ANL-21/28 | NRIC 22-ENG-0001
Revision 0

Development of Advanced Reactor Models for Virtual Test Bed using NEAMS Tools

July 2021

Argonne National Laboratory

B. Feng, J. Fang, N. Stauff, T. Hua, L. Zou, R. Hu, D. Shaver, and A. Novak

NRIC | INL Idaho National Laboratory

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

DISCLAIMER

REVISION LOG

Revision No.	Date	Affected Pages	Description
0	July 9, 2021	all	Initial submission to PICSNE

Page intentionally left blank

ABSTRACT

This report documents and organizes all advanced reactor example inputs (or models) that were developed by Argonne National Laboratory (ANL) staff, while coordinating and collaborating with colleagues from Idaho National Laboratory (INL), using NEAMS tools and submitted to the Virtual Test Bed in fiscal year 2021. The specific models of interest were targeted based on stakeholder feedback and to support the broader U.S. reactor developer industry. All models were generated based on public literature without any proprietary information to facilitate access. The models include:

- A Molten Salt Fast Reactor (MSFR) core model for the computational fluid dynamics (CFD) code Nek5000
- A Modular High Temperature Gas-cooled Reactor (MHTGR) core model for the systems thermal hydraulic code SAM
- A Pebble Bed Fluoride High Temperature Reactor (PB-FHR) core model for SAM
- A PB-FHR bypass flow reflector model for the CFD code NekRS coupled to the MOOSE framework
- A multiphysics model of a heat pipe microreactor that involved coupling the heat pipe code Sockeye to the fuel performance code BISON.

The models are described in detail with a focus on guiding a new user on how to use and run these models with the various codes.

ACKNOWLEDGMENTS

The authors are very grateful for the support and invitation to participate in this activity from colleagues at the Idaho National Laboratory. In particular, Abdalla Abou-Jaoude, Derek Gaston, Guillaume Giudicelli, and others have been key partners in the development of this work.

Page intentionally left blank

CONTENTS

FIGURES

TABLES

Page intentionally left blank

1 Introduction

This report documents and organizes all advanced reactor example inputs (or models) that were developed or improved by Argonne National Laboratory (ANL) staff, while coordinating and collaborating with colleagues from Idaho National Laboratory (INL), using NEAMS tools and submitted to the Virtual Test Bed (VTB) in fiscal year 2021. The specific models of interest were targeted based on stakeholder feedback and to support the broader U.S. reactor developer industry. All models were generated based on public literature without any proprietary information. The models include:

- A Molten Salt Fast Reactor (MSFR) core model for the computational fluid dynamics (CFD) code Nek5000
- A Modular High Temperature Gas-cooled Reactor (MHTGR) core model for the systems thermal hydraulic code SAM
- A Pebble Bed Fluoride High Temperature Reactor (PB-FHR) core model for SAM
- A PB-FHR bypass flow reflector model for the CFD code NekRS coupled to the MOOSE framework
- A multiphysics model of a heat pipe microreactor that involved coupling the heat pipe code Sockeye to the fuel performance code BISON.

The models are described in detail with a focus on guiding a new user on how to use and run these models with the various codes. The intended users of these models are stakeholders from industry and government agencies for both the DOE NRIC and DOE-NE NEAMS programs. It is presumed that such users would be new to the NEAMS suite of tools but have nuclear engineering backgrounds in at least one of the main fields of physics (reactor physics, thermal fluids, *etc.*). Many of the models were developed or refined based on pre-existing models that were developed from other government-funded activities. Such models were not easily shareable or documented sufficiently for new users to use them efficiently. Thus, the VTB served as an ideal mechanism to achieve an outward facing repository of curated models generated using NEAMS tools to promote the use of NEAMS tools via access to example models developed from public information.

2 Molten Salt Fast Reactor (Nek5000)

2.1 Introduction

Nek5000 is an open source CFD code based on the spectral element method (SEM) with a long history of application in reactor thermal-hydraulics research (Merzari et al., 2017). SEM combines the accuracy of spectral methods with the domain flexibility of the finite element method. In Nek5000 calculations, the domain is discretized into ***E*** curvilinear hexahedral elements, in which the solution is represented as a tensor product of N^{th}-order Lagrange polynomials based on the Gauss-Lobatto-Legendre (GLL) nodal points, leading to a total number of grid points $n=EN^3$. Nek5000 was designed from the outset of distributed-memory platforms. It is highly parallel and has been previously applied to a wide range of problems to gain unprecedented insight into the physics of turbulence in complex flows. The time-stepping scheme of Nek5000 is semi-implicit: the diffusion terms of the Navier-Stocks equations are treated implicitly by using a k^{th}-order backward difference formula (***BDFk***), while nonlinear terms are approximated by a k^{th}-order extrapolation (***EXTk***). Nek5000 was originally developed for simulating turbulent flows with very high fidelity, i.e. DNS and LES. More recently, the RANS capability has been implemented in the form of the regularized ***k-ω*** models and the ***k-τ*** model (Fang et al., 2021; Tomboulides et al., 2018). Both LES and DNS require proper resolution of turbulent length scales, which can be very expensive computationally. Considering the problem size involved in MSFR core flow simulations, the RANS model would be the most practical option. A 2-D axisymmetric MSFR core model is documented herein, which will hopefully serve as a useful example for anyone who is interested in using Nek5000 for MSFR related fluid problems. Under the support of DOE-NE's Nuclear Energy Advanced Modeling and Simulation (NEAMS) - Center of Excellence for Thermal Fluids Applications in Nuclear Energy, an effort had been pursued to support Molten Salt Fast Reactor (MSFR) modeling and simulation needs, as part of industry engagement efforts.

2.2 Governing equations and the RANS model

The incompressible formulation of Nek5000 is used in the current study which assumes a Newtonian fluid with constant properties. The corresponding continuity, momentum and energy equations are listed below:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i u_j) = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\nu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right] \tag{2}$$

$$\frac{\partial T}{\partial t} + u_j\frac{\partial T}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\frac{\nu}{Pr}\frac{\partial T}{\partial x_j}\right) + \frac{q}{\rho C_p} \tag{3}$$

where u is the velocity, p is the pressure, T is the temperature, ρ is the fluid density, ν stands for the kinematic viscosity, Pr is the Prandtl number, q is the heat generation in the fluid while C_p is the heat capacity. To model the highly turbulent flows, the RANS models have been recently implemented in Nek5000, including the ***k-ω*** model and its variation, the ***k-τ*** model. The equations for ***k*** and $\boldsymbol{\tau}$ are derived from the ***k-ω*** equations (Tomboulides et al., 2018) by using the definition $\boldsymbol{\tau = 1/\omega}$. The corresponding transport equations are as follows

$$\frac{\partial(\rho k)}{\partial t} + \nabla \cdot (\rho k \boldsymbol{v}) = \nabla \cdot \left[\left(\mu + \frac{\mu_t}{\sigma_k}\right)\nabla k\right] + P - \rho\beta^* \frac{k}{\tau} \quad (4)$$

$$\frac{\partial(\rho \tau)}{\partial t} + \nabla \cdot (\rho \tau \boldsymbol{v}) = \nabla \cdot \left[\left(\mu + \frac{\mu_t}{\sigma_\tau}\right)\nabla \tau\right] - \gamma \frac{\tau}{k} P + \rho\beta - 2\frac{\mu}{\tau}(\nabla\tau \cdot \nabla\tau) \quad (5)$$

In contrast to the original form of the ***k-ω*** model, in which the ***ω*** equation contains terms that become singular close to wall boundaries, all terms in the right-hand side of the ***k*** and ***τ*** equations reach a finite limit at walls and do not need to be treated asymptotically; that is, they do not require regularization for numerical implementation. The ***k-τ*** model in Nek5000 has been benchmarked extensively across a variety of canonical cases including channel flow and the backward facing step, and it has been also applied to fuel rod bundle geometry. Interested readers can refer to the recent publication (Fang et al., 2021) for more details regarding the ***k-τ*** application and validation in fuel rod bundles.

2.3 Geometric model and the meshing

A 2-D axisymmetric core model is first studied with the Nek5000 RANS solver. Based on the core dimensions listed in the description of the reactor, a 2-D geometric model is generated using the open source meshing software, GMSH. The entire model include the core region and part of the inlet and outlet channels. A pure hexahedral mesh is produced with 2,700 elements (as shown in Figure 1). In addition, to facilitate the axisymmetric solver in Nek5000, the core centerline is aligned along the x direction. The molten salt flow comes from the bottom channel (low x side) and exits from the other side.

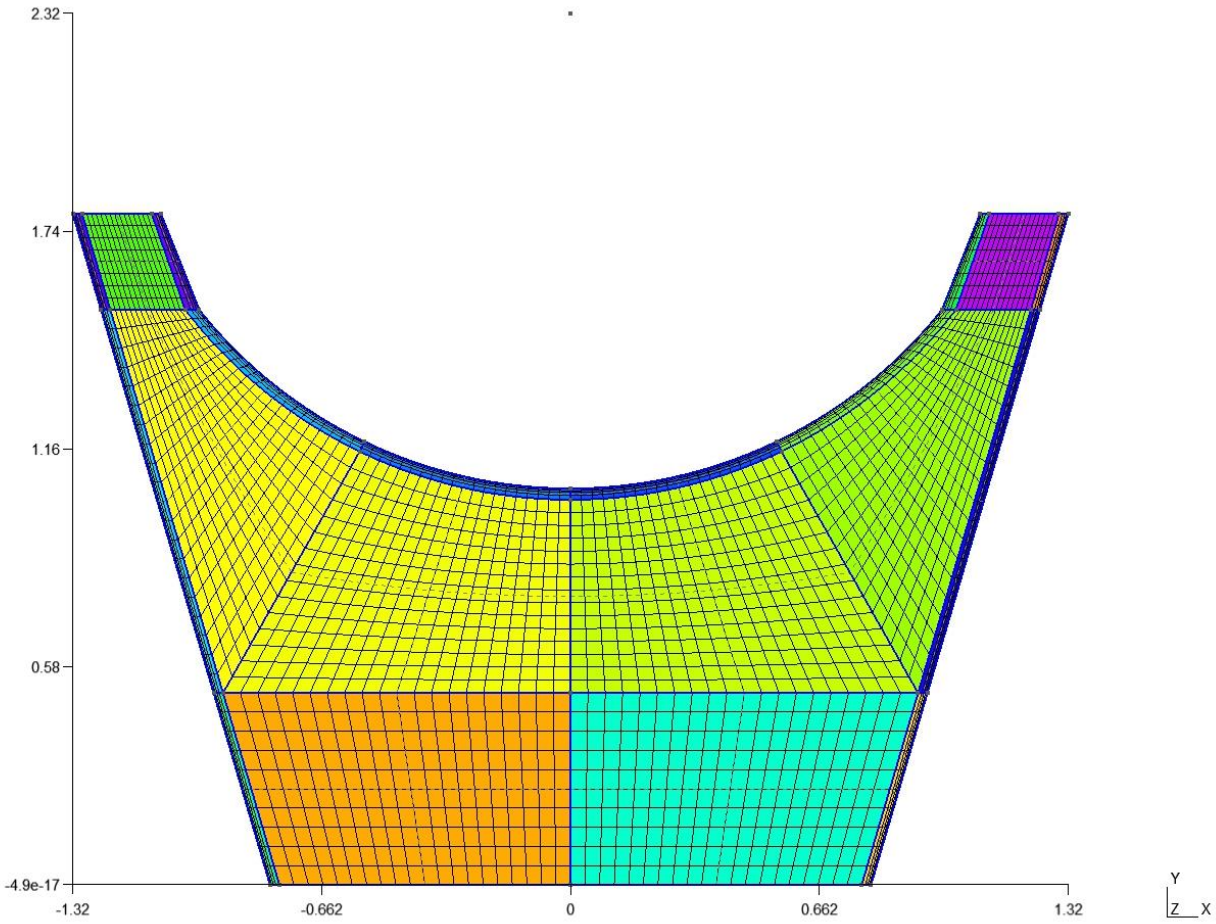

Figure 1: The computational grid of 2-D MSFR core.

2.4 Nek5000 case setups

In this section, we are going through the detailed setup of the Nek5000 case. The 2-D axisymmetric core case will be solved with the ***k-τ*** RANS model. To turn on the ***k-τ*** model, one can specify the model flag to 4 in the Nek user file. An example is given below.

```
      subroutine usrdat3()

c      implicit none

      include 'SIZE'
      include 'TOTAL'

      real wd(lx1,ly1,lz1,lelv)
      common /walldist/ wd

      logical ifcoefs

C     initialize the RANS model
      ifld_k = 3 !address of tke
      ifld_t = 4 !address of tau
      ifcoefs = .false.

C     Supported models:
c     id_m = 0 !regularized high-Re k-omega (no wall functions)
c     id_m = 1 !regularized low-Re k-omega
c     id_m = 2 !regularized high-Re k-omega SST (no wall functions)
c     id_m = 3 !regularized low-Re k-omega SST
      id_m = 4 !standard k-tau

C     Wall distance function:
c     id_w = 0 ! user specified
c     id_w = 1 ! cheap_dist (path to wall, may work better for periodic bounda-
ries)
      id_w = 2 ! distf (coordinate difference, provides smoother function)

      call rans_init(ifld_k,ifld_t,ifcoefs,coefs,id_w,wd,id_m)

      return
      end
```

As for the initial conditions, one can either utilize the trivial fields of zero for velocity and scalars, or load the existing solutions by providing the path to the restart file in the *par* file. In the case of user-specified initial conditions, one can update the *useric* function in the user file.

```
subroutine useric(ix,iy,iz,eg) ! set up initial conditions
implicit none
include 'SIZE'
include 'TOTAL'
include 'NEKUSE'

integer ix,iy,iz,e,eg

e = gllel(eg)

ux   = 0.0
uy   = 0.0
uz   = 0.0
temp = 0.0

return
end
```

For a mesh produced by GMSH, the Nek5000 relies on the physical group ids to specify the boundary conditions. There are 4 groups in the current case, which covers the boundary edges of domain inlet(1), outlet(2), wall(3), and the axisymmetric axis(4). The following code snippet showcases how the boundary condition tags are given to the model entities.

```
do iel=1,nelt
do ifc=1,2*ndim
   id_face = bc(5,ifc,iel,1)
   if (id_face.eq.1) then          ! inlet
       cbc(ifc,iel,1) = 'v  '
   elseif (id_face.eq.2) then      ! outlet
       cbc(ifc,iel,1) = 'O  '
   elseif (id_face.eq.3) then      ! wall
       cbc(ifc,iel,1) = 'W  '
   elseif (id_face.eq.4) then      ! centerline (axisymmetric)
       cbc(ifc,iel,1) = 'A  '
   endif
enddo
enddo

do i=2,ldimt1
do e=1,nelt
do f=1,ldim*2
  cbc(f,e,i)=cbc(f,e,1)
  if(cbc(f,e,1).eq.'W  ') cbc(f,e,i)='t  '
  if(cbc(f,e,1).eq.'v  ') cbc(f,e,i)='t  '
enddo
enddo
enddo
```

The specific values of boundary condition can be provided in the *userbc()* funciton. Since the settings of domain outlet, no-slip wall, and the axisymmetric axis can be handled in a default manner, the *userbc()* only contains the specifications of inlet boundary. A constant velocity is given at the inlet, and the direction of inlet velocity is parallel to the inlet channel. Note that a *turb_in* function is used to compute the proper BC values for k and τ variables on the inlet boundary.

```
      subroutine userbc(ix,iy,iz,iside,eg) ! set up boundary conditions
      implicit none
      include 'SIZE'
      include 'TOTAL'
      include 'NEKUSE'
c
      real wd
      common /walldist/ wd(lx1,ly1,lz1,lelv)

      integer ix,iy,iz,iside,e,eg
      real tke_tmp,tau_tmp
      e = gllel(eg)

      ux   = 0.38981288981
      uy   =-1.32536382536
      uz   = 0.0
      temp = 0.0

      call turb_in(wd(ix,iy,iz,e),tke_tmp,tau_tmp)
      if(ifield.eq.3) temp = tke_tmp
      if(ifield.eq.4) temp = tau_tmp

      return
      end
```

It is recommended to use non-dimensional parameters for Nek5000 simulations. As a result, a set of dimensionless input parameters are provided in the current case. One of key parameters is the Reynolds number (UL/ν), which indicates the level of turbulence intensity in the system. All the key input parameters are given in the *par* file, and a code snippet of the velocity solving is as follows

```
[VELOCITY]
density = 1.0
viscosity = -5.0E+04
residualTol = 1e-8
residualPROJ = yes
```

2.5 MSFR Nek5000 CFD results

The steady-state solutions of velocity and turbulent kinetic energy (TKE) field in the MSFR core from the 2-D axisymmetric RANS case are shown below.

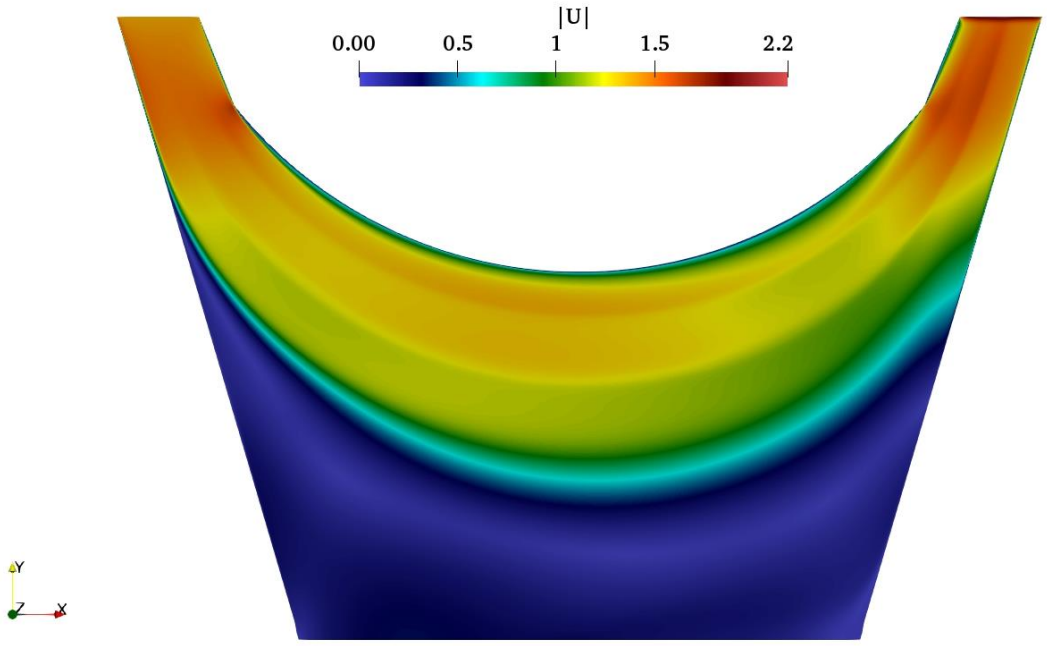

Figure 2: The steady-state velocity field from the 2-D MSFR RANS simulation.

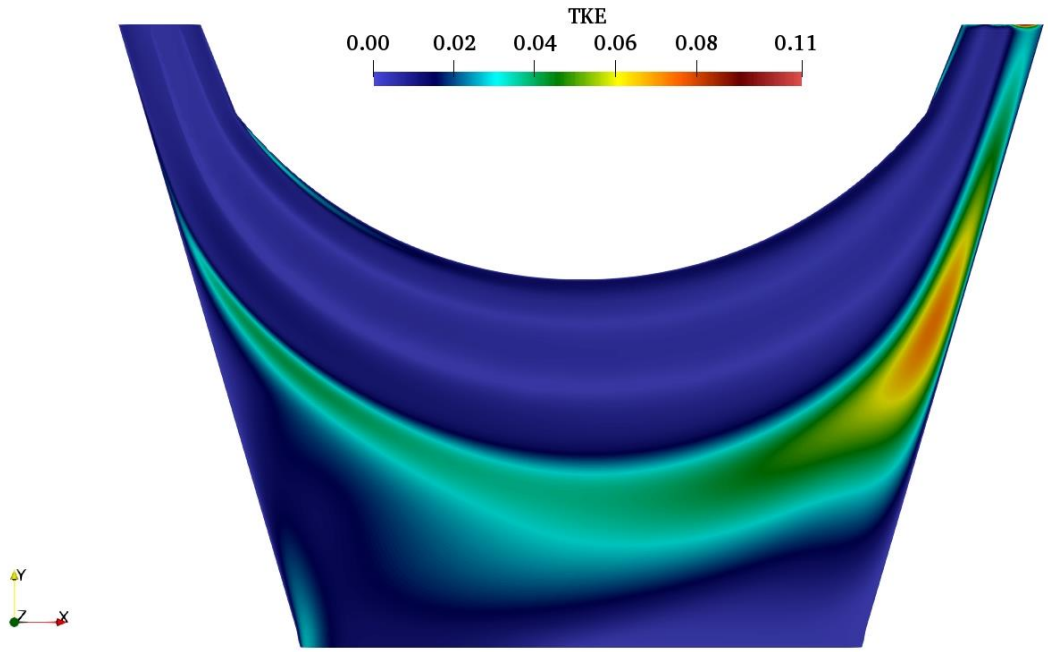

Figure 3: The steady-state TKE field from the 2-D MSFR RANS simulation.

It is noticed that the velocity magnitude is higher in the peripheral region compared to core center, which is related to the velocity boundary condition given at the inlet. As for the TKE field, the regions of high TKE value indicate strong local flow mixing. A velocity field with arrows is shown in Figure 4 to illustrate the velocity directions and magnitudes of specific locations inside the core.

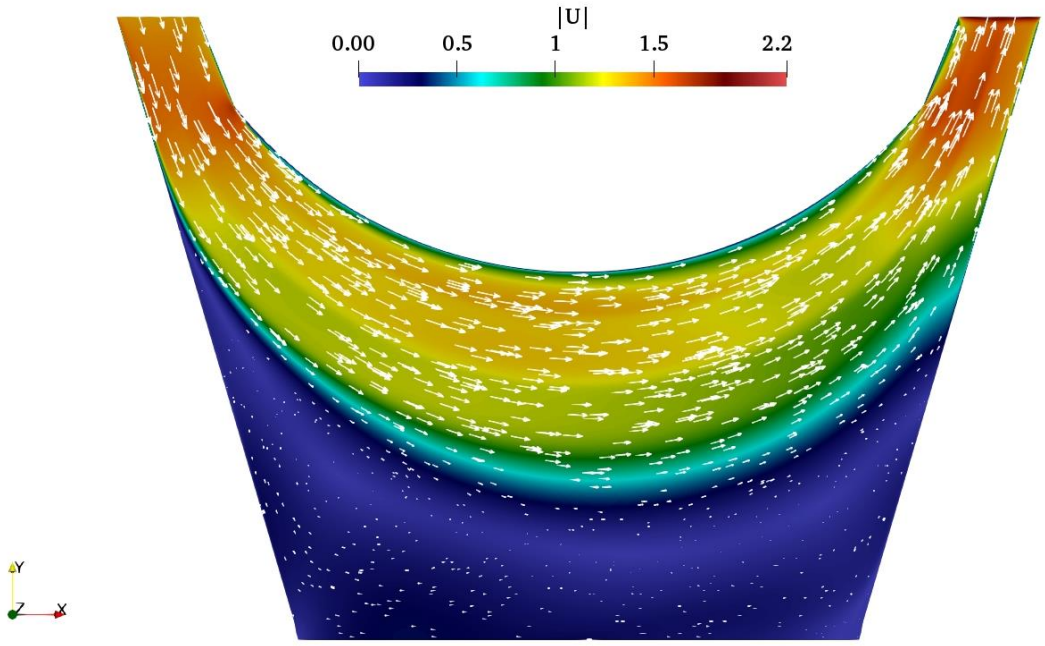

Figure 4: The steady-state velocity field with arrows indicating local velocity directions.

It has been noticed that the flow field solutions are sensitive to the inlet boundary conditions. The results shown above are the steady-state solutions with a constant inlet velocity, which is not necessarily the case in the actual MSFR. Additional studies are being conducted to reproduce the realistic inlet BC, which will help further improve the results.

3 Modular High Temperature Gas-cooled Reactor (SAM)

3.1 Problem Definition

The input file (MHTGR.i) is a model for the General Atomic's 600-MW_t Modular High Temperature Gas-Cooled Reactor [INL 2011]. The SAM model [Hu et al., 2021, Vegendla et al., 2019] was developed via NEAMS activities in previous years and uses the ring approach based on a specified coolant channel pitch of fuel assembly. In this approach, all components including fuel, reflectors, coolant channels, core barrel, reactor pressure vessel (RPV), and reactor cavity cooling system (RCCS) are modeled as concentric cylindrical rings. The active core consists of three fuel rings; inner, middle and outer ring. Each fuel ring is represented by 11 coolant channels and 22 heat structures. Thus the active core is simulated with 99 circular rings where 66 rings for homogenized fuel heat structure and 33 rings for gas coolant. Each coolant ring is sandwiched between two heat structure rings. The two surfaces of these two heat structure rings that form the walls of the coolant ring are thermally equilibrated via surface coupling to prevent unphysical temperature discontinuity. In addition, six more rings are included to represent the inner reflector, outer reflector, core barrel, RPV, RPV coolant channel, and RCCS riser wall. Note that the 600-MWt conceptual design is similar to a previous 350-MWt design developed in the 1980s [Turner et al., 1988, Neylan et al., 1988, GA 1988, NEA 2018].

The output files consist of: (1) a csv file that writes all user-specified variables at each time step; (2) a checkpoint folder that saves the snapshots of the simulation data including all meshes, solutions, and stateful object data. They are saved for restarting the run if needed; and (3) a ExodusII file that also has all mesh and solution data. Users can use Paraview to visualize the solution and analyze the data.

This write-up describes the content of the input file, the output files and how the model can be run using the SAM code.

3.2 Input Description

SAM uses a block-structured input syntax. Each block begins with square brackets which contain the type of input and ends with empty square brackets. Each block may contain sub-blocks. The blocks are described in the order as they appear in the input file. Before the first block entries, users can define variables and specify their values which are subsequently used in the input model. For example,

```
rad_R-1     =   0       # the radius of Ring 1
w_R-1       =   1.48        # the thickness of Ring 1
power_total     =       600e6       # the total power
```

In SAM, comments are entered after the # sign

3.2.1 Global Parameters

This block contains the parameters such as global initial pressure, velocity, and temperature conditions, the scaling factors for primary variable residuals, etc. For example, to specify global pressure of 1.e5 Pa, the users can input

```
global_init_P    =  1.e5
```

This block also contains a sub-block *PBModelParams* which specifies the modeling parameters associated with the primitive-variable based fluid model. New users should leave this sub-block unchanged.

3.2.2 EOS

This block specifies the Equation of State. The users can choose from built-in fluid library for common fluids like air, nitrogen, helium, sodium, molten salts, etc. The users can also input the properties of the fluid as constants or function of temperature. For example, the built-in eos for air can be input as

```
[./eos_air]
   type = AirEquationOfState
 [../]
```

Water is used as coolant at the RCCS, and its properties in SI unites are input as follow.

```
[./eos_water]
   type = PTConstantEOS
   p_0 = 70e5
   rho_0 = 905
   beta = 0
   cp = 4330
   h_0 = 705000
   T_0 = 439
   mu = 0.00016
   k = 0.68
[../]
```

3.2.3 Functions

Users can define functions for parameters used in the model. These include temporal, spatial, and temperature dependent functions. For example, users can input enthalpy as a function of temperature, power history as a function of time, or power profile as a function of position. The input below specifies graphite thermal conductivity as a function of temperature (in K)

```
[./kgraphite]                    #G-348 graphite therm. cond; x- Temperature [K], y-
Thermal condiuctivity [W/m-K]
  type = PiecewiseLinear
  x ='295.75   374.15472.45574.75674.75774.75874.75974.851074.45   1173.95
  1274.05'
  y ='133.02   128.54117.62106.0396.7   88.61  82.22  76.52  71.78  67.88  64.26'
   [../]
```

3.2.4 MaterialProperties

Material properties are input in this block. The values can be constants or temperature dependent as defined in the *Functions* block. For example, the properties of graphite are input as

```
[./graphite-mat]                 # Material name
   type = SolidMaterialProps
   k = kgraphite                   # Thermal conductivity
   Cp = cpgraphite                 # Specific heat
   rho = rhographite               # Density
```

```
[../]
```

The thermal conductivity is defined by the function *kgraphite* which appears under the *Functions* block.

3.2.5 ComponentInputParameters

This block is used to input common features for C*omponents* (section 2.6) so that these common features do not need to be repeated in the inputs for C*omponents* later on. For example, if pipes are used in various parts of the model and the pipes all have the same diameter, then the diameter can be specified in C*omponentsInputParameters* and it applies to all pipes used in the model.

3.2.6 Components

This block provides the specifications for all components that make up the primary and secondary loops. The components consist of: reactor, coolant channels, heat structures (fuel, reflectors, core barrel, RPV and RCCS), pipes, heat exchanger, blower and junctions for connecting components. In the reactor component, the reactor power is an input and this includes normal operating power and decay heat.

```
[./reactor]
    type = ReactorPower
    initial_power = 600e6
#    decay_heat = power_history
  [../]
```

The coolant channels are modeled as 1-D fluid flow components, and heat structures are modeled as 2-D components. Table 1 shows the names of the rings in the model starting from the center of the core toward the RCCS.

Table 1 Names of rings in the SAM model

Name	Description	Note
R1	Inner reflector	
R2_n-L	Inner fuel ring left structure	"n" ranges from 1 to 11
R2C-n	Inner fuel ring coolant channel	"n" ranges from 1 to 11
R2_n-R	Inner fuel ring right structure	"n" ranges from 1 to 11
R3_n-L	Middle fuel ring left structure	"n" ranges from 1 to 11
R3C-n	Middle fuel ring coolant channel	"n" ranges from 1 to 11
R3_n-R	Middle fuel ring right structure	"n" ranges from 1 to 11
R4_n-L	Outer fuel ring left structure	"n" ranges from 1 to 11
R4C-n	Outer fuel ring coolant channel	"n" ranges from 1 to 11
R4_n-R	Outer fuel ring right structure	"n" ranges from 1 to 11
R5	Outer reflector	
R6	Core barrel	
R6_C	Upcomer coolant channel	
R7	Reactor pressure vessel	
RCCS_AC	Reactor cavity cooling system	

In this configuration, each coolant channel communicates with its two adjacent heat structures through the variable *HT_surface_area_density_right* and *HT_surface_area_density_left* such as shown below

```
[./R4_9-L]
   type = PBCoupledHeatStructure
   input_parameters = R4_LHS_param
   name_comp_right= R4C-9
   HT_surface_area_density_right = ${aw_R4-9-L}
   width_of_hs = ${w_R4-9-L}
   radius_i = ${rad_R4-9-L}
 [../]

[./R4C-9]
   type = PBOneDFluidComponent
   position = '0 ${rad_R4C-9} 0'
   input_parameters = R4_channelParam
 [../]

 [./R4_9-R]
   type = PBCoupledHeatStructure
   input_parameters = R4_RHS_param
```

```
        name_comp_left= R4C-9
        HT_surface_area_density_left = ${aw_R4-9-R}
        width_of_hs = ${w_R4-9-R}
        radius_i = ${rad_R4-9-R}
    [../]
```

Adjacent heat structures are connected using *SurfaceCoupling* to assure temperature continuity

```
    [./Gap_R4_9]
        type = SurfaceCoupling
        use_displaced_mesh = true
        coupling_type = GapHeatTransfer
        surface1_name = 'R4_9-R:outer_wall'
        surface2_name = 'R4_10-L:inner_wall'
        width = 1e-20
        radius_1 = ${rad_R4-9-L}
        length = ${axial_length_1}
        eos = eos
        h_gap =  ${hGap_cond}
    [../]
```

In SAM, 1-D components are connected using *PBSingleJunction*. The following input is used to connect the outlet of component *R4C-1* to the inlet of component *R4CUP-1*

```
    [./Branch_R4CUP-1]
  type = PBSingleJunction
 inputs = 'R4C-1(out)'
 outputs = 'R4CUP-1(in)'
  eos = eos
    [../]
```

Heat exchangers are modeled using *PBHeatExchanger* including the fluid flow in the primary and secondary sides, convective heat transfer, and heat conduction in tube wall. Pumps are modeled using *PBPump*.

3.2.7 Postprocessors

This block is used to specify the output variables written to a csv file that can be further processed in Excel. For example, to output the temperature and velocity in R3C-11:

```
    [./R3C11_T_in]
        type = ComponentBoundaryVariableValue
        variable = temperature
        input = R3C-11(in)
      [../]

      [./R3C11_V_in]
        type = ComponentBoundaryVariableValue
        variable = velocity
        input = R3C-11(in)
      [../]
```

To output the maximum temperature in R6:

```
[./R6_max]
  type = NodalMaxValue
  block = 'R6:hs0'
  variable = T_solid
[../]
```

3.2.8 Preconditioning

This block describes the preconditioner used by the solver. New users can leave this block unchanged.

3.2.9 Executioner

This block describes the calculation process flow. The users can specify the start time, end time, time step size for the simulation. Other inputs in this block include PETSc solver options, convergence tolerance, quadrature for elements, etc which can be left unchanged.

3.3 Results

There are three types of output files:

- MHTGR_csv.csv: this is a csv file that writes the user-specified scalar and vector variables to a comma-separated-values file. The data can be imported to Excel for further processing.
- MHTGR_checkpoint_cp: this is a sub-folder that save snapshots of the simulation data including all meshes, solutions. Users can restart the run from where it ended using the file in the checkpoint folder.
- MHTGR_out.displaced.e: this is a EXodusII file that has all mesh and solution data. Users can use Paraview to open this .e file to visualize, plot, and analyze the data.

Figure 5 shows the Paraview output for the primary loop. Figure 6 shows the coolant temperature profile during normal operation

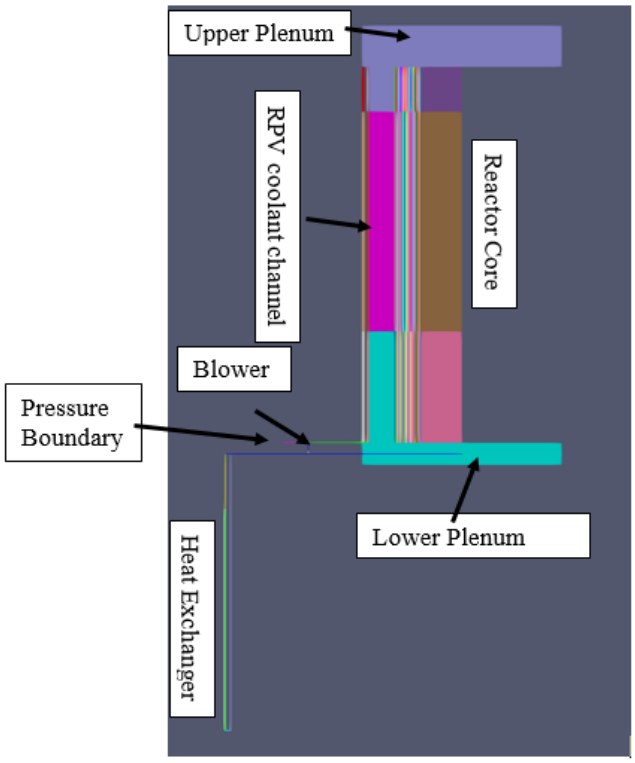

Figure 5: SAM model for the MHTGR primary loop

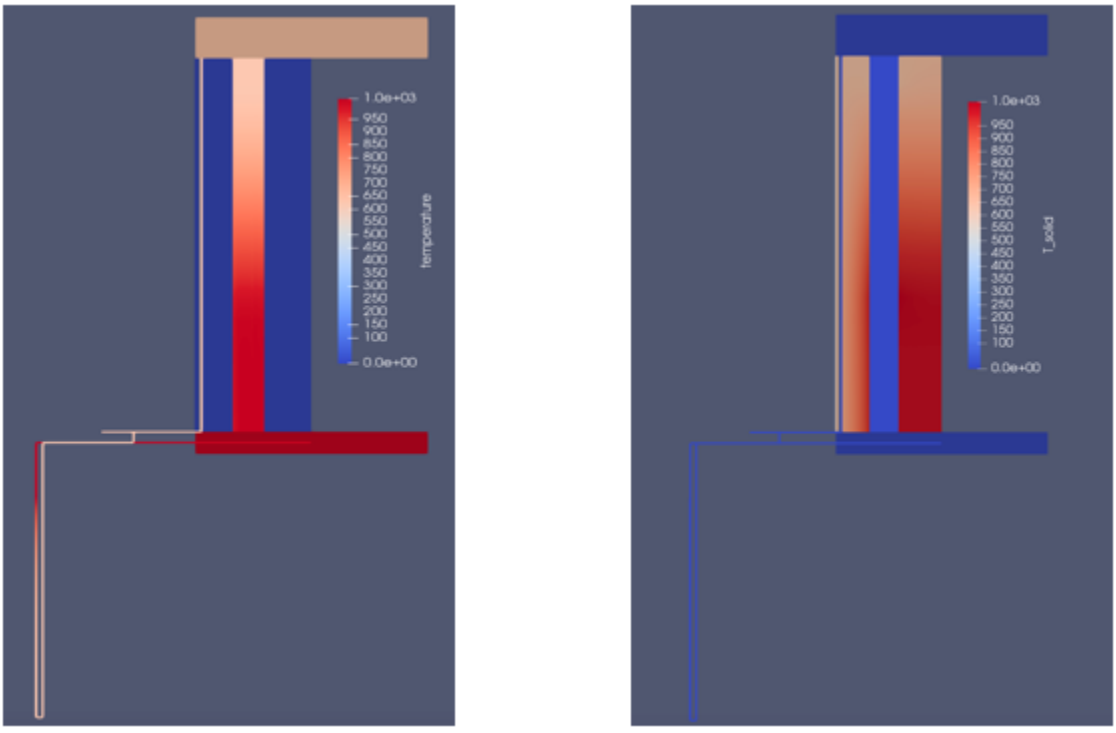

Figure 6: Temperature profiles of helium coolant (left) and heat structure (right) during normal operation

3.4 Running the Input File

SAM can be run in Linux, Unix, and MacOS. Due to its dependence on MOOSE, SAM is not compatible with Windows machine. SAM can be run from the shell prompt as shown below

sam-opt -i mhtgr.i

4 Pebble Bed Fluoride High temperature Reactor (SAM)

4.1 Introduction

The SAM [Hu 2021] input files (PBFHR-SS.i for steady state and PBFHT-TR.i for loss of flow transient) were built to model the "Mark 1" pebble-bed fluoride-salt-cooled high temperature reactor (FHR) which was developed by the University of California Berkeley [UCB 2014]. A schematic of the reactor is shown in Figure 7. FHRs exhibit different thermal hydraulic phenomenon compared to conventional advanced nuclear reactor concepts, such as decay heat removal through natural circulation using Direct Reactor Auxiliary Cooling System (DRACS) loops. Therefore, there is a need for modeling and simulation tools to accurately predict the thermal response of FHRs for a range of postulated transient events. A SAM mode of the UCB Mk-1 design was developed in a study [Ahmed 2017], which focuses on modeling the FHR core under normal operation and a loss of forced flow with SCRAM event.

The output files consist of: (1) a csv file that writes all user-specified variables at each time step; (2) a checkpoint folder that saves the snapshots of the simulation data including all meshes, solutions, and stateful object data. They are saved for restarting the run if needed; and (3) a ExodusII file that also has all mesh and solution data. Users can use Paraview to visualize the solution and analyze the data.

This write-up describes the content of the input file, the output files and how the model can be run using the SAM code.

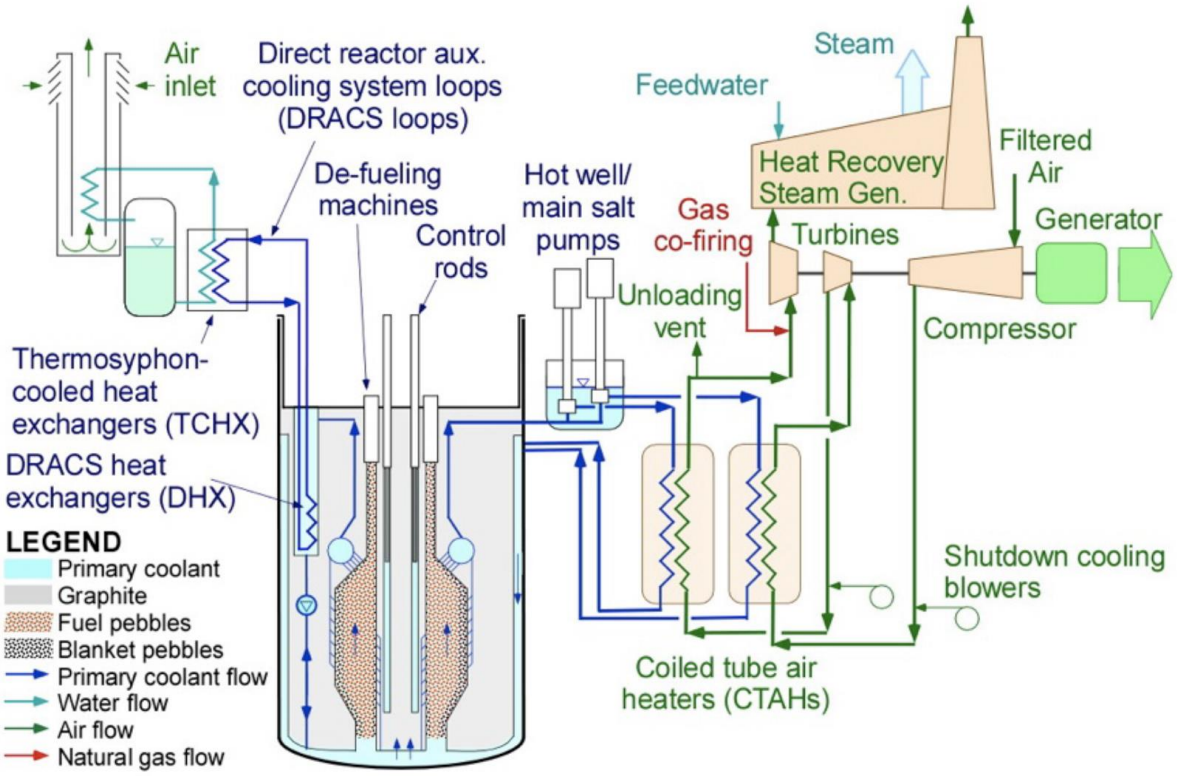

Figure 7: Mk-1 PB-FHR schematic [UCB 2014] (used with permission).

4.2 Input File Description

SAM uses a block-structured input syntax. Each block begins with square brackets which contain the type of input and ends with empty square brackets. Each block may contain sub-blocks. The blocks are described in the order as they appear in the input file.

4.2.1 Global Parameters

This block contains the parameters such as global initial pressure, velocity, and temperature conditions, the scaling factors for primary variable residuals, etc. For example, to specify global pressure of 1.e5 Pa, the use can input

```
global_init_P    =  1.e5
```

This block also contains a sub-block *PBModelParams* which specifies the modeling parameters associated with the primitive-variable based fluid model. New users should leave this sub-block unchanged.

4.2.2 EOS

This block specifies the Equation(s) of State. The users can choose from built-in fluid library for common fluids like air, nitrogen, helium, sodium, molten salts, etc. The users can also input the properties of the fluid as constants or function of temperature. For example, the built-in eos for FLIBE can be input as

```
[./eos_air]
   type = SaltEquationOfState
 [../]
```

4.2.3 MaterialProperties

Material properties are input in this block. The values can be constants or temperature dependent as defined in the *Functions* block. For example, the properties of stainless steel are input as

```
[./ss-mat]                          # Material name (stainless steel)
   type = SolidMaterialProps
   k = 40                           # Thermal conductivity in W/m-K
   Cp = 583.33                       # Specific heat
   rho = 6.e3                        # Density
 [../]
```

Note that all units are in SI by default.

4.2.4 Functions

Users can define functions for parameters used in the model. These include temporal, spatial, and temperature dependent functions. For example, users can input enthalpy as a function of temperature, power history as a function of time, or power profile as a function of position. The input below specifies decay heat as a function of time

```
[./shutdownPower]
   type = PiecewiseLinear
```

```
    x = '0  100    101    102    104    108    116    124    132    140    148    160
    220    340    580    1060  1540  2020  2500  2980  3460  3700  12300'
      y = '1.0000   1.0000   0.0530   0.0508   0.0479   0.0441   0.0403   0.0378
0.0361   0.0347   0.0336   0.0322   0.0279   0.0242   0.0210   0.0179   0.0161
0.0148   0.0138   0.0130   0.0124   0.0121   0.0081'
     [../]
```

4.2.5 Components

This is the main block in the input file. It provides the specifications for all components that make up the DRACS and primary loops. The primary loop, shown schematically in Figure 8, consists of three branches: core, DRACS heat exchanger (DHX), and coiled tube air heater (CTAH). The components and their nodalization IDs in each branch are listed in Table 1. The nodalization IDs are also specified in the input file. The main components in the primary loop are a reactor, the core channel, a heat exchanger, pump, plena, tank, and piping. The reactor power and decay heat profile are user-input

```
[./reactor]
   type = ReactorPower
   initial_power = 2.36e8
   decay_heat = shutdownPower
 [../]
```

The core channel is modeled using *PBCoreChannel* in which the cylindrical fuel elements are modeled as three heat structures: fuel sandwiched between an inner and outer material (h451). The thickness of each structure is specified by the users and because power is generated only in the fuel, the power fraction in the three structures are input as '0 1 0'. The power distribution in the axial direction is defined by the users in the *Paxial* function. Explanation for other input variables can be found in the user manual.

Pipings are modeled as one-dimensional fluid flow component, *PBOneDFluidComponent*. Their locations are specified with variables *position* and *orientation*. Flow area, hydraulic diameter, and pipe length are the main variables that define the element. An example of a piping is as follows

```
[./pipe040]                    #Hot salt extraction pipe (4)
   type = PBOneDFluidComponent
   eos = eos
   position = '0 3.96445 -0.76'
   orientation = '0 0 1'
   roughness = 0.000015
   A = 0.2512732
   Dh = 0.5656244
   length = 3.77
   n_elems = 11
   initial_V = 2.050
   initial_T = 970
   initial_P = 1.3e5
[../]
```

Components are connected using *PBSingleJunction*, or *PBBranch*. For example

```
[./Branch611]                    #In to TCHX manifold
   type = PBSingleJunction
   inputs = 'pipe210(out)'
   outputs = 'pipe220(in)'
   eos = eos
 [../]
```

The DHX is modeled using the *PBHeatExchanger* component which models a shell-and-tube heat exchanger including the fluid flow in the primary and secondary sides, convective heat transfer, and heat conduction in tube wall. Either co-current or counter-current configuration can be modeled. Care should be taken when specifying the heat transfer surface area density (*HT_surface_area_density*). Users are advised to consult the SAM manual for further explanation. The heat transfer coefficients for both the shell side and tube side are calculated internally. However, users can override them using variables *Hw* and *Hw_secondary* (commented out in the input file)

```
[./DHX]                          # DHX shell side (17), DHX tube side (19), DHX tubes structure
   type = PBHeatExchanger
   eos = eos
   eos_secondary = eos
   hs_type = cylinder

   radius_i = 0.00545
   position = '0 0.5 0'
   orientation = '0 0 1'
   A = 0.2224163
   Dh = 0.01085449
   A_secondary = 0.1836403
   Dh_secondary = 0.0109
   roughness = 0.000015
   roughness_secondary = 0.000015
   length = 2.5
   n_elems = 7 #14

   initial_V = 0.122 #0.11969487
   initial_V_secondary = 0.029349731
   initial_T = 870
   initial_T_secondary = 830
   initial_P = 1.9e5
   initial_P_secondary = 2.0e5

   HT_surface_area_density =  441.287971
   HT_surface_area_density_secondary = 458.715596
   #Hw = 526.266
   #Hw_secondary = 440
   HTC_geometry_type = Pipe
   HTC_geometry_type_secondary = Pipe
   PoD = 1.1
```

```
    Twall_init = 900
    wall_thickness = 0.0009
    dim_wall = 2
    material_wall = ss-mat
    n_wall_elems = 4
  [../]
```

The salt pump is modeled using the *PBPump* component. Users specify a constant pump head or pump head function (*Phead*) which is time dependent. Large reverse pump loss coefficients are input to prevent reverse flow.

```
[./Pump]
    type = PBPump
    inputs = 'pipe060(out)'
    outputs = 'pipe070(in)'
    eos = eos
    K = '0 0'
    K_reverse = '2000000 2000000'
    Area = 0.3041
    Head_fn = Phead
    initial_V = 1.783
    initial_T = 970
    initial_P = 2.7e5
  [../]
```

The DRACS loop, shown schematically in Figure 9, consists of the tube side of the DHX heat exchanger, manifold and piping. The nodalization of the components are included in Table 2.

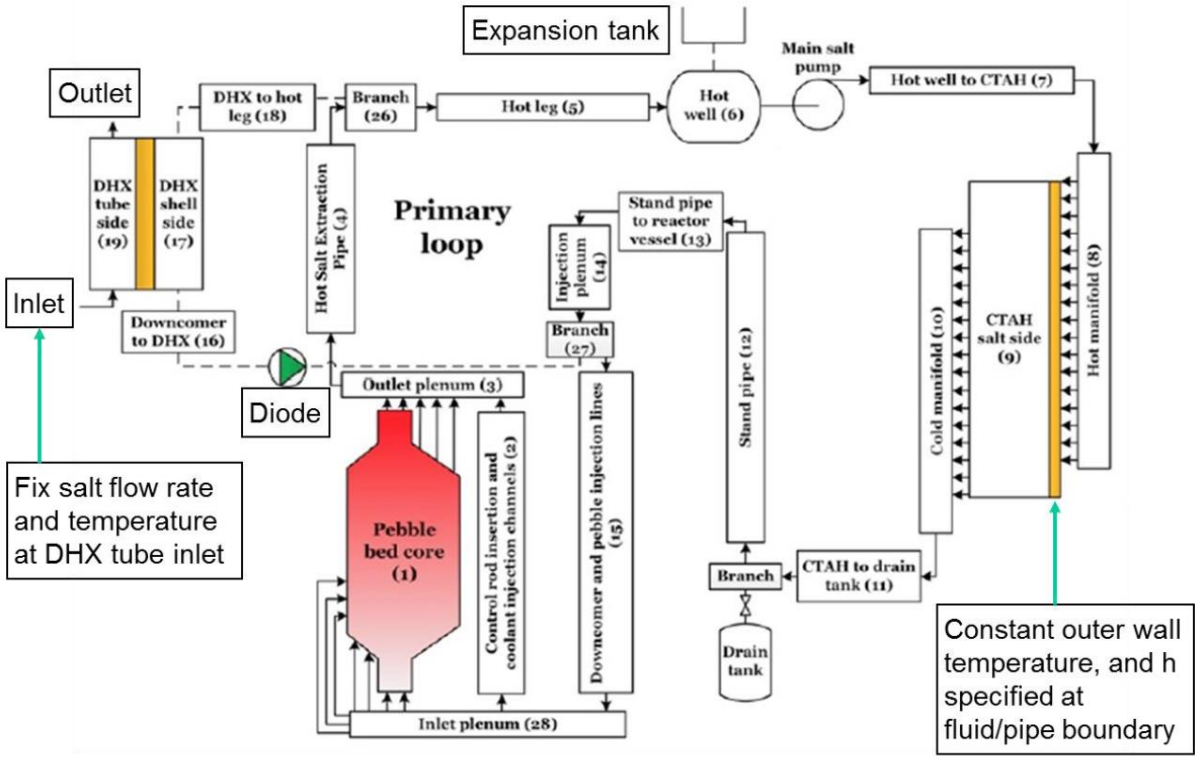

Figure 8: Nodalization of the primary loop model in SAM model (used with permission [Zweibaum 2015])

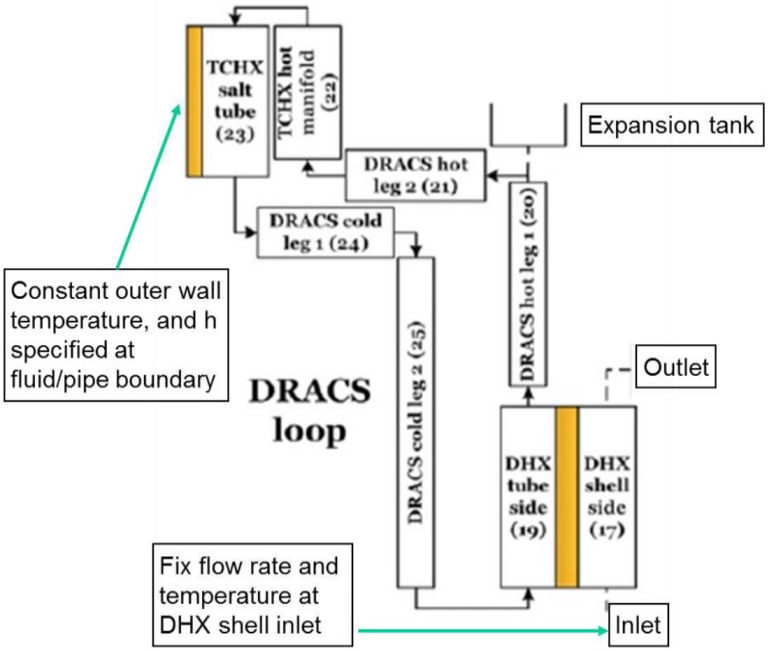

Figure 9: Nodalization of the DRACS loop model in SAM model (used with permission [Zweibaum 2015])

Table 2 Components in Mk-1 PB-FHR Primary and DRACS loops

Core branch				
Pebble bed	1	4.58	0.03	1.33
Core bypass	2	4.58	0.01	0.133
Hot salt collection ring	3	3.96	0.567	0.252
Hot salt extraction pipe	4	3.77	0.566	0.251
Branch	26	0.5	0.58	0.264
CTAH branch				
Reactor vessel to hot salt well	5	3.73	0.58	0.264
Hot salt well	6	2	1.45	3.31
Hot salt well to CTAH	7	3.23	0.44	0.304
CTAH hot manifold	8	3.418	0.28	0.493
CTAH salt side	9	18.47	0.00457	0.449
CTAH cold manifold	10	3.418	0.175	0.192
CTAH to drain tank	11	3.48	0.438	0.302
Standpipe	12	6.51	0.438	0.302
Standpipe to reactor vessel	13	6.603	0.438	0.302
Injection plenum	14	3.04	0.438	0.302
Downcomer	15	4.76	0.056	0.304
Branch	27	0.5	0.056	0.304
Inlet plenum	28	0.2	0.03	1.33
DHX branch				
Downcomer to DHX	16	0.58	0.15	0.0353
DHX shell side	17	2.5	0.0109	0.222
DHX to hot leg	18	3.008	0.15	0.0353
DRACS loop				
DHX tube side	19	2.5	0.0109	0.184
DRACS hot leg 1	20	3.45	0.15	0.0353
DRACS hot leg 2	21	3.67	0.15	0.0353
TCHX manifold	22	2.6	0.15	0.0353
TCHX salt tube	23	6	0.0109	0.175
DRACS cold leg 1	24	4.43	0.15	0.0353
DRACS cold leg 2	25	5.95	0.15	0.0353

4.2.6 Postprocessors

This block is used to specify the output variables written to a csv file that can be further processed in Excel. For example, to output the exit temperature on the secondary side of the DHX:

```
[./DHXTubeTop]
    type = ComponentBoundaryVariableValue
    input = 'DHX:secondary_pipe(out)'
    variable = 'temperature'
  [../]
```

To output the velocity and density of the flow exiting the core:

```
  [./Corev]
    type = ComponentBoundaryVariableValue
    input = 'pipe010(in)'
    variable = 'velocity'
  [../]
  [./Corerho]
    type = ComponentBoundaryVariableValue
    input = 'pipe010(in)'
    variable = 'rho'
  [../]
```

4.2.7 Preconditioning

This block describes the preconditioner used by the solver. New users can leave this block unchanged.

4.2.8 Executioner

This block describes the calculation process flow. The users can specify the start time, end time, time step size for the simulation. Other inputs in this block include PETSc solver options, convergence tolerance, quadrature for elements, etc which can be left unchanged

4.2.9 Restart

A new run can be restarted from a previous run. For example, input file PBFHT-TR.i simulates a transient that starts from the steady state results after running PBFHT-SS.i.

```
[Problem]
  restart_file_base = 'pbfhr-ss_out_cp/0402'
[]
```

4.3 Output Files Description and Results

There are three types of output files:

- PBFHR-SS.csv: this is a csv file that writes the user-specified scalar and vector variables to a comma-separated-values file. The data can be imported to Excel for further processing.

- PBFHR-SS_checkpoint_cp: this is a sub-folder that save snapshots of the simulation data including all meshes, solutions. Users can restart the run from where it ended using the file in the checkpoint folder.
- PBFHR-SS_out.displaced.e: this is a EXodusII file that has all mesh and solution data. Users can use Paraview to open this .e file to visualize, plot, and analyze the data.

Figure 10 shows the Paraview output for the primary loop temperature profile during normal operation and the salt flow pattern is indicated by green arrows. A companion plot of elevation vs. temperature traversing a closed loop of part of the system is also shown. In this regime, the DHX operates as a co-current heat exchanger. The area enclosed in the elevation-temperature plot shows that significant buoyancy force exists to drive natural circulation, since density is a linear function of temperature. This is important after transient initiation, when rapid flow reversal and establishment of natural circulation cooling is essential to minimize the maximum temperatures achieved by the system.

In this simulation, a protected loss of forced flow occurs at t = 100 s. The power drops immediately to about 5% of nominal power and decay heat is the sole heat source in the core. Figure 11 shows the temperature profile and flow pattern at t=1400 s. Natural circulation is established in the primary-to-DHX loop as indicated by the area enclosed in the elevation-temperature plot. The flow driven upward by the core through pipe 2 then enters path 3-4-5, so that DHX behaves as a countercurrent heat exchanger.

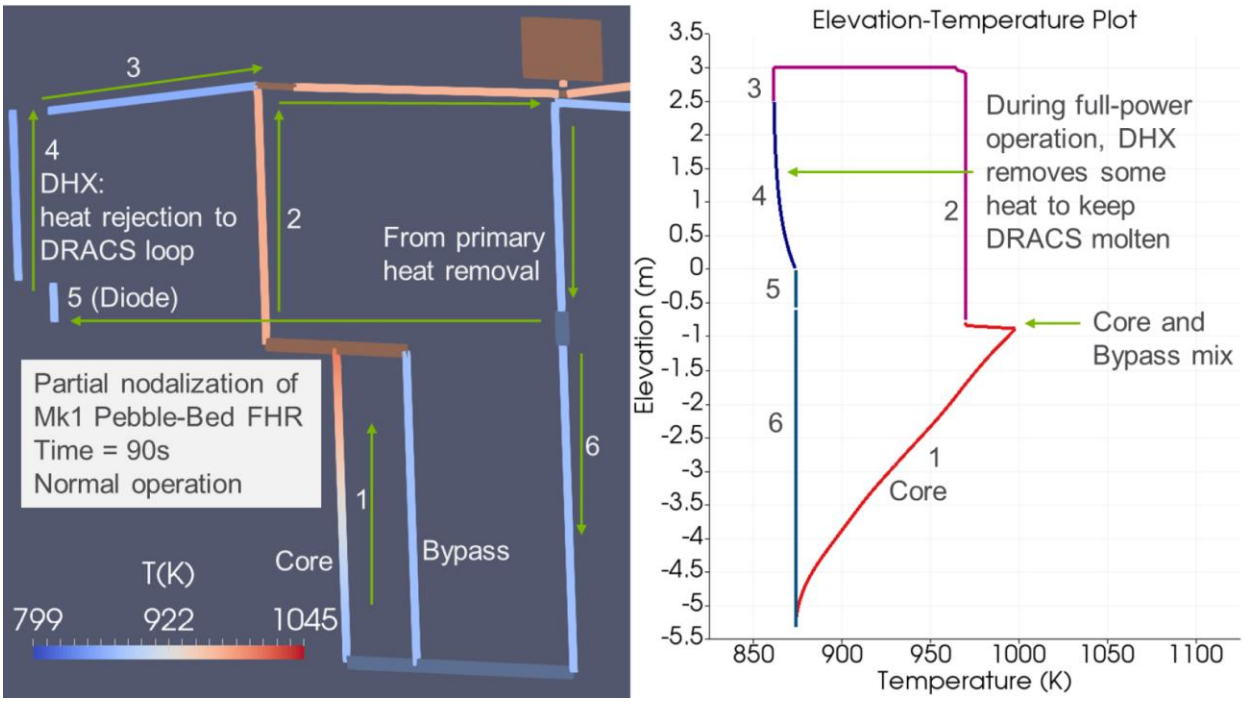

Figure 10: Coolant temperature during steady state (at t=90s).

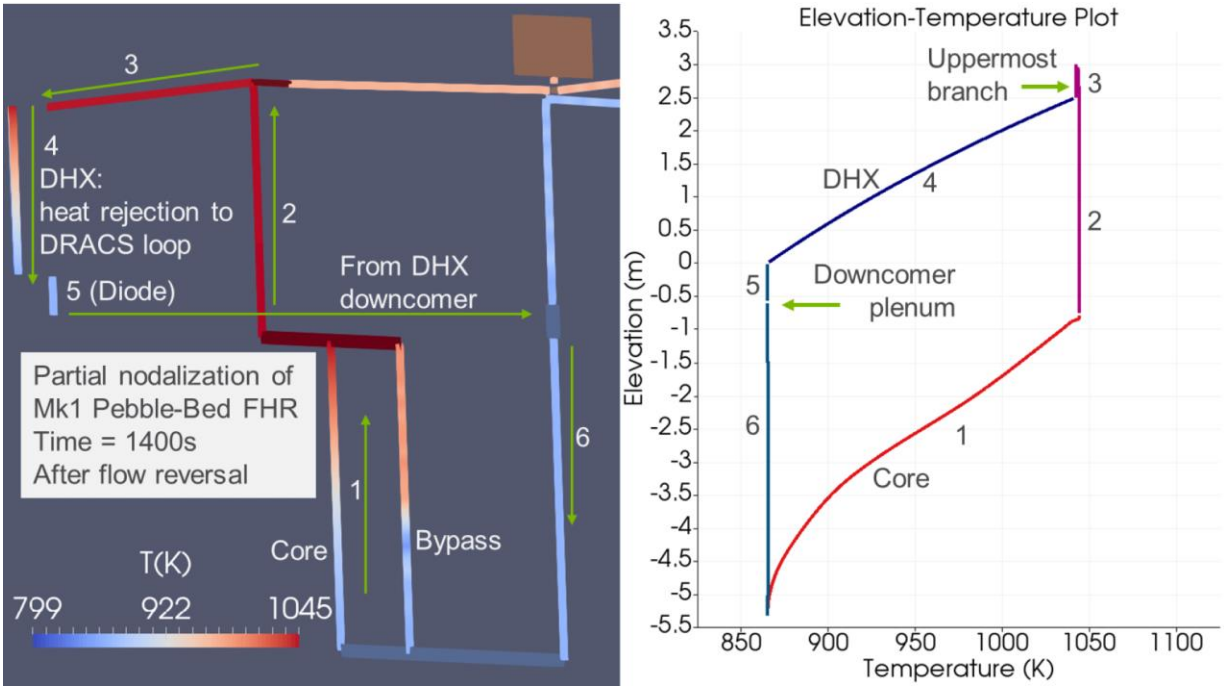

Figure 11: Coolant temperature during loss of forced flow transient (at t=1400s).

4.4 Running the Input File

SAM can be run in Linux, Unix, and MacOS. Due to its dependence on MOOSE, SAM is not compatible with Windows machine. SAM can be run from the shell prompt as shown below

sam-opt -I pbfhr-ss.i

5 PB-FHR - Bypass Flow Reflector Model (NekRS + MOOSE)

5.1 Introduction

The pebble region in the Pebble Bed Fluoride-Salt-Cooled High-Temperature Reactor (PB-FHR) is enclosed by an outer graphite reflector that constrains the pebble geometry while also serving as a reflector for neutrons and a shield for the reactor barrel and core externals. In order to keep the graphite reflector within allowable design temperatures, the reflector contains several bypass flow paths so that a small percentage of the coolant flow, usually on the order of 5 to 10% of the total flow, can remove heat from the reflector and maintain the graphite within allowable temperature ranges. This bypass flow, so-named because coolant is diverted from the pebble region, is important to quantify during the reactor design process so that accurate estimates of core cooling and reflector temperatures can be obtained.

This page shows an example of using NekRS, a GPU based CFD code developed by Argonne National Laboratory, coupled to MOOSE to explore the conjugate heat transfer in a small region of the PB-FHR reflector. With input from a full-core coupled Pronghorn-Griffin model, this example can be used to evaluate hot spots in the reflector and ensure that temperatures remain within allowable limits. The meshes for this example are created with programmable scripts; by running these mesh scripts for a range of bypass gap widths and repeating the flow simulations for a range of Reynolds numbers, this model can be used to provide friction factor and Nusselt number correlations as inputs to a full-core Pronghorn model.

The objectives of the present analysis are to:

- Show an example of how non-MOOSE codes can be coupled to MOOSE, and some capabilities in this space for thermal-hydraulics analysis. You should come away with an understanding of what a "MOOSE-wrapped app" looks like and the extent to which MOOSE-wrapped applications behave like other natively-developed MOOSE applications.
- Predict the pressure, velocity, and temperature distribution in the PB-FHR outer reflector.
- Describe how high-resolution tools such as NekRS can be used to generate closure terms (such as friction factor models) for the coarse-mesh tools in the MOOSE framework, such as Pronghorn.

NekRS uses the OCCA API, which allows NekRS to run on both CPU and GPU systems. However, in this example, NekRS is used with resolved boundary layers, requiring a fine mesh. This analysis therefore requires HPC resources. If you do not have access to HPC resources, you will still find the analysis procedure useful in understanding the MOOSE-wrapped app concept and opportunities for high-resolution thermal-hydraulics within the MOOSE framework.

5.2 Cardinal and MOOSE-Wrapped Apps

The analysis shown here is performed with Cardinal, a MOOSE application that "wraps" NekRS, a CFD code, and OpenMC, a Monte Carlo particle transport code. The Cardinal project is funded by the DOE-NE NEAMS program, while NekRS is primarily funded through the DOE Exascale Computing Project, with additional contributions from NEAMS. As this example focuses on heat transfer modeling, there will be no further discussion of the OpenMC wrapping in Cardinal. "Wrapping" means that, for all intents and purposes, NekRS simulations can be run within the

MOOSE framework and interacted with as if the physics and numerical solution performed by NekRS were a native MOOSE application. Cardinal contains source code that facilitates data transfers in and out of NekRS and runs NekRS within a MOOSE-controlled simulation. At a high level, Cardinal's wrapping of NekRS consists of:

- Constructing a "mirror" of the NekRS mesh through which data transfers occur with MOOSE. For conjugate heat transfer applications such as those shown here, a MooseMesh is created by copying the NekRS surface mesh into a format that all native MOOSE applications can understand.
- Adding MooseVariables to represent the NekRS solution. In other words, if NekRS stores the temperature internally as an std::vector<double>, with each entry corresponding to a NekRS node, then a MooseVariable is created that represents the same data, but that can be accessed in relation to the MooseMesh mirror.
- Writing multiphysics feedback fields in/out of NekRS's internal solution and boundary condition arrays. So, if NekRS represents a heat flux boundary condition internally as an std::vector<double>, this involves reading from a MooseVariable representing heat flux (which can be transferred with any of MOOSE's transfers to the NekRS wrapping) and writing into NekRS's internal vectors.

Accomplishing the above three tasks requires an intimate knowledge of how NekRS stores its solution fields and mesh. But once the wrapping is constructed, NekRS can then communicate with any other MOOSE application via the MultiApp and Transfer systems in MOOSE, enabling complex multiscale thermal-hydraulic analysis and multiphysics feedback. The same wrapping can be used for conjugate heat transfer analysis with any MOOSE application that can compute a heat flux; that is, because a MOOSE-wrapped version of NekRS interacts with the MOOSE framework in a similar manner as natively-developed MOOSE applications, the agnostic formulations of the MultiApps and Transfers can be used to equally extract heat flux from Pronghorn, BISON, the MOOSE heat conduction module, and so on.

Cardinal allows NekRS and OpenMC to couple seamlessly with the MOOSE framework, enabling in-memory coupling, distributed parallel meshes for very large-scale applications, and straightforward multiphysics problem setup. Cardinal has capabilities for conjugate heat transfer coupling of NekRS and MOOSE, concurrent conjugate heat transfer and volumetric heat source coupling of NekRS and MOOSE, and volumetric density, temperature, and heat source coupling of OpenMC to MOOSE. Together, the OpenMC and NekRS wrappings augment MOOSE by expanding the framework to include high-resolution spectral element CFD and Monte Carlo particle transport. For conjugate heat transfer applications in particular, the libMesh-based interpolations of fields between meshes enables fluid-solid heat transfer simulations on meshes that are not necessarily continuous on phase interfaces, allowing mesh resolution to be specified based on the underlying physics, rather than rigid continuity restrictions in single-application heat transfer codes. The remainder of this page will describe the conjugate heat transfer capabilities of Cardinal.

5.3 Geometry and Computational Model

This section describes the PB-FHR reflector geometry and the simplifications made in constructing a computational model of this system. This computational model is a simplified version of the actual reflector geometry and reactor state. Due to limitations in NekRS's boundary conditions, some geometry approximations will be made. This model is also run independently of Pronghorn and Griffin, so several assumptions are made regarding fluid boundary conditions at the inlet to the

reflector and along the reflector-pebble bed interface, even though in reality the flow and heat transfer in the reflector is tightly coupled to the flow and heat transfer in the bed and the neutron transport over the entire reactor.

It is important to be aware of these simplifications when assessing the physical predictions of this model. Nevertheless, these simplifications do not detract from the ability to use this model to learn about high-resolution conjugate heat transfer capabilities in the MOOSE framework as long as the simplifications are acknowledged. The details of this model can be refined for follow-on studies as needed. A top-down view of the PB-FHR reactor vessel is shown below. The center region is the pebble bed core, which has an outer radius of 2.6 m. Surrounding the pebble region are two rings of graphite reflector blocks, staggered with respect to one another in a brick-like fashion. Each ring contains 24 blocks. The inner ring blocks have a width (i.e. radial thickness) of 0.3 m, while the outer ring blocks have a width of 0.4 m. The gaps between the blocks are 0.002 m wide. In black is shown the core barrel, which is 0.022 m thick. The gap between the outer ring of blocks and the barrel is ten times larger than the gap between the inner and outer ring of blocks, or 0.02 m.

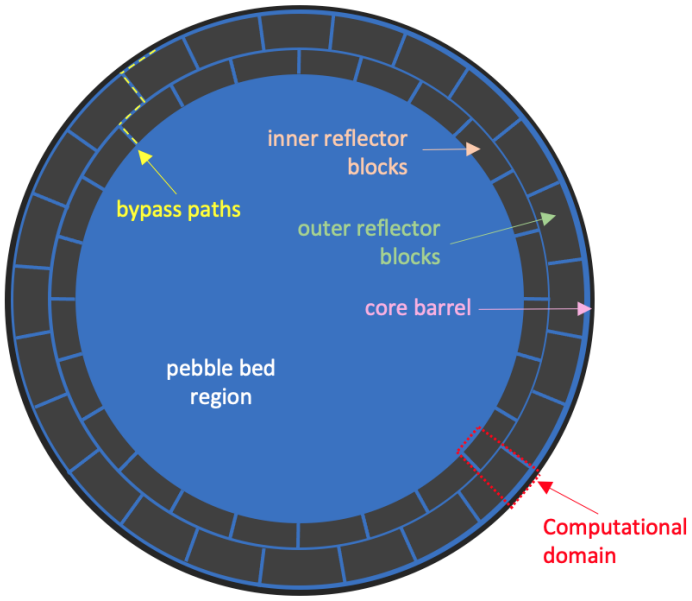

Figure 12: Top-down schematic of the PB-FHR reactor core (only roughly to scale).

To ease the difficulty of meshing very narrow slots, as well as to make visualization and model depiction easier, the width of the gap between blocks is increased in this example to 0.006 m. Of course, over the lifetime of the reactor, the gap width changes due to graphite material response to fluence such that the gap width is not a fixed value. Therefore, we take this liberty here to improve visualization of the thin gaps, but allow setting the gap width as an input to the meshing scripts. Translating this percentage to the PB-FHR geometry corresponds to gaps of approximately 0.005 m thickness.

To form the entire axial height of the reflector, rings of blocks are stacked vertically; each block is 0.52 m tall. The entire core height is 5.3175 m, or about 10 vertical rings of blocks (additional structures at the core inlet and outlet compose any remaining height not evenly divisible by 0.52 m). In both the pebble bed region and through the small gaps between the reflector blocks and between the reflector blocks and the barrel, FLiBe coolant flows. This bypass flow path is shown as

yellow dashed lines; in these gaps, coolant flows both from the pebble bed and outward in the radial direction, as well as vertically from lower reflector rings.

Each ring of blocks is separated from the rings above and below it by very thin horizontal gaps that form as the graphite thermally expands; these flow paths also contribute to bypass flows, allowing a second radial flow path from the pebble bed towards the barrel. For simplicity, this second radial flow path is neglected here.

The total power of the PB-FHR is 236 MW; any power deposition in the reflector is neglected. The FLiBe inlet temperature to the core is 600°C, while the nominal outlet temperature is 700°C.

To reduce computational cost, the coupled NekRS-MOOSE simulation is conducted for a single ring of reflector blocks (i.e. a height of 0.52 m), with azimuthal symmetry assumed to further reduce the domain to half of an inner ring block, half of an outer ring block, and the vertical and radial bypass flow paths between the blocks and the barrel. This computational domain is shown outlined with a red dotted line in the figure above. As already mentioned, the second radial flow path (along fluid "sheets" between vertically stacked rings of blocks) is not considered. This geometry, as well as the boundary conditions imposed, will be described in much greater detail later when the meshes for the fluid and solid phases are discussed.

5.4 Governing Equations

This section describes the physics solved in this coupled conjugate heat transfer problem. Because there are no transient source terms or transient boundary conditions, the MOOSE heat conduction module will solve the steady state energy conservation equation for a solid,

$$-\nabla \cdot (k_s \nabla T_s) = \dot{q}_s$$

where k_s is the solid thermal conductivity, T_s is the solid temperature, and $\dot{q}_s$ is the solid volumetric heat source, which for this application is zero. Solving the steady energy conservation in lieu of the transient equation will accelerate the approach to the coupled pseudo-steady state. NekRS solves the incompressible Navier-Stokes equations,

$$\nabla \cdot \vec{u} = 0$$

$$\rho_f \left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} \right) = -\nabla P + \nabla \cdot \tau + \rho \vec{f}$$

$$\rho_f C_{p,f} \left(\frac{\partial T_f}{\partial t} + \vec{u} \cdot \nabla T_f \right) = \nabla \cdot (k_f \nabla T_f) + \dot{q}_f$$

where $\vec{u}$ is the fluid velocity, ρ_f is the fluid density, P is the pressure, τ is the viscous stress tensor, $\vec{f}$ is a source vector, $\dot{q}_f$ is a volumetric heat source, $C_{p,f}$ is the fluid heat capacity, T_f is the fluid temperature, and k_f is the fluid thermal conductivity. In this example, both $\vec{f}$ and $\dot{q}_f$ are set to zero for simplicity.

Because the expected temperature range is small (on the order of the nominal 100°C temperature rise divided by 10, since the reflector is approximately 10 rings of blocks high), all material properties are taken as constant (though this is not a limitation of NekRS). In addition, the expected flowrate through the reflector block is sufficiently low to be laminar, so the k-τ turbulence model in NekRS is not needed.

NekRS can be solved in either dimensional or non-dimensional form; in the present case, NekRS is solved in non-dimensional form. That is, characteristic scales for velocity, temperature, length, and time are defined and substituted into the governing equations so that all solution fields (velocity, pressure, temperature) are of order unity. A full derivation of the non-dimensional governing equations in NekRS is available with the NekRS documentation, and is not repeated here. Cardinal will handle conversions from a non-dimensional solution to a dimensional MOOSE heat conduction application, but awareness of this non-dimensional formulation will be important for describing the NekRS input files.

In this model, the reference length scale is selected as the block gap width. Therefore, to obtain a mesh with a gap width of unity (in non-dimensional units), the entire NekRS mesh must be multiplied by $1/0.006$. The characteristic velocity is selected to obtain a Reynolds number of 100, which corresponds to $U_{ref} = 0.0575$ based on the values for density and viscosity for FLiBe evaluated at 650°C. A Reynolds number of 100 corresponds to a bypass fraction of about 7%. Further parametric studies for other bypass fractions can be performed by varying the fluid Reynolds and Peclet numbers in the NekRS input files.

5.4.1 Boundary Conditions

This section describes the boundary conditions imposed on the fluid and solid phases. When the meshes are described later, descriptive words such as "inlet" and "outlet" will be directly tied to side sets in the mesh to enhance the verbal description here. As described earlier, this simulation is performed as a standalone case, independent of Pronghorn and Griffin - despite the fact that the reflector flow and heat transfer is tightly coupled to the core thermal-hydraulics. Two simplifications are made:

- The heat flux at the pebble bed-reflector interface is given as a fixed value of 5 kW/m^2, though this value can also be extracted using a flux postprocessor evaluated over the bed-reflector interface in a full-core Pronghorn model.
- The coupled Pronghorn-Griffin PB-FHR model elsewhere in this repository does not model flow through the outer reflector - the reflector is treated as a solid conducting block. Therefore, this particular full-core PB-FHR model cannot directly provide temperature and velocity boundary conditions to the present reflector model. Instead, the inlet boundary conditions are also taken as representing some average conditions in the PB-FHR at an a priori specified bypass fraction and inlet temperature.

For the solid domain, a fixed heat flux of 5 kW/m^2 is imposed on the block surface facing the pebble bed. On the surface of the barrel, a heat convection boundary condition is imposed,

$$q'' = h(T_s - T_\infty)$$

where q'' is the heat flux, h is the convective heat transfer coefficient, and T_∞ is the far-field ambient temperature. Between the reflector blocks, the MOOSE heat conduction module is used to apply quadrature-based radiation heat transfer across a transparent fluid. For a paired set of boundaries, each quadrature point on boundary A is paired with the nearest quadrature point on boundary B. Then, a radiation heat flux is imposed between pairs of quadrature points as

$$q'' = \sigma \frac{\left(T^4 - T_{gap}^4\right)}{\frac{1}{\varepsilon_A} + \frac{1}{\varepsilon_B} - 1}$$

where σ is the Stefan-Boltzmann constant, T is the temperature at a quadrature point, T_{gap} is the temperature of the nearest quadrature point across the gap, and ε_A and ε_B are the emissivities of boundary A and B, respectively.

At fluid-solid interfaces, the solid temperature is imposed as a Dirichlet condition, where NekRS computes the surface temperature. Finally, the top and bottom of the block, as well as all symmetry boundaries, are treated as insulated for simplicity. At the inlet, the fluid temperature is taken as 650°C, or the nominal median fluid temperature. The inlet velocity is selected such that the Reynolds number is 100. At the outlet, a zero pressure is imposed. On the $\theta = 0^\circ$ boundary (i.e. the $y = 0$ boundary), symmetry is imposed such that all derivatives in the y direction are zero. All other boundaries are treated as no-slip.

The $\theta = 7.5^\circ$ boundary (i.e. 360° divided by 24 blocks, divided in half because we are modeling only half a block) should also be imposed as a symmetry boundary in the NekRS model. However, NekRS is currently limited to symmetry boundaries only for boundaries aligned with the x, y, and z coordinate axes. Here, a no-slip boundary condition is used instead, so the correspondence of the NekRS computational model to the actual reactor system is imperfect. At fluid-solid interfaces, the heat flux is imposed as a Neumann condition, where MOOSE computes the surface heat flux.

5.4.2 Initial Conditions

Because the NekRS mesh contains very small elements in the fluid phase, fairly small time steps are required to meet CFL conditions related to stability. Therefore, the approach to the coupled, pseudo-steady conjugate heat transfer solution can be accelerated by obtaining initial conditions from a pure conduction simulation. Then, the initial conditions for the conjugate heat transfer simulation use the temperature obtained from the conduction simulation, with a uniform axial velocity and zero pressure. The process to run Cardinal in conduction mode is described next.

5.5 Meshing

This section describes how the meshes are generated for MOOSE and NekRS. For both applications, the Cubit meshing software is used to programmatically create meshes with user-defined geometry and customizable boundary layers. Journal files, or Python-scripted Cubit inputs, are used to create meshes in Exodus II format. The MOOSE framework accepts meshes in a wide range of formats that can be generated with many commercial and free meshing tools; Cubit is used for this example because the content creator is most familiar with this software, though similar meshes can be generated with your preferred meshing tool.

5.5.1 Solid MOOSE Heat Conduction Mesh

The solid.jou file is a Cubit script that is used to generate the solid mesh. At the top of this file, is a #!python shebang that allows Python to be used to programmatically create the mesh. Any valid Python code (including imported modules) can be used; the actual commands passed to the Cubit meshing tool are written as cubit.cmd(<string command>), where <string command> is a string containing the Cubit command that, in interactive mode, would be manually typed into the Cubit command line in the GUI. For complex meshes, scripting the mesh process is essential for reproducibility and easing the burden of making modifications to the geometry or mesh.

The complete solid mesh (before a series of refinements) is shown below; the boundary names are illustrated towards the right by showing only the highlighted surface to which each boundary corresponds. Names are shown in Courier font. A unique block ID is used for the set of elements corresponding to the inner ring, outer ring, and barrel. Material properties in MOOSE are typically

restricted by block, and setting three separate IDs allows us to set different properties in each of these blocks.

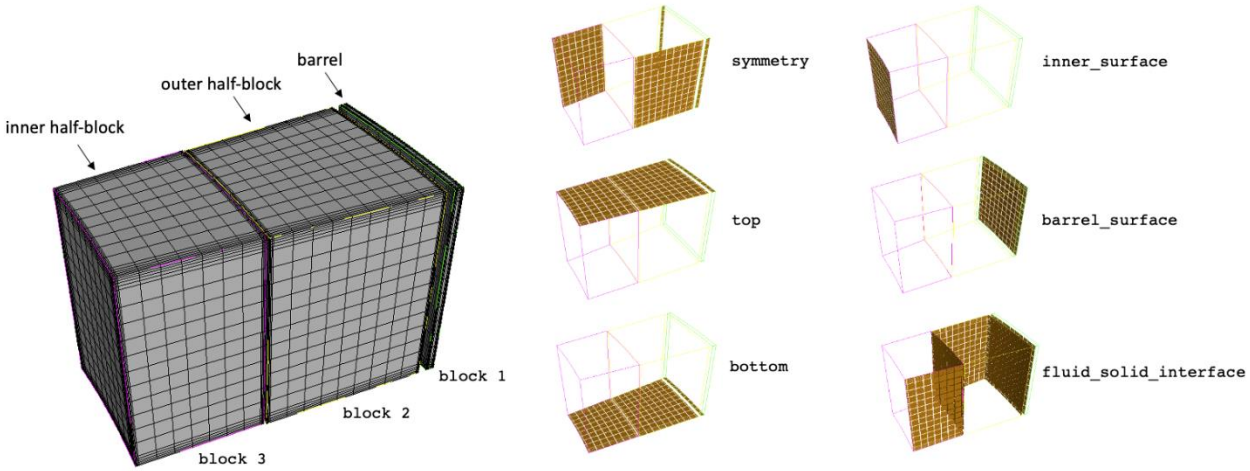

Figure 13: Solid mesh for the reflector blocks and barrel and a subset of the boundary names, before a series of mesh refinements

Unique boundary names are set for each boundary to which we will apply a unique boundary condition; we define the boundaries on the top and bottom of the block, the symmetry boundaries that reflect the fact that we've reduced the full PB-FHR reflector to a half-block domain, and boundaries at the interface between the reflector and the bed and on the barrel surface. To facilitate radiation heat transfer between the thin block gaps, additional boundaries must be defined on the faces on either side of the gaps. Two boundaries are defined per gap - one on either side of the gap. These are shown below, where the naming convention three_to_two indicates a boundary on block 3, across a gap from block 2.

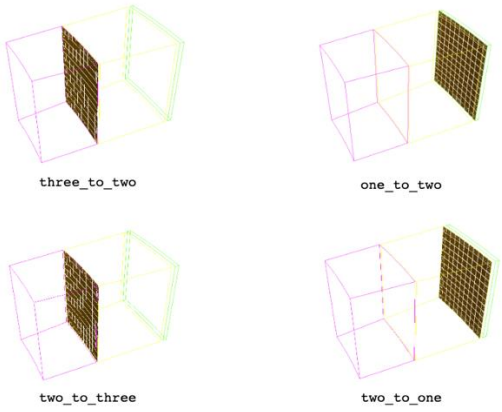

Figure 14: Sidesets defined for enforcing radiation heat transfer boundary conditions

One convenient aspect of MOOSE is that the same elements can be assigned to more than one boundary ID. To help in applying heat flux and temperature boundary conditions between NekRS and MOOSE, we define another boundary that contains all of the fluid-solid interfaces through

which we will exchange heat flux and temperature, as fluid_solid_interface. Some of the elements on the fluid_solid_interface boundary are also present on the boundaries between blocks used for the radiation boundary conditions.

Now that the overall structure of the mesh has been introduced, a brief description of the process of how the mesh was actually constructed is provided. First, the block geometry is formed by creating cylinders and bricks and subtracting them from one another to get angular sectors of cylindrical annuli. Then, the Cubit boundary_layer feature is used to programmatically refine the mesh near surfaces; we perform this refinement because, on some of these surfaces (those exchanging heat with the fluid and radiation across gaps), we expect to have higher thermal gradients that we would like to resolve with a finer mesh. For each boundary layer, we specify a starting element width perpendicular to the surface, a growth factor, and a number of boundary layers we would like to mesh. Finally, the mesh is saved in Exodus II format to disk.

5.5.2 Fluid NekRS Mesh

The fluid.jou is a Cubit script that is used to generate the fluid mesh. The complete fluid mesh is shown below; the boundary names are illustrated towards the right by showing only the highlighted surface to which each boundary corresponds. Names are shown in Courier font. While the names of the surfaces are shown in Courier font, NekRS does not directly use these names - rather, NekRS assigns boundary conditions based on the numeric value of the boundary name; these are shown as "ID" in the figure. An important restriction in NekRS is that the boundary IDs be ordered sequentially beginning from 1.

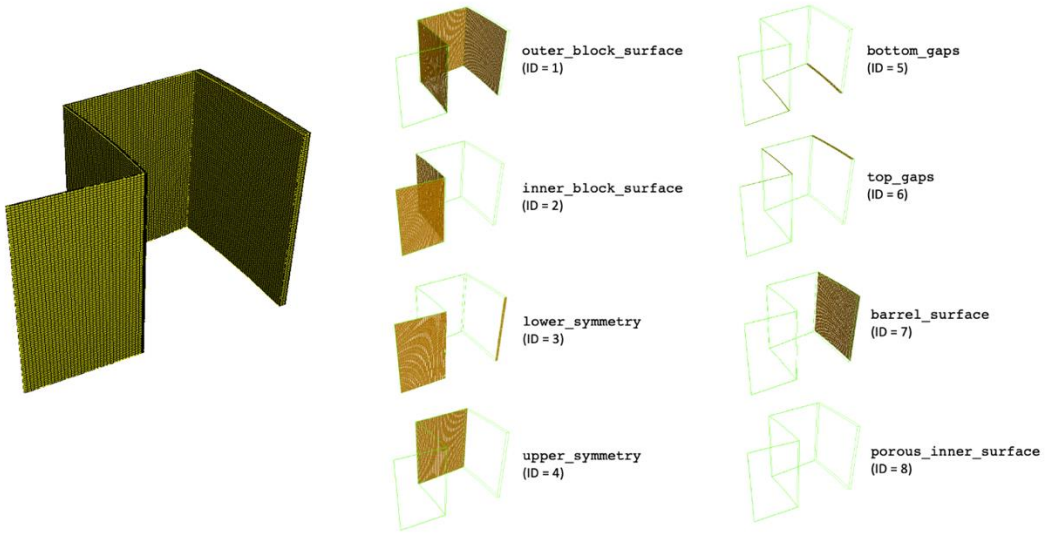

Figure 15: Fluid mesh for the FLiBe flowing around the reflector blocks, along with boundary names and IDs. It is difficult to see, but the porous_inner_surface boundary corresponds to the thin surface at the interface between the reflector region and the pebbles.

A strength of Cardinal for conjugate heat transfer applications is that the fluid and solid meshes do not need to share nodes on a common surface; libMesh mesh-to-mesh data interpolations apply to surfaces of very different refinement and position in space; meshes may even overlap, such as for curvilinear applications. The next image shows a zoom-in of the two mesh files (for the fluid and

solid phases); rather than being limited with a continuous mesh mapping from the fluid phase inwards to the solid phase, each phase can be meshed according to its physics requirements.

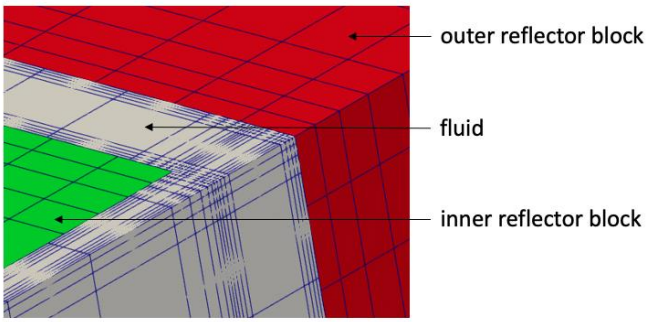

Figure 16: Zoomed-in view of the fluid and solid meshes, overlaid in Paraview. Lines are element boundaries.

Creating the fluid mesh is significantly more involved than creating the solid mesh. First, a series of boolean operations is performed to obtain an un-meshed volume that the fluid will flow through. Rather than create a single volume, the fluid region is created as 7 separate volumes; this allows full customization of the meshing near the corners where we would like the boundary layer scheme to extend from the interior corners (with angles greater than 180 degrees) across the fluid gaps. Like the solid mesh, boundary layers are applied on all surfaces (even if the particular boundary condition on a surface is not necessarily "no-slip"). Finally, the boundary IDs are created; while names are assigned in the Cubit script, these names are not accessible to NekRS, and the numeric boundary IDs are instead used for designating boundaries.

Because NekRS uses a custom binary mesh format with a .re2 extension, a conversion utility must be used to convert an Exodus II format to NekRS's .re2 NekRS format. This utility, or the exo2nek program, requires the Exodus mesh elements to be type HEX20 (a twenty-node hexahedral element), so we set this element type from Cubit.

5.6 Part 1: Initial Conduction Coupling

In this section, NekRS and MOOSE are coupled for conduction heat transfer in the solid reflector blocks and barrel, and through a stagnant fluid. The purpose of this stage of the analysis is to obtain a restart file for use as an initial condition to accelerate the NekRS calculation for conjugate heat transfer, since the energy equation is slowest to converge. Because this initial condition is only used for accelerating a later calculation, applying the radiation quadrature-based boundary conditions is deferred. All input files for this stage of the analysis are present in the pbfhr/reflector/conduction directory. The following sub-sections describe these files.

5.6.1 Solid Input Files

The solid phase is solved with the MOOSE heat conduction module, and are described in the solid.i input. At the top of this file, the core heat flux is defined as a variable local to the file. The value of this variable can then be used anywhere else in the input file with syntax like ${core_heat_flux}, similar to bash syntax. Next, the solid mesh is specified by pointing to the Exodus mesh generated

previously.The heat conduction module will solve for temperature, which is defined as a nonlinear variable.

The Transfer system in MOOSE is used to communicate auxiliary variables across applications; a boundary heat flux will be computed by MOOSE and applied as a boundary condition in NekRS. In the opposite direction, NekRS will compute a surface temperature that will be applied as a boundary condition in MOOSE. Therefore, both the flux (flux) and surface temperature (nek_temp) are declared as auxiliary variables. The solid app will compute flux, while nek_temp will simply receive a solution from the NekRS sub-application. The flux is computed as a constant monomial field (a single value per element) due to the manner in which material properties are accessible in auxiliary kernels in MOOSE.

In this example, the overall calculation workflow is as follows:

- Run MOOSE heat conduction with a given surface temperature distribution from NekRS.
- Send heat flux to NekRS as a boundary condition.
- Run NekRS with a given surface heat flux distribution from MOOSE.
- Send surface temperature to MOOSE as a boundary condition.

The above sequence is repeated until convergence of the coupled domain. For the very first time step, an initial condition should be set for nek_temp; this is done using a function with an arbitrary, but not wholly unrealistic, distribution for the fluid temperature. That function is then used as an initial condition with a FunctionIC object.

Next, the governing equation solved by MOOSE is specified with the Kernels block as the HeatConduction kernel, or $-\nabla \cdot (k\nabla T) = 0$. An auxiliary kernel is also specified for the flux variable that specifies that the flux on the fluid_solid_interface boundary should be computed as $-k\nabla T \cdot \hat{n}$.

Next, the boundary conditions on the solid are applied. On the fluid-solid interface, a MatchedValueBC applies the value of the nek_temp receiver auxiliary variable to T in a strong Dirichlet sense. Insulated boundary conditions are applied on the symmetry, top, and bottom boundaries. On the boundary at the bed-reflector interface, the core heat flux is specified as a NeumannBC. Finally, on the surface of the barrel, a heat flux of $h(T - T_\infty)$ is specified, where both h and T_∞ are specified as material properties.

The HeatConduction kernel requires a material property for the thermal conductivity; material properties are also required for the heat transfer coefficient and far-field temperature for the ConvectiveHeatFluxBC. These material properties are specified in the Materials block. Here, different values for thermal conductivity are used in the graphite and steel.

Next, the MultiApps and Transfers blocks describe the interaction between Cardinal and MOOSE. The MOOSE heat conduction module is here run as the master application, with the NekRS wrapping run as the sub-application. We specify that MOOSE will run first on each time step. Allowing sub-cycling means that, if the MOOSE time step is 0.05 seconds, but the NekRS time step is 0.02 seconds, that for every MOOSE time step, NekRS will perform three time steps, of length 0.02, 0.02, and 0.01 seconds to "catch up" to the MOOSE master application. If sub-cycling is turned off, then the smallest time step among all the various applications is used.

Three transfers are required to couple Cardinal and MOOSE; the first is a transfer of surface temperature from Cardinal to MOOSE. The second is a transfer of heat flux from MOOSE to

Cardinal. And the third is a transfer of the total integrated heat flux from MOOSE to Cardinal (computed as a postprocessor), which is then used internally by NekRS to re-normalize the heat flux (after interpolation onto its GLL points).

Next, postprocessors are used to compute the integral heat flux as a SideIntegralVariablePostprocessor. Next, the solution methodology is specified. Although the solid phase only includes time-independent kernels, the heat conduction is run as a transient because NekRS ultimately must be run as a transient (NekRS lacks a steady solver). A nonlinear tolerance of 1E-6 is used for each solid time step, and the overall coupled simulation is considered converged once the relative change in the solution between steps is less than 5E-4. Finally, an output format of Exodus II is specified.

5.6.2 Fluid Input Files

The fluid phase is solved with Cardinal, which under-the-hood performs the solution with NekRS. The wrapping of NekRS as a MOOSE application is specified in the nek.i file. Compared to the solid input file, the fluid input file is quite minimal, as the specification of the NekRS problem setup is mostly performed using the NekRS input files that would be required to run NekRS as a standalone application.

First, a local variable, fluid_solid_interface, is used to define all the boundary IDs through which NekRS is coupled via conjugate heat transfer to MOOSE for convenience. Next, a NekRSMesh, a class specific to Cardinal, is used to construct a "mirror" of the surfaces in the NekRS mesh through which boundary condition coupling is performed. In order for MOOSE's MultiAppNearestNodeTransfer to correctly match nodes in the solid mesh to nodes in this fluid mesh mirror, the entire mesh must be scaled by a factor of L_{ref} to return to dimensional units.

Next, the Problem block describes all objects necessary for the actual physics solve; for the solid input file, the default of FEProblem was implicitly assumed. However, to replace MOOSE finite element calculations with NekRS spectral element calculations, a Cardinal-specific NekRSProblem class is used. Like all MOOSE-wrapped apps, the problem class that wraps NekRS derives from ExternalProblem. To allow conversion between a non-dimensional NekRS solve and a dimensional MOOSE coupled heat conduction application, the characteristic scales used to establish the non-dimensional problem are provided.

Next, a Transient executioner is specified. This is the same executioner used for the solid case, except now a Cardinal-specific time stepper, NekTimeStepper is provided. This time stepper simply reads the time step specified in NekRS's .par file (to be described shortly), and converts it to dimensional form if needed. An Exodus II output format is specified. It is important to note that this output file only outputs the temperature and heat flux solutions on the surface mirror mesh; the solution over the entire NekRS domain is output with the usual .fld field file format used by standalone NekRS calculations.

Finally, several postprocessors are included. A postprocessor named flux_integral is added automatically by NekRSProblem to receive the value of the heat flux integral from MOOSE for internal normalization in NekRS. The other three postprocessors are all Cardinal-specific postprocessors that perform integrals and global min/max calculations over the NekRS domain. Here, the boundary_flux postprocessor computes $-k\nabla T \cdot \hat{n}$ over a boundary in the NekRS mesh. This value should approximately match the imposed heat flux, flux_integral, though perfect agreement is not to be expected since flux boundary conditions are only weakly imposed in the spectral element method. The max_nek_T and min_nek_T then compute the maximum and

minimum temperatures throughout the entire NekRS domain (i.e. not only on the conjugate heat transfer coupling surfaces).

Note that fluid.re2 does not appear anywhere in nek.i - the fluid.re2 file is a mesh used directly by NekRS, while NekRSMesh is a mirror of the boundaries in fluid.re2 through which boundary coupling with MOOSE will be performed.

You will likely notice that many of the almost-always-included MOOSE blocks are absent from the nek.i input file - for instance, there are no nonlinear or auxiliary variables. The NekRSProblem assists in input file setup by declaring many of these coupling fields automatically. For this example, two auxiliary variables named temp and avg_flux are added automatically; these variables receive incoming and outgoing transfers to/from NekRS. You will see both temp and avg_flux referred to in the solid input file Transfers section.

In addition, the nek.i input file only describes how NekRS is wrapped within the MOOSE framework; with the fluid.re2 mesh file, each NekRS simulation requires at least three additional files, that share the same case name fluid but with different extensions. The additional NekRS files are:

- fluid.par: High-level settings for the solver, boundary condition mappings to sidesets, and the equations to solve.
- fluid.udf: User-defined C++ functions for on-line postprocessing and model setup
- fluid.oudf: User-defined OCCA kernels for boundary conditions and source terms

A detailed description of all of the available parameters, settings, and use cases for these input files is available on the NekRS documentation website. Because the purpose of this analysis is to demonstrate Cardinal's capabilities, only the aspects of NekRS required to understand the present case will be covered. First, begin with the fluid.par file.

The input consists of blocks and parameters. The [GENERAL] block describes the time stepping, simulation end control, and the polynomial order. Here, a time step of 0.025 (non-dimensional) is used; a NekRS output file is written every 100 time steps. Because NekRS is run as a sub-application to MOOSE, the stopAt and numSteps fields are actually ignored, so that the steady state tolerance in the MOOSE master application dictates when a simulation terminates. Because the purpose of this simulation is only to obtain a reasonable initial condition, a low polynomial order of 2 is used.

Next, the [VELOCITY] and [PRESSURE] blocks describe the solution of the pressure Poisson equation and velocity Helmholtz equations. In the velocity block, setting solver = none turns off the velocity solution; therefore, none of the parameters specified here are of consequence, so their description will be deferred. Finally, the [TEMPERATURE] block describes the solution of the temperature passive scalar equation. $\rho_f C_{p,f}$ is set to unity because the solve is conducted in non-dimensional form.

The coefficient on the diffusive equation term is equal to $1/Pe$. In NekRS, specifying conductivity = -1500.5 is equivalent to specifying conductivity = 0.00066644 (i.e. $1/1500.5$), or a Peclet number of 1500.5. Next, residualTol specifies the solver tolerance for the temperature equation to 10^{-8}. Finally, the boundaryTypeMap is used to specify the mapping of boundary IDs to types of boundary conditions. NekRS uses short character strings to represent the type of boundary condition; boundary conditions used in this example include:

- W: no-slip wall
- symy: symmetry in the y-direction
- v` user-defined velocity
- o: zero-gradient outlet
- f: user-defined flux
- I: insulated
- t: user-defined temperature

Therefore, in reference to the fluid mesh, boundaries 1, 2, and 7 are flux boundaries (these boundaries will receive a heat flux from MOOSE, boundaries 3, 4, and 8 are insulated, boundary 5 is a specified temperature, and boundary 6 is a zero-gradient outlet. The actual assignment of numeric values to these boundary conditions is then performed in the fluid.oudf file. The fluid.oudf file contains OCCA kernels that will be run on a GPU (if present). Because this case does not have any user-defined source terms in NekRS, these OCCA kernels are only used to apply boundary conditions.

The names of these functions correspond to the boundary conditions that were applied in the .par file - only the user-defined temperature and flux boundaries require user input in the .oudf file. For each function, the bcData object contains all information about the current boundary that is "calling" the function; bc->id is the boundary ID, bc->s is the scalar (temperature) solution at the present GLL point, and bc->flux is the flux (of temperature) at the present GLL point. The bc->wrk array is a scratch space to which the heat flux values coming from MOOSE are written. These OCCA functions then get called directly within NekRS.

Finally, the fluid.udf file contains C++ functions that are user-defined functions through which interaction with the NekRS solution are performed. Here, the UDF_Setup function is called once at the very start of the NekRS simulation, and it is here that initial conditions are applied. The fluid.udf file is shown below.

5.6.3 Execution and Postprocessing

After converting the NekRS output files to a format viewable in Paraview, the simulation results can be displayed. The solid temperature, surface heat flux, and fluid temperature are shown below. Note that the fluid temperature is shown in nondimensional form based on the selected characteristic scales. The image of the fluid solution is rotated so as to better display the temperature variation around the inner reflector block.

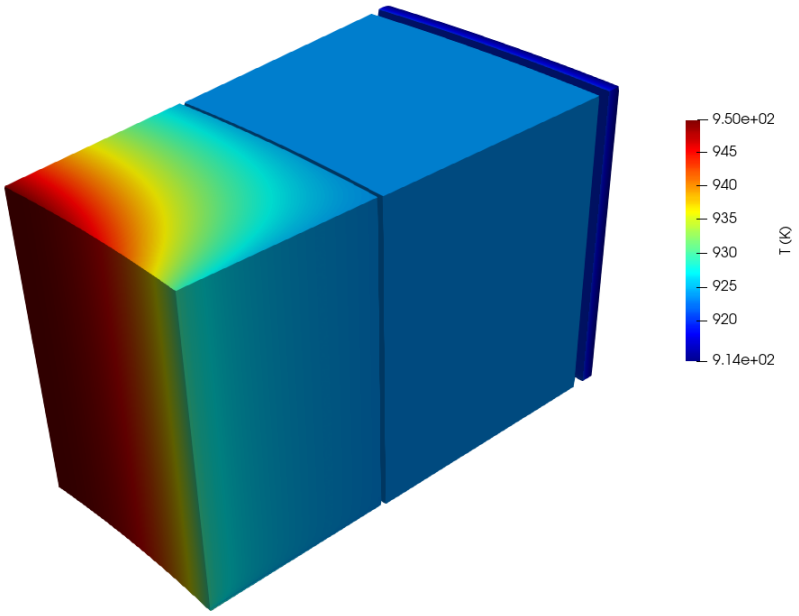

Figure 17: Solid temperature for steady state conduction coupling between MOOSE and NekRS

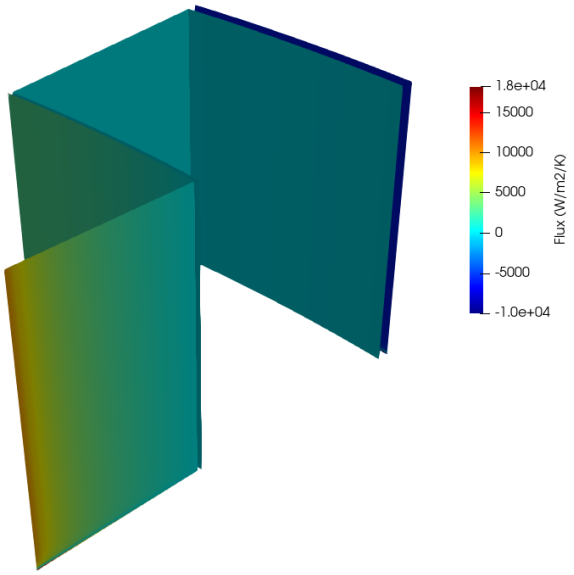

Figure 18: Solid surface heat flux for steady state conduction coupling between MOOSE and NekRS

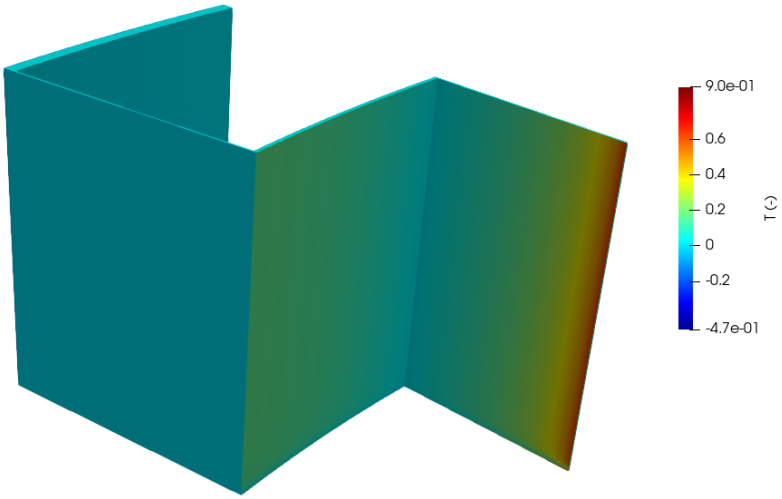

Figure 19: Fluid temperature (nondimensional) for steady state conduction coupling between MOOSE and NekRS

The solid blocks are heated by the pebble bed along the bed-reflector interface, so the temperatures are highest in the inner block. As can be seen in the solid heat flux, the heat flux into the fluid is in some places positive (such as near the inner reflector block) and is in other places negative (such as near the barrel) where heat leaves the system.

5.7 Part 2: Conjugate Heat Transfer Coupling

In this section, NekRS and MOOSE are coupled for conjugate heat transfer between the FLiBe coolant and the reflector blocks and barrel. All input files for this stage of the analysis are present in the pbfhr/reflector/cht directory. The following sub-sections describe all of these files; for brevity, most emphasis will be placed on input file setup that is different or extends the conduction case from earlier.

5.7.1 Solid Input Files

The solid phase is again solved with the MOOSE heat conduction module. The input file is largely the same as the conduction case, except that now the gap radiation heat flux between blocks is included with a ThermalContact action.

This adds radiation heat transfer across the gap between the inner and outer reflector blocks and across the gap between the outer reflector block and the barrel. The emissivity for all surfaces is assumed to be 0.8. This action then adds all requisite MOOSE objects to add quadrature-based radiation boundary conditions, such as paired auxiliary variables to match a node on surface A with a node on surface B and the ensuing heat flux boundary condition objects.

5.7.2 Fluid Input Files

The fluid phase is again solved with NekRS wrapped as a MOOSE app via Cardinal. The input file is largely the same as the conduction case, except that additional postprocessors are added to query both the thermal and hydraulic aspects of the NekRS solution.

We have added postprocessors to compute the average inlet pressure, and the average inlet and outlet mass flowrates. Like the NekVolumeExtremeValue, these postprocessors operate directly on NekRS's internal solution arrays to provide diagnostic information. Because the outlet pressure is set to zero, pressure_in corresponds to the pressure drop in the fluid.

As seen earlier, four additional files are required to set up the NekRS simulation - fluid.re2, fluid.par, fluid.udf, and fluid.oudf. These files are largely the same as those used in the steady conduction model, so only the differences will be emphasized here. startFrom provides a restart file, conduction.fld and specifies that we only want to read temperature from the file (by appending +T to the file name). We increase the polynomial order as well. In the [VELOCITY] block, the density is set to unity, because the solve is conducted in nondimensional form.

The coefficient on the diffusive term is equal to $1/Re$. In NekRS, specifying diffusivity = -100.0 is equivalent to specifying diffusivity = 0.01 (i.e. $1/100.0$), or a Reynolds number of 100.0. All other parameters have similar interpretations as before. The fluid phase is again solved with NekRS wrapped as a MOOSE app via Cardinal. The input file is largely the same as the conduction case, except that additional postprocessors are added to query both the thermal and hydraulic aspects of the NekRS solution.

5.7.3 Execution and Postprocessing

The temperature and heat flux distributions are qualitatively quite similar to those for the conduction case because temperature differences across the block gaps are not very large. Therefore, only the fluid pressure and velocity distributions are shown below, both in non-dimensional form.

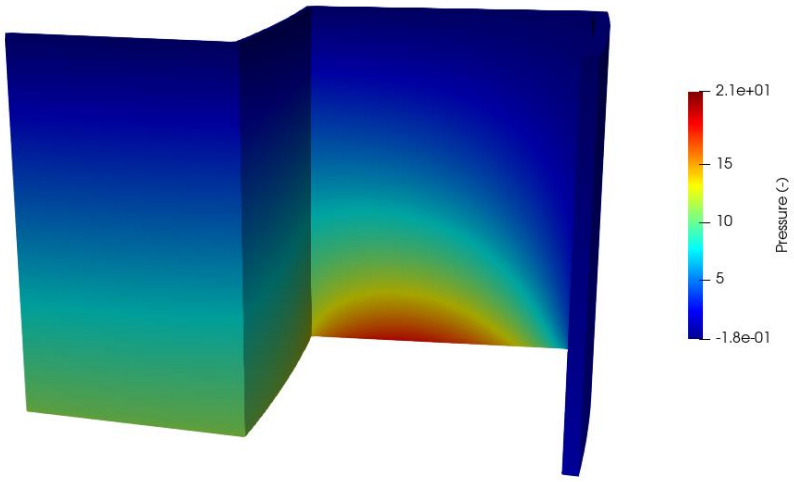

Figure 20: Pressure (nondimensional) for conjugate heat transfer coupling between MOOSE and NekRS

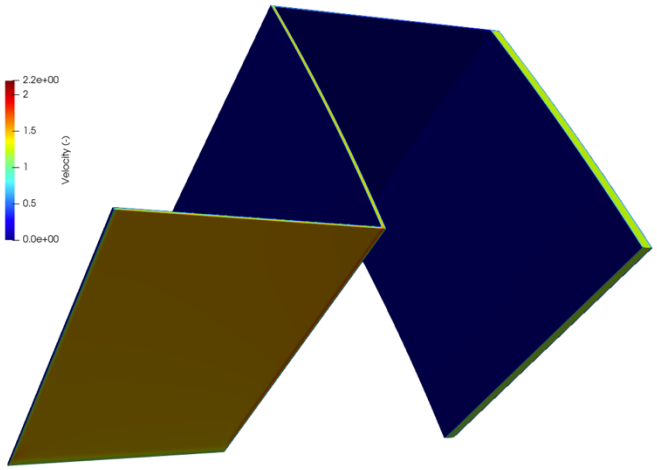

Figure 21: Velocity (nondimensional) for conjugate heat tranfser coupling between MOOSE and NekRS

The no-slip condition on the solid surface, and the symmetry condition on the $y = 0$ surface, are clear in the velocity image. The pressure loss is highest in the gap along the $\theta = 7.5°$ boundary due to the imposition of no-slip conditions on both sides of the half-gap width due to limitations in NekRS's boundary conditions. Therefore, these pressure predictions are likely to change once NekRS's symmetry conditions are expanded to non-x/y/z-aligned surfaces.

5.8 Pronghorn Closures

This analysis has only predicted the conjugate heat transfer solution for a single value of Reynolds number and a single value of gap width. This section briefly describes the procedure necessary to use Cardinal for generating porous media closures for Pronghorn - specifically, the friction coefficient. Pronghorn's momentum conservation equation contains a distributed loss term that relates the friction pressure drop to a closure, W,

$$\epsilon \frac{\partial \mathrm{P}}{\partial \mathrm{x_i}} = -\mathrm{W_{ij}} \rho_\mathrm{f} \mathrm{V_j}$$

where ϵ is the porosity, W is a diagonal tensor that is related to the conventional definition of a friction coefficient, and V is the interstitial velocity. The present model can be used to predict W as a function of the block geometry and the Reynolds number. The calculation workflow would be as follows:

1. Re-run the solid.jou and fluid.jou scripts for a range of block gap widths.
2. Re-run the steady conduction initial condition calculation and the conjugate heat transfer calculation for the range of block gap widths, for a range of Reynolds numbers.
3. For each individual run, compute the overall porosity of the block-fluid domain, and express $\partial P / \partial x_i$ as $\Delta P / H$, where ΔP is the pressure drop given by the pressure_in postprocessor in the nek.i input file, and H is the height of the block, or 0.502 m.

4. Correlate the pressure drop results and obtain a functional fit to W (with terms linear and quadratic in velocity, according to dimensional considerations for pressure loss).
5. Use a FunctionAnisotropicDragCoefficients friction factor model in Pronghorn, and provide the functional fit in the Pronghorn input file for the linear and quadratic terms in Darcy_coefficient (the linear proportionality) and Forchheimer_coefficient (the quadratic proportionality).

Similar calculation procedures would be used to predict the Nusselt number, or other porous media closures such as effective solid conductivities - parameterize the operating space and geometry that affects the physics of interest, run repeated high-resolution calculations, and correlate the data into a form that can then be inserted into Pronghorn using existing material property classes.

6 Heat Pipe Microreactor (Sockeye + BISON)

The VTB Heat Pipe Microreactor model is based off of the Micro-Reactor design under analysis by the NEAMS Micro-Reactor Application Drivers area [Stauff *et al.*, 2021]. As a modeling exercise, a micro-reactor concept was designed at ANL to gather some of the most pressing modeling challenges faced by the micro-reactor industry: a) the use of heat pipe technologies to remove the nuclear heat; b) the use of TRi-structural ISOtropic (TRISO) fuel to enable operations at very high temperatures; c) the use of rotating control rod drums in the radial reflectors.

6.1 Description of the reactor

This core has a rated power of 2 MW thermal and its layout is shown in Figure 22 below. A traditional TRISO fuel with 19.95 at% Low-Enriched Uranium (LEU) in UCO form was adopted in a hexagonal graphite matrix with a 40% packing fraction. The core length is set to 160 cm with 20 cm upper and lower axial reflectors made of beryllium metal. 30 fuel assemblies are surrounded by one ring of beryllium reflector and 12 control drums. Only one sixth of the core geometry was modeled to reduce the size of the modeling problem.

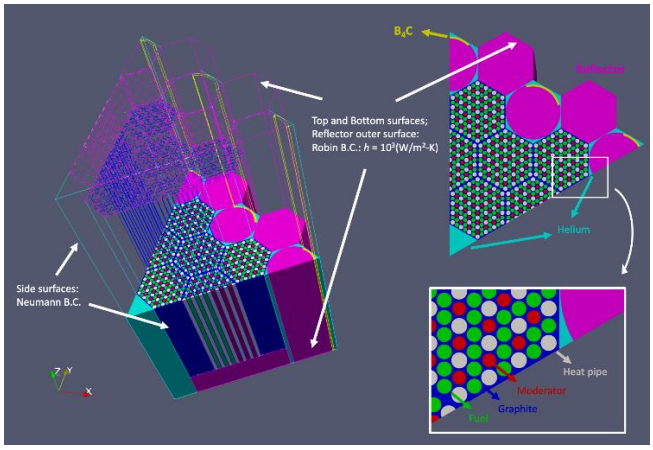

Figure 22: Description of Micro-Reactor Model

To achieve an optimum level of moderation, yttrium-hydride (YH2) pins are employed in addition to graphite structure component as YH2 provides more efficient neutron slowing-down capability enabling the design of more compact core. The yttrium-hydride is surrounded in two layers of helium gaps separating the moderator, the stainless steel envelope and the graphite monolith.

This concept employs heat pipes with a stainless-steel envelope and potassium heat-transfer fluid. The heat pipe is separated from the graphite monolith by another layer of helium gap. The heat pipe region has been divided into three zones representing the fluid at the phase of vapor, liquid and wick respectively.

The control system of the core includes 12 control drums located in the radial reflector that are capable of bringing the core to cold shutdown throughout the operation of the reactor. For redundancy purposes, a shutdown rod is located in the central core location.

6.2 Multiphysics model

6.2.1 Mesh File

Mesh generation was performed with Cubit toolkit and the mesh file is used in BISON, and in the MultiApp (coupling BISON and Sockeye). A simplified 1/6 core was generated for preliminary assessment (Figure 23). This mesh does not contain the helium gaps and stainless steel envelops for moderators and heat pipes, both of which will be included in the later version of full core model. The mesh density in radial direction is high as multiple small features (fuel rods, moderators, heat pipes and control drums) are involved.

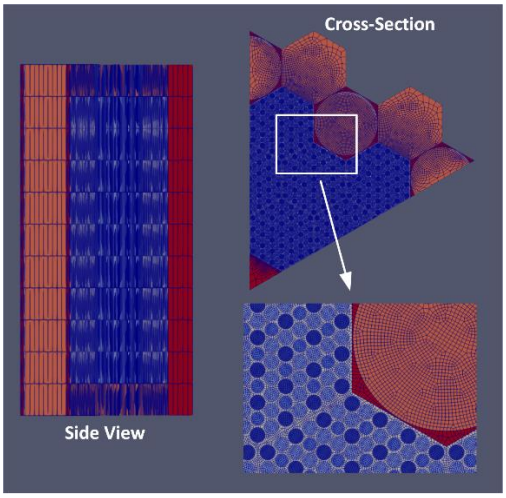

Figure 23: Mesh description for microreactor model

6.2.2 BISON Model (Appendix A)

The present BISON simulation utilized the heat conduction module in MOOSE, supported with materials models in BISON including the thermal properties of TRISO fuel, thermal and mechanical properties of SS316 (cladding material for YH2). Convective boundary conditions were defined at the top and bottom of the model with an external temperature of 800K and h=100 W/m^2-K.

6.2.3 Sockeye Model (Appendix B)

Sockeye is used for steady-state heat pipe thermal performance using the effective thermal conductivity model, i.e., a 2D axisymmetric conduction model with a very high thermal conductivity of 2×10^5 W/m-K is applied to the vapor core. A heat flux boundary condition is applied to the exterior of the casing in the evaporator section, which is provided by the bulk conduction model. A convective boundary condition is applied to the exterior of the envelope in the condenser section, with an external temperature of 800 K and h=10^6 W/m^2-K. The list and location of each heat-pipe in the 1/6 code is provided in the repository.

6.2.4 MultiApp

Multiphysics simulations performed leverage the MOOSE MultiApps system to couple thermo-mechanics and heat pipe heat energy transfer systems, which are simulated by BISON, and

Sockeye, respectively. Each code has its own mesh and corresponding space and time scales. The MOOSE MultiApp system is leveraged to tightly couple the different codes via Picard iterations as illustrated in Figure 24. In the absence of a neutronic model, constant power density is transferred to the BISON sub-application into the thermal simulation. Before solving for the temperature, BISON passes the surface temperature at the heat pipes to the Sockeye sub-apps, which compute the temperature distribution in every single heat pipe. Based on the calculated temperature gradient in each heat pipe, the consequent heat flux is passed back to BISON and converted into a heat sink for the thermal calculation.

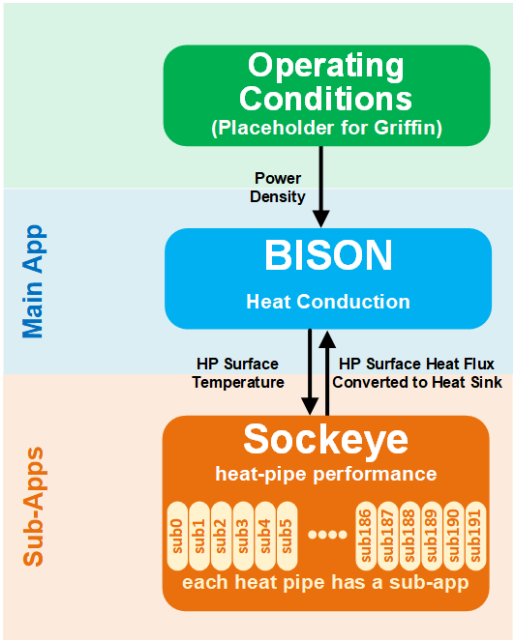

Figure 24: Multi-physics coupling strategy.

6.3 Multiphysics results

A multiphysics steady-state analysis was completed in the full-core model by coupling the Bison thermo-mechanical solver to the Sockeye heat-pipe thermal performance solver. Figure 25 shows a picture obtained with Paraview displaying the temperature throughout the core at steady-state. Temperatures are displayed in Kelvin.

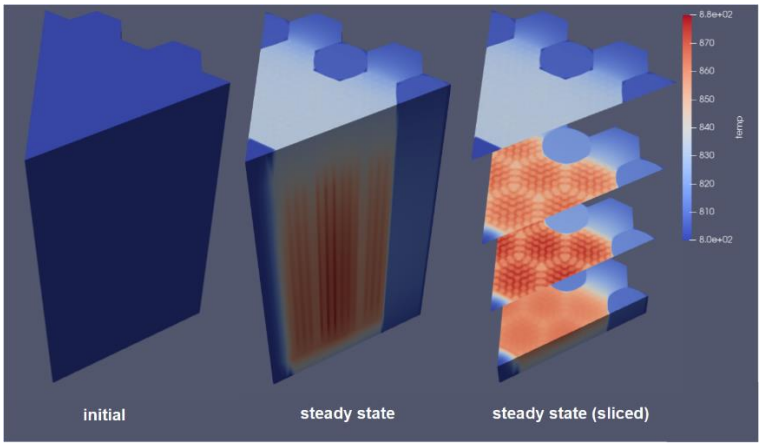

Figure 25: Temperature distribution of microreactor core

Figure 26 displays the temperature obtained with Sockeye along the interface of the wick and vapor regions of the most inner heat-pipe at steady-state.

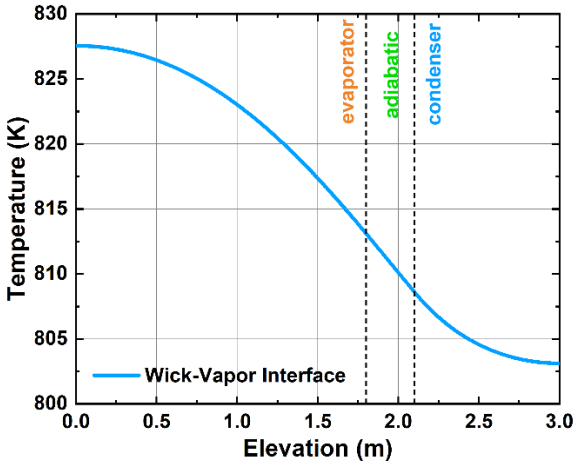

Figure 26: Sockeye-calculated temperature distribution of heat pipe

7 Summary

In fiscal year 2021, five reference models of advanced reactors were developed and submitted to the Virtual Test Bed. These models have all received confirmed interest from industry stakeholders and were prioritized in terms of the model generation schedule. Many of the models were developed or refined based on pre-existing models that were developed from other government-funded activities. Such models were not easily shareable or documented sufficiently for new users to use them efficiently. Thus, the VTB served as an ideal mechanism to achieve an outward facing repository of curated models generated using NEAMS tools to promote the use of NEAMS tools via access to example models developed from public information.

8 References

Ahmed 2017 K. K. Ahmed, R. O. Scarlat, and R. Hu, "Benchmark Simulation of Natural Circulation Cooling System with Salt Working Fluid Using SAM," NURETH-17, Xian, China, September 3-8, 2017.

Fang et al., 2021 Fang, J., Shaver, D. R., Min, M., Fischer, P., Lan, Y.-H., Rahaman, R., Romano, P., Benhamadouche, S., Hassan, Y. A., Kraus, A., & Merzari, E. (2021). Feasibility of Full-Core Pin Resolved CFD Simulations of Small Modular Reactor with Momentum Sources. *Nuclear Engineering and Design*, *378*, 111143. https://doi.org/10.1016/j.nucengdes.2021.11114

GA 1988 General Atomic's Prismatic High Temperature Gas Cooled Reactor, https://aris.iaea.org/PDF/PrismaticHTR.pdf (1988).

Hu 2021 R. Hu *et al.*, SAM Theory Manual, Argonne National Laboratory, *ANL/NE-17/4* Rev. 1 (2021).

INL 2011 HTGR technology course for the nuclear regulatory commission. Module 5a, "Prismatic HTGR core design description". Idaho National Laboratory, *May 24-27, 2011*.

Merzari et al., 2017 Merzari, E., Obabko, A., Fischer, P., Halford, N., Walker, J., Siegel, A., & Yu, Y. (2017). Large-scale large eddy simulation of nuclear reactor flows: Issues and perspectives. *Nuclear Engineering and Design*, *312*, 86–98. https://doi.org/10.1016/J.NUCENGDES.2016.09.028

NEA 2018 "NEA Benchmark of the Modular High Temperature Gas-Cooled Reactor – 350 MW Core Design Volumes I and II," Nuclear Energy Agency, Nuclear Science Committee, NEA/NSC/R(2017)4, 2018. http://www.oecd.org/officialdocuments/publicdisplaydocumentpdf/?cote=NEA/NSC/R(2017)4&docLanguage=En

Neylan et al., 1988 A.J. Neylan, D.V. Graf, and A.C. Millunzi, "The Modular High-Temperature Gas-Cooled Reactor in The U.S.," *Nucl. Engr. Des.* 109 (1988) 99-105.

Stauff et al., 2021 Nicolas E. Stauff, Kun Mo, Yan Cao, Justin W. Thomas, Yinbin Miao, Changho Lee, Christopher Matthews, Bo Feng, "Preliminary Applications of NEAMS Codes for Multiphysics Modeling of a Heat Pipe Microreactor," proceedings of ANS Annual 2021 meeting, June (2021).

Tomboulides 2018 Tomboulides, A., Aithal, S. M., Fischer, P. F., Merzari, E., Obabko, A. V., & Shaver, D. R. (2018). A novel numerical treatment of the near-wall regions in the k− ω class of RANS models. *International Journal of Heat and Fluid Flow*, *72*, 186–199. https://doi.org/10.1016/J.IJHEATFLUIDFLOW.2018.05.017

Turner et al., 1988 R.F. Turner, A.M. Baxter, O.M. Stansfield, and R.E. Vollman, "Annular Core for the Modular High-Temperature Gas-Cooled Reactor (MHTGR)," *Nucl. Engr. Des.* 109 (1988) 227-231.

UCB 2014 P. Petersen et. al., "Technical Description of the "Mark 1" Pebble-Bed Flouride-Salt-Cooled High Temperature Reactor (PB-FHR) Power plant". University of California Berkeley, UCBTH-14-002, September 2014.

Vegendla et al., 2019 P. Vegendla, R. Hu and L. Zou, "Multi-Scale Modeling of Thermal-Fluid Phenomenon Related to Loss of Forced Circulation Transient in HTGRs," Argonne National Laboratory, ANL-19/35, September 2019

Zweibaum 2015 N. Zweibaum, "Experimental Validation of Passive Safety Systems Models: Application to Design and Optimization of Fluoride-Salt-Cooled, High-Temperature Reactors," PhD dissertation, University of California Berkeley, 2015.

Page intentionally left blank