ORNL/TM-2017/325

Warthog: Coupling Status Update

Shane W. D. Hart
Bradley T. Rearden

Approved for public release.
Distribution is unlimited.

June 30, 2017

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

DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via US Department of Energy (DOE) SciTech Connect.

Website http://www.osti.gov/scitech/

Reports produced before January 1, 1996, may be purchased by members of the public from the following source:

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Telephone 703-605-6000 (1-800-553-6847)
TDD 703-487-4639
Fax 703-605-6900
E-mail info@ntis.gov
Website http://www.ntis.gov/help/ordermethods.aspx

Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange representatives, and International Nuclear Information System representatives from the following source:

Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831
Telephone 865-576-8401
Fax 865-576-5728
E-mail reports@osti.gov
Website http://www.osti.gov/contact.html

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM-2017/325

Reactor and Nuclear Systems Division

WARTHOG: COUPLING STATUS UPDATE

Shane W. D. Hart
Bradley T. Rearden

Date Published: June 30, 2017

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

ABSTRACT

The Warthog code was developed to couple codes that are developed in both the Multi-Physics Object-Oriented Simulation Environment (MOOSE) from Idaho National Laboratory (INL) and SHARP from Argonne National Laboratory (ANL). The initial phase of this work, focused on coupling the neutronics code PROTEUS with the fuel performance code BISON. The main technical challenge involves mapping the power density solution determined by PROTEUS to the fuel in BISON. This presents a challenge since PROTEUS uses the MOAB mesh format, but BISON, like all other MOOSE codes, uses the libMesh format.

When coupling the different codes, one must consider that Warthog is a light-weight MOOSE-based program that uses the Data Transfer Kit (DTK) to transfer data between the various mesh types. Users set up inputs for the codes they want to run, and then Warthog transfers the data between them. Currently Warthog supports XSProc from SCALE or the Sub-Group Application Programming Interface (SGAPI) in PROTEUS for generating cross sections. It supports arbitrary geometries using PROTEUS and BISON. DTK will transfer power densities and temperatures between the codes where the domains overlap.

In the past fiscal year (FY), much work has gone into demonstrating two-way coupling for simple pin cells of various materials. XSProc was used to calculate the cross sections, which were then passed to PROTEUS in an external file. PROTEUS calculates the fission/power density, and Warthog uses DTK to pass this information to BISON, where it is used as the heat source. BISON then calculates the temperature profile of the pin cell and sends it back to XSProc to obtain the temperature corrected cross sections. This process is repeated until the convergence criteria (tolerance on BISON solve, or number of time steps) is reached. Models have been constructed and run for both uranium oxide and uranium silicide fuels. These models demonstrate a clear difference in power shape that is not accounted for in a stand-alone BISON run.

Future work involves improving the user interface (UI), likely through integration with the Nuclear Energy Advanced Modeling and Simulation (NEAMS) Workbench. Furthermore, automating the input creation would ease the user experience. The next priority is to continue coupling the work with other codes in the SHARP package. Efforts on other projects include work to couple the Nek5000 thermo-hydraulics code to MOOSE, but this is in the preliminary stages.

1. INTRODUCTION

The Nuclear Energy Advanced Modeling and Simulation (NEAMS) program in the US Office of Nuclear Energy at the US Department of Energy (DOE) funds two comprehensive frameworks that can be used by developers to provide for advanced simulation techniques for various aspects of simulating a nuclear reactor. The Fuels Product Line (FPL) at the Idaho National Laboratory (INL) is developing the Multi-Physics Object-Oriented Simulation Environment (MOOSE) framework[1] upon which the BISON fuel performance code[2] is based. In parallel, the Reactor Product Line (RPL) at Argonne National Laboratory (ANL) has been developing the SHARP framework[3], which facilitates the use of multiple codes including PROTEUS[4] and Nek5000[5]. MOOSE focuses on providing a multi-physics framework which uses a top-down approach in which the framework is constructed first, and then the applications are built on top of the framework. SHARP provides a framework based on a bottom-up approach in which applications are developed, and then a framework is developed to tie them together. Both are valid approaches to multi-physics problems, but their independent development has resulted in framework components that are difficult to mix and match.

Previous work[6] led to the development of the coupling code *Warthog* to couple these two different frameworks together. Initial work has focused on coupling the fuel performance code BISON, built on the MOOSE framework, to the neutronics code PROTEUS, built on the SHARP framework. This work demonstrated the feasibility of coupling a simple pin cell between the codes. Work was performed to generate cross sections at the correct temperatures using the XSProc code of the SCALE[7] code system. However, at this earlier stage, there was no two-way coupling: PROTEUS was able to generate a fission distribution for BISON, but results of the BISON calculation would not update the PROTEUS temperatures.

This report documents work that builds upon the previous Warthog work to fully enable two-way coupling between PROTEUS and BISON. Cross sections are fully updated between each time step using the temperatures calculated by BISON, and the cross sections are used in each subsequent PROTEUS run to provide an updated fission power density to BISON. This report contains an example of a pincell complete with two-way coupling, and it provides descriptions of requirements needed to expand this to a full assembly model using MOOSE's MultiApp feature. In addition, future work is discussed, including coupling MOOSE with other codes of the SHARP suite.

2. PREVIOUS WORK

2.1 COUPLING

Previously a simple coupling scheme was present in Warthog that allowed the transfer of data between PROTEUS and BISON. However, there was no way to update the cross sections used by PROTEUS when BISON was finished calculating the temperatures. With the advice of the MOOSE team, Warthog was moved to a MultiApp approach in which each component of the coupling run was moved to a separate MooseApp. While the high-level flow of Warthog is the same as that shown in Figure 1, the control is now managed by MultiApps instead of having just one MasterApp that contains modules to run each code. The use of MultiApps allows for MOOSE to control running the codes in parallel and to ease transfers of values on the mesh.

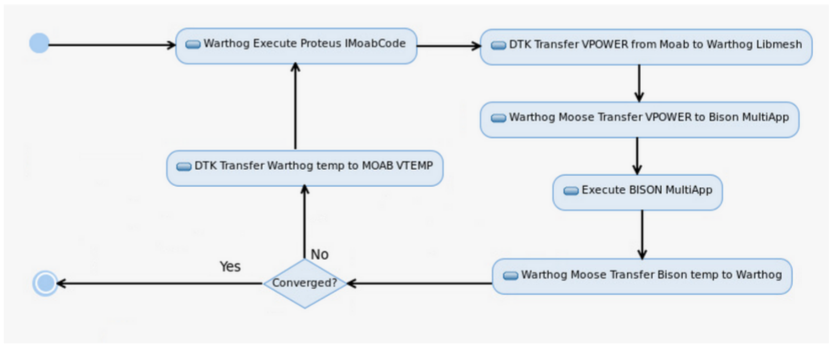

Warthog Execute Proteus IMoabCode
DTK Transfer VPOWER from Moab to Warthog Libmesh
Warthog Moose Transfer VPOWER to Bison MultiApp
DTK Transfer Warthog temp to MOAB VTEMP
Execute BISON MultiApp
No
Yes
Converged?
Warthog Moose Transfer Bison temp to Warthog

Figure 1: High level flow for a Warthog execution.

While MOOSE can use built-in functions to transfer the mesh data between the various MultiApps, Warthog requires the use of Data Transfer Kit (DTK)[8] to transfer data between MOOSE/BISON and PROTEUS due to the different mesh libraries used by the two codes. MOOSE and all other MOOSE "animals" use the libMesh mesh format[9]. PROTEUS and other SHARP codes use the ANL-developed MOAB format[10]. DTK is developed at Oak Ridge National Laboratory (ORNL) and supports transferring variables and other mesh data between a wider variety of mesh types.

Part of the previous work involved refactoring the Warthog code to allow all external codes to be called as MOOSE MultiApps. Each external code (XSProc, PROTEUS, BISON) is a separate MooseApp that can be called from a main Warthog input file. This main Warthog input file can control the parallelism and the data transfers to the MultiApps. See Appendix A.1 for an example master input file.

2.2 EXTERNAL CODE MULTIAPPS

Each external code MultiApp added by Warthog is discussed briefly below.

2.2.1 XSPROC

PROTEUS can obtain cross sections in a couple of different formats, including either an ASCII text file in the ANLXS format or a binary ISOTXS-formatted file. These file formats are described in the PROTEUS

User's Guide. Cross sections can also be calculated by a special Sub-Group Application Programming Interface (SGAPI)[11] provided with some versions of PROTEUS.

The SGAPI can take a significant amount of time to calculate the cross sections. To aid in the rapid development of Warthog, the SCALE module XSProc was wrapped as a external code callable through a Warthog MultiApp. XSProc can create multigroup self-shielded cross sections based on various fuel parameters. XSProc takes up to a minute to run, which is much faster than subgroup methods. XSProc is written in C++ and can be used through a library API.

Originally XSProc was added to Warthog as an ExternalCode that could be called through a special input block parameter. With the refactoring of Warthog, XSProc is now called through a MultiApp that wraps the XSProc code in a special XSProc Executioner. Executioners in MOOSE programs are program units that control the execution of the underlying solve. The XSProc Executioner reads in some parameters from the input file and runs the XSProc solve. After the XSProc solve, the AMPX-formatted[12] multigroup cross sections are converted to the ASCII-formatted ANLXS format that PROTEUS can read.

After every PROTEUS/BISON run, XSProc can be run again with the updated temperatures to calculate new cross sections. The overall flow of the process is shown in Figure 2.

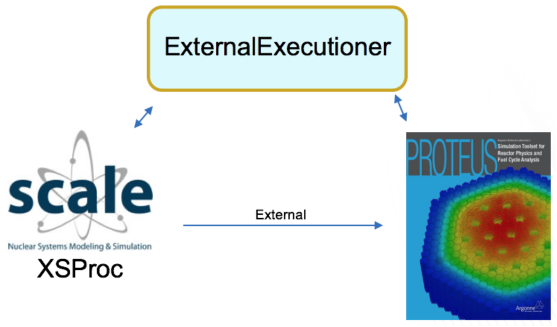


Figure 2: Flow of XSProc to PROTEUS execution.

2.2.2 PROTEUS

As mentioned above, PROTEUS can use the MOAB mesh format for the solve. Since BISON and other MOOSE programs use the libMesh format, DTK must be used to transfer the temperature and fission power data between the two programs. The input format for the PROTEUS MultiApp is similar to that of the XSProc MultiApp. Various parameters can be given to PROTEUS through the input blocks of the file. However, most parameters will be set in the PROTEUS input file itself. When PROTEUS is initialized through Warthog, it will read this input file for the majority of the PROTEUS parameters.

PROTEUS works fastest when boundary conditions are void, but it can solve a reflected boundary problem. This allows the simulation of a pin cell that is reflected on all sides. An example mesh for such a problem is shown in Figure 3. PROTEUS can use both tetrahedral and quadratic meshes, but it only supports lower order meshes.

2.2.3 BISON

Because BISON is already a MooseApp, it does not require modifications to run inside Warthog as a MultiApp. To successfully run BISON, the user must provide a BISON input file and set up the mesh

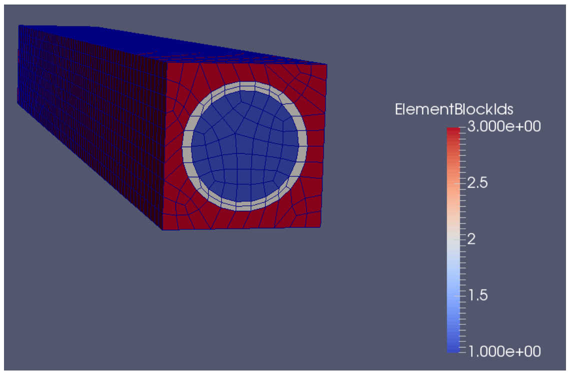


Figure 3: Mesh used by PROTEUS.

transfer inside the Warthog master input file. To ease the mesh transfer, it is beneficial to have the PROTEUS and BISON meshes to be as similar as possible. The main difference between the two meshes will be the lack of a moderator in the BISON mesh. Also, the BISON mesh supports higher order mesh elements if the user wishes to enable them.

An example mesh for BISON showing the temperature distribution at the beginning of a run is shown in Figure 4.

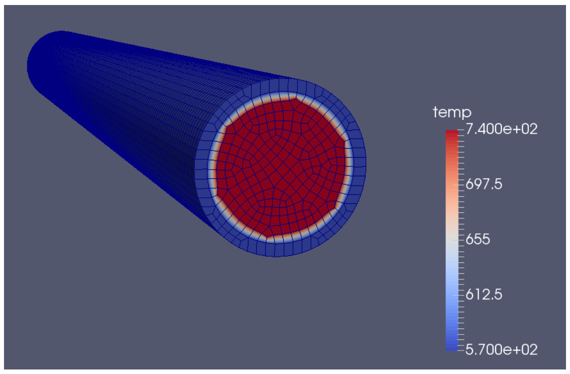


Figure 4: Temperature distribution at beginning of Warthog run.

3. CURRENT WORK

This section describes the work performed on Warthog this fiscal year (FY).

3.1 COUPLING PROGRESS

At the end of the last FY, Warthog was able to run the neutronics code PROTEUS and send the fission power density to BISON through DTK and the built-in mesh transfer capabilities. BISON was then able to run with the given power density. However, the process did not include the ability of PROTEUS to update the cross sections based on the temperature feedback given by BISON.

Adding the ability to recalculate the cross sections using the latest temperature data and then pass that information to PROTEUS was one of the main focuses of the work that was performed during this FY. The cross sections required recalculating after each step to capture the temperature effects. As before, XSProc was used to calculate the self-shielded cross sections, and a conversion tool was used to convert the AMPX formatted cross sections to the ANLXS format that could be read by PROTEUS. Updating the cross sections inside PROTEUS required identifying the subroutines of PROTEUS that managed the cross sections and determining then how to call them to update the cross sections without having the other data become corrupted or deleted.

PROTEUS stores the cross sections in Fortran data containers consisting mostly of plain arrays of values. These values are initialized through various reading and initialization routines that do the following:

- Optionally read the ANLXS file, convert it internally to ISOTXS, and write it to disk.
- Read the ISOTXS file to size the appropriate arrays, and prepare to store the data.
- Read the ISOTXS file to obtain the cross sections, and place them into the Fortran arrays.

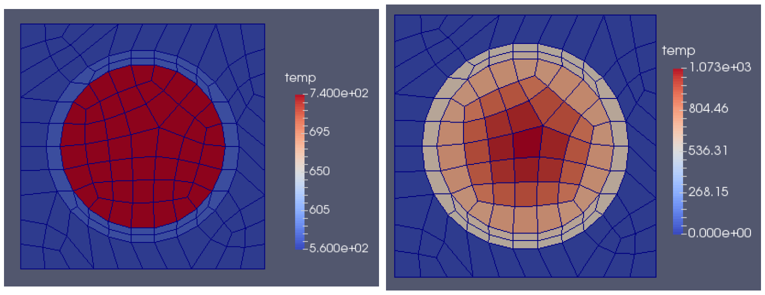
To update the cross sections between time steps, these routines must be called so that new data can be retrieved from XSProc. Currently PROTEUS can only read new cross sections from disk, and there is no easy way to pass through new cross sections using an Application Programming Interface (API) or other programming method. Also, there is no easy way to change the mesh. PROTEUS does allow some mesh deformation, but this has yet to be investigated.

3.2 COUPLING EXAMPLES

3.2.1 Light Water Reactor (LWR) FUEL PIN

The initial coupling problem was to model an LWR pin cell. For ease of the neutronics, the pin cell calculation was reflected on all sides. This should approximate the power distribution of an interior assembly pin cell and enable testing of the coupling approach. The mesh used for this problem is the same as that shown in Section 2.2.2. For ease of meshing and calculation, the gap has been homogenized with the clad. This will ensure that the solver used in PROTEUS is able to converge quickly.

The initial temperature distribution for the case is given in Figure 5a. The case was run using 60-second time steps until the temperatures had reached a stable state. For each time step, the cross sections were calculated using XSProc and passed to PROTEUS for the neutronics calculation. The fission distribution was then passed to BISON to calculate a new temperature, which then went back to XSProc for new cross sections. This process is repeated until a predefined time, usually after the temperatures in BISON have converged to some final temperature. The final temperatures are shown in Figure 5b.



(a) Initial **(b) Final**

Figure 5: Temperature for LWR fuel pin in XY plane.

Since there is no gap between the fuel and the clad, there is no fuel swelling or clad contact modeled with this approach. In addition, boundary conditions are set so that the fuel does not move. This case is mainly a demonstration of the two-way coupling feature in Warthog.

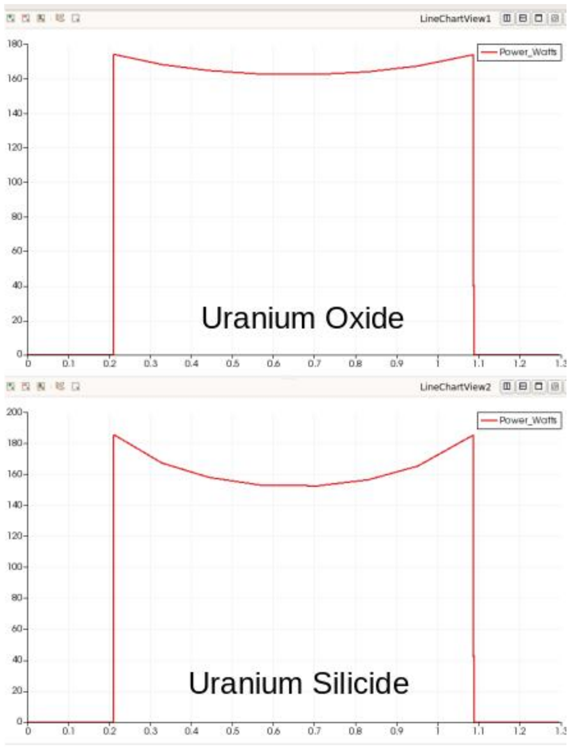
3.2.2 URANIUM SILICIDE FUEL PIN

Currently BISON uses the same default power profile for all fuel types. Two of the goals of this effort is to show that using the same profile is inadequate and to demonstrate the differences in the power profile between different types of fuel. Therefore a model was created with the same physical dimensions as the LWR fuel pin described above, except uranium silicide was used as the fuel material. The differences in the power shape can then be captured and are shown in Figure 6a. For both of the cases, the power was normalized to 300 W, and the output from PROTEUS was viewed. It is clear that the default uranium silicide model has a significantly different power shape than the uranium oxide model. The power peaking at the edge of the fuel is more pronounced, as is the trough in the center of the pin. These power shapes can be compared to the fission rate used in BISON as shown in Figures 6b and 6c. BISON starts with a flat fission rate, and moves towards a U-shape as the fuel is burned. This fission rate is the same for all fuel materials.

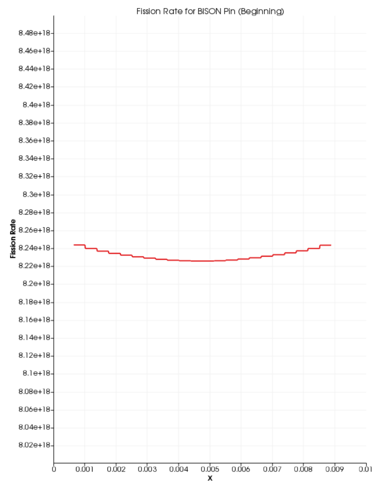
paragraphNew Geometry Mesh

The fuel/clad interaction is of particular interest in BISON. Since the previous demonstration model had no gap between the fuel and the clad, this phenomenon cannot be modeled. BISON includes a Python script that can be used to easily generate a variety of fuel geometry meshes. The tool is described fully in the BISON manual. While the tool can generate both 2D and 3D meshes, only 3D meshes will be investigated at this stage due to limitations in PROTEUS which arise when converting a 2D mesh from the Exodus format to the PROTEUS internal format.

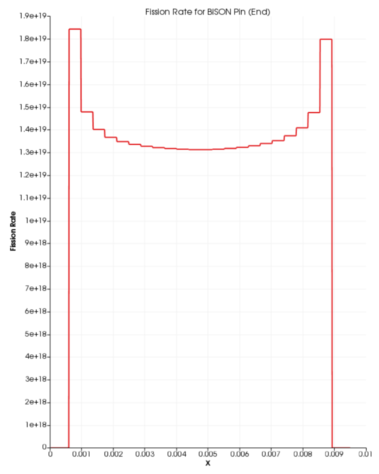
When creating a 3D mesh using the Python script, an angle can be given that determines how much of the pin is actually meshed. 360° meshes are created by setting the angle to 360° and 180° half meshes are created by setting the option to 180°. The 180° mesh created for this step is shown in Figure 7 which shows a clearly visible gap between the ends of the fuel and the clad. Radially there is also a small gap, but this is hard to see in the given figure. The expectation is that as time passes and the fuel heats up, the fuel will expand and swell to contact the cladding.



(a) Power shape for uranium oxide and uranium silicide fuels.



(b) BISON fission rate at beginning of run.



(c) BISON fission rate at end of run.

Figure 6: Power profiles.

Like the original mesh, the BISON mesh must be modified to be used with PROTEUS. The water moderator must be added, and the sides must be reflected to ensure that the power distribution is accurate. In addition, to ease meshing, the full pin was meshed instead of just 180° of the pin.

To generate the cross sections, the XSProc input was modified to use uranium silicide fuel with 5% enrichment, and the dimensions of the fuel, clad, and gap were modified to match that of the geometry generated by the BISON Python script. From the XSProc input, it is trivial to move to a full continuous energy (CE) Monte Carlo KENO-VI input. Running a KENO case would allow verification of what the eigenvalue produced by PROTEUS should be. Views of the KENO model are given in Figure 8. The eigenvalue calculated by a fully reflected KENO model is 1.49183 +/- 0.00019, this compares favorably with the PROTEUS eignvalue of 1.4544.

PROTEUS had some problems with convergence due to the low density material (helium) in the gap. The small size of the gap may also be an issue. The PROTEUS code seems to have convergence issues in solving this case. While the eigenvalue begins to approach the solution generated by KENO, the time taken

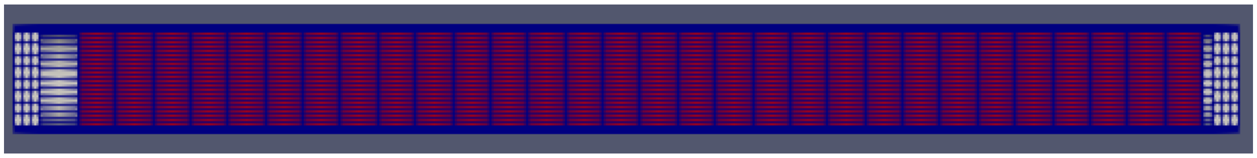
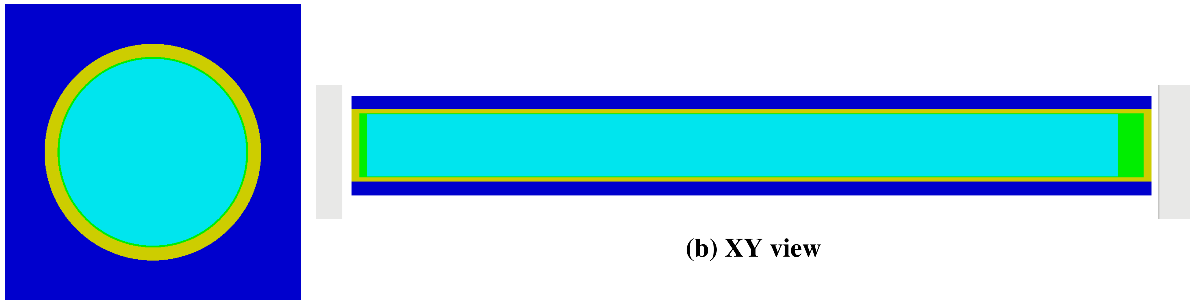


Figure 7: 180° BISON fuel mesh.



(a) XZ view

(b) XY view

Figure 8: KENO geometry of fuel.

is unacceptable for a quick coupling example, and the default iteration limit is eventually reached. Another indication of convergence issues is present in the power profile produced. Unlike the power profiles shown in Figure 6a, the power profile produced by this run is flat. The radial power profile is shown in Figure 9.

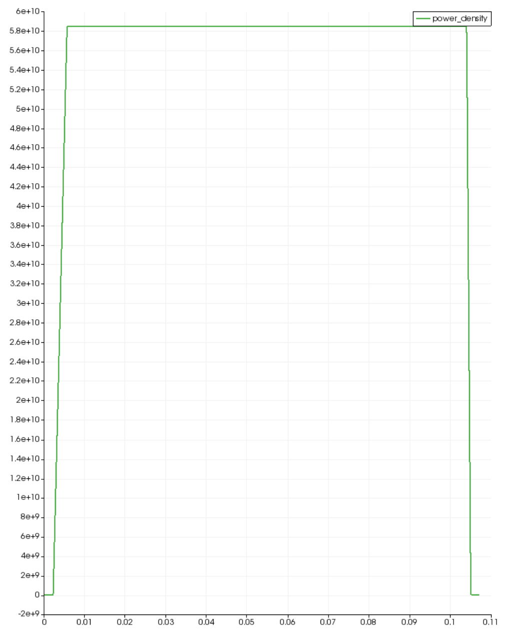


Figure 9: PROTEUS power distribution in uranium silicide fuel with gap.

Due to the nature of the power shape, it was deemed unnecessary at this stage to continue to investigate the impact of the differing power shape on the long-term coupling. The coupling has been demonstrated to work in models without the gap present, so efforts will be focused on ensuring ease of use and the ability to create accurate, complex PROTEUS models.

4. CONCLUSIONS AND FUTURE WORK

In recent work for this FY, the Warthog code has been able to show that it can couple simple PROTEUS and BISON pin cell cases. There is nothing inherently preventing more complicated geometries from being coupled, except for the limitations on what the user is willing to do to get timely results from PROTEUS. For example, a user could model an entire assembly in PROTEUS and then individually couple the power profiles to various BISON pin cells to determine the fuel performance of various pins in the assembly. As long as the different BISON pin meshes have varying geometries that overlap with a specific PROTEUS pin, then DTK will only map the appropriate power profile to the BISON case.

In addition, much work remains to be done to create a better user interface (UI). Currently the user must manually create the PROTEUS and BISON input files. There is an option to use an automatically created XSProc pin cell input, but the user will usually be required to provide that input, too. In addition to providing the inputs, the user must ensure that the mesh geometries are created and that they are sufficient for the problem at hand. Also, since DTK transfers mesh data only on domains that exist in both geometries, it is up to the user to ensure that the geometry meshes for both PROTEUS and BISON overlap in the regions of interest.

Some of the work towards creating a better UI will be accomplished by the continued development of the NEAMS Workbench[13]. This tool is being developed to bring together various NEAMS tools under a single UI. With the release of this tool, it should be easier for users to create their own Warthog inputs and tie them to other NEAMS tools.

Continuing work during this FY also includes coupling BISON with the Nek5000 code from ANL. Some preliminary work has been performed to couple Nek5000 with BISON, but that work is in the early stages. There are some key differences between the current coupling with Nek5000 and the coupling with PROTEUS. Nek5000 usually operates on a separate domain from BISON: that of the coolant channel outside of the fuel. The current coupling approached devised at ANL and INL uses transfer coefficients on the boundary between the two different meshes to transfer the required data from Nek5000 to BISON. This is different from the PROTEUS transfer, which uses DTK to transfer the actual data to the same domain on the BISON mesh.

Work will also continue on including Warthog as part of the MOOSE family of recognized programs. Coordination with INL will help this task succeed. The end goal is a program that enables easy transfer of data between the disparate frameworks of the NEAMS program.

5. REFERENCES

[1] D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandie, “MOOSE: A parallel computational framework for coupled systems of nonlinear equations,” *Nuclear Engineering and Design*, 239(10), October 2009, 1768–1778.

[2] D. Perez, R. Williamson, S. Novascone, G. Pastore, J. Hales, and B. Spender, *Assessment of BISON: A Nuclear Fuel Performance Analysis Code*, INL/MIS-13-30314, 2013.

[3] T. J. Tautges et al., *SHARP Assembly-Scale Multiphysics Demonstration Simulations*, ANL/MCS-NE-13-9, March 2013.

[4] E. R. Shemon, M. A. Smith, and C. Lee, *PROTEUS-SN User Manual*, ANL/NE-14/6 (Rev 3.0), February 2016.

[5] P. Fischer, et al., *Nek5000 User Documentation* ANL/MCS-TM-351.

[6] S. W. D. Hart, *Warthog: Progress on Coupling BISON and PROTEUS*, ORNL/TM-2016/579, September 2016.

[7] B. T. Rearden and M. A. Jessee, Eds., *SCALE Code System*, ORNL/TM-2005/39, Version 6.2.1, Oak Ridge National Laboratory, Oak Ridge, Tennessee (2016).

[8] S. R. Slattery and P. P. H. Wilson, *The Data Transfer Kit: A Geometric Rendevous-Based Tool for Multiphysics Data Transfer*, M&C 2013, Sun Valley, ID., 2013.

[9] B. S. Kirk, J. W. Peterson, R. H. Stogner, G. F. Carey, “libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations”, *Engineering with Computers*, Vol. 22, Issue 3, pp 237–254, Dec. 2006.

[10] T. J. Tautges, R. Meyers, K. Merkley, C. Stimpson, and C. Ernst, *MOAB: A mesh-oriented database*, SAND2004-1592. Sandia National Laboratories, Albuquerque, NM., 2004.

[11] C. H. Lee, A. Marin-Lafleche, and M. A. Smith, *Development of Cross Section Library and Application Programming Interface (API)*, ANL/NE-13/15, September 2013.

[12] D. Wiarda, M. E. Dunn, N. M. Greene, C. Celik, and L. M. Petire, *AMPX-6: A Modular Code System Processing ENDF/B*, ORNL/TM-2016/43, Oak Ridge National Laboratory, Oak Ridge, Tennessee (2016).

[13] B. T. Rearden, R. A. Lefebvre, et al,. “Introduction to the Nuclear Energy Advanced Modeling and Simulation Workbench”, *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering* (M&C 2017), April 2017

APPENDIX A. WARTHOG INPUT FILES

A.1 MASTER INPUT

An example input file for Warthog is given below. Note how this input file controls the various MultiApps that make up the coupling of Warthog.

```
[Mesh]
  type = FileMesh
  file = PROTEUS_Mesh.exo
[]

[AuxVariables]
  [./temp]
    family = monomial
    order = constant
  [../]
  [./power_density]
  [../]
  [./material_density]
    family = monomial
    order = constant
  [../]
[]

[MultiApps]
  [./my_xsproc]
    type = TransientMultiApp
    app_type = WarthogApp
    execute_on = timestep_begin
    positions = ’0.0 0.0 0.0’
    input_files = ’xsproc.i’
    max_procs_per_app = 1
  [../]
  [./my_proteus]
    type = TransientMultiApp
    app_type = WarthogApp
    execute_on = timestep_begin
    positions = ’0.0 0.0 0.0’
    input_files = ’proteus.i’
    max_procs_per_app = 1
  [../]
  [./my_bison]
    type = TransientMultiApp
    app_type = WarthogApp
    execute_on = timestep_end
    positions = ’0.0 0.0 0.0’
    input_files = ’bison.i’
    max_procs_per_app = 4
  [../]
[]

[Transfers]
  [./to_bison_pd]
    type = MultiAppMeshFunctionTransfer
    direction = to_multiapp
    multi_app = my_bison
    source_variable = power_density
    variable = power_density
  [../]
  [./from_bison]
```

```
    type = MultiAppMeshFunctionTransfer
    direction = from_multiapp
    multi_app = my_bison
    source_variable = temp
    variable = temp
  [../]
  [./temps_to_xsproc]
    type = MultiAppMeshFunctionTransfer
    direction = to_multiapp
    multi_app = my_xsproc
    source_variable = temp
    variable = temp
  [../]
  [./from_proteus]
    type = MultiAppMeshFunctionTransfer
    direction = from_multiapp
    multi_app = my_proteus
    source_variable = power_density
    variable = power_density
  [../]
[]

[ICs]
  [./mod]
    type = ConstantIC
    variable = temp
    value = 560.0
    block = mod
  [../]
  [./clad]
    type = ConstantIC
    variable = temp
    value = 570.0
    block = clad
  [../]
  [./fuel]
    type = ConstantIC
    variable = temp
    value = 740.0
    block = fuel
  [../]
  [./dens]
    type = ConstantIC
    variable = material_density
    value = 1.0
  [../]
[]

[Problem]
  solve = False
  type = FEProblem
[]

[Executioner]
  type = Transient
  num_steps = 60
  dt = 600
[]
```

```
[Outputs]
  exodus = 1
[]
```

A.2 XSPROC INPUT

An input for the XSProc MultiApp is given below. The `[Postprocessors]` input block is optional and serves to show the block temperatures at various time steps. The `[AuxKernels]` block ensures that the moderator temperature stays at a constant level.

```
[Mesh]
    type = FileMesh
    file = PROTEUS_Mesh.exo
[]

[AuxVariables]
  [./temp]
    family = monomial
    order = constant
  [../]
[]

[Postprocessors]
  [./temp_fuel]
    type = ElementAverageValue
    block = fuel
    execute_on = timestep_end
    variable = temp
  [../]
  [./temp_clad]
    type = ElementAverageValue
    block = clad
    execute_on = timestep_end
    variable = temp
  [../]
  [./temp_mod]
    type = ElementAverageValue
    block = mod
    execute_on = timestep_end
    variable = temp
  [../]
[]

[AuxKernels]
  [./set_mod_temp]
    type = ConstantAux
    block = mod
    execute_on = 'timestep_begin timestep_end initial linear nonlinear'
    value = 560.0
    variable = temp
  [../]
[]

[Problem]
  solve = false
  type = FEProblem
[]
```

```
[Executioner]
  type = XSProcExecutioner
  dt = 600
[]

[Outputs]
  exodus = 1
[]
```

A.3 PROTEUS INPUT

An input for the PROTEUS MultiApp is given below. This is not the input for the PROTEUS itself, merely the MOOSE MultiApp code that wraps the PROTEUS execution.

```
[Mesh]
    type = FileMesh
    file = PROTEUS_Mesh.exo
[]

[AuxVariables]
  [./temp]
  [../]
  [./power_density]
  [../]
  [./material_density]
  [../]
[]

[Problem]
  solve = false
  type = FEProblem
[]

[Materials]
  [./protMat]
    type = ProteusMaterial
  [../]
[]

[Executioner]
  type = ProteusExecutioner
  generateInputFile = false
  generateMaterialsFile = false
  dt = 600
[]

[Outputs]
  exodus = 1
[]
```