Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013 (SNA + MC 2013)
La Cité des Sciences et de l'Industrie, Paris, France, October 27-31, 2013

Python-based framework for coupled MC – TH reactor calculations

A. Travleev [1*], R. Molitor [1], V. Sanchez [1]

[1] *Institute for Neutron Physics and Reactor Technology (INR), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany*

[*] Corresponding Author, anton.travleev@kit.edu

Abstract

We develop a set of python packages to provide modern programming interface to neutronics and TH codes. Currently implemented interfaces to the MCNP and SCF codes allow efficient description of neutronics and thermo-hydraulics domains and provide framework for coupling.

KEYWORDS: Python, MCNP, coupled MC-TH, SCF

I. Introduction

The High Performance Monte Carlo Reactor Core Analysis (HPMC) project aims full-core coupled calculations including neutronics, thermo-hydraulics, burnup and time dependence. This goal will be achieved in steps, starting from coupling of two physics domains (e.g. neutronics and thermo-hydraulics, neutronics and nuclide kinetics, etc.) for simple pin and assembly geometries. The transition from simple geometries to more general and complex implies that the coupling code developed for simple geometry at the first stage of the project should be reused on later stages.

It is assumed in the HPMC project that on certain stage cluster computers will be used for Monte-Carlo neutronics calculations. This means that the developed code should work on local desktops as well on cluster nodes.

These two aspects of the project led us to development of a python-based framework for coupling. The concept of this framework, the current development stage, code examples and some calculation results are presented in this work.

II. Concept of the coupling framework

The coupling framework includes (a) means to describe a general calculation model, where data common for all physics domain can be specified, and (b) particular code back-ends. The code back-ends understand the general model, provide interface to code-specific data, and handle the code input and output files.

This framework structure allows to independently develop code back-ends, once the structure of the general model is defined. To a user (a nuclear reactor physicist performing computations), this framework structure provides convenient possibility to define common data only once, as a general model, and to use these data in each code involved into coupled calculations.

1. General model

The general model, as currently implemented, describes geometry of the calculation domain.

To represent a PWR- or BWR-like geometry, two types of solids are defined: a rectangular parallelepiped (box), and a vertical cylinder. There are several rules that the solids obey: A solid can be inserted into another one, and can be positioned arbitrarily with respect to its container; Several solids can be inserted into the same container, the recently inserted covers the previously inserted; A solid can be inserted into only one container at the same time.

For example, a general model describing one fuel pin with surrounded coolant can have the following elements:

- A box representing the coolant,
- a cylinder inserted into the coolant box, representing the cladding,
- a cylinder inserted into the cladding cylinder, representing the gap,
- a cylinder inserted into the gap cylinder, representing the fuel pellets.

Each element of the general model has attributes to represent state variables. Currently implemented are heat, temperature and density axial profiles. Each state variable attribute has its own axial meshing. Each axial mesh of every state variable has its own value. Thus, axial profiles for heat, density and temperature are represented by piecewise-constant functions.

A material name can be assigned to each general model element. Currently, these are just strings; the actual meaning of the material name must be specified to every particular code backend.

The following code is an example of a pin model with the structure as described above:

```
from hpmc import Box, Cylinder
# water box
w = Box(X=1.27, Y=1.27, Z=390+50)
# cladding
c = w.insert('clad', Cylinder(R=0.475, Z=3400))
# gap
g = c.insert('gap',  Cylinder(R=0.411, Z=390))
# fuel pellets
f = g.insert('fuel', Cylinder(R=0.4025, Z=g.Z))
# material names
w.material = 'water'
c.material = 'zirc'
f.material = 'uo2'
# water density
w.dens.set_grid([1]*10)  # 10 equal mesh elements
w.dens.set_values(1)     # const. dens., g/cm3
# water temperature
w.temp.set_grid([1]*7)   # 7 mesh elements
w.temp.set_values(580.)  # const. temp, K
# fuel temperature
f.temp.set_grid([1]*6)   # 6 mesh elements
f.temp.set_values(1200)
# mesh for heat:
f.heat.set_grid([1]*10)  # 10 mesh elements
f.heat.set_values([1, 2, 3, 4, 5,
   ....:           5, 4, 3, 2, 1])
# fuel and clad densities are constant:
c.dens.set_values(4.)
f.dens.set_values(10.)
```

The ‘hpmc’ package is the package where the solids and axial mesh classes are defined. The insert() method puts a solid given as its second argument with the key, specified as the first argument, and returns the inserted solid. The set_grid() and set_values() methods of the heat, temp(erature) and dens(ity) attributes are used to specify the piecewise constant representation of the correspondent state variables.

2. MCNP backend

The MCNP[1)] backend is implemented in two steps. The stand-alone python package ‘mcnp’ provides object-oriented description of cells, surfaces, tallies and materials. The MCNP interface, defined as a part of the ‘hpmc’ package describing the general model, can convert solids of the general model to cells and surfaces of the ‘mcnp’ package.

The MCNP interface needs certain MCNP-specific data to convert a general model to a valid MCNP input file. This includes material composition, boundary conditions, source specification.

In the following example we show the definition of water for MCNP:

```
import mcnp
# natural element compositions
h = mcnp.Material('H')
o = mcnp.Material('O')
# water chemical composition
water = h*2 + o
# thermal data for H in water
water.thermal = 'lw'
```

The Material class has predefined natural isotopic compositions[2)]. Instances of this class can be mixed using weight, atomic or volume (if material density is specified) fractions. Cross-section suffices are not set directly. Instead, a user specifies the path to an xsdir file and sets material temperature. Based on the content of the xsdir file, proper suffices are chosen automatically. One can also specify interpolation law (in this case material temperature is represented as a mixture of cross-sections at two different temperatures). To illustrate this functionality let us see the MCNP material specification generated by the water material defined above:

```
In [1]: print water.card()
m{0:<} $ mixture H-O at 300.0 K
       1001.31c 1.9997700e+00
       1002.31c 2.3000000e-04
       8016.31c 9.9757000e-01
       8017.31c 3.8000000e-04
       8016.31c 2.0500000e-03
mt{0:<} lwtr01.31t $ thermal data at 293.606K
```

Note the use of thermal data. No temperature interpolation is implemented for thermal data, the code only choose the data with most close temperature.

A user must provide correspondence between the MCNP materials and material names of the general model. In the next example it is shown how to create an MCNP interface for the general model defined above and how to specify the material composition, relevant to MCNP:

```
from hpmc import McnpInterface
mci = McnpInterface(w)
mci.materials['water'] = water
```

The McnpInterface class provides also means to define lateral and axial boundary conditions, to specify initial neutron source and number of cycles in a criticality run.

After all relevant data are specified, one can start MCNP with the run() method. This method requires one argument, which specifies the MCNP execution mode. For example, the following code creates a folder, generates the input file corresondent to the geometry of the general model, starts MCNP in the initial run execution mode and returns results of calculations as the copy of the input general mode:

```
mc_result = mci.run('r')
```

The lower-cased mode ‘r’ means that the MCNP workplace (i.e. a folder with all necessary files) is prepared, but MCNP is not actually started. Even in this case the returned model has some arbitrary heat profile. This option is useful to test scripts, when not actual results, but only the formal coupling and coding is checked.

3. SCF backend and coupling to MCNP

An interface to the SCF[3)] code is implemented similar to the MCNP interface. There is stand-alone package ‘scf’ whose classes describe object-oriented representation of SCF input data. And there is an ScfInterface class that ‘knows’ how to

convert general model geometry into the SCF geometry.

Currently, the SCF interface is in the development stage and some of the material properties as well as some calculation control parameters are hardcoded. However, already on this stage, the CSF interface can handle the pin general model described above.

An example of the SCF interface to the pin model from above:

```
from hpmc import ScfInterface
sci = ScfInterface(mc_result)
sci.Tin = 580.  # coolant inlet temp, K
sci.Ptot = 30e3 # total rod power, J/s
sci.Gr = 3.6e2  # mass flow rate, g/cm3
sc_result = sci.run('r')
```

Note that we passed to the SCF interface the general model returned by the MCNP interface. in this way, results of MCNP run appear in the input for SCF. This technique provides the basis for effective and transparent data handling between codes.

Additionally, one can perform mathematical operations (currently implemented addition, subtraction, multiplication) on the attributes representing density, temperature and heat. For example, a relaxation formula of this kind:

$$P_{r,i} = \alpha\, P_{r,i-1} + (1 - \alpha) P_m \qquad (1)$$

where subscript r denotes relaxed power used as input for SCF, and the 'm' subscript denotes the power computed by MCNP at i-th iteration, can be described as the following code:

```
# i-1 heat, used for scf input
Pr = sci.gm.values()[-1].heat

# i-th mcnp result:
Pm = mc_result.values()[-1].heat

# relaxation factor
a = 0.5

# new relaxed power:
Pr = a*Pr  +  (1. - a)*Pm
```

III. Results of illustrative coupled calculations

To test the developed framework and to provide a real-world example, we define a model to represent a PWR fuel pin. Two coupling schemes are coded: one utilizes relaxation of the power axial profile with MCNP statistics increase on each iteration[4)], the other utilizes relaxation of the fuel temperature axial profile with the constant MCNP statistics.

Figure 1 shows axial profiles of fuel heat and temperature after the 22-nd iteration. At this iteration MCNP was scheduled to sample about 2400 cycles, 50 of them are inactive; each cycle has 500 particle histories. Black line on the upper plot shows the relaxed power obtained on this iteration, computed as superposition of the MCNP result (yellow line) and previous relaxed power (grey line). One can see that on this iteration the new MCNP result does not introduce considerable changes to the relaxed distribution.

The lower plot shows behavior of the fuel temperature as computed with SCF for the relaxed power. For comparison, the correspondent axial profiles obtained on the previous iteration, is shown.

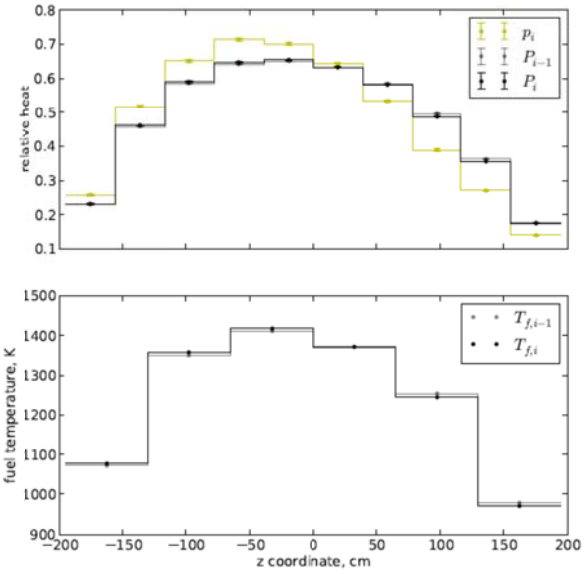

Figure 1: Results of the first coupling scheme

The second figure shows results on the 30-th iteration of the second coupling scheme. The upper plot shows the relaxed fuel temperature (black line), which is obtained as superposition of SCF result (yellow line) and relaxed fuel temperature on the previous step (grey line). The lowest plot shows the fuel heat axial profile, as computed by MCNP for the model with relaxed fuel temperature, water temperature and density.

It interesting to note that both results are obtained with almost the same number of cycles over the whole iterations. In the first coupling scheme, there were 200 cycles and this number increased by about 100 each iteration. Thus, after the 22-nd iteration, where about 2400 cycles were scheduled, the cumulative number of cycles reached about 29000. This is almost the same as the total number of cycles after 30 iterations of the second coupling scheme, where on each iteration, MCNP was scheduled to sample 1000 cycles.

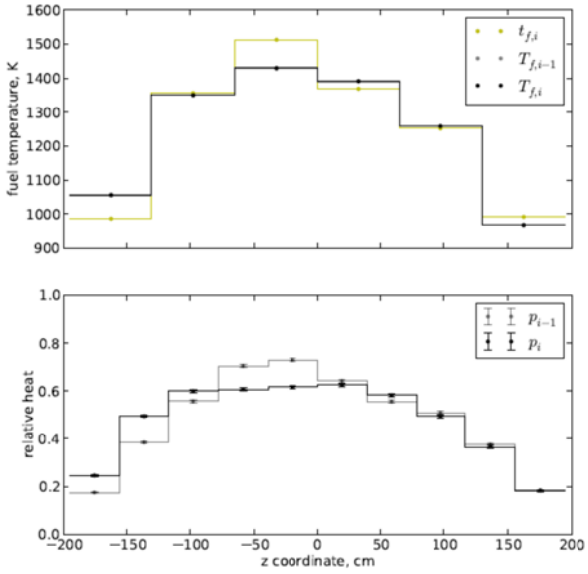

Figure 2: Results of the second coupling scheme

IV. Current state and outlook

The example code snippets above show the most important already implemented features of the framework. Additionally, there are some auxiliary mechanisms that simplify everyday life of a user running coupled calculations. Among them is the ability to dump current iteration, so it can be continued later, and an automatic generation of figure reports similar to the plots shown above.

The documentation is developed in parallel with the packages, with a small delay necessary to exclude documenting of experimental stuff. The Sphinx[5] system is used to write documentation. This system provides an environment, where the code examples written in the documentation can be run through the Python interpreter, ensuring that the documentation is intact with the code.

The presented framework, although already can be used to perform coupled simulations of a stand-alone pin, is under development. The nearest plans, according to the goals of the HPMC project, is to improve MCNP and SCF interfaces to the level that allows modeling and coupling calculations of a nuclear reactor core detailed down to assembly- and pin-level. Additionally, an interface to the Serpent[6] code must be provided.

Since the full reactor core simulations with Monte-Carlo codes are feasible only when run in parallel, transition from desktops (currently, the packages work both on Windows and Linux machines) to clusters is unavoidable. A basement for this transition is ready, however, implementation details will depend on the cluster's job scheduling environment.

Further plans may include coupling to the parts of KANEXT[7] system and some burnup codes. This, however lies beyond the HPMC project.

Acknowledgement

This work is funded by the European Commission via the FP7 project HPMC "High-Performance Monte Carlo Reactor Core Analysis" under contract no. 295971.

References

1) X-5 Monte Carlo Team, MCNP — A General Monte Carlo N-Particle Transport Code, Version 5, LANL (2003).
2) Pure Appl. Chem., vol. 83, No 2, pp. 397-410, 2011
3) V. Sanchez, U. Imke, A. Ivanov, R. Gomez, "SUBCHANFLOW: A Thermal-Hydraulic Sub-Channel Program to Analyse Fuel Rod Bundles and Reactor Cores", 17th Pacific Basin Nuclear Conference, Cancún, México, 2010.
4) Dufek, J. and Gudowski, W., "Stochastic Approximation for Monte Carlo Calculation of Steady-State Conditions in Thermal Reactors," Nucl. Sci. Eng., Vol. 152, 2006, pp. 274.
5) Sphinx documentation, http://sphinx-doc.org/
6) Serpent 2, a Continuous-energy Monte Carlo Reactor Physics Burnup Calculation Code, http://montecarlo.vtt.fi/
7) Becker, M., Criekingen, S. Van, Broeders, C.H.M.: The KArlsruhe PROgram System KAPROS and its successor the Karlsruhe Neutronic Extendable Tool KANEXT, http://inrwww.fzk.de/kanext.html, 2008.