INL/EXT-16-39874

Integration of OpenMC Methods into MAMMOTH and Serpent

Leslie Kerby[1,2], Mark DeHart[1], Aaron Tumulak[1,3]

[1]Reactor Analysis and Design Department
Idaho National Laboratory
1955 N. Fremont Avenue
Idaho Falls, ID 83415

[2]Department of Nuclear Engineering and Health Physics
Idaho State University
995 University Boulevard
Idaho Falls, ID 83401

[3]Department of Nuclear Engineering & Radiological Sicences
University of Michigan
500 S. State Street
Ann Arbor, MI 48109

September 2016

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

INL/EXT-16-39874

Integration of OpenMC Methods into MAMMOTH and Serpent

Leslie Kerby, Mark DeHart, Aaron Tumulak

September 2016

Idaho National Laboratory
Nuclear Systems Design and Analysis
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Contents

List of Figures

List of Tables

1 Introduction

Modeling nuclear reactors is complex, requiring multiphysics solutions between neutronics, thermal hydraulics, and fuel behavior. Monte Carlo methods are becoming more desirable and feasible with advances in parallel computing technology, yet they are still impractical for full-core simulations. However, aspects of the simulation where a higher level of modeling fidelity is required could be performed with Monte Carlo codes, and then these solutions could be coupled to deterministic codes.

Serpent 2, a three-dimensional continuous-energy Monte Carlo reactor physics burnup calculation code [1], has been tested and produces accurate cross sections for the Advanced Test Reactor (ATR) and the Transient Reactor Test Facility (TREAT) [2], both here at INL, and is desirable for use in future TREAT simulations. MAMMOTH is being developed at INL to produce a full-core solution: neutronics, thermal hydraulics, and fuel behavior [3]. Preferably, a coupled solution will be obtained which contains Serpent 2 neutronics coupled with MAMMOTH thermal hydraulics and fuel performance.

This research builds upon and utilizes work performed by both OpenMC and Serpent development teams. OpenMC [4], a Monte Carlo particle transport simulation code focused on neutron criticality calculations, has already completed significant research in coupling OpenMC and MOOSE [5, 6, 7], which we wish to adapt to Serpent 2. The OpenMC coupling scheme utilizes a MultiApp and the MOOSE application Cerberus for thermal feedback. OpenMC also uses Functional Expansion Tallies (FETs), allowing for a more efficient passing of multiphysics data between OpenMC and MOOSE. Recent development of the multi-physics interface for Serpent 2 eases the complexity of coupling Serpent with thermal hydraulic feedback [8]. Details of a Serpent–MOOSE prototype developed several years ago were graciously provided to us by Ville Valtavirta of the Serpent development team. This prototype utilizes Userobjects and contains basic heat conduction. We chose to model our coupling on this prototype.

Coupling Serpent 2.1.26 with MOOSE proved to be complex. We have preliminary results, but further work is necessary. One of the pre-eminent issues in coupling two codes is accurately and efficiently transferring between their different meshes. The current Serpent–MOOSE coupling uses the unstructured OpenFOAM mesh on the Serpent side, and the structured hexahedral mesh on the MOOSE side. Implementing Functional Expansion Tallies (FETs) in Serpent enables us to have a mesh-free fission power distribution in Serpent which can more easily be transferred to any desired mesh within MOOSE (and eventually BISON and MAMMOTH). Both of these capabilities have been accomplished. However,

FETs are implemented in the standalone Serpent and need to be merged with our coupled Serpent–MOOSE version.

2 Introduction

Modeling nuclear reactors is complex, requiring multiphysics solutions between neutronics, thermal hydraulics, and fuel behavior. Monte Carlo methods are becoming more desirable and feasible with advances in parallel computing technology, yet they are still impractical for full-core simulations. However, aspects of the simulation where a higher level of modeling fidelity is required could be performed with Monte Carlo codes, and then these solutions could be coupled to deterministic codes.

Serpent 2, a three-dimensional continuous-energy Monte Carlo reactor physics burnup calculation code [1], has been tested and produces accurate cross sections for the Advanced Test Reactor (ATR) and the Transient Reactor Test Facility (TREAT) [2], both here at INL, and is desirable for use in future TREAT simulations. MAMMOTH is being developed at INL to produce a full-core solution: neutronics, thermal hydraulics, and fuel behavior [3]. Preferably, a coupled solution will be obtained which contains Serpent 2 neutronics coupled with MAMMOTH thermal hydraulics and fuel performance.

This research builds upon and utilizes work performed by both OpenMC and Serpent development teams. OpenMC [4], a Monte Carlo particle transport simulation code focused on neutron criticality calculations, has already completed significant research in coupling OpenMC and MOOSE [5, 6, 7], which we wish to adapt to Serpent 2. The OpenMC coupling scheme utilizes a MultiApp and the MOOSE application Cerberus for thermal feedback. OpenMC also uses Functional Expansion Tallies (FETs), allowing for a more efficient passing of multiphysics data between OpenMC and MOOSE. Recent development of the multi-physics interface for Serpent 2 eases the complexity of coupling Serpent with thermal hydraulic feedback [8]. Details of a Serpent–MOOSE prototype developed several years ago were graciously provided to us by Ville Valtavirta of the Serpent development team. This prototype utilizes Userobjects and contains basic heat conduction. We chose to model our coupling on this prototype.

Coupling Serpent 2.1.26 with MOOSE proved to be complex. We have preliminary results, but further work is necessary. One of the pre-eminent issues in coupling two codes is accurately and efficiently transferring between their different meshes. The current Serpent–MOOSE coupling uses the unstructured OpenFOAM mesh on the Serpent side, and the structured hexahedral mesh on the MOOSE side. Implementing Functional Expansion Tallies (FETs) in Serpent enables us to have a mesh-free fission power distribution in Serpent which can more easily be transferred to any desired mesh within MOOSE (and eventually BISON and MAMMOTH). Both of these capabilities have been accomplished. However,

FETs are implemented in the standalone Serpent and need to be merged with our coupled Serpent–MOOSE version.

3 Coupling of Serpent and MOOSE

As discussed in the Introduction, coupling the neutronics from Serpent 2 with the thermal hydraulics and fuel performance in MAMMOTH is the desired path. A coupled Serpent 2 and MOOSE code is a first step towards this. We have completed a parallelizable coupled Serpent 2.1.26–MOOSE code and tested it successfully on a single fuel pin.

3.1 Implementation

3.1.1 Userobjects

The coupled code utilizes three MOOSE userobjects: ElementTransfer, RunSerpent, and HeatToMoose. These are built off of the userobjects Ville Valtavirta wrote to couple Serpent 2.1.15 with MOOSE several years ago.

Figure 1 illustrates the flow of the coupled Serpent–MOOSE code. The userobject HeatToMoose transfers the Serpent fission heat generated, per element volume, to the MOOSE mesh. It reads the power production from the multiphysics interface output file produced by Serpent into an array and transforms it into the MOOSE 3D mesh. This array can later be accessed by MOOSE to calculate heat production in a certain element. An initial guess for the fission power generation (along with the geometry specified by Serpent), which can be accomplished by running standalone Serpent, must be supplied. MOOSE then runs a simple heat conduction solution given the fission heat and geometry dictated by Serpent. The userobject ElementTransfer averages the MOOSE temperature solution field for each element, transforms this to an OpenFOAM mesh, and transfers this to the Serpent interface input file. Next, the userobject RunSerpent runs either a full Serpent calculation, or if Serpent has already been run once and does not need to be initialized, a shorter transport cycle. The initial guess for fission heat is then overwritten by the coupled Serpent when it calculates new fission power distributions, and the process starts over again, iterating the number of times specified in the MOOSE input file.

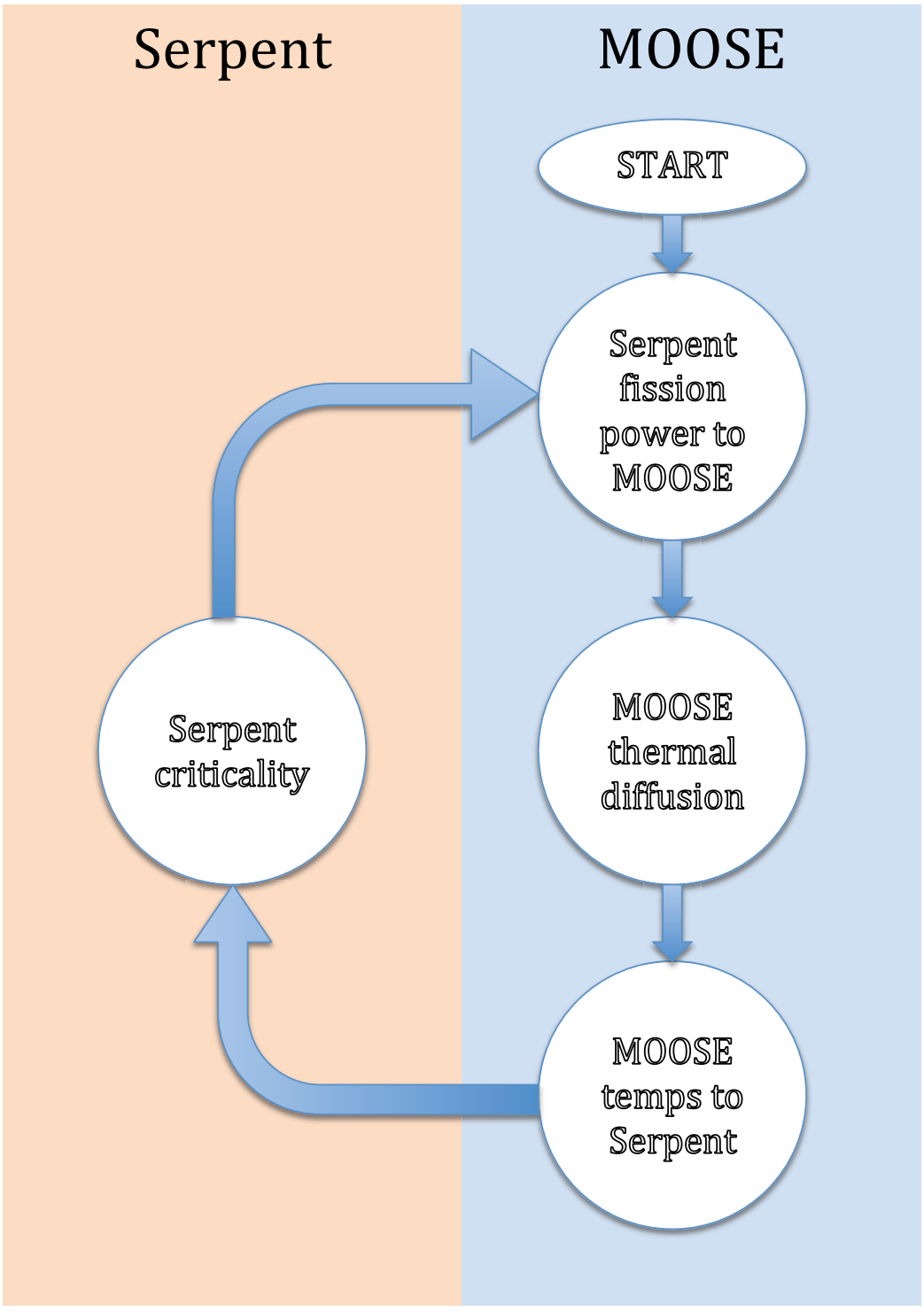

Figure 1: Flow of the coupled Serpent–MOOSE code.

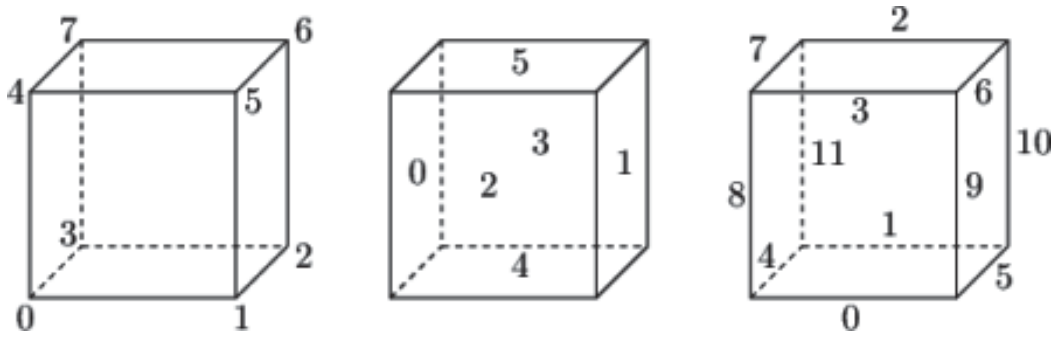

Figure 2: OpenFOAM hexahedron vertices, faces, and edge numbering.

3.1.2 Meshes

The current impementation utilizes an OpenFOAM mesh on the Serpent side. The OpenFOAM mesh uses a *polyMesh* object, which has five attributes: points, faces, owners, neighbors, and boundaries. These are separated into four separate files (boundaries are not used) in the Serpent multiphysics interface. The points file contains a list of vectors describing the cell vertices, where the first vector in the list represents vertex 0, the second vector represents vertex 1, etc. The faces file contains a list of faces, each face being a list of indices to vertices in the points list, where again, the first entry in the list represents face 0, etc. The owner file is a list of owner cell labels, the index of entry relating directly to the index of the face, so that the first entry in the list is the owner label for face 0, the second entry is the owner label for face 1, etc. And lastly, the neighbor file contains a list of neighbor cell labels. Fig. 2 shows an example of a hexahedron in the *cellShape* class, which is the specific type of *polyMesh* used on the Serpent side in this coupling.

In the present coupled code, MOOSE uses a *GeneratedMesh* with dim=3 and elem_type=PRISM6, or simply a rectangular-prism (ie, "box") mesh, with sides numbered as depicted in Fig. 3.

3.1.3 Code Modifications

MOOSE is written in C++ and Serpent 2 in C. There are two main options for compiling the coupled code:

1. Use the MOOSE Makefile to direct the compilation of Serpent 2;

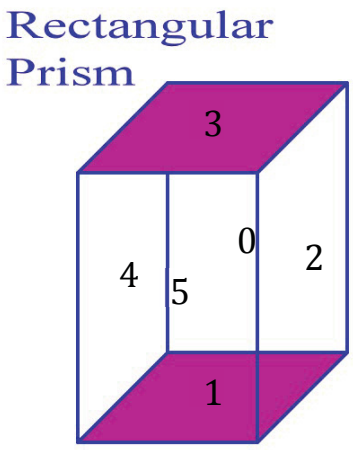

Figure 3: Example of a *GeneratedMesh* MOOSE rectangular prism.

2. Add the Serpent 2 source code with the MOOSE source code.

The first option is preferable as it allows specific instructions in how the Serpent 2 C code is compiled. However, the shared libraries were not communicating correctly across the coupled code so this method was abandoned. With the second option, Serpent 2 C code is automatically compiled by MOOSE. This means that GFX within Serpent is not compiled correctly, and therefore Serpent cannot access the GFX graphical functions.

Several modifications to the Serpent 2 code were necessary to create this coupled version. The main changes are summarized as follows:

- Protect header files from multiple definitions;
- Extern global variables declared in header files;
- Create C file to define those global variables;
- Change main to cmain;
- #define OPEN_MP and NO_GFX_MODE;
- Create DATA_EXT_MODE option;
- Add extern clause to all *.c and *.h files;

- Fix Newton's method bug in TMS.

The standard wrapper used to protect header files from multiple definitions is shown below.

```
#ifndef HEADER_NAME
#define HEADER_NAME
  ...
  {header file contents}
  ...
#endif
```

The extern clause used in all files is as follows.

```
#ifdef __cplusplus
extern "C" {
#endif
  ...
  {file contents}
  ...
#ifdef __cplusplus
} // closing curly bracket
#endif
```

In testing the single fuel pin case, the analog k_{eff} was converging to the standalone Serpent value, but the implicit k_{eff} was not. In exploring the cause of this we discovered the culprit in a hidden bug in the TMS method, in tmpmajorants.c. In the Newton's method solve for E_{min} and E_{max} the derivative was sometimes going toward zero, causing it to blow up and producing a NaN for E_{min} or E_{max}, resulting in a segmentation fault.

Interestingly, this bug is also found in the standalone Serpent 2.1.26, but it did not cause a problem with the k_{eff} calculations, nor does it cause a segmentation fault. Perhaps the different way the standalone version of Serpent 2 is compiled affords it extra protection. However, $E_{min/max}$ does still go to $\pm\infty$.

```
Standalone Serpent 2.1.26:
n=108
iter=0 E_max=2.371008e+01 derivative=-7.079385e-01
iter=1 E_max=-3.141819e+197 derivative=-5.162824e-202
iter=2 E_max=3.509165e+00 derivative=-8.152219e+00
iter=3 E_max=3.840001e+00 derivative=-2.672801e+00
iter=4 E_max=4.114408e+00 derivative=-1.070173e+00
```

```
iter=5 E_max=4.354007e+00 derivative=-4.194301e-01
iter=6 E_max=4.568968e+00 derivative=-1.624786e-01
iter=7 E_max=4.764411e+00 derivative=-6.260353e-02
iter=8 E_max=4.941769e+00 derivative=-2.418251e-02
iter=9 E_max=5.097684e+00 derivative=-9.520114e-03
iter=10 E_max=5.221080e+00 derivative=-3.974104e-03
iter=11 E_max=5.294504e+00 derivative=-1.920526e-03
iter=12 E_max=5.316691e+00 derivative=-1.227412e-03
iter=13 E_max=5.318361e+00 derivative=-1.069755e-03
iter=14 E_max=5.318370e+00 derivative=-1.058697e-03
iter=15 E_max=5.318370e+00 derivative=-1.058639e-03
n=109
iter=0 E_max=2.472143e+01 derivative=-6.868821e-01
iter=1 E_max=-4.732367e+216 derivative=-3.472301e-221
iter=2 E_max=3.518731e+00 derivative=-8.271520e+00
iter=3 E_max=3.850415e+00 derivative=-2.728911e+00
iter=4 E_max=4.125319e+00 derivative=-1.092971e+00
iter=5 E_max=4.365260e+00 derivative=-4.284353e-01
iter=6 E_max=4.580483e+00 derivative=-1.659826e-01
iter=7 E_max=4.776144e+00 derivative=-6.395567e-02
iter=8 E_max=4.953719e+00 derivative=-2.470318e-02
iter=9 E_max=5.109901e+00 derivative=-9.722264e-03
iter=10 E_max=5.233697e+00 derivative=-4.055133e-03
iter=11 E_max=5.307649e+00 derivative=-1.955961e-03
iter=12 E_max=5.330193e+00 derivative=-1.246294e-03
iter=13 E_max=5.331919e+00 derivative=-1.083861e-03
iter=14 E_max=5.331928e+00 derivative=-1.072285e-03
iter=15 E_max=5.331928e+00 derivative=-1.072223e-03
n=110
iter=0 E_max=2.924542e+01 derivative=-6.074701e-01
iter=1 E_max=-inf derivative=-1.029385e-317
iter=2 E_max=3.557129e+00 derivative=-8.748723e+00

It would then repeat the cycle and go to infinity again.
```

The fix for this was fairly simple. The $derivative == 0$ check was modified to be $|derivative| < 1.0e-50$, for both E_{min} and E_{max}. This fixed the problem and, in addition, should make the method more efficient.

```
Standalone Serpent 2.1.26 with fix
n=108
iter=0 E_max=2.371008e+01 derivative=-7.079385e-01
iter=1 E_max=3.509165e+00 derivative=-8.152219e+00
iter=2 E_max=3.840001e+00 derivative=-2.672801e+00
iter=3 E_max=4.114408e+00 derivative=-1.070173e+00
iter=4 E_max=4.354007e+00 derivative=-4.194301e-01
iter=5 E_max=4.568968e+00 derivative=-1.624786e-01
iter=6 E_max=4.764411e+00 derivative=-6.260353e-02
iter=7 E_max=4.941769e+00 derivative=-2.418251e-02
iter=8 E_max=5.097684e+00 derivative=-9.520114e-03
iter=9 E_max=5.221080e+00 derivative=-3.974104e-03
iter=10 E_max=5.294504e+00 derivative=-1.920526e-03
iter=11 E_max=5.316691e+00 derivative=-1.227412e-03
iter=12 E_max=5.318361e+00 derivative=-1.069755e-03
iter=13 E_max=5.318370e+00 derivative=-1.058697e-03
```

```
iter=14 E_max=5.318370e+00 derivative=-1.058639e-03
n=109
iter=0 E_max=2.472143e+01 derivative=-6.868821e-01
iter=1 E_max=3.518731e+00 derivative=-8.271520e+00
iter=2 E_max=3.850415e+00 derivative=-2.728911e+00
iter=3 E_max=4.125319e+00 derivative=-1.092971e+00
iter=4 E_max=4.365260e+00 derivative=-4.284353e-01
iter=5 E_max=4.580483e+00 derivative=-1.659826e-01
iter=6 E_max=4.776144e+00 derivative=-6.395567e-02
iter=7 E_max=4.953719e+00 derivative=-2.470318e-02
iter=8 E_max=5.109901e+00 derivative=-9.722264e-03
iter=9 E_max=5.233697e+00 derivative=-4.055133e-03
iter=10 E_max=5.307649e+00 derivative=-1.955961e-03
iter=11 E_max=5.330193e+00 derivative=-1.246294e-03
iter=12 E_max=5.331919e+00 derivative=-1.083861e-03
iter=13 E_max=5.331928e+00 derivative=-1.072285e-03
iter=14 E_max=5.331928e+00 derivative=-1.072223e-03
n=110
iter=0 E_max=2.924542e+01 derivative=-6.074701e-01
iter=1 E_max=3.557129e+00 derivative=-8.748723e+00
```

3.2 Results

The coupled Serpent 2.1.26–MOOSE was tested with a single fuel pin surrounded by water, as shown in Fig. 4 and detailed in Table 1. Results for k_{eff} for standalone Serpent 2.1.26 and the coupled Serpent 2.1.26–MOOSE are displayed in Table 2. Each were run with 20 inactive cycles and 200 active cycles of 4000 neutrons/cycle, and with OpenMP with 3 threads. The coupled Serpent–MOOSE was run for 5 iterations; only results from the final iteration are shown.

Fuel		Water	
Isotope	% mass	Isotope	%mass
^{235}U	2.9971	^{1}H	66.6667
^{238}U	85.153	^{16}O	33.3333
^{16}O	11.85		
5 cm^2 square		10 cm^2 square	

Table 1: Fuel pin properties.

The analog and implicit k_{eff} calculated from the standalone Serpent and coupled Serpent–MOOSE are within a standard deviation of each other. Still, there is a slight decrease in

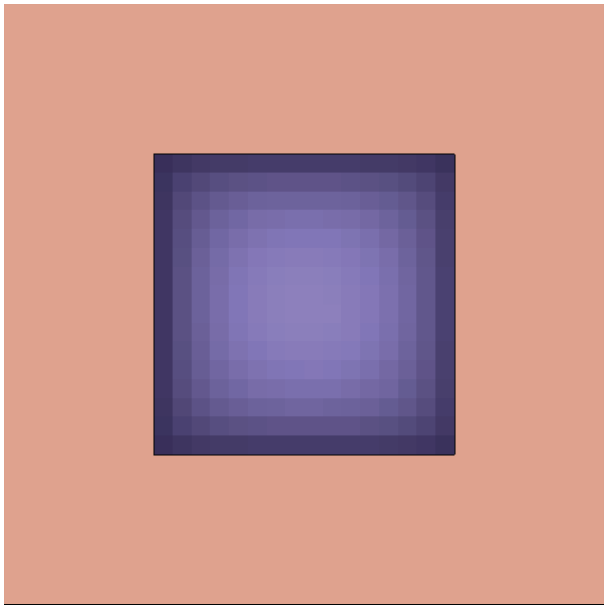

Figure 4: Geometry of single fuel pin.

	Standalone Serpent 2.1.26	Coupled Serpent–MOOSE
k_{eff} (analog)	0.20592 +/- 0.00080	0.20535 +/- 0.00085
k_{eff} (implicit)	0.20598 +/- 0.00060	0.20542 +/- 0.00061
Transport comp. time	20.9	20.3

Table 2: Fuel pin k_{eff} values for the standalone Serpent and coupled Serpent–MOOSE.

the k_{eff} of the coupled Serpent–MOOSE and this could be due to thermal feedback of increasing fuel temperatures.

4 Implementation of FETs

One of the pre-eminent issues in coupling two codes is accurately and efficiently transferring between their different meshes. The current Serpent–MOOSE coupling uses the unstructured OpenFOAM mesh on the Serpent side, and the structured hexahedral mesh on the MOOSE side. Implementing Functional Expansion Tallies (FETs) in Serpent enables us to have a mesh-free fission power distribution in Serpent which can more easily be transferred to any desired mesh within MOOSE (and eventually BISON and MAMMOTH).

Serpent is a three-dimensional continuous-energy Monte Carlo reactor physics code that has been used to produce homogenized multi-group cross-sections for deterministic calculations [1]. The Multiphysics Object-Oriented Simulation Environment (MOOSE) framework developed at Idaho National Laboratory provides a high-level interface for solving systems of coupled, nonlinear partial differential equations [5], lending itself useful for modeling multiphysics phenomena found in reactor physics problems. Coupling Serpent to MOOSE requires an efficient method of receiving and sending spatially-dependent information. An implementation of functional expansion tallies (FETs) to represent temperature, density, and local power in a single 3-D fuel pin is implemented in Serpent 2. Preliminary results show that the method is feasible and produces qualitatively acceptable results.

4.1 Theory

Fuel pins are usually cylindrical. In cylindrical geometry, a scalar-valued function $f(r,\theta,z)$ can be expanded as the sum of the product of Legendre and Zernike polynomials

$$f(r,\theta,z) = \sum_i \sum_j c_{ij} Z_j(r,\theta) P_i(z) \tag{1}$$

where the Zernike polynomials $Z_j = Z_n^m$ are defined for even $n-m$ and $n \geq m$ as

$$
\begin{aligned}
Z_n^m(r,\theta) &= \begin{cases} \sqrt{2(n+1)} R_n^m(r)\cos(m\theta) & \text{for } m > 0 \\ \sqrt{2(n+1)} R_n^{-m}(r)\sin(-m\theta) & \text{for } m < 0 \\ \sqrt{n+1} R_n^0(r) & \text{for } m = 0 \end{cases} \\
R_n^m(r) &= \sum_{k=0}^{\frac{n-m}{2}} (-1)^k \binom{n-k}{k} \binom{n-2k}{\frac{n-m}{2}-k} r^{n-2k}
\end{aligned}
\tag{2}
$$

where the second and third factors in parenthesis are binomial coefficients.

For brevity in notation and convenience in code implementation, the radial and radial indices n and m are mapped to a single index j using Noll's indexing. The rules are as follows:

1. The first entry $(n = 0, m = 0)$ is $j = 1$.
2. (n,m) with greater n have greater j.
3. (n,m) with $m < 0$ have odd-numbered j.
4. (n,m) with $m > 0$ have even-numbered j.
5. Within a given n, (n,m) with greater $|m|$ have greater j.

The Legendre polynomials P_i are defined for integers $i \geq 0$ as

$$
P_i(z) = \sqrt{\frac{2i+1}{2}} \sum_{k=0}^{i} \binom{i}{k} \binom{-i-1}{k} \left(\frac{1-z}{2}\right)^k \tag{3}
$$

where the first two factors in parenthesis are binomial coefficients and the last factor is a real number.

From the orthogonality of Zernike and Legendre polynomials, the product $Z_j(r,\theta)P_i(z)$ also satisfies

$$\int_{-1}^{1} dz \int_{0}^{1} dr \int_{0}^{2\pi} d\theta \left(Z_j(r,\theta)P_i(z) \right) \left(Z_{j'}(r,\theta)P_{i'}(z) \right) = \delta_{i,i'}\delta_{j,j'} \tag{4}$$

where δ is the Kronecker delta function. Note that the forms of the polynomials defined in Eqs. [2] and [3] are normalized so that their inner products are one. The constants c_{ij} can then be defined as

$$c_{ij} = \int_{-1}^{1} dz \int_{0}^{1} dr \int_{0}^{2\pi} d\theta \; f(r,\theta,z)Z_j(r,\theta)P_i(z) \tag{5}$$

For the purpose of tallying a score E with weight w at position (r,θ,z) using functional expansion tallies, the tally for coefficient c_{ij} can be incremented by a value

$$Z_j(r,\theta)P_i(z)Ew \tag{6}$$

taking care to also increase the total weight W by an amount w.

4.2 Implementation

Serpent 2 provides a multi-physics interface for receiving temperature and density distributions for external coupling with thermal-hydraulic codes. One interface, *Type 3*, allows an arbitrary number `np` of parameters $\mathtt{n}_0$, $\mathtt{n}_1$, ..., $\mathtt{n}_{\mathtt{np}-1}$ to be passed into the serepent executable `sss2` during runtime by means of file I/O. The parameters are parsed and used in function `UserIFC()` which is called whenever Serpent samples a given material's temperature and density at a point (`x`, `y`, `z`). `UserIFC()` was modified to accept spatially-dependent temperature and density through coefficients given by Eq. [5].

Using Alg. [1], temperature and densities from an external code can be passed into Serpent as continuous functions represented only by a handful of coefficients. With updated material properties, Serpent can perform another iteration of a Monte Carlo solution

Algorithm 1 Outline of `UserIFC()` function using `nz` Legendre polynomials $\{P_0(z), P_1(z), \ldots, P_{\text{nz}-1}(z)\}$ and `nr` Zernike polynomials $\{Z_1(r,\theta), Z_2(r,\theta), \ldots, Z_{\text{nr}}(r,\theta)\}$. Note that `*f` is a pointer to either temperature or density at location (x,y,z).

function USERIFC(*f, x, y, z, nz, nr, $c_{0,1}$, ..., c_{ij}, ..., $c_{nz\text{-}1,\,nr}$)
    *f = 0
    $i = 0$
    for $i <$ nz **do**
        $j = 1$
        for $j \leq$ nr **do**
            *f = *f + $c_{ij} \times Z_j(r(x,y),\theta(x,y)) \times P_i(z)$
        end for
    end for
end function

and form a new power distribution. Normally, the local power can only be returned as volume-averaged quantities using `nz` equally-spaced axial bins and `nr` equal-volume radial bins for each fuel pin. The fission power `E` with weight `w` is scored in the function `ScoreInterfacePower()` using `nz` $\times$ `nr` bins.

Algorithm 2 Original `ScoreInterfacePower()` function using volume-averaged bins. `DetermineBinIndex()` is a function that returns the index of the bin (i,j) that contains position (x,y,z).

function SCOREINTERFACEPOWER(E, w, nz, nr, x, y, z)
    i, j = `DetermineBinIndex`(nz, nr, x, y, z)
    `AddScoreToBin`(E, w, i, j)
end function

Since the underlying data structure for storing lists in Serpent was found to be rather unwieldy, the function `ScoreInterfacePower()` was modified to store `nz` $\times$ `nr` coefficients rather than volume-averaged powers. The parameter `nz` in the input file now sets the number of Legendre polynomials to use in the axial direction and `nr` now sets the number of Zernike polynomials to use for the radial and azimuthal directions. The evaluated Legendre/Zernike polynomial is incorporated into the energy E of the score.

The coefficients are returned in the same format as the volume-averaged bin powers.

Algorithm 3 Modified `ScoreInterfacePower()` function using `nz` Legendre polynomials $\{P_0(z), P_1(z), \ldots, P_{\text{nz}-1}(z)\}$ and `nr` Zernike polynomials $\{Z_1(r,\theta), Z_2(r,\theta), \ldots, Z_{\text{nr}}(r,\theta)\}$. The coefficients are stored in `nz` $\times$ `nr` bins.

```
function SCOREINTERFACEPOWER(E, w, nz, nr, x, y, z)
    i = 0
    for i < nz do
        j = 1
        for j ≤ nr do
            E = E × Z_j(r(x,y), θ(x,y)) × P_i(z)
            AddScoreToBin(E, w, i, j)
        end for
    end for
end function
```

4.2.1 Axial Functional Expansion

A handful of test cases were performed to check that the updated interface was behaving properly. A radially infinite, axially finite assembly of BWR fuel pins 100 cm in height (Fig. [3]) was used to examine the axial power profile. Only the coolant densities in the axial directions $\rho(z)$ were tested. Only one Zernike polynmial, $Z_1(r,\theta) = 1$, corresponding to the average value in the (r,θ) direction, was used. Legendre polynomials $\{P_0(z), P_1(z), \ldots, P_{\text{np}-1}(z)\}$ were used to expand the functional form of the density

$$\rho(z) \approx n_0 P_0(z) + n_1 P_1(z) + \ldots + n_{\text{nz}-1} P_{\text{nz}-1}(z) \tag{7}$$

using the first `nz` terms. The functional forms of the axially-dependent densities are listed in Table 4.

Parameter	**Value**
Fuel	UO_2 (1.83% ^{235}U)
Clad	Zircalloy
Coolant	H_2O
Fuel Outer Radius	4.335×10^{-1} cm
Void Outer Radius	4.420×10^{-1} cm
Clad Outer Radius	5.025×10^{-1} cm
Assembly Pitch	1.295 cm
Assembly Height	100 cm
Fuel Nominal Density	$10.424\,\mathrm{g\,cm^{-3}}$
Coolant Nominal Density	$1.0\,\mathrm{g\,cm^{-3}}$

Table 3: Geometric and material parameters of fuel assembly. Note that the coolant nominal density is multiplied by a factor $0 \leq f \leq 1$ calculated from the current position r, θ, z and the coefficients passed into Serpent during runtime.

Case	n_0	n_1	n_2	n_3	n_4
0	1.4142	0	0	0	0
1	0.70711	0.40825	0	0	0
2	0.56569	0.0	0.37947	0	0
3	0.70711	-0.24495	0.31623	0.16036	0
4	0.53033	-0.16330	0.11068	0.10690	0.21213

Table 4: Coefficients used to test axial functional expansion. The listed values are coefficients used to impose an axial density profile in the coolant.

Case	n_0	n_1	n_2	n_3	n_4	n_5	n_6
0	0.944	-0.00125	-0.405	9.65e-4	0.0118	3.02e-4	-0.00350
1	0.712	0.372	-0.217	-0.228	-0.0563	0.00440	0.00397
2	0.912	-4.24e-4	0.168	1.93e-4	-0.294	6.20e-4	-0.0619
3	0.608	-0.357	-0.0207	0.234	-0.168	0.00182	0.0117
4	0.737	-0.281	-0.273	0.214	0.0915	-0.0658	-0.0749

Table 5: The first seven coefficients returned from the functional expansion tally of axial fission power profile in a fuel pin assembly with coolant densities given by Table 4.

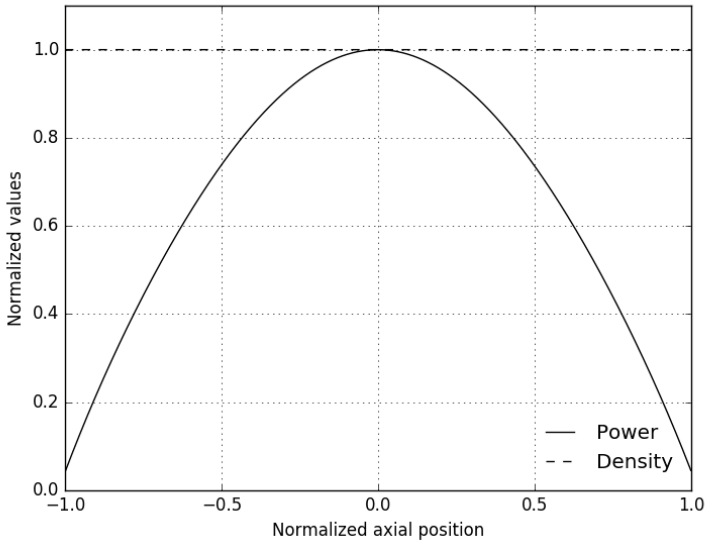

Figure 5: Density and corresponding power for Case 0, the 0^{th} order Legendre expansion

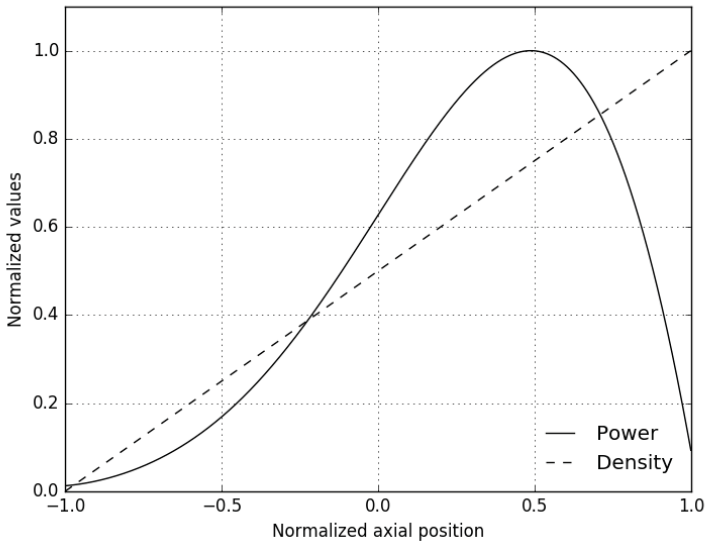

Figure 6: Density and corresponding power for Case 1, the 1^{st} order Legendre expansion

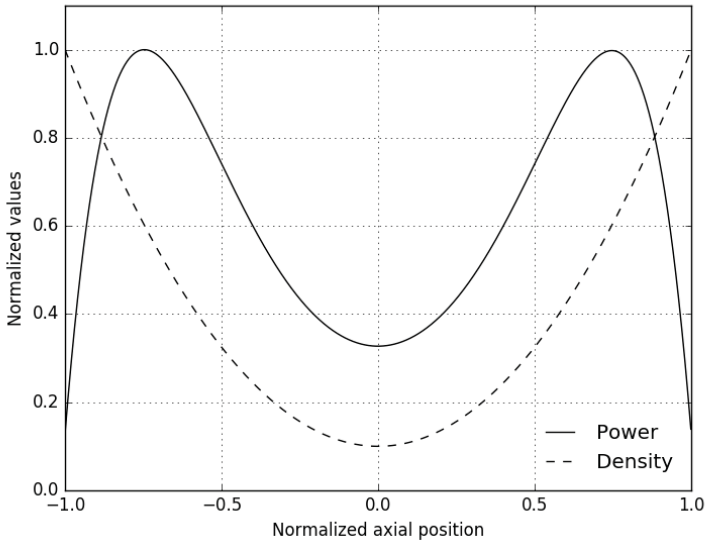

Figure 7: Density and corresponding power for Case 2, the 2nd order Legendre expansion

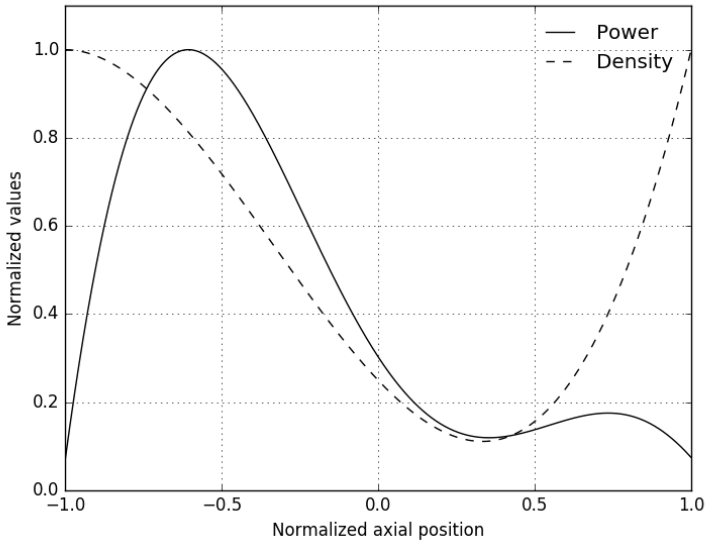

Figure 8: Density and corresponding power for Case 3, the 3rd order Legendre expansion

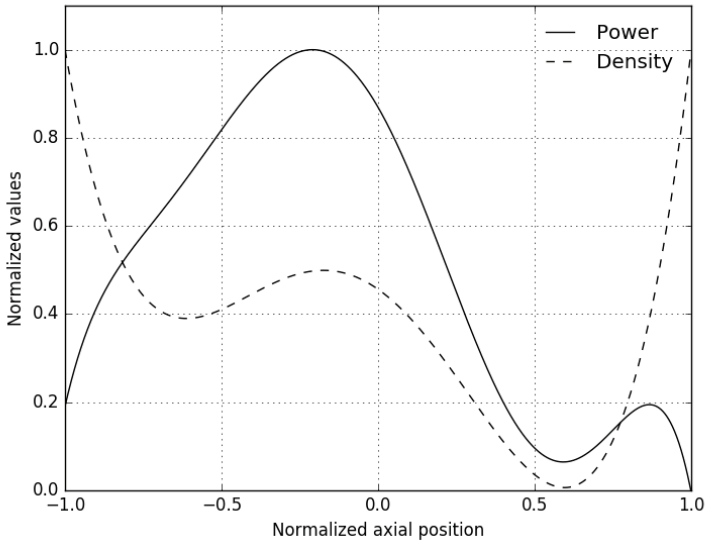

Figure 9: Density and corresponding power for Case 4, the 4^{th} order Legendre expansion.

The results obtained look reasonable since power appears to peak close to where there is denser coolant. The 0^{th} order solution follows a cosine power distrubition which matches analytical solutions. Even in asymmetric density distributions such as Figs. [6], [8], and [9], the power distributions peak where density is at a maximum and dip somewhat near where density is at a minimum. It is interesting to note that the transport equation for this particular problem tends to introduce a strong even-order $(n_0, n_2, n_4, \ldots)$ response in the fission power shown in Table 5.

4.2.2 Radial & Azimuthal Functional Expansion

In most real reactor problems, the radial and azimuthal dependence of fission power comes mostly from the distribution of the flux in and around the fuel pin rather than the density of the fuel pin itself. To test if the Zernike polynomials were properly capturing the radial and azimuthal dependence on neutron flux, a simple test problem was contrived with a single cylinder of uranium metal partially covered by a strong absorber in water (Table 6).

Zernike polynomials were used to score the fission power

$$P(r,\theta) \approx n_1 Z_1(r,\theta) + n_2 Z_2(r,\theta) + \ldots + n_{\mathrm{nr}} Z_{\mathrm{nz}}(r,\theta) \tag{8}$$

using the first `nr` terms. In the following results, `nr` $= 21$, corresponding to a sixth-order radial expansion.

Parameter	**Value**
Fuel	Uranium metal (1.0% ^{235}U)
Absorber	Boron-10
Coolant	H_2O
Fuel Outer Radius	25.0 cm
Absorber Outer Radius	30.0 cm
Assembly Height	100 cm
Fuel Density	10.0 g cm^{-3}
Absorber Density	2.0 g cm^{-3}
Coolant Density	1.0 g cm^{-3}

Table 6: Geometric and material parameters of fuel cylinder. Densities were constant for each material.

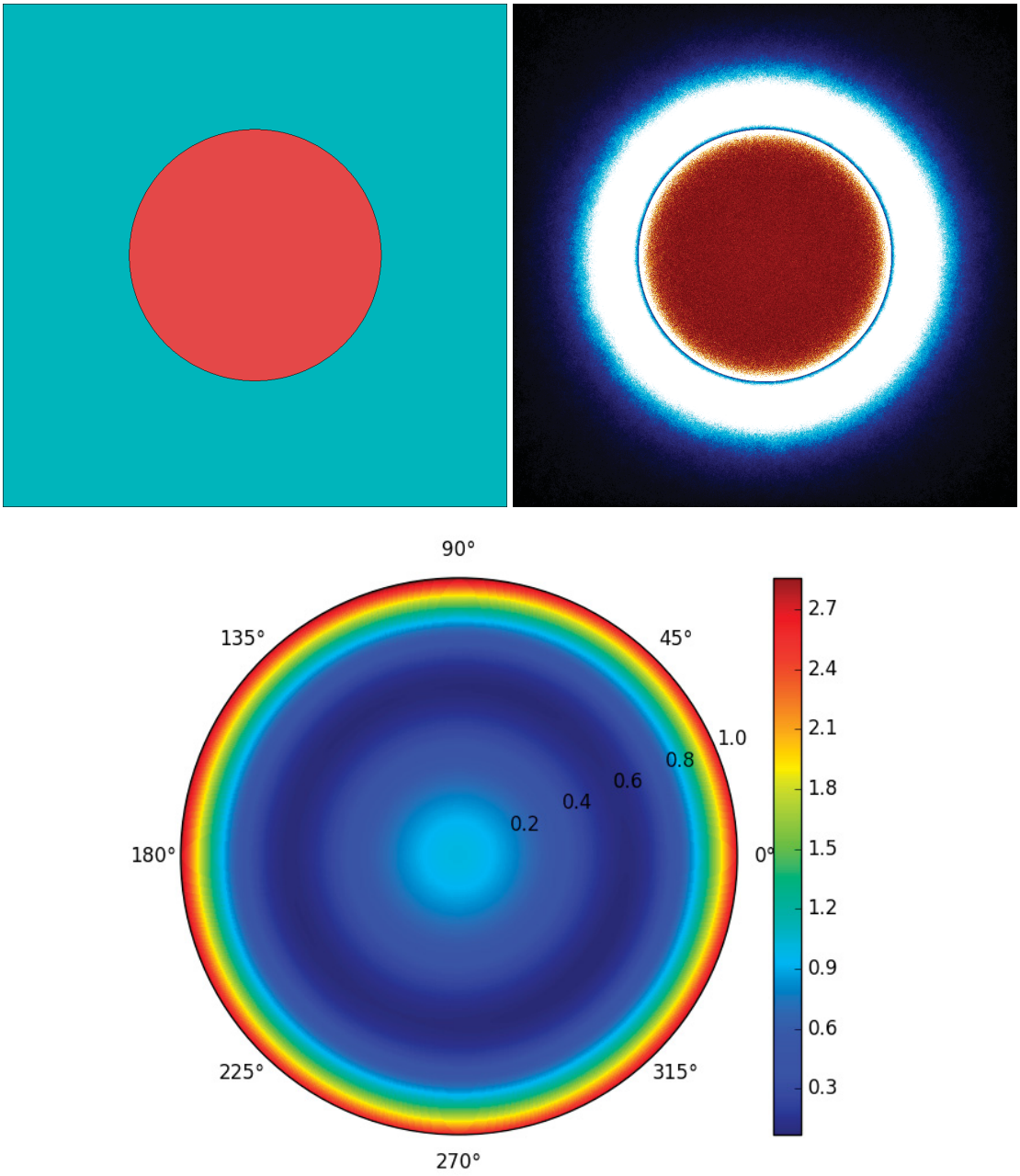

Figure 10: Bare fuel cylinder. *Top-left:* Geometry. *Top-right:* Power/flux plot. *Bottom:* Fission power peaking factor reconstruction.

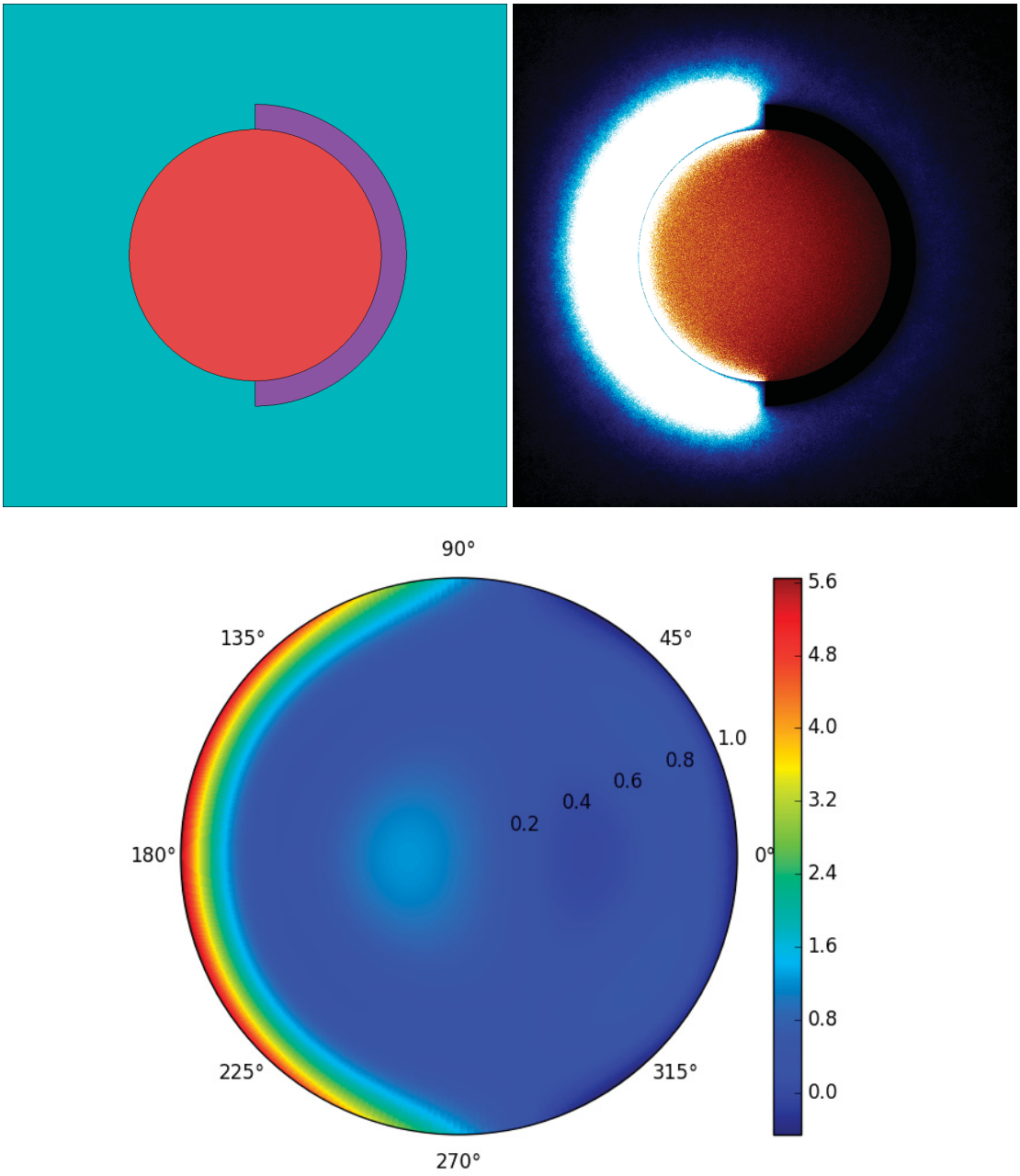

Figure 11: Half-covered fuel cylinder. *Top-left:* Geometry. *Top-right:* Power/flux plot. *Bottom:* Fission power peaking factor reconstruction.

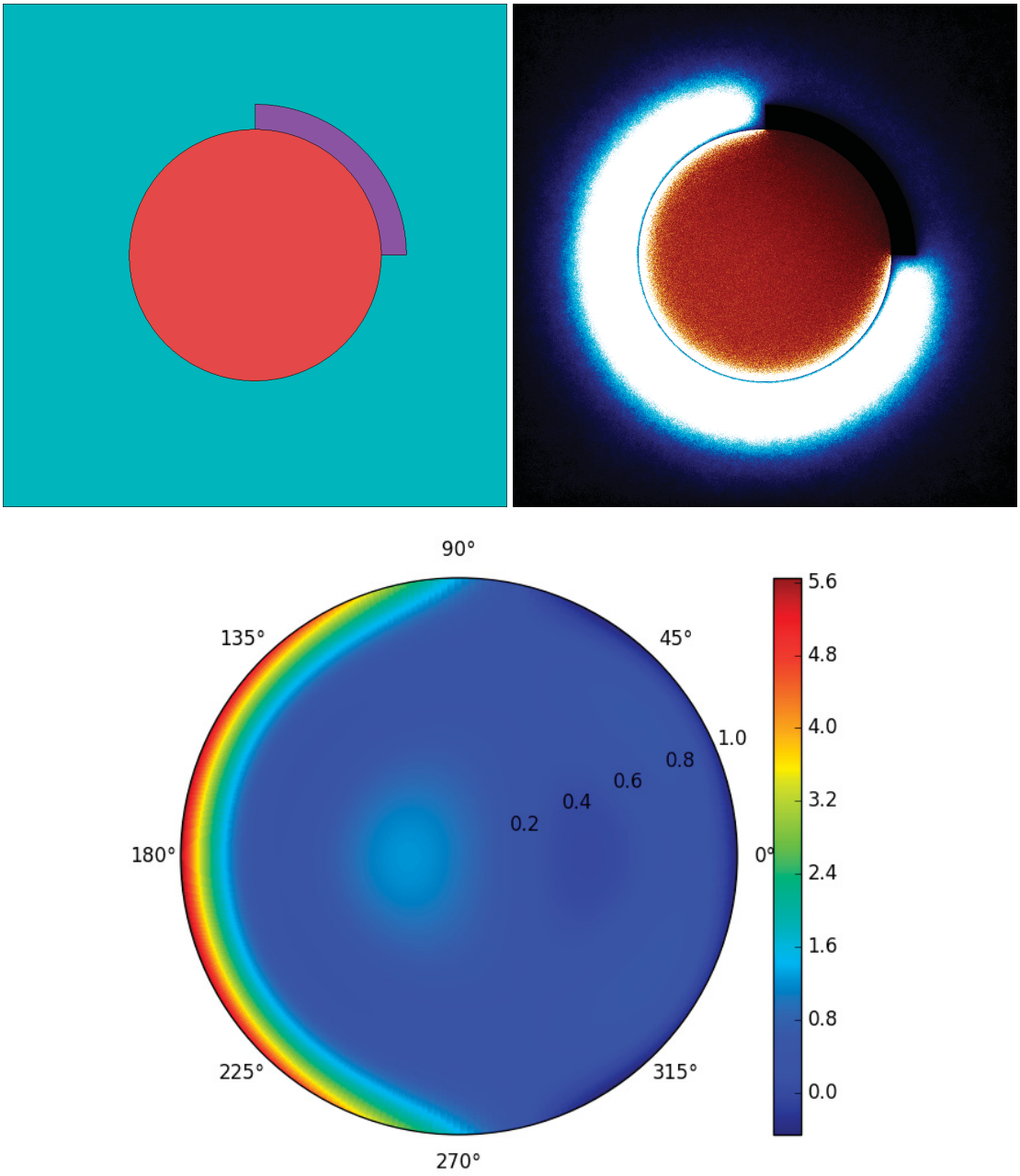

Figure 12: Quarter-covered fuel cylinder. *Top-left:* Geometry. *Top-right:* Power/flux plot. *Bottom:* Fission power peaking factor reconstruction.

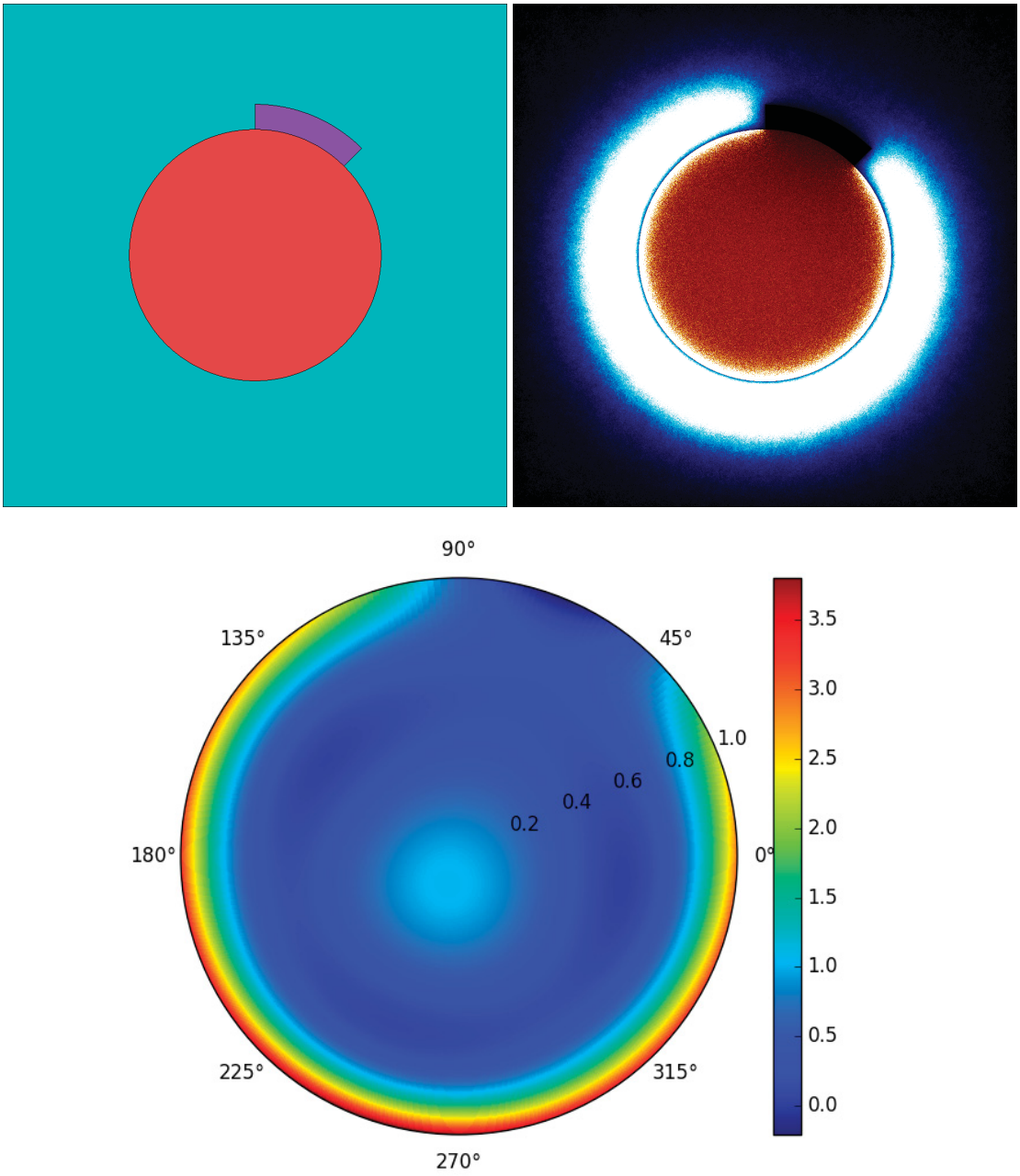

Figure 13: Eighth-covered fuel cylinder. *Top-left:* Geometry. *Top-right:* Power/flux plot. *Bottom:* Fission power peaking factor reconstruction.

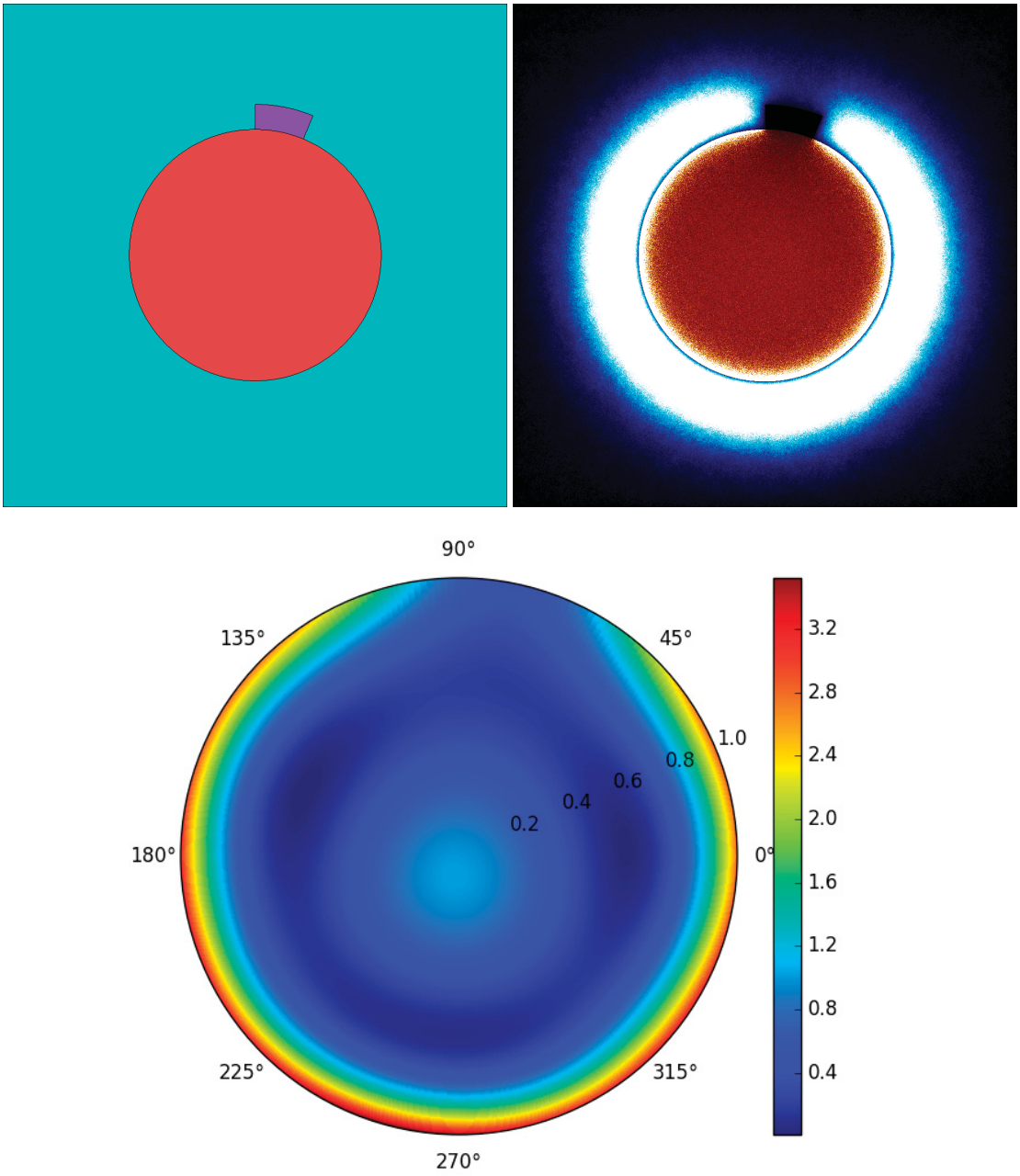

Figure 14: Sixteenth-covered fuel cylinder. *Top-left:* Geometry. *Top-right:* Power/flux plot. *Bottom:* Fission power peaking factor reconstruction.

Figures [10] to [14] show the radial and azimuthal peaking factors that were obtained at the axial midpoint $z = 0$ of the fuel cylinder. Its behavior appears to make sense, with power production being depressed locally where the absorber is present. Using 21 Zernike polynomials appears to resolve small peturbations in the neutron flux as shown in Fig. [14].

4.3 Results

To quantitatively measure the accuracy of functional expansion tallies in resolving the spatial distribution of the fission power, a benchmark PWR fuel assembly was developed. A radially infinite, axially finite assembly of PWR fuel pins 366 cm in height was used to examine how an axially linear density and temperature distribution in the coolant would affect the fission power distrubtion in the fuel.

Parameter	**Value**
Fuel	UO_2 (4.5% ^{235}U)
Clad	Zircalloy
Coolant	H_2O
Fuel Outer Radius	$4.095\,75 \times 10^{-1}$ cm
Void Outer Radius	$4.178\,30 \times 10^{-1}$ cm
Clad Outer Radius	$4.749\,80 \times 10^{-1}$ cm
Assembly Pitch	1.259 84 cm
Active Fuel Height	366 cm
Fuel Nominal Density	10.424 g cm^{-3}
Coolant Inlet Density	0.742 76 g cm^{-3}
Coolant Outlet Density	0.664 52 g cm^{-3}
Coolant Inlet Temperature	291.9 °C
Coolant Outlet Temperature	325.8 °C

Table 7: Geometric and material parameters of fuel assembly. Note that the coolant temperature and density is linearly interpolated between the inlet and outlet values.

To compare results, the fuel pin was partitioned into 10 equally-spaced axial zones and 20 radial zones. The first five innermost radial zone have the same volume and the last fifteen outermost zones have the same (smaller) volume. This was done since the change in neutron flux tends to change significantly near the outer edge of the fuel pin. With 10 axial

zones and 20 radial zones, the neutron neutron fission power in 200 zones was measured using Serpent detector cards. The functional expansion tally was carried out to 5^{th} order Legendre and 4^{th} order Zernike polynomials.

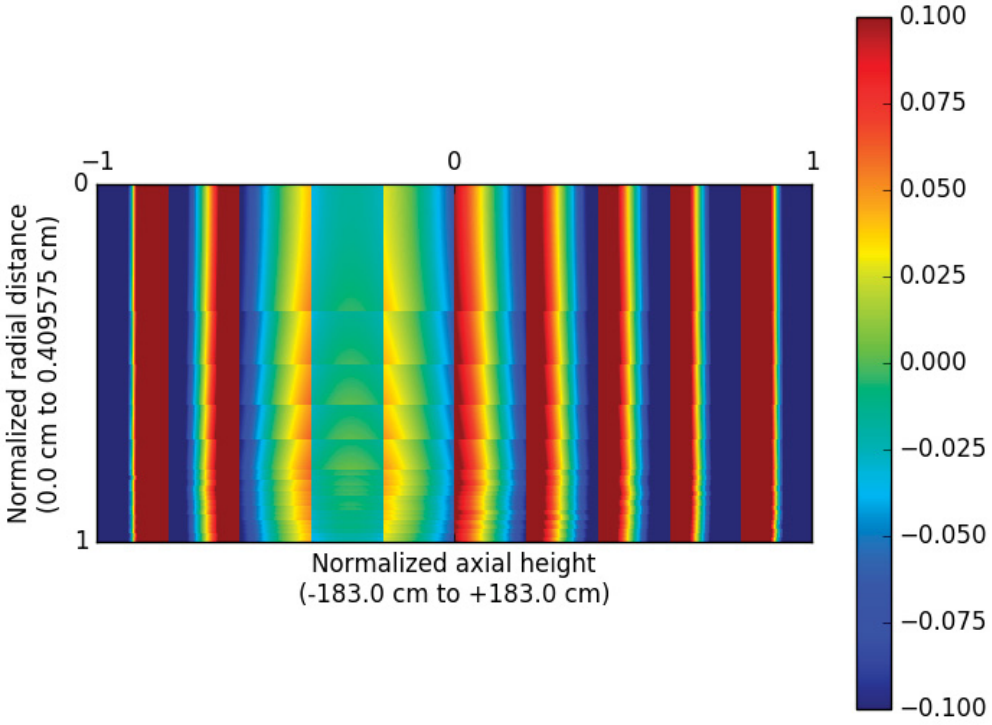

Figure 15: Comparison of axial distribution of fission power in fuel pin.

Figure [15] shows the relative error between the discrete Serpent detector and continuous Functional Expansion Tally for the fissioin power. It is computed by

$$\frac{\text{FET Power Density} - \text{Detector Power Density}}{\text{Detector Power Density}}. \tag{9}$$

Note that areas in dark red represent where the FET overestimates the fission power relative to the detector and areas in dark blue are areas where the FET underestimates the

fission power relative to the detector. There are nine dicontinuities in the relative error corresponding to transitions between detector regions. In general, the FET seems to match the detector values near their midpoints. The regions with the largest relative errors are at the top and bottom of the fuel pin where the fission power is small and the denominator in Eq. [9] approaches zero. Since the plot of relative error is quite unwieldy, the fission power density was plotted along one-dimensional cross sections of the fuel pin.

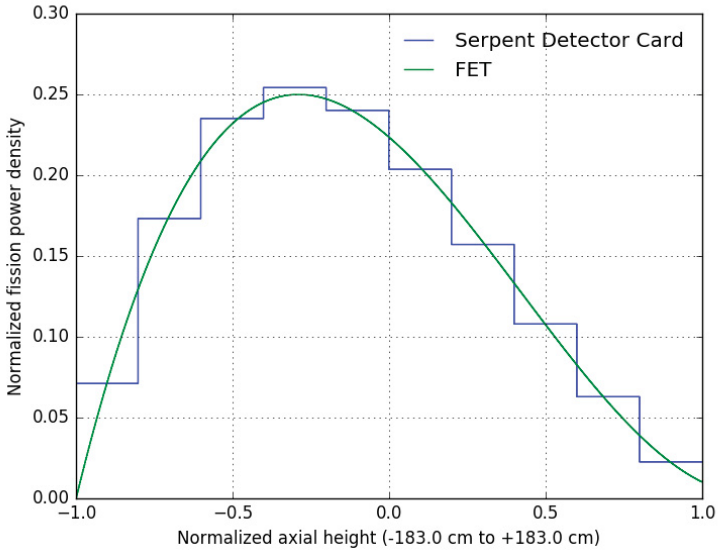

Figure 16: Axial distribution of fission power density along centerline ($r = 0$).

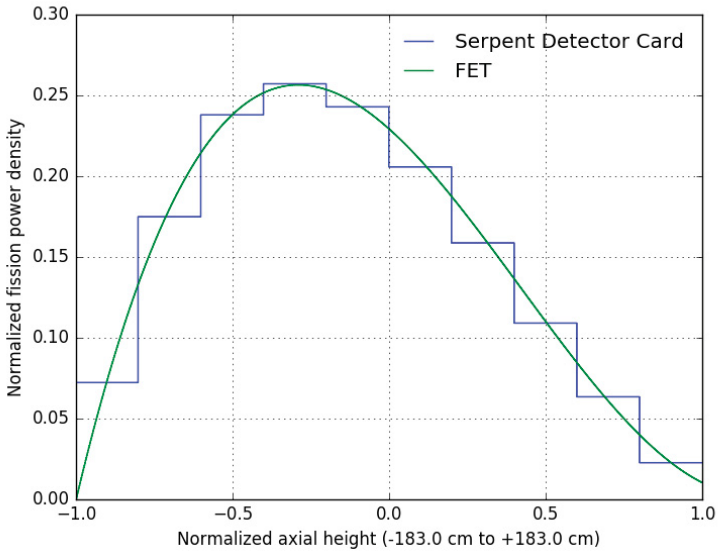

Figure 17: Axial distribution of fission power density at radial distance between centerline and outer edge ($r = 0.5$).

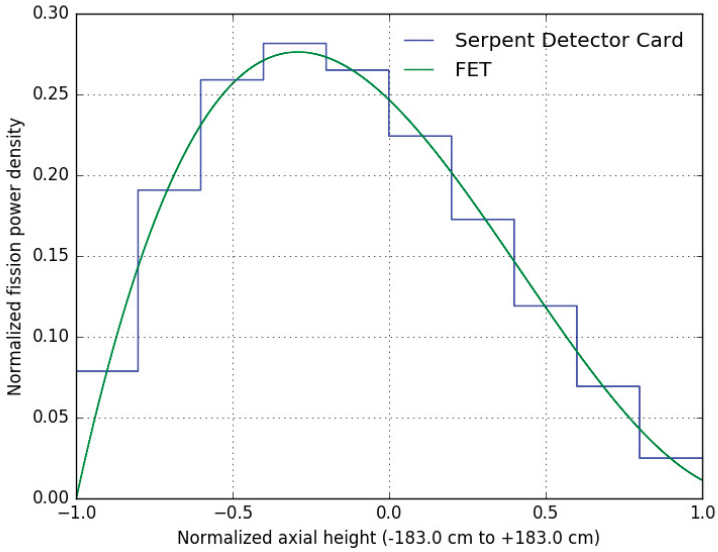

Figure 18: Axial distribution of fission power density along outer edge of fuel pin ($r = 1.0$).

Figures [16] to [18] show the fission power density along different radial sections of the fuel pin. The fission power reaches a maximum closer to the coolant inlet ($z = -1.0$) due to increased moderation from denser water. Closer to the edge of the fuel pin in Fig [18], there is a significant increase in fission power density due to the fuel's proximity to the moderator.

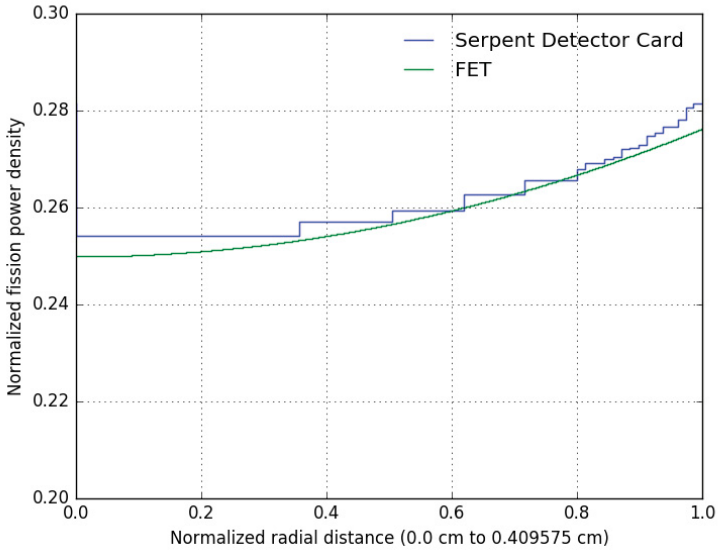

Figure 19: Radial distribution of fission power density at maximum axial power ($r = -0.28$).

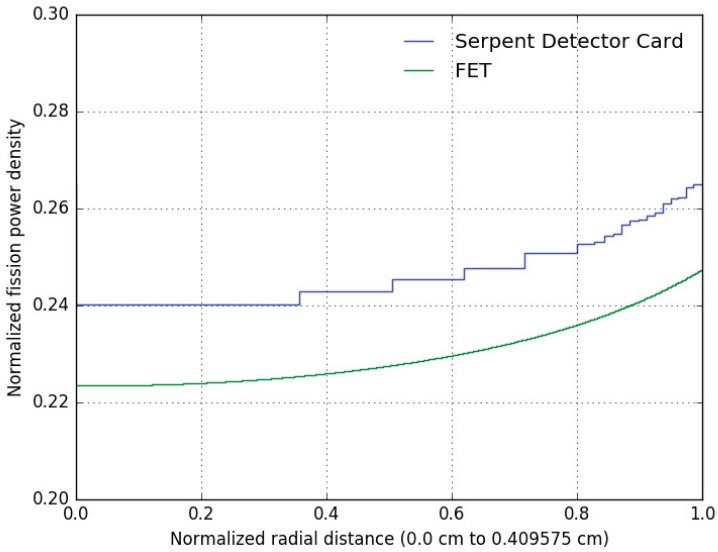

Figure 20: Radial distribution of fission power density at axial point just below midpoint ($r \approx 0$).

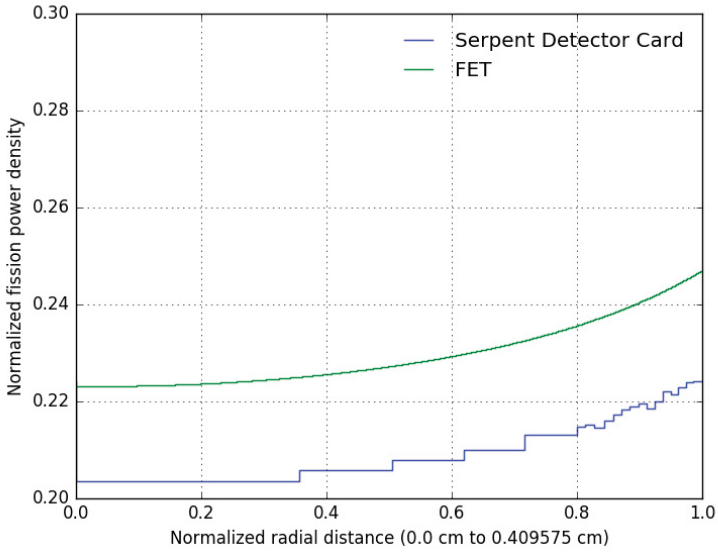

Figure 21: Radial distribution of fission power density at axial point just above midpoint ($r \approx 0$).

Figures [19] to [21] show the fission power density along different axial cross sections of the fuel pin. The difference between Figs. [20] and [21] demonstrate the discontinuity in relative error shown in Fig. [15]. Immediately before the interface in Fig. [20], the FET underestimates the fission power density relative to the detector. Immediately after the interface in Fig. [21], the FET overestimates the fission power density relative to the detector. Since the FET appears to match with the midpoints of the detectors, this indicates that smaller detector sizes would still agree with FET results.

5 Conclusions and Future Work

A parallelizable coupled Serpent–MOOSE code has been created and Functional Expansion Tallies (FETs) have been implemented within Serpent. The coupled Serpent–MOOSE code simulated a slightly lower k_{eff} for the single fuel pin case than the standalone Serpent. At this point it is unclear if this is due to thermal feedback of the rising fuel temperatures or to statistical variance. The FETs implemented within Serpent provide a smooth, continuous function which has many advantages over the discontinuous tally system it replaces. The FETs were accurate representations of fission power in fuel pin geometries. Both methods show promise of usefulness in reactor analysis at INL.

Further work remains to implement FETs into the coupled Serpent–MOOSE version. An interface system which not only outputs FETs from Serpent to MOOSE, but also outputs FETs from MOOSE to Serpent, would be desirable. It is true that the term Functional Expansion Tally is not absolutely correct when used to describe output from MOOSE, as MOOSE, not being a Monte Carlo code, does not have "tallies" in the sense of how FETs are theoretically employed. Nevertheless, it would be possible to distill the MOOSE distributions into functional expansions and therefore we will continue to use the term "FETs" to describe this final functional-expansion solution. In addition, such an interface system would also make it easy to store BISON fuel pin power profiles as functional expansions instead of full meshes, leading to considerable memory savings.

Coupling of Serpent and BISON is also a "next step" in this work, in order to make the coupled Serpent more useful. Furthermore, investigation of the use of MultiApps instead of Userobjects should be undertaken. Lastly, at present the coupling uses the single-material multiphysics Serpent interface. This should be expanded to include multiple materials.

References

[1] J. Leppänen, M. Pusa, T. Viitanen, V. Valtavirta, and T. Kaltiaisenaho. The serpent monte carlo code: Status, development and applications in 2013. *Ann. Nucl. Energy*, 82:142–150, 2015.

[2] Javier Ortensi, Mark D. DeHart, Frederick N. Gleicher, Yaqi Wang, Sebastian Schunert, Anthony L. Alberti, and Todd S. Palmer. Full core treat kinetics demonstration using rattlesnake/bison coupling within mammoth. INL Report, INL/EXT-15-36268, 2015.

[3] Frederick N. Gleicher, Javier Ortensi, Benjamin W. Spencer, Yaqi Wang, Stephen R. Novascone, Jason D. Hales, Derek Gaston, Richard L. Williamson, and Richard C. Martineau. The coupling of the neutron transport application rattlesnake to the nuclear fuels performance application bison under the moose framework. PHYSOR 2014, The Role of Reactor Physics toward a Sustainable Future, Kyoto, Japan, September 28 – October 3, 2014.

[4] Paul K. Romano, Nicholas E. Horelik, Bryan R. Herman, Adam G. Nelson, Benoit Forget, and Kord Smith. Openmc: A state-of-the-art monte carlo code for research and development. *Ann. Nucl. Energy*, 82:90–97, 2015.

[5] D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandie. Moose: A parallel computational framework for coupled systems of nonlinear equations. *Nucl. Engrg. Design*, 239:1768–1778, 2009.

[6] M. Ellis, B. Forget, and K. Smith. Preliminary coupling of the monte carlo code openmc and the multiphysics object-oriented simulation environment (moose) for analyzing doppler feedback in monte carlo simulations. ANS MC2015, Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method, Nashville, Tennessee, April 19–23, 2015.

[7] M. Ellis, B. Forget, and K. Smith. Continuous temperature representation in coupled openmc/moose simulations. PHYSOR 2016, Unifying Theory and Experiments in the 21st Century, Sun Valley, Idaho, May 1–5, 2016.

[8] Jaakko Leppänen, Ville Valtavirta, and Tuomas Viitanen. Unstructured mesh based multi-physics interface for cfd code coupling in the serpent 2 monte carlo code. PHYSOR 2014, The Role of Reactor Physics toward a Sustainable Future, Kyoto, Japan, September 28 – October 3, 2014.