Kraken workshop

Python level introduction to coupled calculations with Kraken using Cerberus

Ville Valtavirta

VTT

Outline

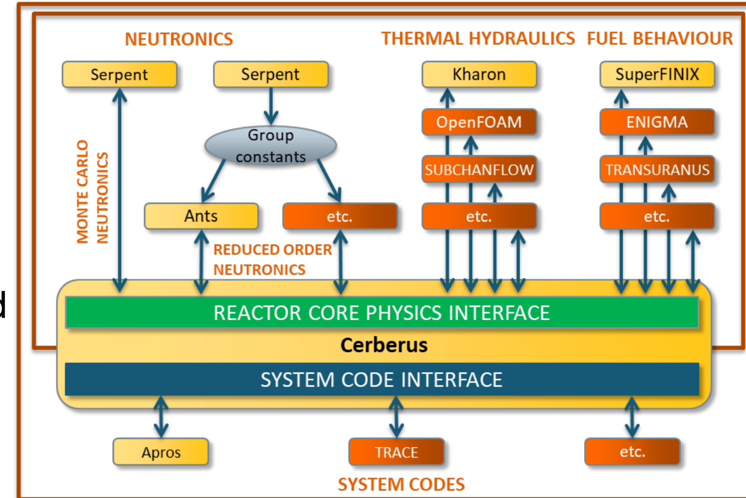
- Cerberus Python package:
  - Idea
  - Capabilities:
    - Solver()
    - Transferrable()
    - About Field()s and Mesh()es
    - Interpolator()
- Serpent and Cerberus
- Setting up coupled calculations with Cerberus:
  - Critical boron iteration.
  - Control rod iteration.
  - Transient simulations.
  - Burnup calculations.

Cerberus Python package

Cerberus Python package

- Code agnostic multi-physics driver of the Kraken framework.
- Provides high-level API for solvers, fields and variables on Python side.
- Python makes building coupled calculation schemes simple and fun.

- Cerberus aims to hide most of the boring and technical stuff from the user.
  - E.g. actual communications between processes, copying files and creating folders.
- Strikes a balance between simplicity and flexibility.
- Aimed for expert users, who can package common calculation sequences into further Python packages/modules for non-expert users.

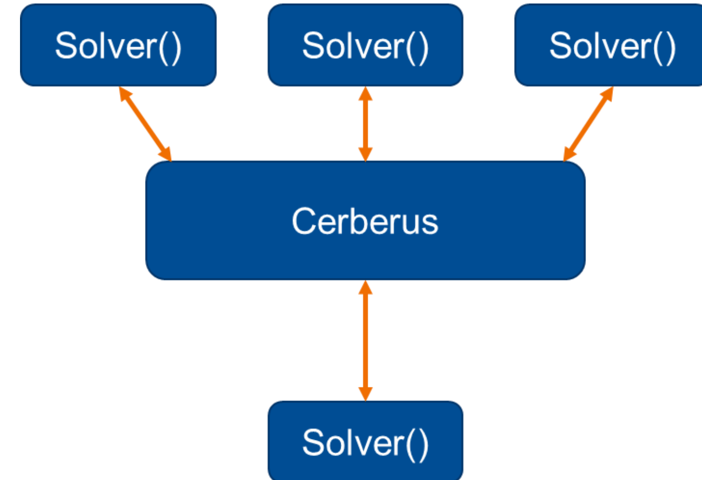


A schematic representation of the plans for the completed Kraken framework. Finnish solver modules developed at VTT are shown in yellow, while potential state-of-the-art third party solvers to be coupled are shown in orange.

Cerberus Python package

- Code agnostic multi-physics driver of the Kraken framework.
- Provides high-level API for solvers, fields and variables on Python side.
- Python makes building coupled calculation schemes simple and fun.

- Cerberus aims to hide most of the boring and technical stuff from the user.
  - E.g. actual communications between processes, copying files and creating folders.
- Strikes a balance between simplicity and flexibility.
- Aimed for expert users, who can package common calculation sequences into further Python packages/modules for non-expert users.



The Solver class of Cerberus

- All solvers participating in the calculation are based on the Solver class that provides methods such as
  - Solver.initialize()
  - Solver.get_transferrable()
  - Solver.solve()
  - Solver.set_current_time()
  - Solver.suggest_next_time()
  - Solver.move_to_time()
  - Solver.write_restart()
  - Solver.read_restart()
  - Etc.

The user does not need to know what happens “under the hood” when calling one of these methods from Python.

All solvers provide the same functionality to Python even if actual implementation in the solver module may differ.

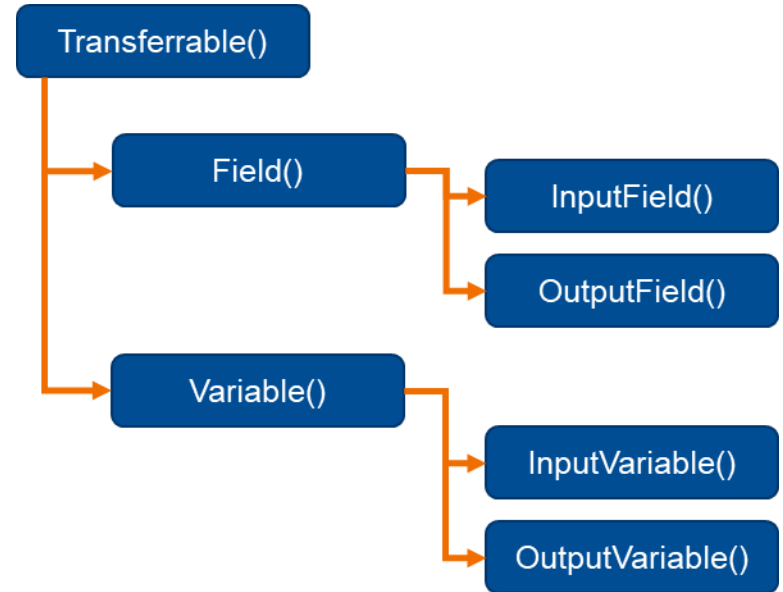
Cerberus does not know (or care) which solver handles neutronics and which thermal hydraulics.

The expert user, of course, does care.

VTT

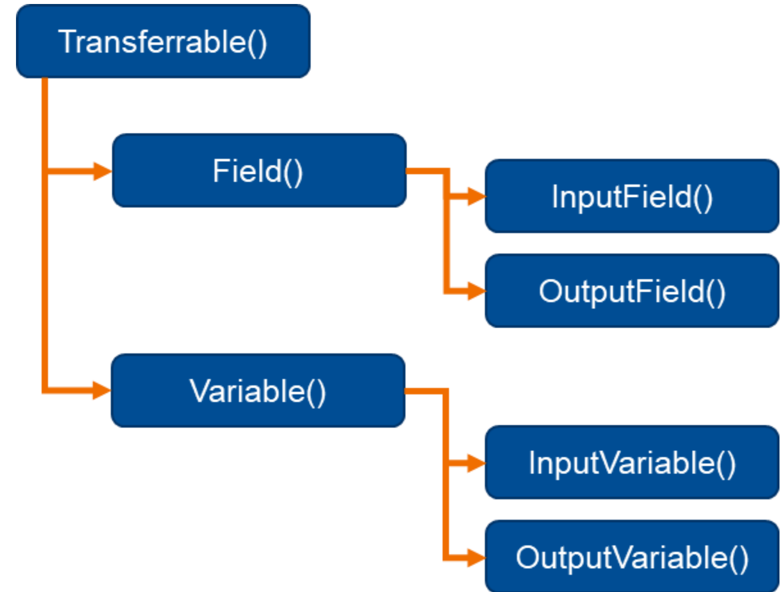
Transferrables in Cerberus

- The transferrable superclass covers all input and output data needs of the solver that can be transferred between the solver and Cerberus.
- Transferrable.communicate(): Exchange data with solver.
- Transferrable.write_simple(): Write data to file.
- Transferrable.write_foam(): Write data to Foam files.
- Transferrable.value_vec: Current values of data.
- Transferrable.get_conv_crit(): Evaluate convergence criterion between current and previous values.
- …



Transferrables in Cerberus

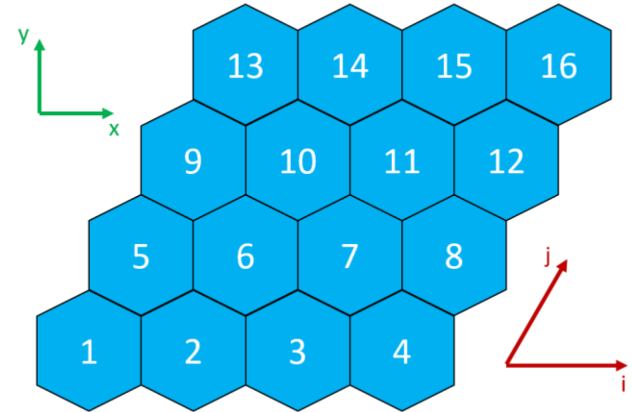
- Field() and Variable() are sub-classes of Transferrable().
- Field is a (physical) dataset with a spatial representation (mesh).
- Variable is a more general set/piece of data. Often single valued.



VTT

On Field()s and Mesh()es

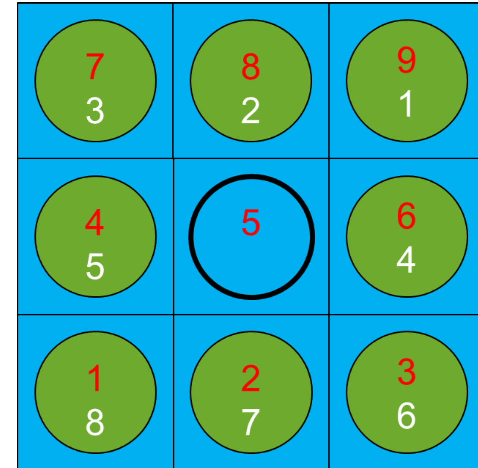
- Field data on Python side (as seen by the user) is in SI-units and uses a default (global) indexing for the mesh.
- Conversions between solver (local) and Cerberus (global) units and indexings are handled automatically by Cerberus based on data Cerberus obtains from the solver.
- Cerberus output can be written to files separately using the global and local indexings (typically only global indexing required).



Global indexing for mesh type 3: One axial layer of structured x-type 60 degree hexagonal mesh.

On Field()s and Mesh()es

- Meshes may be used for automatic generation of interpolations between fields in the future. At the moment, interpolations need to be pre-generated by the user.
- Mesh information is also written in files and can be re-created in postprocessing using krakentools.kraken.Mesh class, which offers some useful functionalities for plotting etc.
- Automated output to FoamFiles is partially supported with additional support added in the future.



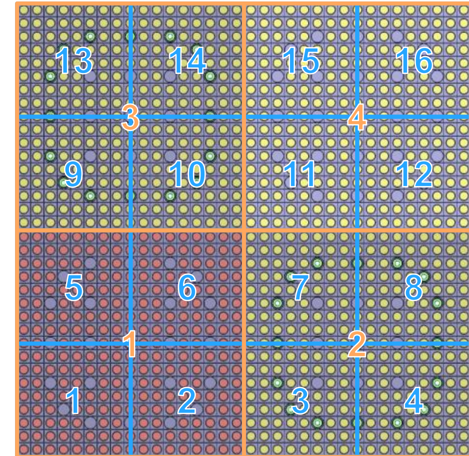
Cerberus will automatically handle conversions between solver indexing to global indexing to provide one uniform indexing scheme across all solvers and fields on the Python side.

The Interpolator() class

- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix $\bar{\bar{A}}$ that describes the production of the destination data field $\bar{d}$ from the source data field $\bar{s}$.

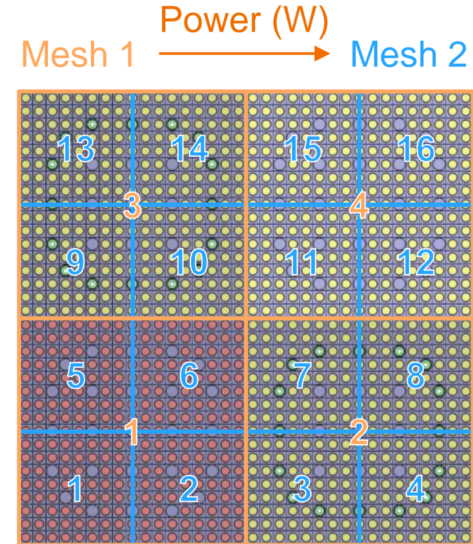
$$\bar{d} = \bar{\bar{A}}\bar{s}$$

$$\begin{pmatrix} d_1 \\ \vdots \\ d_{N_d} \end{pmatrix} = \begin{pmatrix} a_{1\to 1} & \cdots & a_{N_s\to 1} \\ \vdots & \ddots & \vdots \\ a_{1\to N_d} & \cdots & a_{N_s\to N_d} \end{pmatrix} \begin{pmatrix} s_1 \\ \vdots \\ s_{N_s} \end{pmatrix}$$



VTT

The Interpolator() class



- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix $\bar{\bar{A}}$ that describes the production of the destination data field $\bar{d}$ from the source data field $\bar{s}$.

$$\bar{d} = \bar{\bar{A}}\bar{s}$$

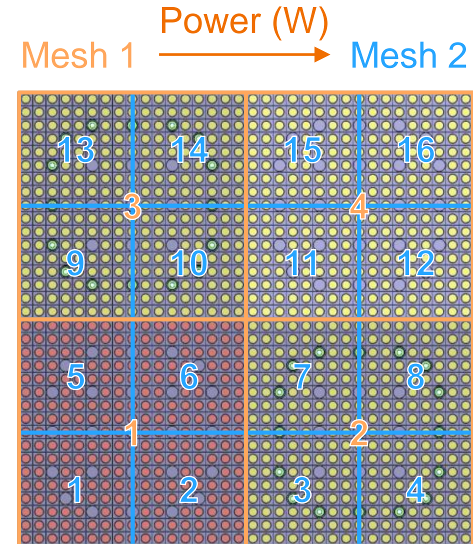
$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ \vdots \\ d_{16} \end{pmatrix} = \begin{pmatrix} a_{1\to1} & a_{2\to1} & a_{3\to1} & a_{4\to1} \\ a_{1\to2} & a_{2\to2} & a_{3\to2} & a_{4\to2} \\ a_{1\to3} & a_{2\to3} & a_{3\to3} & a_{4\to3} \\ a_{1\to4} & a_{2\to4} & a_{3\to4} & a_{4\to4} \\ a_{1\to5} & a_{2\to5} & a_{3\to5} & a_{4\to5} \\ a_{1\to6} & a_{2\to6} & a_{3\to6} & a_{4\to6} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1\to16} & a_{2\to16} & a_{3\to16} & a_{4\to16} \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$$

The Interpolator() class

- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix $\bar{\bar{A}}$ that describes the production of the destination data field $\bar{d}$ from the source data field $\bar{s}$.

$$\bar{d} = \bar{\bar{A}}\bar{s}$$

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ \vdots \\ d_{16} \end{pmatrix} = \begin{pmatrix} 0.25 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0.25 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$$



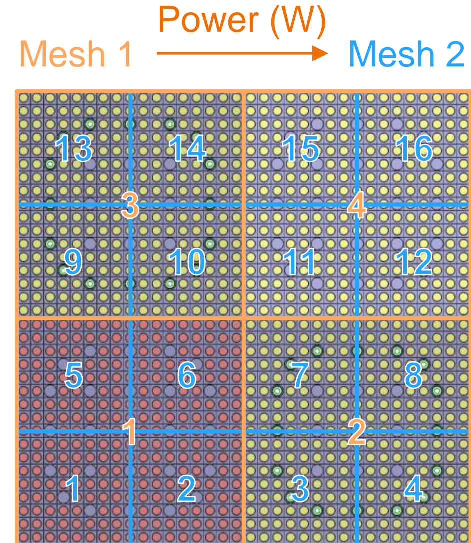
The Interpolator() class

- Handles data transfer between two Field()s using a chosen interpolation scheme:
  - One-to-one mapping.
  - User supplied interpolation matrix $\bar{\bar{A}}$ that describes the production of the destination data field $\bar{d}$ from the source data field $\bar{s}$.

$$\bar{d} = \bar{\bar{A}}\bar{s}$$

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ \vdots \\ d_{16} \end{pmatrix} = \begin{pmatrix} 0.25 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0.25 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$$

mapping.txt:
1 0.25
1 0.25
2 0.25
2 0.25
1 0.25
1 0.25
…
4 0.25



Serpent and Cerberus

VTT

VTT

Serpent and Cerberus

- Cerberus provides a (runtime) Python interface for Serpent allowing the user to:
  - Access Serpent results as output variables and fields.
  - Adjust Serpent model using input variables and fields.
  - Control Serpent solution flow (iterate current point / burn forward etc.)

- Examples of output variables:
  - All Serpent STAT variables (generally all tallied results such as IMP_KEFF, INF_FLX, B1_ABS, …)
    - (Except some material based variables, which all have the same name.)
  - Iterated boron concentration (if using “set iter nuc”).
  - Current normalization.

  - Material data:
    - Burnup
    - ZAI list
    - Nuclide densities
    - Domain
  - Pebble data
  - Memory consumption.

← More on these in presentation by Yves Robert

VTT

Serpent and Cerberus

- Cerberus provides a (runtime) Python interface for Serpent allowing the user to:
  - Access Serpent results as output variables and fields.
  - Adjust Serpent model using input variables and fields.
  - Control Serpent solution flow (iterate current point / burn forward etc.)

- Examples of input variables:
  - Values for transformation cards (for moving things).
  - Equilibrium xenon/samarium mode: Off, On, Fixed.
  - Boron concentration in coolant (and iteration of atomic densities)
  - Current normalization.
  - Neutron population, number of active cycles.

  - Burnup step type: Burn, decay, activation.
  - Material data:
    - Burnup
    - Nuclide densities
    - Domain
  - Pebble data.

← More on these in presentation by Yves Robert

Serpent and Cerberus

- Cerberus provides a (runtime) Python interface for Serpent allowing the user to:
  - Access Serpent results as output variables and fields.
  - Adjust Serpent model using input variables and fields.
  - Control Serpent solution flow (iterate current point / burn forward etc.)

- Input and output fields:
  - Temperature and density fields for multi-physics interfaces as input fields.
  - Related tallied power fields as output fields.

  - Equilibrium xenon and samarium fields as input and output fields.
    - Can set to zero at some point and then back to earlier values for e.g. xenon reactivity estimation.

Serpent and Cerberus

- Cerberus provides a (runtime) Python interface for Serpent allowing the user to:
  - Access Serpent results as output variables and fields.
  - Adjust Serpent model using input variables and fields.
  - Control Serpent solution flow (iterate current point / burn forward etc.)
- Cerberus is a powerful tool for using Serpent in complex tasks as simulation control and adjustments can be done already during runtime without need to restart Serpent afterwards.
- New input/output variables and fields are rather easy to add:
  - Expect the amount of accessible data to grow in the future.
- Cerberus may make existing Serpent power users into superpower users.
  - Ask Yves Robert for comments on first hand user experience.

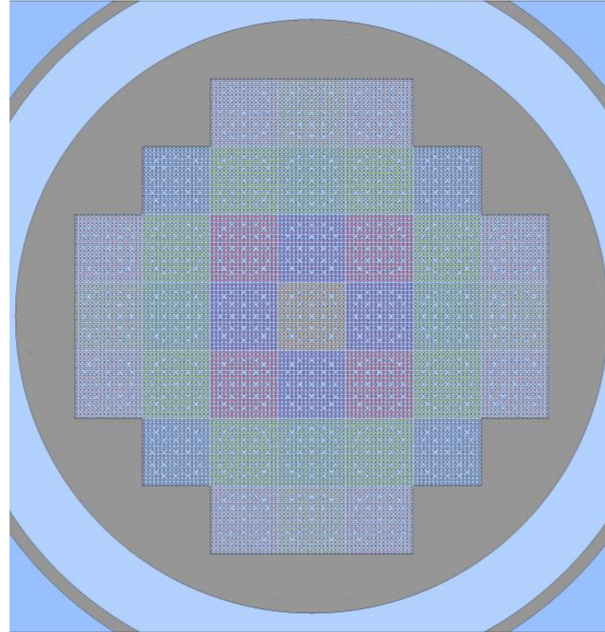
Setting up coupled calculations with Cerberus

VTT

Reactor model for investigation

- Early design for a district heating SMR core.
  - Low power (50 MW), low temperature (360 K inlet), low pressure (< 10 bar).
  - 37 fuel assemblies.
  - 150 cm active height.
- Simplified from original:
  - Removed spacer grids
  - Simplified control rods
  - Moved to boron based reactivity control
- Intended to test the simulator capabilities of Kraken.
- Details of analyses published in

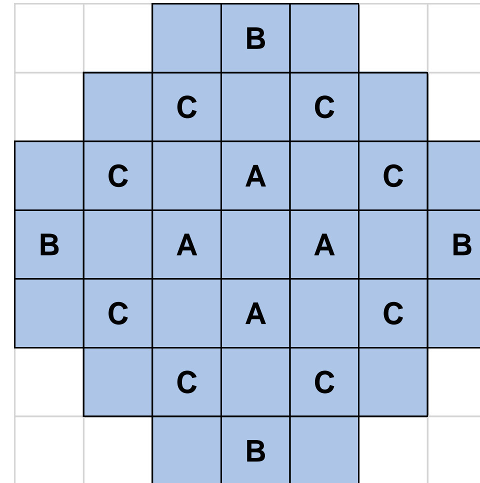
Valtavirta, V., Tuominen, R.
"A simple reactor core simulator based on VTT's Cerberus Python package"
ANS M&C 2021, April 11-15, 2021, Raleigh, NC



Reactor model for investigation

- Early design for a district heating SMR core.
  - Low power (50 MW), low temperature (360 K inlet), low pressure (< 10 bar).
  - 37 fuel assemblies.
  - 150 cm active height.
- Simplified from original:
  - Removed spacer grids
  - Simplified control rods
  - Moved to boron based reactivity control
- Intended to test the simulator capabilities of Kraken.
- Details of analyses published in

Valtavirta, V., Tuominen, R.
"A simple reactor core simulator based on VTT's Cerberus Python package"
ANS M&C 2021, April 11-15, 2021, Raleigh, NC



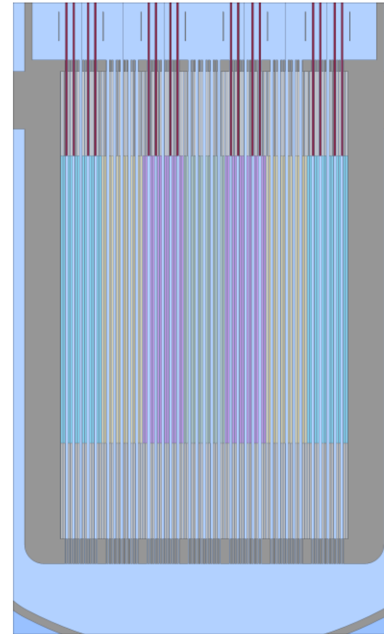
Control rod groups in the reactor core.

VTT

Reactor model for investigation

- Early design for a district heating SMR core.
  - Low power (50 MW), low temperature (360 K inlet), low pressure (< 10 bar).
  - 37 fuel assemblies.
  - 150 cm active height.
- Simplified from original:
  - Removed spacer grids
  - Simplified control rods
  - Moved to boron based reactivity control
- Intended to test the simulator capabilities of Kraken.
- Details of analyses published in

Valtavirta, V., Tuominen, R.
"A simple reactor core simulator based on VTT's Cerberus Python package"
ANS M&C 2021, April 11-15, 2021, Raleigh, NC

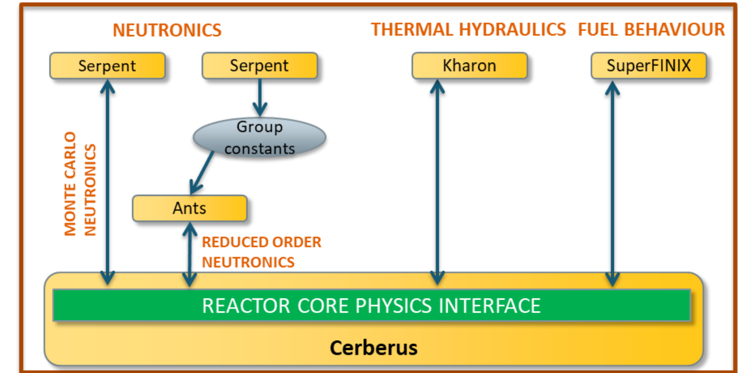


VTT

Applied calculation chains

Calculation setup:

- Neutronics (**Serpent** OR **Ants**):
  - Continuous energy Monte Carlo **OR**
  - Eight group nodal diffusion with ¼ assembly subnodalization.
- Thermal hydraulics (**Kharon**):
  - Porous medium closed channel single phase.
  - One channel per assembly.
- Fuel behaviour (**SuperFINIX**):
  - Traditional 1.5 dimensional approach.
  - One representative fuel rod per assembly.
- Coupled calculation:
  - Coupled fields exchanged at assembly level.

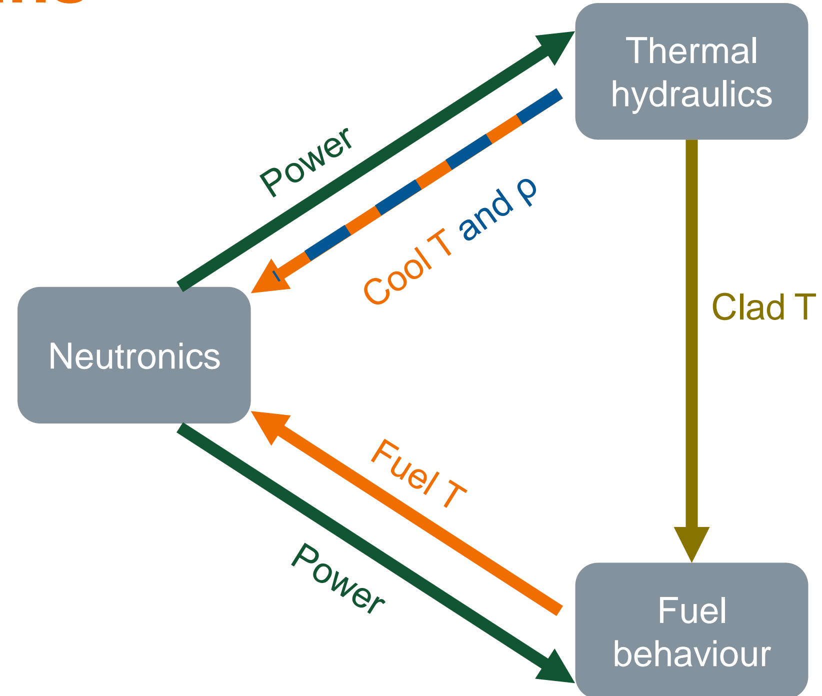


The Kraken based core simulator in this work used SuperFINIX for fuel behaviour, Kharon for thermal hydraulics and either Serpent or Ants for the neutronics.

Applied calculation chains

Calculation setup:

- Neutronics (**Serpent** OR **Ants**):
  - Continuous energy Monte Carlo **OR**
  - Eight group nodal diffusion with ¼ assembly subnodalization.
- Thermal hydraulics (**Kharon**):
  - Porous medium closed channel single phase.
  - One channel per assembly.
- Fuel behaviour (**SuperFINIX**):
  - Traditional 1.5 dimensional approach.
  - One representative fuel rod per assembly.
- Coupled calculation:
  - Coupled fields exchanged at assembly level.



VTT

Available inputs

- group_constants/new8g_noppr.xs
- inputs/
  - ants/
  - kharon/
  - serpent/
  - superfinix/
  - mappings/

Available inputs

- group_constants/new8g_noppr.xs
- inputs/
  - ants/
    - Ants8g.inp
    - includes/
      - control_rods.inc
      - core.inc
      - fuel_assemblies.inc
      - radial_reflector.inc
  - kharon/
  - serpent/
  - superfinix/
  - mappings/

Available inputs

- group_constants/new8g_noppr.xs
- inputs/
  - ants/
  - kharon/
    - Kharon.inp
  - serpent/
  - superfinix/
  - mappings/

Available inputs

- group_constants/new8g_noppr.xs
- inputs/
  - ants/
  - kharon/
  - serpent/
    - Serpent.inp
    - includes/
      - …
  - superfinix/
  - mappings/

Available inputs

- group_constants/new8g_noppr.xs
- inputs/
  - ants/
  - kharon/
  - serpent/
  - superfinix/
    - SuperFINIX.inp
    - finixfiles/
      - finix.options
      - finix.rods
      - finix.scenario
  - mappings/

VTT

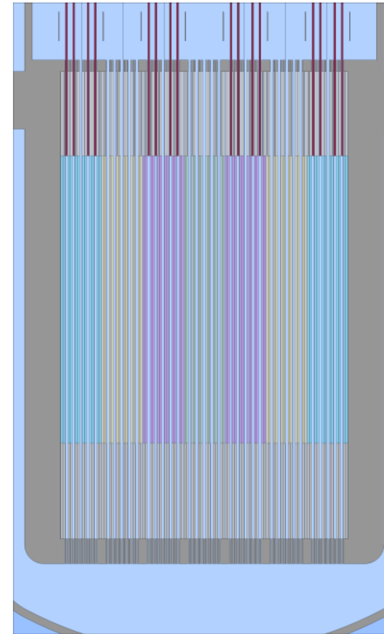
Available inputs

- group_constants/new8g_noppr.xs
- inputs/
  - ants/
  - kharon/
  - serpent/
  - superfinix/
  - mappings/
    - ants_to_kharon.txt
    - ants_to_sf.txt
    - kharon_to_ants.txt
    - kharon_to_sf.txt
    - sf_to_ants.txt
    - sf_to_neutronics.txt

First step: ARO HZP CBC

- All rods out (ARO) hot zero power (HZP) critical boron concentration.

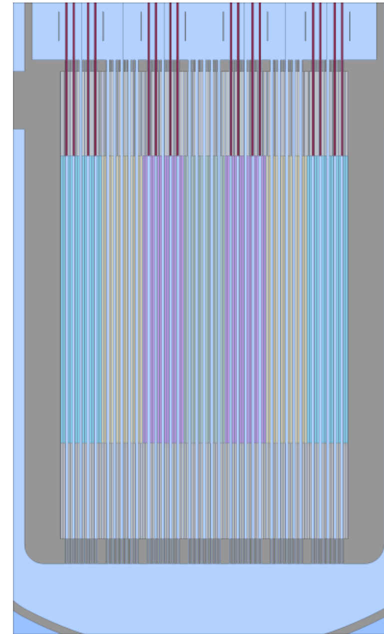
- Initialize neutronics solver and set boron iteration on.
- Set HZP state.
- Converge neutronics solution.
- Get and store results.



Second step: ARO HFP CBC

- All rods out (ARO) hot full power (HFP) critical boron concentration.

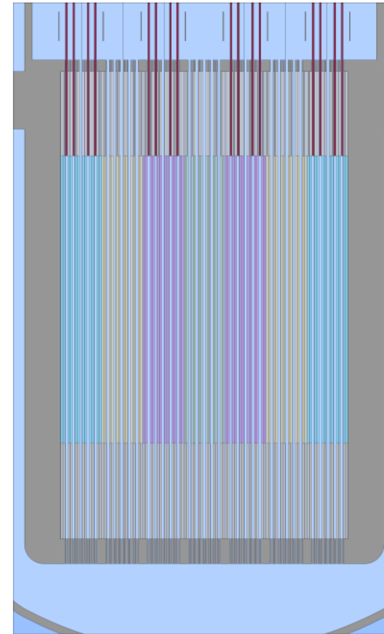
- Initialize required solvers.
- Supply initial guess for fields.
- Converge coupled solution.
- Get and store results.



VTT

Third step: ARO HFP CBC + control rod worths

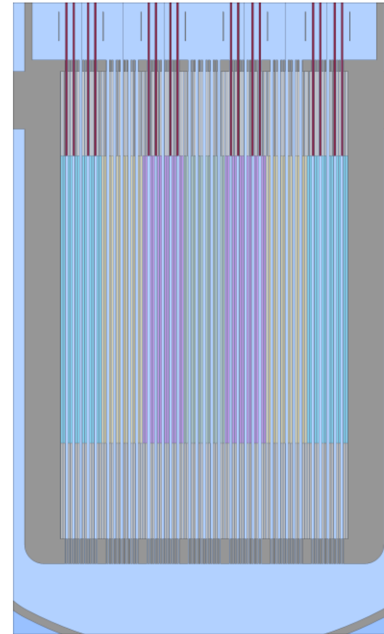
- Initialize required solvers.
- Supply initial guess for fields.
- Converge coupled solution.
- Evaluate control rod worths.



VTT

Final step: Coupled burnup calculation with full power (constant extrapolation)

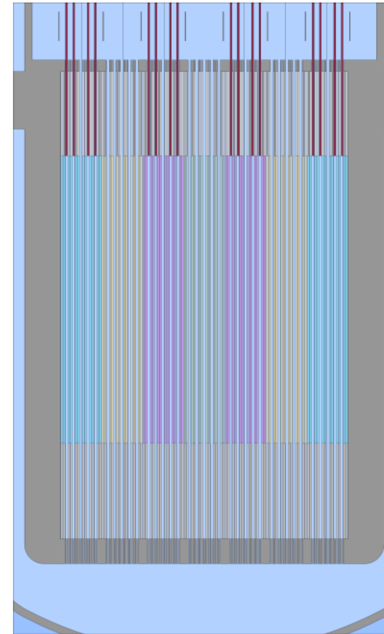
- Initialize required solvers.
- Supply initial guess for fields.
- For each time point:
  - Converge coupled solution.
  - Time integrate solution to next time point.



What about Serpent?

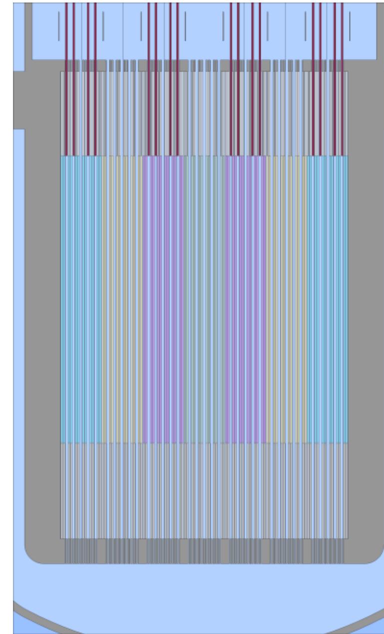
- Serpent is still a Solver.
- Field and variable names are different:
  - Ants_iv_xenon_state
  - sss_iv_fixed_xenon

  - Ants_ov_boron
  - sss_ov_critical_boron
- Internal workings of solvers are different.
- Meshing of fields may be different:
  - Ants meshing based on node structure.
  - Serpent meshing more freely chosen (multi-physics interface based).
- The same calculation can be repeated with Serpent (including the same variations).



Summary

- Cerberus offers a high level Python API for solvers and their fields and variables in Kraken.
- Coupled calculation chains can be easily created in Python.
- More complex calculation chains quickly get tedious to write from scratch.
  - Common calculation sequences and sub-sequences can be packaged into their own modules.
- Cerberus.simulator module contains a relatively simple reactor core simulator setup that has been utilized with Serpent/Ants, Kharon, SuperFINIX.



VTT

beyond the obvious

Ville Valtavirta
Ville.Valtavirta@vtt.fi
Kraken@vtt.fi

@VTTFinland

www.vtt.fi