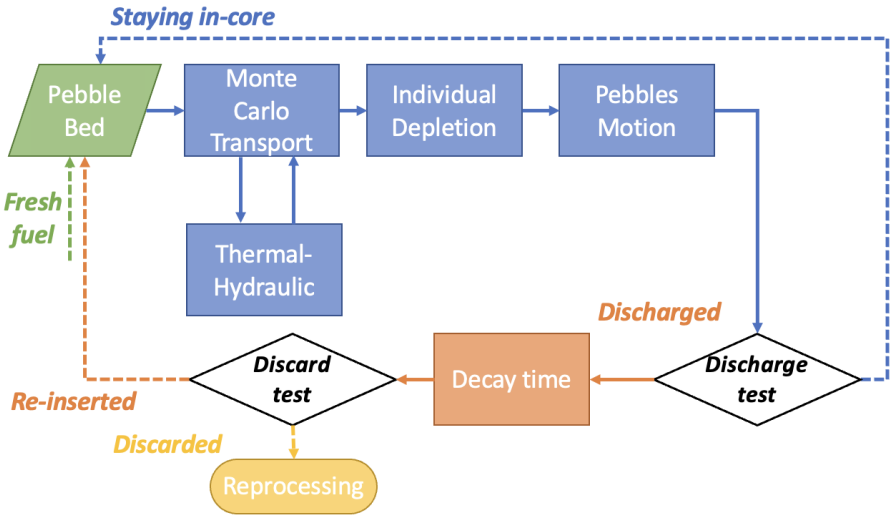

Figure 1: HxF workflow [3].

5 Setup and Environment Configuration

5.1 Setting Up Cerberus: The `setup_Cerberus` Script

The `setup_Cerberus` script is a crucial component for configuring the environment required to run the simulation tool. It ensures that all necessary dependencies are installed and properly configured before executing any simulations. This script should be executed once before using the tool for the first time.

5.1.1 Purpose and Impact

The script automates the setup of the computing environment, handling:

- Unloading conflicting software modules and loading required ones.
- Installing and configuring Miniconda if it is not already present.
- Creating a dedicated Conda environment for running the simulation.
- Installing necessary Python packages and dependencies.
- Ensuring compatibility with Cerberus and KrakenTools frameworks.

5.1.2 Step-by-Step Breakdown

The script follows these main steps:

1. Module Management To prevent software conflicts, the script first unloads any existing Python, GCC, CMake, and Nano modules before loading the correct versions required for the simulation.

2. Miniconda Installation If Miniconda is not found in the user's home directory, the script downloads and installs it. This ensures a controlled Python environment without affecting system-wide installations.

3. Creating a Custom Conda Environment The script configures a Conda environment named `kraken`, placing it in the scratch directory to optimize storage usage. It also configures Conda to store package installations in a designated location.

4. Installing Dependencies Within the new Conda environment, the script installs essential Python libraries such as:

- `serpentTools` for processing Serpent Monte Carlo outputs.
- `numpy, scipy, pandas` for numerical and data manipulation.
- `matplotlib, seaborn` for visualization.
- `mpi4py` for parallel computing.

5. Linking Cerberus and KrakenTools To ensure proper integration with the simulation framework, the script links the Cerberus and KrakenTools directories to the Conda environment.

6. Verification Finally, the script deactivates and reactivates the environment to confirm that Cerberus is correctly installed by running a simple import test.

5.1.3 Execution

To set up the environment, run:

```
bash ./Utils/setup_Cerberus
```

This should only be run once before using the simulation tool. If any updates are made to the environment, re-executing the script ensures a clean and correct setup.

5.2 Activating Kraken: The `sss_environment` Script

The `sss_environment` helps you to load the correct simulation for your coming simulation, export the good serpent path, and activate kraken, the environment used to run the searc_equilibrium tool.

5.3 Testing the environment and kraken activation

After sourcing the `sss_environment` script, you should be able to do a few checks. The first one will be to see what the current path related to the accessible Python module:

```
[ludovicjantzen@ln002 search_equilibrium]$ which python
/global/scratch/users/ludovicjantzen/conda/kraken/bin/python
```

Then by running accessing the python through the Kraken environment:

```
[ludovicjantzen@ln002 search_equilibrium]$ python
Python 3.11.11 | packaged by conda-forge |
(main, Mar  3 2025, 20:43:55) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cerberus
>>> cerberus
<module 'cerberus' from '/global/home/groups/co_nuclear/
HxF_tools/Cerberus_HxF/cerberus/__init__.py'>
>>>
```

You should be able to import Cerberus and access its location without error. This means that your Kraken environment has been set up correctly.

6 Input Scripts

This section explains the role of the input scripts. They take the form of a Python script where a few dozens of variables are defined. You can find a template under `HxF/Source /Default_Input.py`. It's also really valuable to spend time understanding the available examples of input scripts available under `Input_scripts`. Here you can find a quick explanation of all the options:

6.1 Calculation Options

- **transport**: Enables neutron transport calculations.
- **resolve_first**: If true, perform an initial resolution step.
- **correct**: Applies correction methods if enabled. Basically, turn on the basic linear pcc in Serpent. The use of this method is still highly debated in PBR simulation since we overlap this method with a motion scheme.
- **domain_decomposition**: Activates domain decomposition for parallel execution. Decompose based on the number of requested nodes.

6.2 Motion Settings

- **discrete_motion**: Enables discrete motion model (DM). If set to False, the DEM positions should be provided.
- **looping**: If true, enables looping motion mode. This technique is compatible with DEM calculation where HxF will match the pebble positions between the first and the last position data set. It helps to simulate long circulation sequences while simulating a handful series of positions

6.3 Restart Settings

- **restart_calculation**: If true, starts from a previously saved state. To restart the calculation, you'll need two types of files. The data frame contains the pebble positions named `core_step.csv` and the Serpent restart file containing the pebble composition. Usually, you restart an HxF calculation using previous HxF results.
- **read_first_compositions**: Reads initial compositions from predefined files for each pebble.

6.4 Output Settings

- **plotting**: Enables visual output of results. These plots are stored in the folder **Cases/your_case/** and are generated at each calculation step. See more in section 11.
- **saving**: Saves results to disk.

- **write_restart**: Enables writing restart files.

6.5 Case Information

- **path_to_case**: Directory where the Serpent input files are stored. Usually, it's stored under **Models/**.
- **case_name**: Name of the case. This name is informative and will be used to name the log file.
- **main_input_file**: Name of the main Serpent input file.

6.6 Serpent Input Data

- **pbed_file**: Pebble position file. This is particularly important when using the Discrete Motion (DM) model, where pebbles follow a structured lattice. In this case, the position file serves as a reference for the simulation, and the DM tool updates pebble compositions accordingly without altering their spatial arrangement.
- **r_pebbles**: Pebble radius (cm).

A crucial part of the input configuration is the `pebbles_dict`, which defines the properties of different pebble types. For example, the following dictionary describes the HTR-10 configuration:

```
pebbles_dict = {'u_fuel_pebble':{'mat_name':'fuel', 'pebbles_frac':0.57,
'r_fuel_kernel':0.025, 'Ntrisos':8335, 'threshold_type':'burnup',
'threshold_dir':+1, 'threshold':72},
            'u_graph_pebble':{'pebbles_frac':0.43}}
```

Each entry in this dictionary corresponds to a specific universe defined in the Serpent input (`u_fuel_pebble` and `u_graph_pebble`). The parameters for each universe are as follows:

- **mat_name**: Material name (only required for burnable pebbles).
- **pebbles_frac**: Fraction of this type of pebble in the core (e.g., in the HTR-10, 57% of the pebbles are fuel pebbles).
- **r_fuel_kernel**: Fuel kernel radius (cm).
- **Ntrisos**: Number of TRISO particles per fuel pebble (used to compute the fuel volume).
- **threshold_type**: Criterion for discarding pebbles (e.g., `burnup`, `passes`, or isotope concentration).
- **threshold_dir**: Direction of the threshold check (`+1` for exceeding the threshold, `-1` for falling below it).

- **threshold**: Numeric value defining the discard limit (e.g., fuel pebbles in the HTR-10 are discarded when their burnup exceeds 72 MWd/kgHM).

You can also add pebble-wise detectors with this format:

```
detectors = {'flux_pebbles_thermal':{'E':[1e-11, 1.86e-6],
'normalized':True},
         'flux_pebbles_fast':{'E':[0.18, 20], 'normalized':True},
         'power_pebbles':{'extra_cards':['dr', -8, 'void']}}
```

Moreover, HxF integrates an integration process deriving fluence from flux and energy from power (time integration over the entire pebble life-cycle).

Different threshold criteria can be used for pebble discard decisions, such as the number of passes through the core, accumulated burnup, or the mass fraction of specific isotopes (Cs-137). For a more comprehensive discussion of available discard criteria, refer to Section 11.

6.7 Transport Calc Settings

- **power_normalization_field**: Field used for power normalization.
- **power_normalization_value**: Total reactor power (W).
- **Nsteps**: Number of calulation steps. Usually, we use a high number of steps (~ 1000). The cluster time limit of 3 days will cut off this high number of steps but at least you're sure your simulation is as far as possible.
- **neutrons_per_cycle**: For each single simulation step, you can define the number of simulated neutrons. this number will be the same assigned to active and inactive cycle of a given HxF calculation step. Regarding the number of active and inactive cycles, these parameters are defined in the Serpent input in **Models/** with the classic card `set pop`.

  Since you define the number of neutrons for each computation step, `neutrons_per_cycle` should be a list with a length of `N_steps`. The common practice is to use a low number of neutrons to approach equilibrium and then refine when the equilibrium is reached. Example:

  ```
  neutrons_per_cycle=[20000]*68*2+[500000]*(Nsteps-68*2)
  ```

- **decay_step**: Decay step applied to every discharged pebble to simulate the off-core time. Most of the time is set to 0 because the recirculation time ends up being really short ($\sim$ 5 minutes).

6.8 Motion Parameters for DM

Discrete Motion (DM) follows predefined movement rules per step, controlling the pebble flow in the reactor. The key parameters for DM are:

- **motion_direction**: Defines the movement direction (+1 or -1).
- **Nrows_to_move**: Number of pebble rows moved per step.
- **time_per_pass**: Time per pass (days) in discrete motion mode. Similar to `neutrons_per_cycle`, this parameter is a list with a length equal to `N_steps`. It influences both the motion increment and the burnup step size. The time step is refined once equilibrium is reached to improve accuracy.

```
Nrows_to_move = [6]*68*2 + [3]*(Nsteps-68*2)
```

6.9 Motion Parameters for DEM

The Discrete Element Method (DEM) relies on externally generated position files, typically named `position_step.csv`, which must be accessible when launching HxF. The parameters controlling DEM-based motion include:

- **recirc_threshold**: Threshold for recirculation (cm). HxF detects pebble recirculation by comparing axial positions between consecutive DEM steps. If the absolute displacement exceeds this threshold, the pebble is considered to have recirculated.
- **positions_folder**: Directory containing DEM-generated position files.
- **DEM_step_increment**: Defines the step increment when reading position files. A value of 1 means all files are used, while 2 means every other file is used.
- **circulation_rate**: Pebble circulation rate (pebbles per day).
- **DEM_circulation_step**: Number of pebbles per DEM circulation step.
- **positions_scale**: Scaling factor for DEM positions.
- **positions_translation**: Translation vector for DEM positions. Since Serpent's reactor geometry must align with the DEM-generated positions, this translation ensures consistency.
- **looper_Nr, looper_Nz, looper_method**: Parameters for the looping method. Details on available looping methods can be found in `./Source/Looping.py`.

6.9.1 Example: HTR-10 Configuration

In this example, the DEM-generated positions cover 90 steps, with each step corresponding to a time increment of 12.5 days. The following input parameters apply:

```
recirc_threshold = 200 # cm
positions_folder = '/global/scratch/users/co_nuclear/pebble_positions_larger/'
DEM_step_increment = 1
circulation_rate = 125 # pebbles/day
DEM_circulation_step = 1564 # pebbles
positions_scale = 100
positions_translation = [0, 0, -610]

if looping:
    DEM_start = 40
    DEM_end = 129
    looper_Nr = 5
    looper_Nz = 10
    looper_method = 'rz'
```

Explanations: A large recirculation threshold is set to detect full pebble recirculations. In the HTR-10, recirculated pebbles move from $z = -610$ cm to $z = 0$ cm. The `circulation_rate` and `DEM_circulation_step` indicate that the DEM time step is around 12.5 days. Since the DEM outputs positions in meters, while Serpent expects centimeters, the `positions_scale` corrects for unit differences. Additionally, the DEM simulation spans $z = 0$ m to $z = 6.10$ m, whereas the Serpent geometry spans $z = -610$ cm to $z = 0$ cm, justifying the need for `positions_translation`. When looping is enabled, HxF applies a repeating pattern to DEM positions. The looping parameters are:

- **looper_Nr**: Number of radial divisions.
- **looper_Nz**: Number of axial divisions. Indeed, the looper assigned the positions per block that are built from the subdivision defined with **looper_Nz** and **looper_Nr**. More detail about the looping tool and the induced error is discussed in [2].
- **looper_method**: Defines how looping is applied. Available methods include:
  - **'rz'**: Minimize r and z distance between the old and new position.
  - **'xyz'**: Minimize the distance between the old and the new position.

Moreover, the position file is stored under the:
`/global/scratch/users/co_nuclear/pebble_positions_larger/` In this directory, the files take this format: `step0000077.csv`. '77' is obviously the step-index of the DEM simulation. One of the options you can add to your

input for instance is `DEM_files_prefix`, which in this case will be `step` (Also the default option if you do not set anything).

`HxF.py` script sorts for you the position files based on their index that comes after the `DEM_files_prefix` which step in this example.

6.10 Output and Data Storage

- **output_folder_name**: Directory where output files are stored in the directory **Cases/**. It's really important to set this name correctly and remember to change it to not overwrite the previous results.
- **verbosity_mode**: Level of verbosity for logging.
- **inventory_names**: List of isotopes tracked in inventory calculations. These isotopes will end up being stored at the pebble-wise level for each step in the `core_step.csv` file. For instance, this isotope inventory can be used as a discard threshold.

6.11 Domain Decomposition

- **allowed_decomposition_types**: Types of domain decomposition allowed.
- **max_domains**: Maximum number of subdomains.

More details are given in the section 13.3

6.12 Restart Writing

- **restart_write_every**: Frequency (in steps) for writing core composition restart files.
- **restart_discarded_write_every**: Frequency (in steps) for writing discarded composition restart files.
- **restart_discharged_write_every**: Frequency (in steps) for writing discharged composition restart files.

6.13 Restart Reading

- **restart_step**: Step number from which to restart.
- **restart_data**: Path to restart data file. It corresponds to the file core_step.csv.
- **restart_binary**: Path to restart binary file. It corresponds to the Serpent restart binary file.

6.14 Saving Options

- **write_global**: Saves global reactor data.
- **write_incore**: Saves data of pebbles inside the core.
- **write_reinserted**: Saves data of reinserted pebbles.
- **write_discarded**: Saves data of discarded pebbles.
- **write_discharged**: Saves data of discharged pebbles.

This detailed parameter list provides an overview of the key input settings in HxF. These options allow users to customize the simulation workflow, neutron transport, depletion, motion, and output settings.

6.15 Final note

We saw the main general options available. Obviously, if your options are not well spelled or do not exist, the script will return you an error. There is a section in the appendix that explains how to test your code before submitting your job. In addition, if you want to double-check the role of a specific parameter or its availability, the best is to look directly at the script `HxF.py`.

7 Models

This section describes the **Models** folder. It contains the Serpent input. Usually, it required the definition of the reactor geometry, and the reactor materials. Then, regarding the simulation parameters such as power normalization, neutron population, and burnup step..., all these parameters will be written down by the tool directly, or set as dynamic variables thanks to Cerberus. The process that HxF is doing is to first delete all the redundant information in the Serpent input, collect the necessary information such as reactor geometry, reference pebble positions for the DM tool, and material composition... and then add the needed simulation parameters derived from the Python input scripts.

The repository contains a few reactor models:

- the gFHR and some more complex models derived from the base model that is referenced here [4]. It's a simplified generic benchmark reactor for neutronic calculation featuring a purely cylindrical shape. The base model involves a DM tool with an FCC structure as a position reference. The gFHR core has been simulated also involving DEM position determination. Also, a model with a more complex geometry featuring conical regions and fueling/defueling chute under the directory **Models/gFHR_cone/**, also simulated with DEM tool.

- the HTR-10 which is a benchmark reactor built in China in 2000 to demonstrate the feasibility of helium-cooled PBR. The main reactor parameters can be found in [5]. This reactor features a complex geometry that requires to use of the DEM tool. Some variations of this model can be found such as a longer core to fit more pebbles. Usually, we only use the main model under **Models/HTR-10/**. This model still includes some specificities such as two types of pebbles, fule and graphite pebbles. The fraction of each of these pebbles usually allows to reach criticality at equilibrium.

- The small PBR is a smaller version of the gFHR, scaled down for test purposes. It produces 5 MWth.

8 The script `HxF.py`

The script `HxF.py` is the main script executing your Python input script. The code can be separated into a few sections:

- First section is about importing modules, especially the Cerberus module.
- Then the code modifies the provided Serpent input and adds the required parameters derived from the Python input.
- The code initializes the simulation environment by creating dynamic variables thanks to the Cerberus/Kraken.
- The loop over the total number of calculation steps that operate transport, brunup, and the pebble motion.

9 The script `config`

9.1 HxF Execution Script

The following Bash script automates the execution of HxF on a computing cluster, ensuring the correct setup of Python and Serpent environments while handling cluster-specific configurations.

9.1.1 Script Breakdown

This script defines and manages the execution of an HxF simulation with the following key components:

- **Python Scripts:** The script specifies the input Python file (`HTR10_coupled.py`) and the main HxF execution script (`HxF.py`), which should not be modified.
- **Conda Environment:** The script activates the Conda environment `kraken`, which was set up using `setup_Cerberus`. Additionally, the script sources `sss_environment`, ensure all necessary environment variables are loaded.
- **Cluster Configuration:** The script is designed to run on a high-performance computing (HPC) cluster, defining:
  - `GROUP_ACCOUNT`: Account name (e.g., `fc_neutronics` or `co_nuclear`).
  - `PARTITION`: HPC partition (e.g., `savio3`).
  - `NNODES`: Number of nodes allocated for the job.
  - `QOS`: Quality of Service setting for resource allocation.
  - `NHOURS`: Maximum runtime in hours.
  - `SUFFIX_JOB`: Optional suffix for job naming.
- **Serpent Configuration:** The script specifies paths and libraries needed to run Serpent:
  - `SERPENT_EXE`: Path to the Serpent executable.
  - `SERPENT_DATA`: Directory containing nuclear data files.
  - `SERPENT_ACELIB`, `SERPENT_DECLIB`, `SERPENT_NFYLIB`: Paths to cross-section, decay, and fission yield libraries.
- **OpenFOAM Configuration:** The script sets `OF_BASHRC`, which loads the OpenFOAM environment for coupled simulations. This is only needed if you run a coupled simulation. In the Appendix B, there is a quick discussion about the TH coupling, its feasibility, and its interest.

This script ensures that all dependencies are correctly set before executing HxF, facilitating seamless integration of Python, Serpent, and OpenFOAM simulations on an HPC cluster.

Note: Any cluster related issues such as bad combinations of cluster parameters may be solved referring to the Savio user manual. Moreover, the Serpent and XS libraries are usually updated continuously and these modifications should be reflected in your `config` file.

10 The script `launch HxF`

10.1 HxF Job Submission Script

This script is responsible for configuring and submitting HxF simulations on an HPC cluster using SLURM. It ensures that all necessary environment variables are set up correctly before launching the job.

10.1.1 Script Breakdown

This script automates job submission for HxF simulations and ensures the correct SLURM configuration.

- **Configuration Loading:** The script requires a configuration file as an argument and sources it, ensuring all environment variables are set.
- **Environment Variables Export:** Essential environment variables such as Python script names, SLURM job details, and Serpent configurations are exported for use in the job execution.
- **SLURM Parameters:**
  - `PARTITION_CPUS_PER_NODE`: Dynamically fetches the maximum number of CPUs per node for the selected partition.
  - `SUFFIX_JOB`: Appends a suffix to the job name if specified.
  - `JOB_NAME`: Generates a job name based on the input script filename.
  - `OUTPUT_FILE`, `ERROR_FILE`: Log files are created under `./Logs/`.
  - `TIMELIMIT`: Converts the requested job time (`NHOURS`) into HH:MM:SS format.
- **Data Cleanup:** Ensures that unset Serpent variables are removed to prevent incorrect configurations.
- **Job Execution and Logging:** The script prints all job-related parameters for verification before submitting the job using `sbatch`. The `Utils/exe.sub` script is executed, running the simulation in the allocated SLURM environment.

This script streamlines the execution of HxF on an HPC system by automating the job submission process and ensuring robust configuration management.

Now looking at the `exe.sub`, we can summarize the role of this file. The script launches the job with the cluster parameters defined in `config`.

The few commands that the cluster has to execute are the following:

```
    #!/bin/bash

# Load the environment
source ${CONFIGURATION_FILE} ${CONDA_ENVIRONMENT}
```

```
if [ -n "$OF_BASHRC" ]; then
    source $OF_BASHRC
fi
# Run the Python script
mpirun -np 1 --report-bindings --bind-to none -oversubscribe
python ${PYTHON_SCRIPT} ${INPUT_SCRIPT} ${
    PARTITION_CPUS_PER_NODE} ${NNODES}
```

This bash script loads the environment (kraken) and exports all the required variables for the cluster. Then it launches the python through a mpirun command. Basically, Cerberus will take as input the PARTITION_CPUS_PER_NODE and NNODES, and spawn the MPI processes through the booked nodes. In this mode, it corresponds to the command:

```
mpirun -np ${NNODES} Serpent.exe -omp ${PARTITION_CPUS_PER_NODE}
Serpent_input
```

So far, it has been the most memory-efficient way to distribute the threads but can lead to some NUMA errors (ORTE DAEMON COMMUNICATION ERROR).

11 The folder Cases/

The folder Cases will be created once you launch your first HxF simulation and store your updated Serpent input and your results under a specific directory that takes the name of the output folder name mentioned in the Python input scripts. Watch out: if you do not change the name of the output folder in the Python input script, you'll overwrite your previous results.

For each Case, you'll have a few types of results. You can first find the initial Serpent input that has been copied from the **Models** folder. It copies also your Python input to keep track of your simulation. Then you'll find three folders:

- The folder **Data/** contains some pebble-wise data stored at each calculation step with the `.csv` format. There are a few types of file: `core_step.csv`, `discharged_step.csv`,`discarded_step.csv`, `reinserted_step.csv`, which respectively contain the pebble-wise information for the pebble in the core, discharged, discarded, and reinserted at step "`step`". The column of the files are the following ones in `core_step.csv`:

```
Index(['Unnamed: 0', 'id', 'x', 'y', 'z', 'r', 'r_dist', 'row_id', 'uni',
       'mat_name', 'isactive', 'initial', 'insertion_step', 'avg_r_dist',
       'passes', 'recirculated', 'discarded', 'residence_time',
       'pebble_type_0', 'burnup', 'flux_pebbles_thermal',
       'flux_pebbles_thermal_rel_unc', 'integrated_flux_pebbles_thermal',
       'integrated_flux_pebbles_thermal_rel_unc',
       'integrated_flux_pebbles_thermal_unc', 'flux_pebbles_fast',
       'flux_pebbles_fast_rel_unc', 'integrated_flux_pebbles_fast',
       'integrated_flux_pebbles_fast_rel_unc',
       'integrated_flux_pebbles_fast_unc', 'power_pebbles',
       'power_pebbles_rel_unc', 'integrated_power_pebbles',
       'integrated_power_pebbles_rel_unc', 'integrated_power_pebbles_unc',
       'pass_residence_time', 'pass_agg_r_dist', 'pass_avg_r_dist',
       'pass_nsteps', 'pass_burnup', 'pass_integrated_flux_pebbles_thermal',
       'pass_integrated_flux_pebbles_thermal_unc',
       'pass_integrated_flux_pebbles_thermal_rel_unc',
       'pass_integrated_flux_pebbles_fast',
       'pass_integrated_flux_pebbles_fast_unc',
       'pass_integrated_flux_pebbles_fast_rel_unc',
       'pass_integrated_power_pebbles', 'pass_integrated_power_pebbles_unc',
       'pass_integrated_power_pebbles_rel_unc', 'domain_id'],
      dtype='object')
```

  As you can see, pebble-wise information is really diverse. You can also see the pebble-wise results of the detectors and their corresponding time-integrated value (fluence for the flux). There are some details about pass-averaged value.

Moreover, any `inventory_names` defined in the Python input will be added as a pebble-wise atomic density value. You can also add some extra fields directly in your Python input setting up `extra_fields=['fima', 'decayheat', 'activity', 'burnup']`.

Finally, in the **Data/** folder, you'll have a `cycle_step.csv`. This file contains some general info about the simulation at each step. The file `cycle_step.csv` includes all the concatenated info about each single step up to step `step`. The columns and available data are:

```
time,passes,recirculated,discarded,keff,keff_relative_uncertainty,
keff_absolute_uncertainty,neutrons_per_cycle,power_normalization_value,
DEM_step_increment,threshold_fuel,time_step
```

- The folder **Plots/** contains multiple plot for each calculation step such as `pass_step.png`, `cumulative_step.png`, `inventory_step.png`, `keff_step.png`. These files are generated only if the option top plot is selected in your Python input.
- The folder **wrk_Serpent/** contains the updated Serpent input and all the usual Serpent input such as the `serpent_res.m`, `serpent_det_step.m` and all the restart files for the core and potentially the discarded and discharged pebbles if the option has been selected to write all the restart files.

Note: An additional folder **Logs/** is created and contains your logs. There are two logs outputted:

- `your_job.o` is the HxF log containing the Serpent logs and the Cerberus log. In this log, you'll see where HxF script is failing and can return a directly related Serpent error.
- `your_job.error` contains cluster errors such as communication errors, memory issues, or environmental problems. For cluster troubleshooting, please refer to the Savio user manual.

12 Submiting your job

The simple command to run will be:

```
./launch_HxF config
```

In this case, `config` is the file used as an argument to launch the calculation. The reason why this file should be set up correctly before being launched. Please refer to the Savio user Manual to obtain the compatible partitions/accounts/QOS/time-limit/resources. Since the HxF calculation can be restarted, reaching the time limit is not a problem in itself.

13 Parallel Computation

13.1 Overview

HxF has been designed to model reactors ranging from small Pebble Bed Reactors (PBRs) with thermal power outputs of approximately 10 MWth to large-scale reactors containing over 100,000 pebbles. Simulating such large-scale systems requires both (1) a significant number of computational cores to accelerate the simulation process and (2) sufficient memory to handle burnup calculations. This is particularly relevant for HxF, which models the full reactor life cycle, starting from fresh fuel and simulating multiple pebble recirculation cycles.

To efficiently utilize available computational resources, HxF leverages the hybrid MPI+OpenMP parallelization capabilities of Serpent. When a job is launched using the `./launch_HxF` script, the requested number of nodes and cores is automatically detected, and HxF dynamically configures Serpent to optimally allocate transport and burnup calculations across the available resources.

13.2 Job Launch and Process Spawning

The `./launch_HxF` script is responsible for submitting the job and initializing the necessary computational environment. The sequence of operations can be summarized as follows:

1. The script detects the number of allocated nodes and cores using environment variables provided by the job scheduler (e.g., SLURM, PBS, or directly from `mpi4py`).
2. It determines the parallelization strategy, specifically:
    - The number of MPI ranks (typically equal to the number of nodes or user-specified domains).
    - The number of OpenMP threads per MPI rank (determined by the number of available cores per node).
3. HxF then distributes computational tasks by assigning different regions of the reactor (e.g., axial, radial, or sectoral domains) to different MPI ranks.
4. The script generates the Serpent input files, ensuring that the appropriate parallel settings are applied. This includes defining the `set dd` command in the Serpent input file, specifying the domain decomposition strategy.
5. Finally, the script executes Serpent using `mpirun` or an equivalent parallel execution command, launching the required number of processes and managing communication between them.

13.3 Domain Decomposition

Domain decomposition is a critical feature that allows HxF to efficiently parallelize simulations by dividing the reactor into smaller computational regions. The decomposition is controlled by user-defined options in the Python input script. It generates a region of equal volumes (not the equal number of pebbles...).

13.3.1 Decomposition Types and Options

The user specifies the domain decomposition strategy through the following parameters:

- `allowed_decomposition_types`: A string defining which dimensions can be used for domain splitting. The possible values are:
  - `'a'` - Axial decomposition
  - `'r'` - Radial decomposition
  - `'s'` - Sectoral (angular) decomposition
- `max_domains`: A list specifying the maximum number of domains in each dimension.
- `nnodes`: The number of MPI ranks (typically equal to the number of computing nodes).

13.4 Example: Domain Decomposition with 8 Nodes

This example assumes a large reactor core with a total of 8 computational nodes available. The domain decomposition is controlled by the variables `allowed_decomposition_types` and `max_domains`, which specify the type and depth of geometric splitting in the pebble bed.

We analyze three realistic options below:

Option 1: Full 3D Decomposition — `ars`, [8, 2, 2]

- **Decomposition types:** Axial (a), Radial (r), Sectoral (s)
- **Max domains:** 8 axial, 2 radial, 2 sectoral
- **Total possible domains:** $8 \times 2 \times 2 = 32$

HxF will search for a combination of $a \times r \times s$ such that the number of domains equals the number of nodes (or MPI ranks). In this case, it selects:

$$4 \text{ (axial)} \times 2 \text{ (radial)} \times 1 \text{ (sectoral)} = 8 \text{ domains}$$

Each domain is assigned to one MPI rank. The ‘initial_domains.txt‘ file will contain the mapping of each active pebble to one of the 8 domain IDs based on spatial location.

Outcome: Well-balanced 3D load distribution using geometry-aware domain partitioning. Ideal for large, asymmetric, or flux-gradient heavy reactors.

Option 2: Radial + Sectoral — `rs`, [4, 4]

- **Decomposition types:** Radial (r), Sectoral (s)
- **Max domains:** 4 radial, 4 sectoral
- **Total possible domains:** $4 \times 4 = 16$

HxF will search for combinations giving 8 domains. It selects:

$$4 \text{ (radial)} \times 2 \text{ (sectoral)} = 8 \text{ domains}$$

Outcome: Suitable for steady-state or radially dominant problems. Axial variation is ignored in the domain split. Fast, but may lead to imbalanced load for transient or depletion cases.

Option 3: Balanced 3D Decomposition — `ars`, [4, 4, 2]

- **Decomposition types:** Axial (a), Radial (r), Sectoral (s)
- **Max domains:** 4 axial, 4 radial, 2 sectoral
- **Total possible domains:** $4 \times 4 \times 2 = 32$

HxF selects:

$$2 \text{ (axial)} \times 2 \text{ (radial)} \times 2 \text{ (sectoral)} = 8 \text{ domains}$$

Outcome: Each MPI process handles a spatial cube of the bed, giving excellent resolution for geometry-driven problems. The decomposition is relatively coarse per dimension but spans all directions.

In all cases, HxF writes a Serpent-compatible domain map:

```
set dd 5 "initial_domains.txt"
```

Each pebble in this file is assigned a domain ID. Serpent will launch one transport solver per domain, using MPI under the hood.

13.4.1 Domain Assignment Algorithm

When domain decomposition is enabled, HxF follows these steps to assign pebbles to computational domains:

1. It generates a list of candidate domain layouts based on `max_domains` and the number of nodes.

2. It selects the best decomposition configuration by evaluating different combinations and choosing the one that maximizes balance across domains.

3. Each pebble is assigned a `domain_id` based on its position within the selected decomposition.

4. The `domain_id` values are written to a file called `initial_domains.txt`, which is later used by Serpent.

13.4.2 Generating the `initial_domains.txt` File

The file `initial_domains.txt` is a crucial component of the domain decomposition process. It contains a single-column list of integers, where each entry corresponds to the domain ID of inactive in the same order as the Serpent input file. The file is generated using the following Python function:

```
first_pbed.data.loc[data['isactive'], 'domain_id'].astype(int)
.to_csv('initial_domains.txt', index=False, header=False)
```

This ensures that only active pebbles (i.e., those participating in the simulation) are included in the decomposition.

13.4.3 Integrating Domain Decomposition in Serpent

To instruct Serpent to use the precomputed domain assignments, HxF appends the following command to the Serpent input file:

```
set dd 5 "initial_domains.txt"
```

Here, the option `5` indicates that domain IDs are read from an external file. Serpent will then use this information to distribute transport and burnup calculations among MPI ranks.

13.5 Conclusion

The parallelization strategy in HxF effectively balances the workload across available computational resources by combining MPI-based domain decomposition with OpenMP-based thread-level parallelism. By leveraging Serpent's built-in support for parallel execution and domain decomposition, HxF enables efficient simulation of complex PBR systems across a wide range of reactor sizes.

Note: In general, it will make more sense to use a decent number of nodes (8+) to take advantage of this feature and also limit memory usage. In addition to providing memory for your burnup calculation, you'll speed up your transport calculation. Moreover, HxF will create one restart file (core composition pebble-wise) per domain (`input_serpnet.wrk_ddi`). If you want to restart your job from these files, you won't need to use the number of nodes since HxF will gather the restart info before redistributing them through a new `initial_domain.txt`.

A Testing your input

Before launching your 3 days 20 nodes calculation, please check your simulation. Firstly, you check the proper importation and integration of the Cerberus Python module. Please see section 5.3.

Then to test your job, you should be inspired by the structure of the file **Utils/exe.sub**. As you may see, first the environment Kraken is activated and then a single mpirun command is launched with the following structure:

```
mpirun -np 1 --report-bindings --bind-to none
-oversubscribe python ${PYTHON_SCRIPT}
${INPUT_SCRIPT} ${PARTITION_CPUS_PER_NODE}
${NNODES}
```

Then two options are available to test your input (after activating Kraken):

- If you run on the log-in node (for a short time) you'll use only one core and run the following command:

  ```
  python HxF.py ./Input_scripts/gFHR.py
  ```

  If you get an interactive session with 4 nodes and 32 cores per node for example you can run the following commands:

  ```
  source ./confing ### help you to export
  the variables such as SERPENT_EXE
  ```

  ```
  mpirun -np 1 python HxF.py ./Input_scripts/gFHR.py 32 4
  ```

  In this case, since all the required variables have been exported from the config file, the script will be running on four nodes. It helps to monitor the CPU usage and memory usage for each single node. To obtain the memory usage, refer to the `htop` command in the Savio User manual.

B Thermal Coupling

So far, the thermal coupling for PBR, providing pebble-wise results in terms of temperatures, has been applied to a static bed. Usually, the equilibrium sate is extracted as a HxF snapshot. The integration of TH tool within HxF was not possible since the spawning process featured by Cerberus and Serpent does not apply to OpenFoam (our current TH solver). One method that is currently coded in `HxF.py` when you set up `thermal_coupling=True` in your Python input script. In this case, after every transport step, HxF pauses and launches an external job to obtain the temperature distribution from the previously calculated power density map. This process means that as long as the OpenFOAM job does not terminate, HxF will be waiting. In addition, the overall HxF run will be drastically extended since Transport and OpenFOAM will iterate until reaching a equilibrium (most likely a steady keff). Then it can move to the burnup and motion stages. As observed in the previous study, only a few iterations of the coupled scheme transport/TH is needed to reach equilibrium. It still means that the overall simulation time (linearly linked to the total number of transport calculation) can be doubled or tripled.

A good alternative was to compute the temperature afterward and keep a loose coupling meaning by the obtained temperatures are not fed back into the transport calculation. This is supported by the fact that usually coupled calculations converged after 1 iteration.

References

[1] Jaakko Leppänen. Serpent - a continuous-energy monte carlo reactor physics burnup calculation code. Technical report, VTT Technical Re- search Centre of Finland, 2004.

[2] Y. Robert, L. Jantzen, and M. Fratoni. Demonstration of coupling between hyper-fidelity depletion and discrete element method for pebble bed rectors. In *Proceedings of M&C 2023*, Niagara Falls, 2023.

[3] Yves Robert, Tatiana Siaraferas, and Massimiliano Fratoni. Hyper-fidelity depletion with discrete motion for pebble bed reactors. *Scientific Reports*, 13(1), August 2023.

[4] Nader Satvat, Fatih Sarikurt, Kevin Johnson, Ian Kolaja, Massimiliano Fratoni, Brandon Haugh, and Edward Blandford. Neutronics, thermal-hydraulics, and multi-physics benchmark models for a generic pebble-bed fluoride-salt-cooled high temperature reactor (fhr). *Nuclear Engineering and Design*, 384:111461, 2021.

[5] William K. Terry, Soon Sam Kim, Leland M. Montierth, Joshua J. Cogliati, and Abderrafi M. Ougouag. Evaluation of the htr-10 reactor as a benchmark for physics code qa. 11 2005.