RAVEN – CTF – FMU Workflow

Wesley Williams, William Gurecky, Vineet Kumar, Dane de Wet

Advanced Reactor Engineering & Development Section, ORNL

Integrated Energy Systems (IES) Tools: Capability Overview and Training Day 4

24th March 2022

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

Overview

- Introduction to Futility based CTF-FMU coupling
- CTF-FMU coupling results
- Introduction to RAVEN-CTF-FMU
- RAVEN-CTF-FMU results
- Conclusions

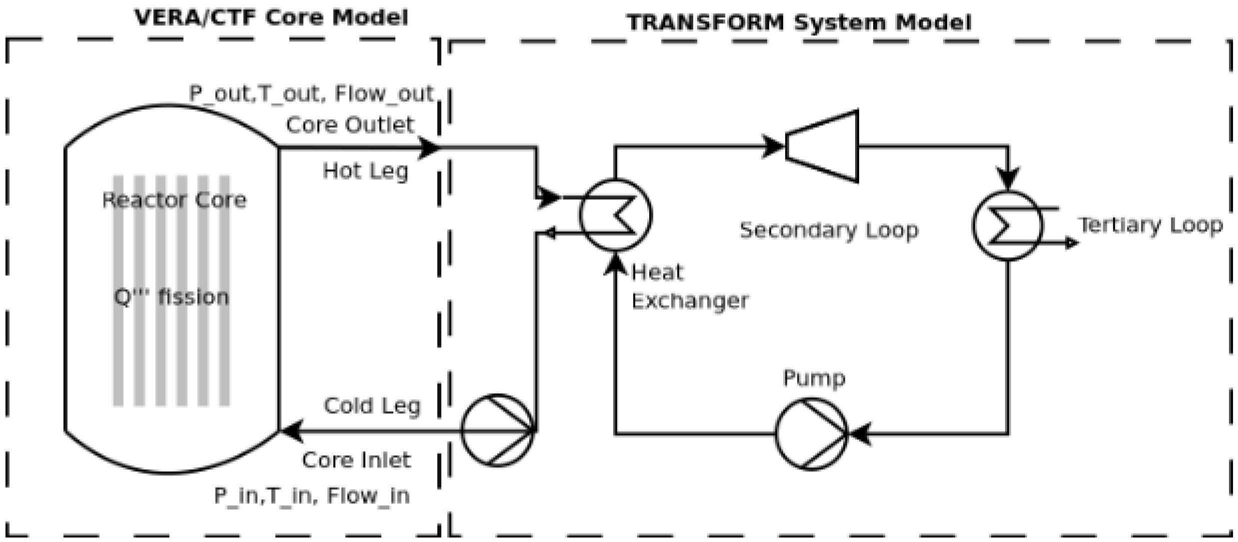
CTF-FMU Coupling

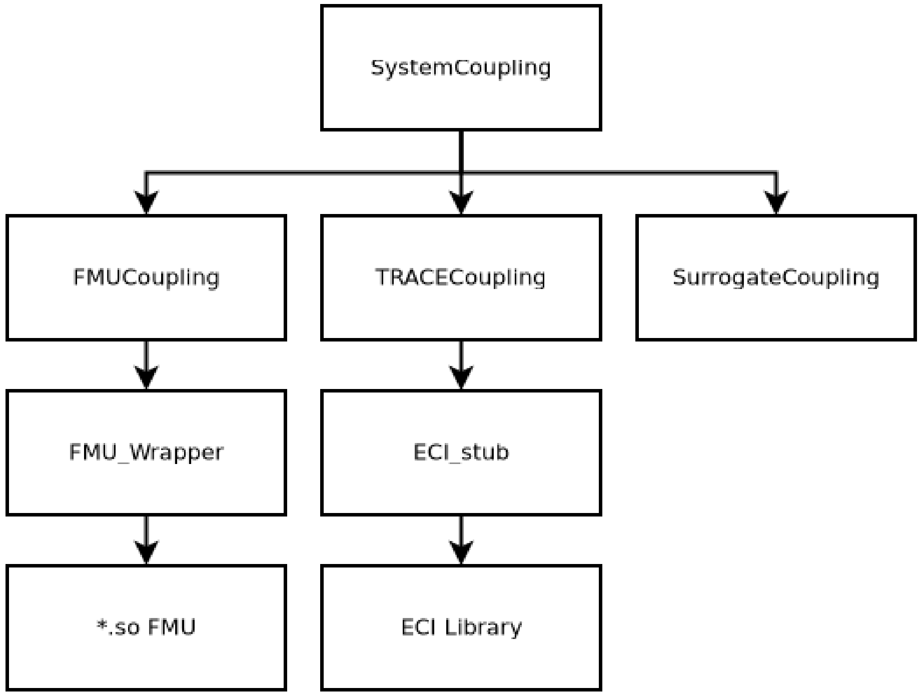
- An in-memory coupling between the sub-channel thermal hydraulics code COBRA-TF (CTF), which is included in the Virtual Environment for Reactor Applications (VERA), and the systems code Transient Simulation Framework of Reconfigurable Models (TRANSFORM) was developed.

- An extensible FORTRAN interface to FMUs was developed and incorporated into the open-source Futility software library as part of this work.

- FMI coupling with the CTF code to perform steady-state and transient simulations between a sub-channel thermal-hydraulic CTF model of the Molten Salt Reactor Experiment (MSRE) core and the TRANSFORM model of the secondary system is used as a demonstration test case.

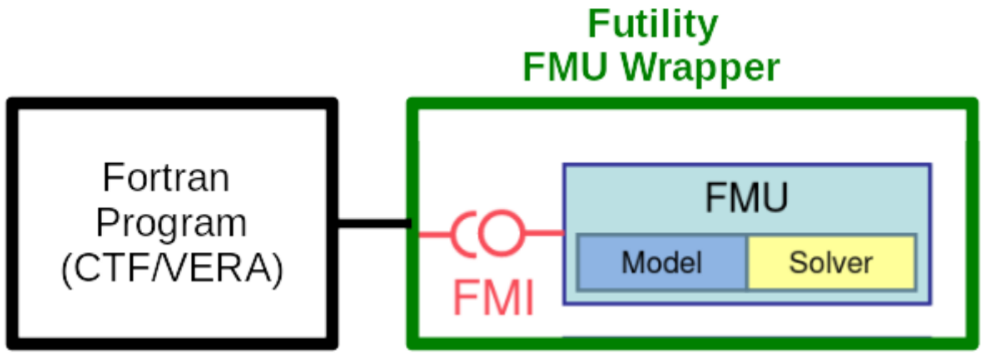
VERA/CTF core model and TRANSFORM FMU





FMU Wrapper in Futility

- Allows interaction with FMUs from a FORTRAN program.
- Parses FMU model description XML to obtain variable names, variable intents, and information on supported FMU capabilities.
- Provides model restart/rewind capability provided the loaded FMU supports this capability.
- Similar to and inspired by existing python (FMPy) and C++ (FMIKit) FMU wrappers. The FMU Wrapper in Futility represents a first-of-kind FORTRAN FMU wrapper capability.
- Futility[1] is an open-source FORTRAN library jointly developed at ORNL and the University of Michigan.

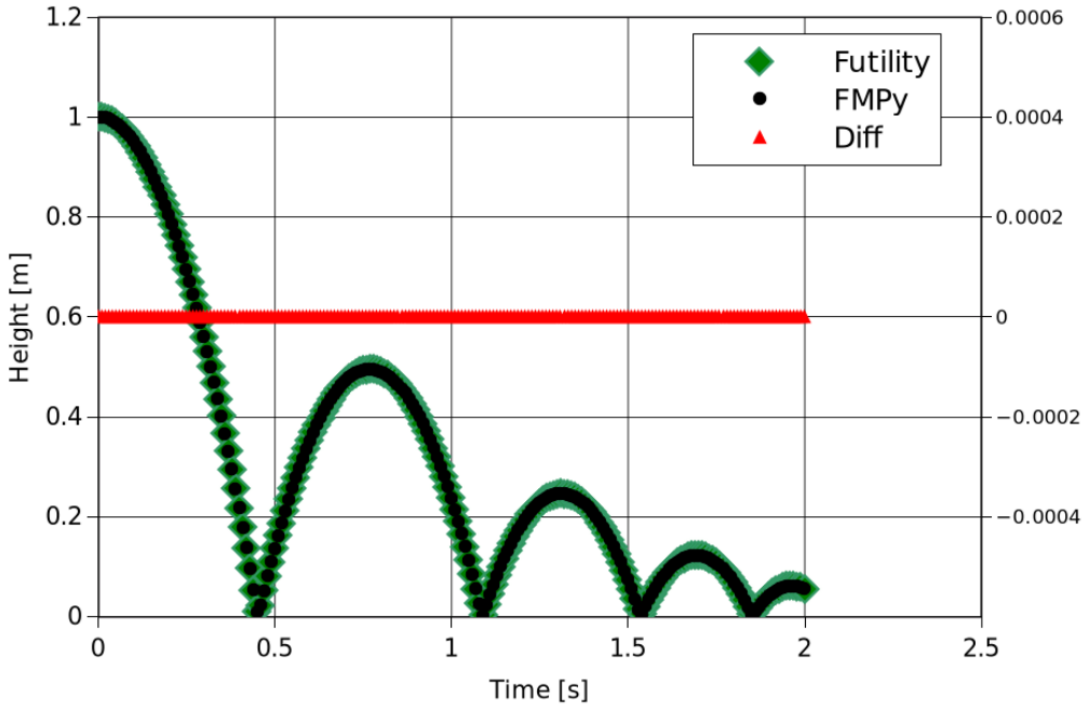


A standard, open source, C FMI implementation is wrapped using ISO C BINDINGS in Futility. The C FMI code implements the white paper standard FMI, as dictated by the official Modelica Association Project documentation. The Futility FMU Wrapper is a thin wrapper around the standard C FMI implementation supplied by the Modelica Software Foundation with some quality of life improvements.

[1] *Futility Development Group, "Futility: FORTRAN Utility,"* https://github.com/CASL/Futility

FMU Wrapper Example & FMPy Comparison

```
USE FMU_Wrapper
TYPE(FMU2_Slave) :: test_fmu2_cs
TYPE(ParamType) :: FMU_params
...
CALL FMU_params%clear()
CALL FMU_params%add('FMU_Wrapper->id',fmu_id)
unzipDirectory='/home/user/exampleFMU/reference_fmu_bouncing_ball'
CALL FMU_params%add('FMU_Wrapper->unzipDirectory', trim(unzipDirectory))
! Initilize the FMU
CALL test_fmu2_cs%init(fmu_id, FMU_params)
CALL test_fmu2_cs%setupExperiment(.TRUE., tol, timeStart, .TRUE., timeEnd)
CALL test_fmu2_cs%setNamedVariable('g', -9.81_SRK)
CALL test_fmu2_cs%setNamedVariable('e', 0.7_SRK)
CALL test_fmu2_cs%setRestart()
WRITE(*,*) "time[s]    height[m]    velocity[m/s]"
DO
  CALL test_fmu2_cs%getNamedVariable('v', ball_velocity)
  CALL test_fmu2_cs%getNamedVariable('h', ball_height)
  WRITE(*,*) time, ball_height, ball_velocity
  CALL test_fmu2_cs%doStep(dt)
  ...
ENDDO
...
```



Futility FMU Wrapper compared to FMPy for same pre-compiled, third-party, Bouncing Ball FMU[2] . 1m initial height. 0 initial velocity, coefficient of restitution of 0.7.

[2] https://github.com/modelica/fmi-cross-check/tree/master/fmus/2.0/cs/linux64/Test-FMUs/0.0.2/BouncingBall

XML coupling specification file

- Coupled CTF-TRANSFORM requires a CTF input deck, an FMU zip archive extracted to the running directory, and an XML coupling specifications file.
- The coupling specifications file specifies names of boundary condition variables exchanged, FMU parameters and solver properties.

```
************************************************************************************
*GROUP 1 - Calculation Variables and Initial Conditions
************************************************************************************
**NGR
   1
**NGAS IRFC EDMD IMIX ISOL           GINIT NOTRN MESH MAPS IPRP MFLX IBTM PPV   NM14
     1    2    0    3    3    1.70995e+02     0    1    0    4    0    0   7      0
*Card 1.2
**              GTOT          AFLUX         DHFRAC
      170.99460        6.40020        0.99990
*Card 1.3
**              PREF              HIN            HGIN          VFRAC1          VFRAC2
        3.44738     -633.88889 288.4200000      1.0000000      0.9999000
*Card 1.4
**GTP(1)   VFRAC(3)  GTP(2) VFRAC(4)  GTP(3) VFRAC(5)  GTP(4) VFRAC(6)
     air     0.0001
*Card 1.5
     {fmu_unzip_directory}  "/home/user/FMUS/Transform/Transform_V3/FMU_Unzipdir"
************************************************************************************
```

CTF Input File

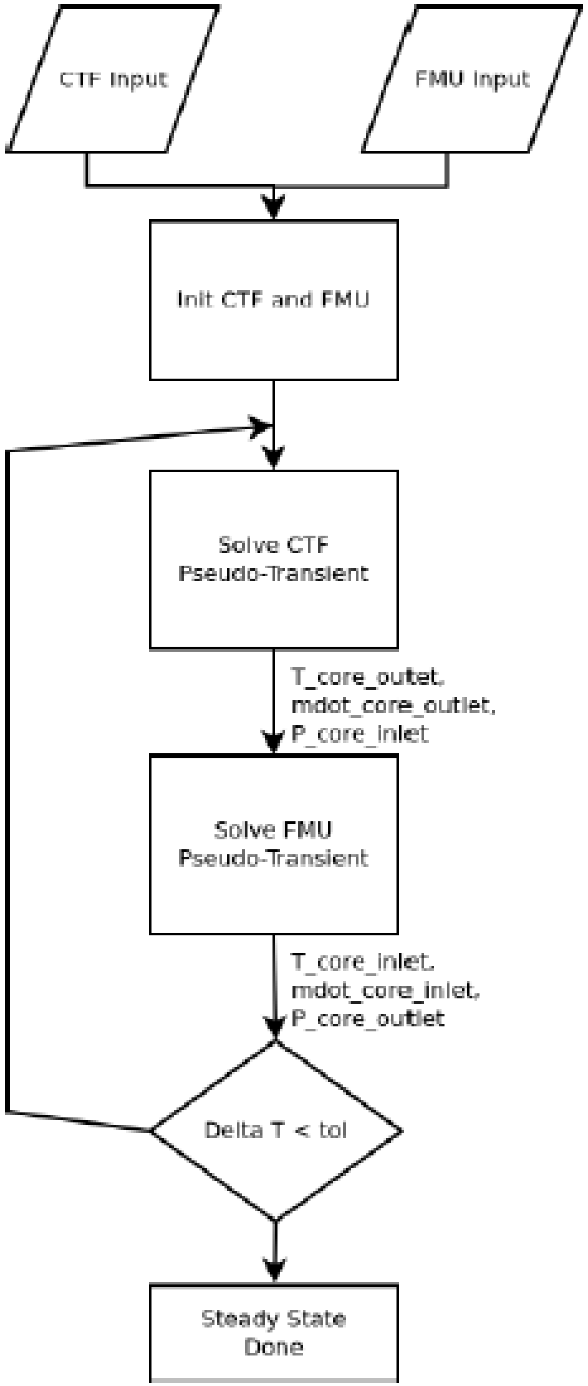
FMU XML coupling specifications File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ParameterList name="CASEID">
    <Parameter name="toltemp" type="double" value="0.01"/>
    <Parameter name="tolmf" type="double" value="2.0"/>
    <Parameter name="tolpress" type="double" value="1.0e1"/>
    <Parameter name="toltemp_FMU" type="double" value="1.0e-4"/>
    <Parameter name="tolmf_FMU" type="double" value="1.0e-4"/>
    <Parameter name="tolpress_FMU" type="double" value="1.0e-4"/>
    <Parameter name="FMU_dt_max" type="double" value="2.0e-1"/>
    <Parameter name="ulax_T_corein" type="double" value="1.0"/>
    <Parameter name="ulax_P_coreout" type="double" value="1.0"/>
    <Parameter name="ulax_mflow_corein" type="double" value="1.0"/>
    <ParameterList name="FMU_VAR_INIT">
       <!-- Set FMU Parameters and inital values -->
       <Parameter name="P_in" type="double" value="101.33e3"/>
       <Parameter name="P_corein" type="double" value="101.33e3"/>
       <Parameter name="T_in" type="double" value="907.0"/>
       <Parameter name="mflow_in" type="double" value="171.0"/>
       <Parameter name="mflow_pumpprimary" type="double" value="171.0"/>
       <Parameter name="mflow_secondary" type="double" value="105.745"/>
    </ParameterList>
    <ParameterList name="BC_VAR_NAMES">
       <!-- Parameter name= CTF_name     value= FMU_name -->
       <Parameter name="T_corein" type="string" value="T_out"/>
       <Parameter name="T_coreout" type="string" value="T_in"/>
       <Parameter name="P_corein" type="string" value="P_corein"/>
       <Parameter name="P_coreout" type="string" value="P_in"/>
       <Parameter name="mflow_corein" type="string" value="mflow_out"/>
       <Parameter name="mflow_coreout" type="string" value="mflow_in"/>
       <Parameter name="mflow_pumpprimary" type="string" value="mflow_pumpprimary"/>
    </ParameterList>
    <ParameterList name="FMU_VAR_TRANSIENT">
       <!-- FMU vars that vary as a fn of time -->
       <Parameter name="time" type="Array(double)" value="{0,10,100}"/>
       <Parameter name="mflow_secondary" type="Array(double)" value="{80,90,105.7}"/>
    </ParameterList>
    <ParameterList name="FMU_VAR_LOG">
       <!-- FMU Variables to log to file -->
       <Parameter name="mflow_pumpprimary" type="bool" value="true"/>
       <Parameter name="mflow_secondary" type="bool" value="true"/>
       <Parameter name="T_in" type="bool" value="true"/>
       <Parameter name="T_out" type="bool" value="true"/>
    </ParameterList>
```

CTF-FMU Steady-State Coupling

Initialization
(1) Set maximum number of outer iterations, N.
(2) Set under relaxation factors, $\omega \in (0, 1]$. Default $\omega = 1$.
(3) Set outer loop convergence tolerance. Default $\varepsilon \cong 0.25$K.
(4) Supply initial guess for $\mathrm{x}_0 = \{T_{0,in}, \dot{m}_{0,in}, P_{0,out}, ...\}$
(5) Initialize CTF and FMU from input
for Outer step: i in $\{0, ...N\}$ **do**
    Execute a pseudo-transient CTF computation, given: x_i:
    $\tilde{\mathrm{x}}_{i+1} \leftarrow \mathscr{G}_{CTF}(\mathrm{x}_i, \theta_{CTF})$
    Execute a pseudo-transient FMU computation:
    $\hat{\mathrm{x}}_{i+1} \leftarrow \mathscr{F}_{FMU}(\tilde{\mathrm{x}}_{i+1}, \theta_{FMU})$
    Update the state vector with under relaxation
    $\mathrm{x}_{i+1} = \omega\hat{\mathrm{x}}_{i+1} + (1 - \omega)\hat{\mathrm{x}}_i$
    if $|\mathrm{x}_{i+1} - \mathrm{x}_i| < \varepsilon$ **then**
        Break
    end if
end for

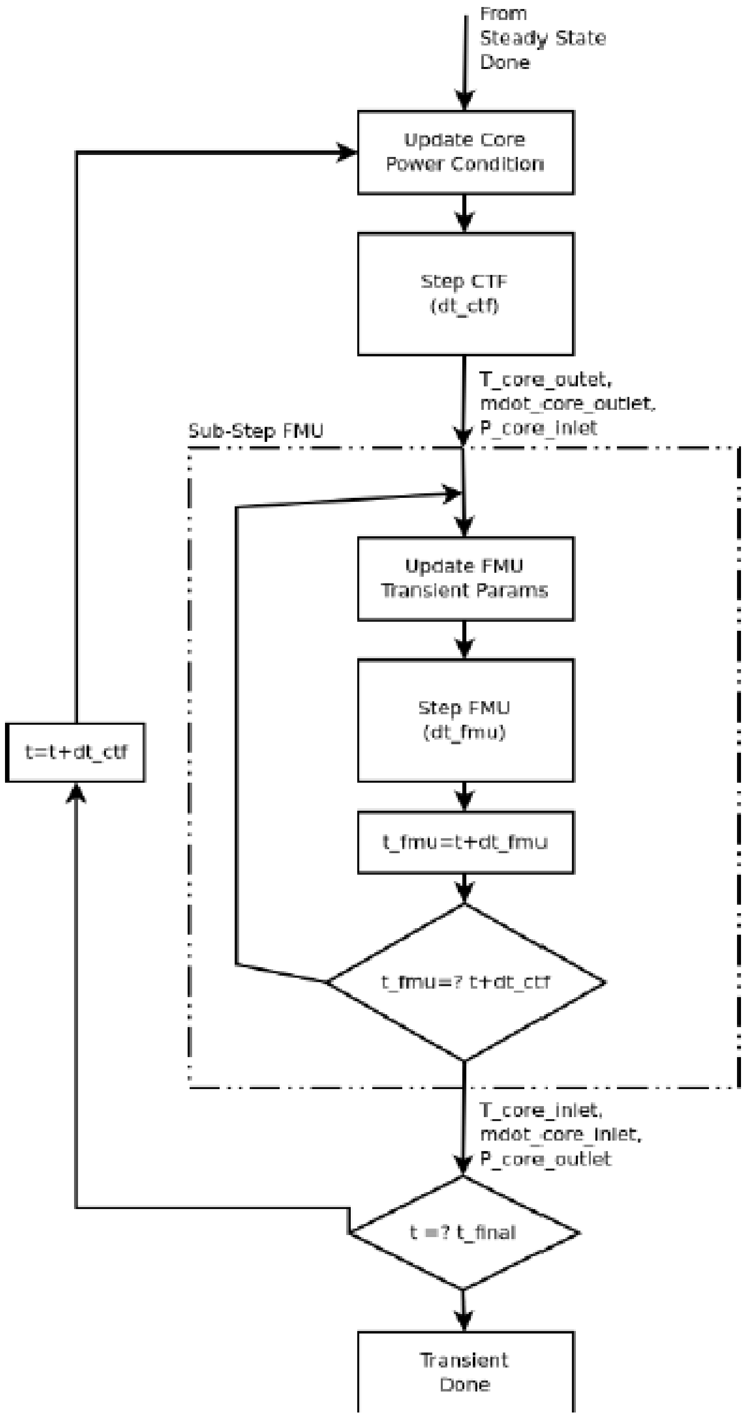
- Alternating pseudo-transient calculations of the core and system code.
- Core mass flow rates, temperatures and pressures exchanged at the boundaries.



CTF-FMU Transient Coupling

Initialization
(1) Set the CTF and FMU time steps, dt.
(2) Supply initial guess for $x_0 = \{T_{0,in}, \dot{m}_{0,in}, P_{0,out}, ...\}$.
(3) Initialize CTF and FMU from input.
(4) Perform an initial steady-state calculation (based on Listing 1).
while $t_{CTF} <= T$ **do**
  Set the FMU solution time to previous CTF time for sub-stepping:
  $t_{FMU} = t_{CTF}$
  Execute a transient CTF computation, given: $x_{t_{CTF}}$:
  $\tilde{x}_{t+dt} \leftarrow \mathscr{G}_{CTF}(x_{t_{CTF}}, \theta_{CTF})$
  Step the transient time step of the CTF solve:
  $t_{CTF} = t_{CTF} + dt_{CTF}$
  while $t_{FMU} <= t_{CTF}$ **do**
    Execute a transient FMU computation using the interpolated transient CTF solution
    $x_{t+dt} \leftarrow \mathscr{F}_{FMU}(\tilde{x}_{t_{FMU}+dt_{FMU}}, \theta_{FMU})$
    Step the transient time step of the FMU solve:
    $t_{FMU} = t_{FMU} + dt_{FMU}$
  end while
end while

- Initial steady-state solution performed before transient time-marching.
- FMU timestep taken to be smaller than CTF timestep (which is CFL dependent) for stability.

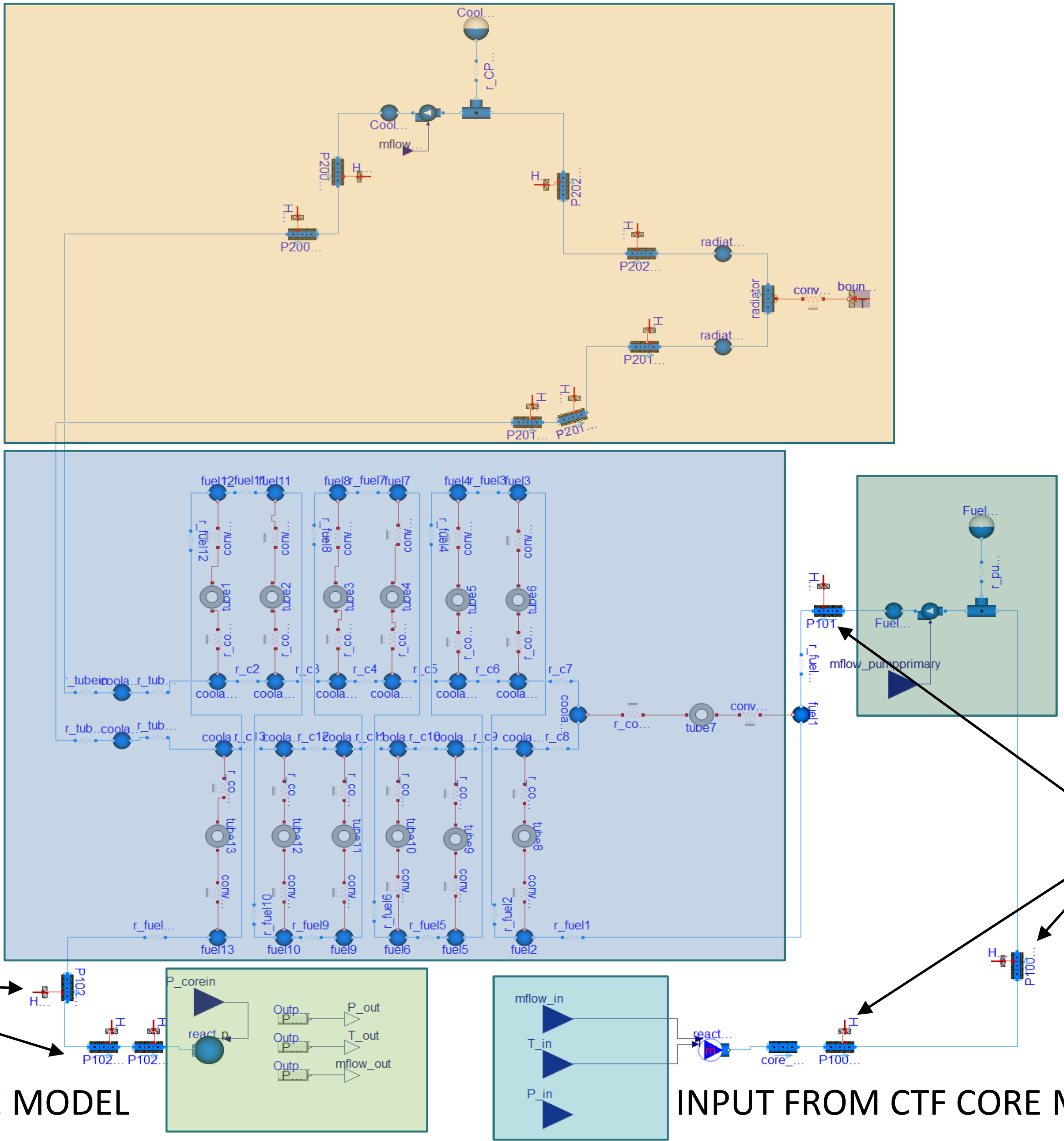


MSRE Secondary Loop : TRANSFORM Model

SECONDARY COOLING LOOP

PRIMARY/ SECONDARY HEAT EXCHANGER

PRIMARY COOLANT FEED PIPING

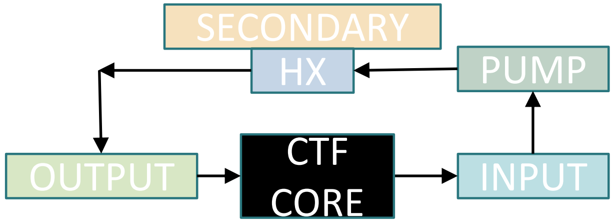


PRIMARY COOLANT PUMP

PRIMARY COOLANT RETURN PIPING

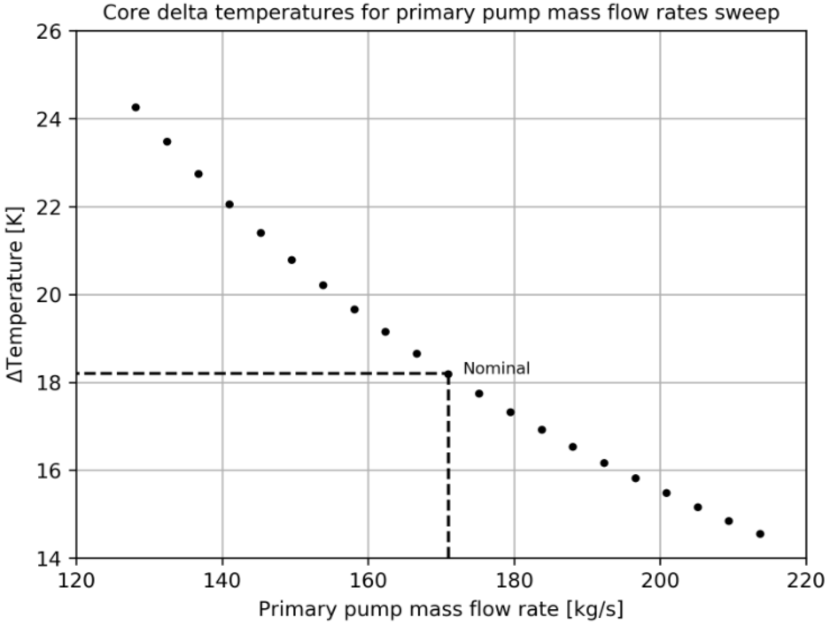
OUTPUT TO CTF CORE MODEL

INPUT FROM CTF CORE MODEL



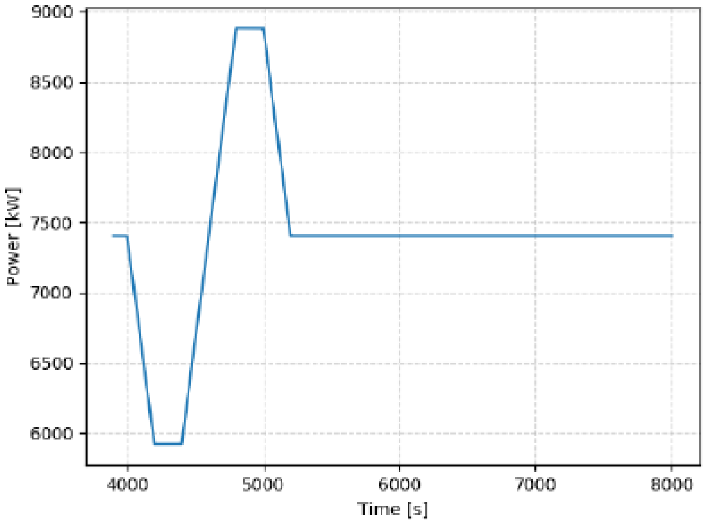
CTF-FMU Coupling results

Coupled Steady-state cases

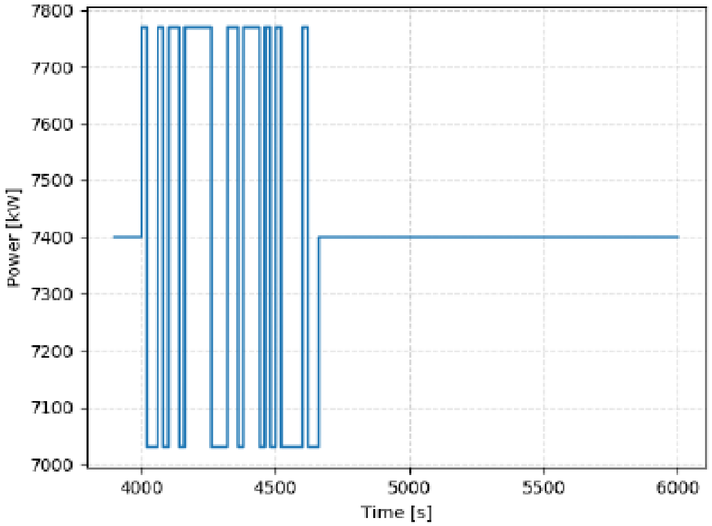


- The predicted temperature change across the core at nominal conditions is 18.2 K.
- This value is near the value reported in the MSRE (ΔT = 17.8 K) at nominal operating conditions.

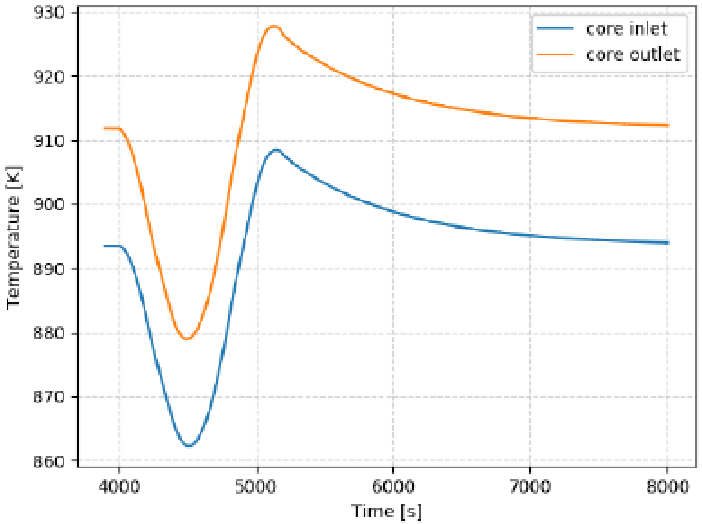
Coupled Transient cases



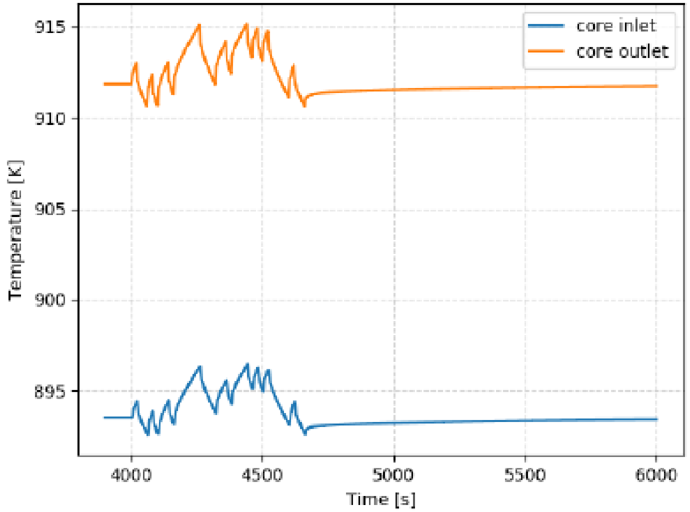
(a) Power vs. time.



(a) Power vs. time.

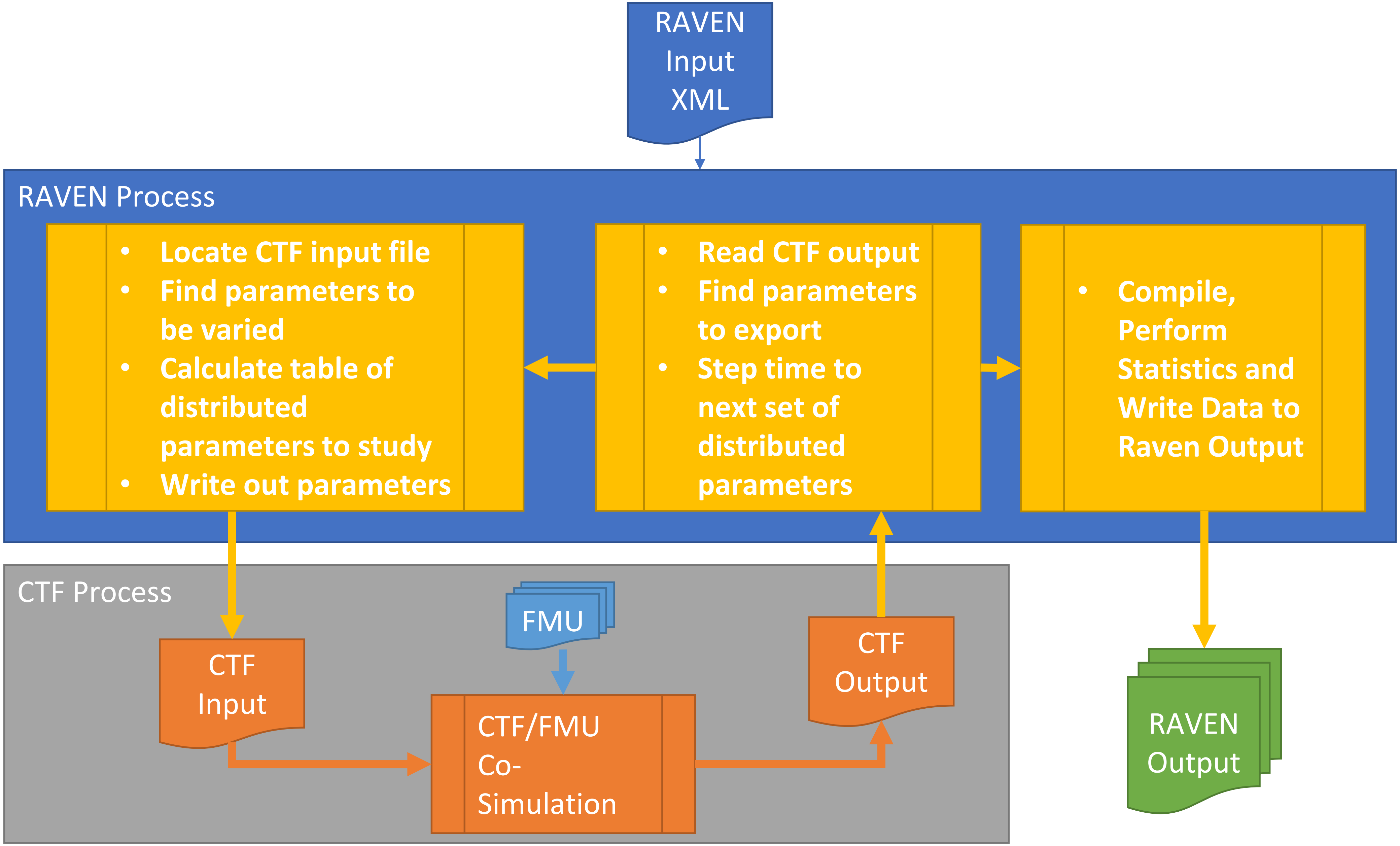


(b) Core inlet and outlet temperatures.



(b) Core inlet and outlet temperatures.

RAVEN – CTF – FMU Workflow



RAVEN – CTF – FMU Workflow

- Two approaches pursued: Futility-based in-memory coupling and external FMPy Python-based coupling.
- Using FMIPy for a looser coupling between CTF and FMU.
- The existing CTF code interface can accomplish most of the above steps by using
  - a ‘**GenericCode**’ interface to look for parameters to be varied in the CTF input files passed from the RAVEN framework
  - Writing new input files with the substituted variables from the ‘GenericCode’ interface
  - Writing a Comma-Separated Value (CSV)–based RAVEN output file by reading the generated standard CTF output file for each set of sampled variable(s).

Modified RAVEN input parser for CTF

```
*******************************************************************************
*GROUP 1 - Calculation Variables and Initial Conditions
*******************************************************************************
**NGR
   1
**NGAS IRFC EDMD IMIX ISOL          GINIT NOTRN MESH MAPS IPRP MFLX IBTM PPV  NM14
     1    2    0    3    0    171.0*$RAVEN-priMF$+171.0     1    1    0    4    0    0
       7     0
*Card 1.2
**          GTOT           AFLUX          DHFRAC
        171.0*$RAVEN-priMF$+171.0        6.40020        0.99990
*Card 1.3
**          PREF             HIN            HGIN          VFRAC1          VFRAC2
          3.44738      -633.88889     288.4200000      1.0000000      0.9999000
```

RAVEN HDF5 Reader for CTF Output

```
class ctfdataHDF5:
  """
    Class that parses CTF output file and reads in (output files type: .ctf.out) and
   write a csv file
  """
  def __init__(self, filein):
    """
      Constructor
      @ In, filen, string, file name to be parsed
      @ Out, None
    """
    # check file existence
    if ("ctf.native.h5" not in filein):
      raise IOError(
            "Check if the supported hdf5 output file (*.ctf.native.h5) is included.")

    self.majorData, self.headerName = self.getData(filein)

  def returnData(self):
    """
      Method to return the data in a dictionary
```


Oak Ridge National Laboratory

RAVEN CTF-FMU In-memory coupling Input file

```
<RunInfo>
  <WorkingDir>Testfmu1</WorkingDir>
  <Sequence>testRun</Sequence>
  <batchSize>25</batchSize>
  <NumThreads>1</NumThreads>
  <expectedTime>1:00:00</expectedTime>
  <CustomMode class="MPIEXECSimulationMode" file="%BASE_WORKING_DIR%/mpi_custom.py">
 mpicust</CustomMode>
  <mode>mpicust</mode>
</RunInfo>

<Files>
  <Input name="cobra_input" type="ctf" >system_coupling_transform.inp</Input>
  <Input name="thermophy" type="thermophy" >thermophysical_properties.dat</Input>
  <Input name="fmu_param" type="fmu_param" >fmu_param.xml</Input>
</Files>

<Models>
  <Code name="MyCobraTF" subType="CTF">
    <executable>
      "/home/vk8/build/install/bin/cobratf"
    </executable>
    <csv>True</csv>
  </Code>
```

RAVEN CTF Coupling Interface

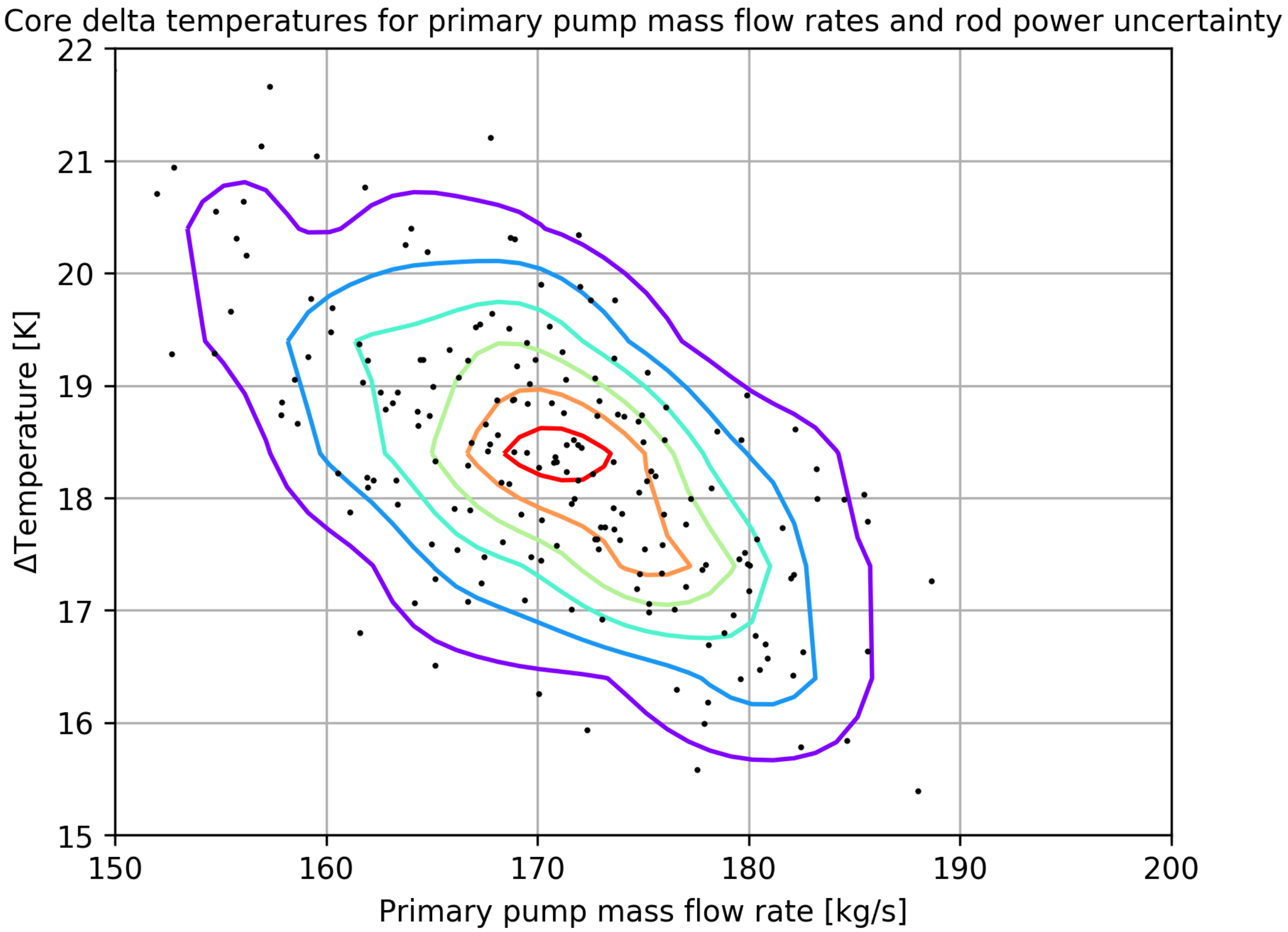
RAVEN CTF-FMU External Coupling (FMPy) input file

```
<RunInfo>
  <WorkingDir>Testfmu1</WorkingDir>
  <Sequence>testRun</Sequence>
  <batchSize>1</batchSize>
</RunInfo>

<Files>
  <Input name="fmuCTF.py" type="">fmuCTF.py</Input>
  <Input name="fmuCTFCouple.py" type="">fmuCTFCouple.py</Input>
  <Input name="make_deck.py" type="subKit">make_deck.py</Input>
  <Input name="thermophysical_properties.dat" type="" >thermophysical_properties.dat
  </Input>
  <Input name="fmu_param.xml" type="" >fmu_param.xml</Input>
</Files>

<Models>
  <ExternalModel ModuleToLoad="fmuCTF" name="PythonModule" subType="">
    <variables>HEAT,AVG_ax21_chan_temp</variables>
    <executable>
      /home/vk8/build/install/bin/cobratf
    </executable>
    <fmuDir>transform_fmus</fmuDir>
  </ExternalModel>
</Models>
```

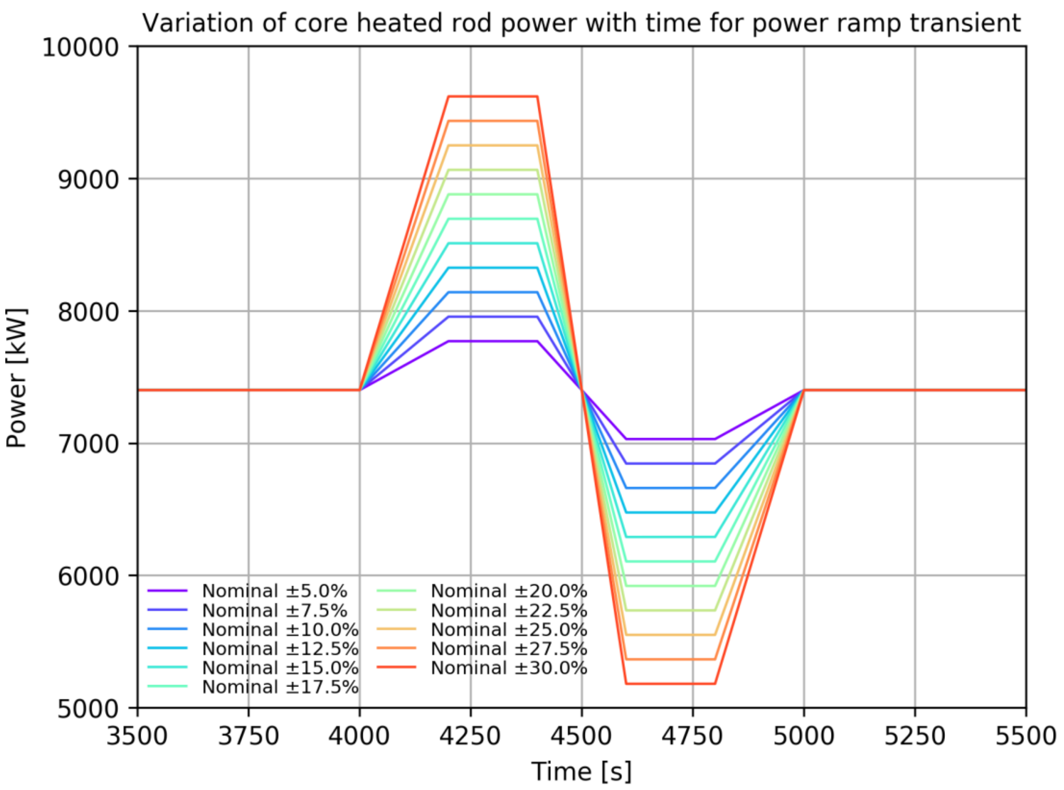
RAVEN CTF External model Interface

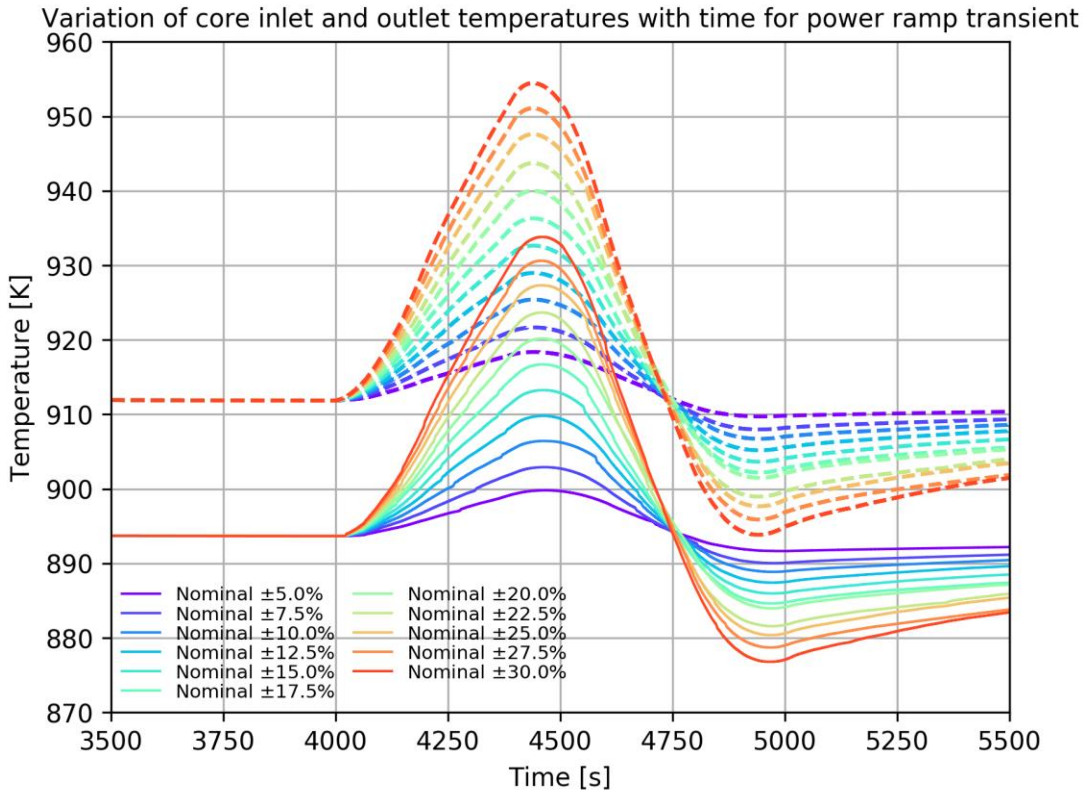
Reactor Power & Pri. Pump mass flow rate variation ± 20%

```
<Distributions>
  <Normal name="dist_for_powerFrac">
    <mean>0.0</mean>
    <sigma>0.05</sigma>
    <upperBound>0.2</upperBound>
    <lowerBound>-0.2</lowerBound>
  </Normal>
  <Normal name="dist_for_priMF">
    <mean>0.0</mean>
    <sigma>0.05</sigma>
    <upperBound>0.2</upperBound>
    <lowerBound>-0.2</lowerBound>
  </Normal>
</Distributions>

<Samplers >
  <MonteCarlo name="MC_samp">
    <samplerInit>
      <limit>200</limit>
    </samplerInit>
    <variable name="powerFrac">
      <distribution>dist_for_powerFrac</distribution>
    </variable>
    <variable name="priMF">
      <distribution>dist_for_priMF</distribution>
    </variable>
  </MonteCarlo>
</Samplers>
```



Transient power ramp sensitivity results

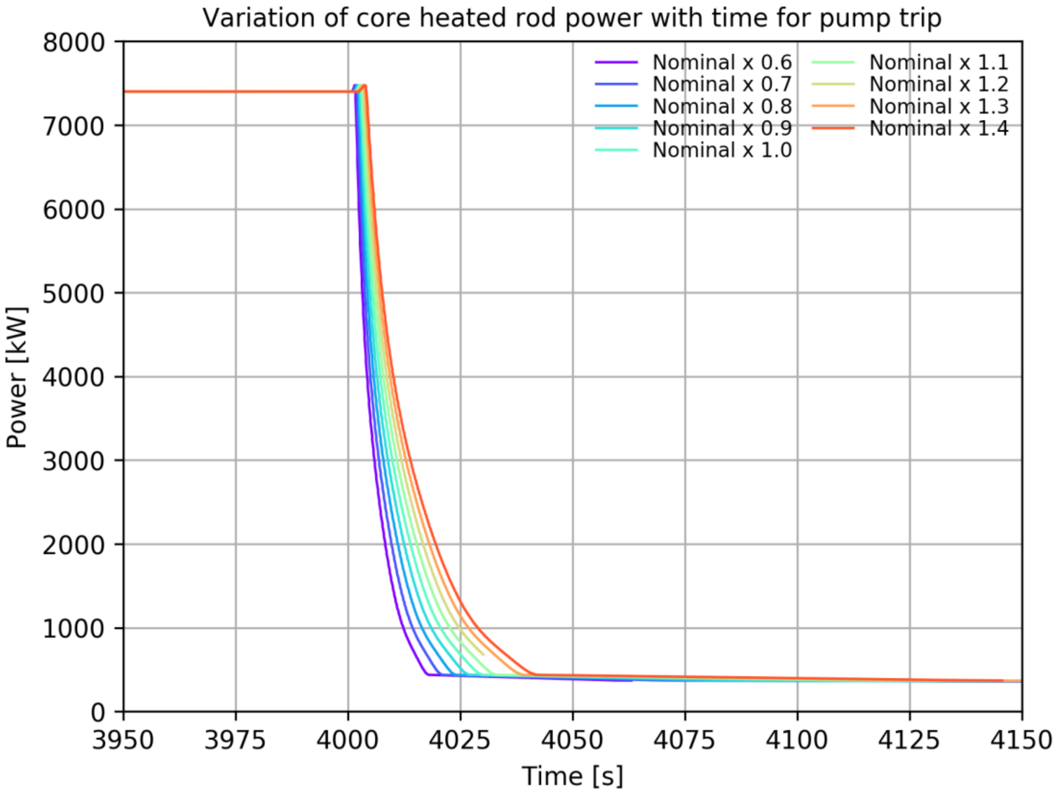


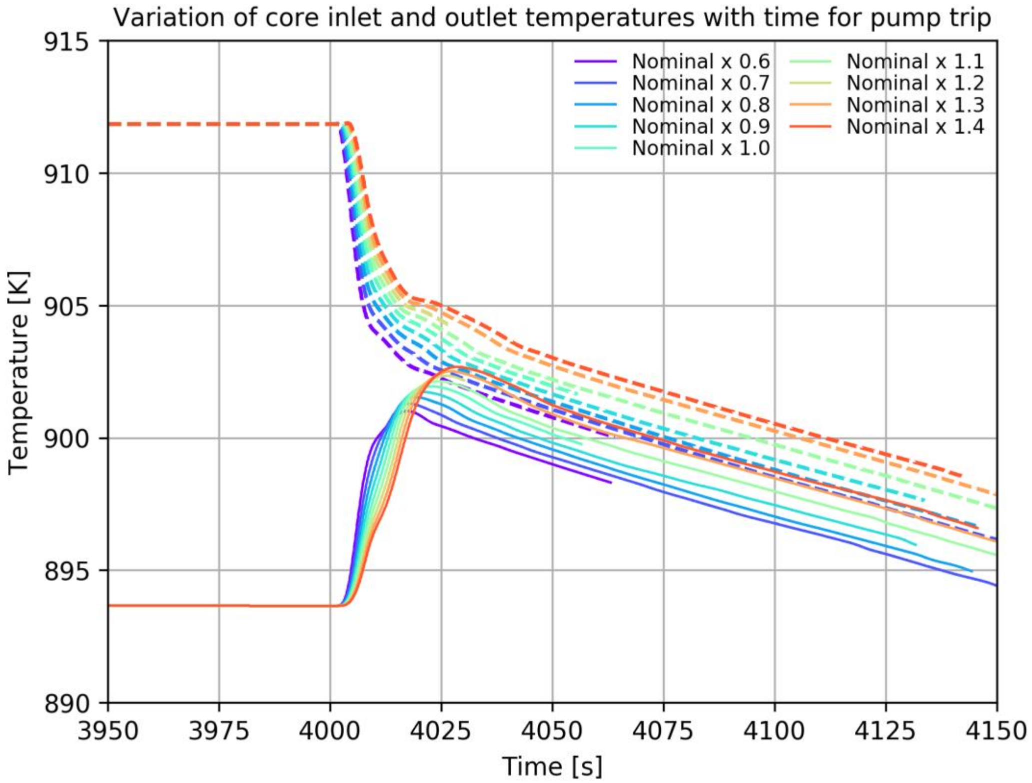


- A 200 s heated rod power ramp transient with mirror image rising and falling power profiles for one cycle.
- Utilizing RAVEN, the amplitude of the profiles were varied to study the system response.
- The system response is sharper for the initial ramp up than the ramp down. The CTF core model is a simplified model, and a more accurate model with heat loss calibration would provide a more accurate representation of the system's thermal inertia.

OAK RIDGE National Laboratory

Pump Trip sensitivity results



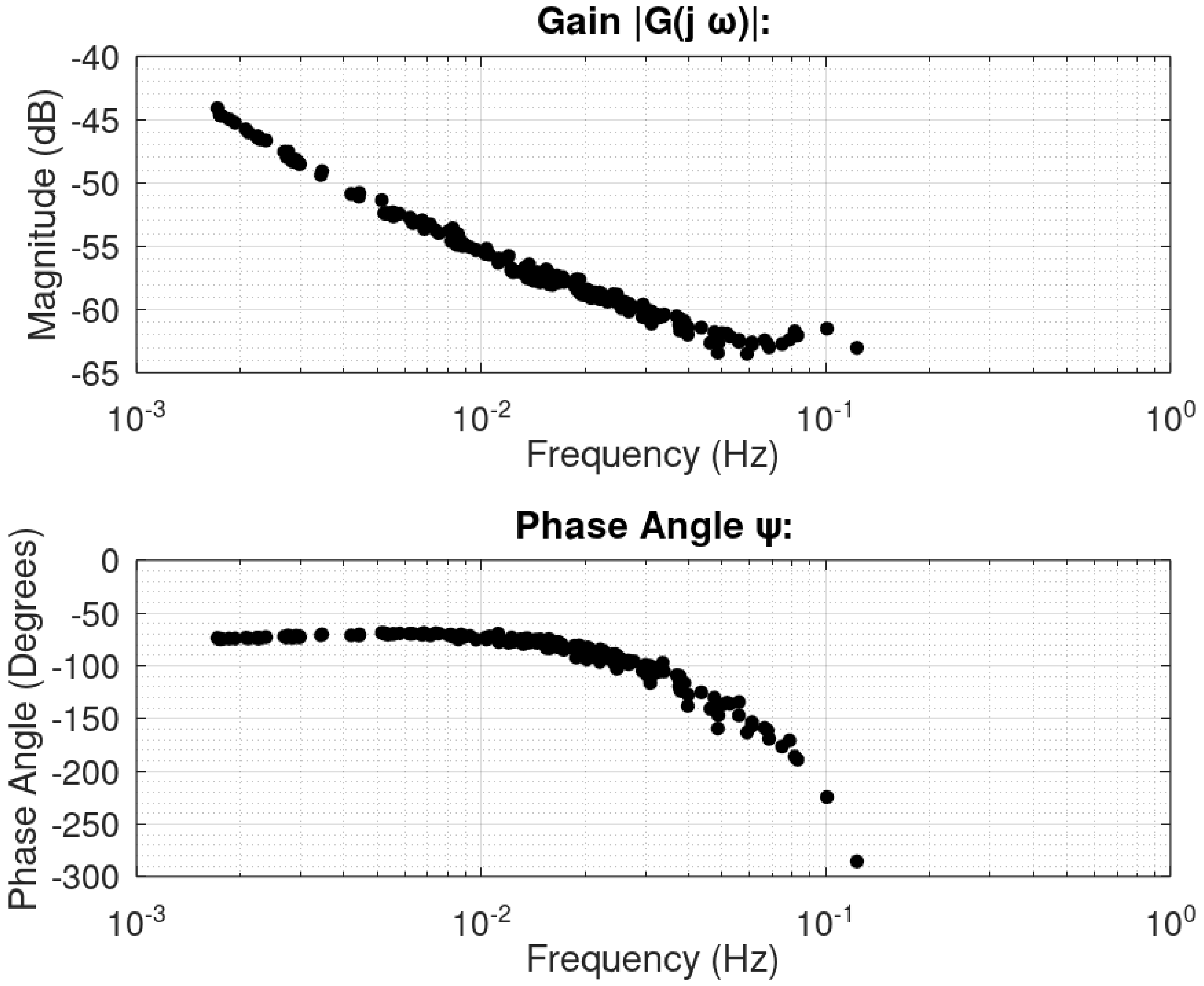


- A secondary loop transient with a consequent reduction in heated rod power, simulating a SCRAM event. The nominal pump coast-down rate was based on an MSRE technical report, and the coast-down rate was varied using RAVEN while preserving the power reduction curve.

- This type of study is useful in understanding the robustness of the coupled solver toward improving its convergence for some of these extreme conditions.

CTF-FMU Frequency analysis

- The RAVEN simulations ran sequences with different fundamental harmonics.
- The frequency domain data from all these tests were combined to provide a frequency-dependent description of the coupled system’s frequency response characteristics.
- This provides a concise description of the transient behavior of this coupling across a large frequency range.
- These analysis steps were performed externally to RAVEN. Incorporating these in RAVEN – readily available for quickly performing this type of characterization with many actuated inputs and measured outputs.



Oak Ridge National Laboratory

Conclusions

- This work demonstrated new functionality and applications that stemmed from the development of high-fidelity to low-fidelity (high-low) coupling for system simulations.
- A new Futility-based CTF-FMU coupling was developed within VERA and allows for the study of coupled system behavior in the reactor core and the supporting primary, secondary, and tertiary loop components.
- The Futility-based coupling and an external Python-based coupling were used in RAVEN for both steady-state and transient sensitivity studies.
- In-memory coupling of the codes was found to provide for much better computational performance in RAVEN. However, this may be due to the unoptimized usage of FMPy in Python. The additional need to exchange data in the out-of-memory implementations could make high-low coupling expensive.
- Future work will continue to explore and expand these capabilities with RAVEN and FMUs such as incorporating frequency analysis, coupling multiple FMUs and ROMs etc.

Thank You