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

Anton A. Travleev[1], Richard Molitor[1], and Victor Sanchez[1]

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

We develop a set of Python packages to provide a modern programming interface to codes used for analysis of nuclear reactors. Currently implemented interfaces to the Monte Carlo (MC) neutronics code MCNP and thermo-hydraulic (TH) code SCF allow efficient description of calculation models and provide a framework for coupled calculations. In this paper we illustrate how these interfaces can be used to describe a pin model, and report results of coupled MCNP-SCF calculations performed for a PWR fuel assembly, organized by means of the interfaces.

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

I. Introduction

The HPMC project[(1)] aims full-core coupled calculations including Monte Carlo (MC) neutronics, thermo-hydraulics (TH), burnup and time dependence. This goal will be achieved in steps, starting from coupling two physics domains (e.g. neutronics and thermo-hydraulics, neutronics and nuclide kinetics, etc.) for simple pin and assembly geometries and gradually increasing complexity. It is assumed in the HPMC project that on certain stage cluster computers will be used for MC neutronics calculations.

These two aspects of the project imply that the coupling procedure need to be reusable for different types of problems (like pin, assembly and whole core models) and should be flexibly applied on different types of computers (like small desktop machines and supercomputers).

Coupled neutronics – TH calculations are widely used in reactor engineering and several different approaches have been established to couple neutronics and TH codes. The fastest in implementation approach to organize data flow between two particular N and TH codes is to prepare input file templates and to write an interface program (this approach is followed, for example, in[(2)]). The interface program reads output from one code, prepares data for the other code and inserts them into the correspondent input file template. Both templates and the interface program are usually written considering particular geometry type, which on the one hand simplifies coding, but on the other hand makes it difficult to reuse the coupling scheme for a different type of geometry.

Another way to couple neutronics and TH codes is to use a common platform that provides means to integrate codes and to organize data flow between them. An example of this approach is the NURESIM system[(3,4)] that utilizes the SALOME platform.[(5)] This approach results in much more flexible solution as compared to the approach described above, but also has some drawbacks. For example, integration of a code into the platform might require changes in the code itself, which is not always possible due to e.g. licensing limitations. Moreover, requirements to computer (hardware and OS) imposed by SALOME are quite restrictive.[(5)]

In the present work we propose a coupling approach that can be applied to different types of calculation geometry and does not require changes in the code to be integrated. This approach is implemented as a set of Python[(6)] packages called PIRS – Python Interfaces for Reactor Simulations that currently provides interfaces to MCNP[(7)] and SCF codes.[(8)]

In this paper we describe the concept of PIRS, give an example of neutronics specifications for a simplified pin model, and report results of coupled MCNP-SCF calculations for a PWR assembly organized by means of PIRS.

II. Concept of the coupling framework

Python classes provided by PIRS can be classified by their functionality into three groups: low-level interfaces, general model classes and high-level interfaces, see figure 1. Classes of the first group describe an interface between Python and a particular code (low-level interfaces). A low-level interface class allows to set any parameter in the input file, generates it, provides means to execute the code and defines procedures to read output files directly from the Python interpreter or a Python script. Low-level classes ensure that the generated input files have correct syntax and often saves a user from specifying auxiliary or non-used parameters.

General model classes can be used to describe calculation geometry and meshes to represent system variables.

High-level interface classes are used to convert geometry described with general model classes to instances of low-level interface classes and to put results of code calculations (read by low-interface classes) back to general model.

To organize coupled neutronics – TH calculations, a user defines parameters, common to both neutronics and TH domains, in a variable (say, `a1`) by means of general model classes. This variable contains information about geometry and axial meshes

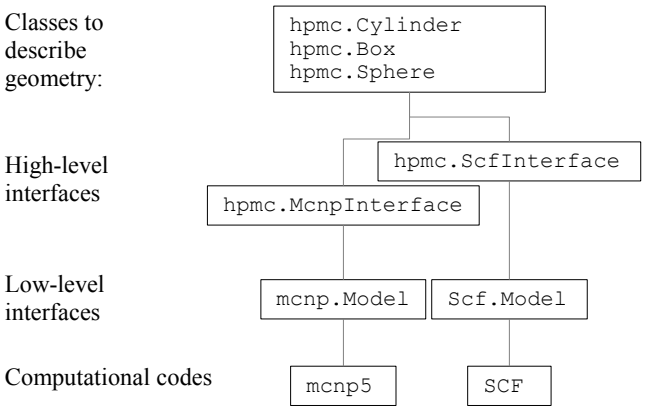

Figure 1: PIRS Classes and their interaction with computational codes.

for heat deposition, temperature and density. An instance of one of the high-level interfaces (let call it `i1`) is used to set code-specific information, like cross-sections data sets for MCNP. Variable `i1` has a method that takes `a1` as an input model, prepares an input file based on the common information stored in `a1` and code-specific data stored in `i1`, executes the code, reads the code output and returns `a2` that is a copy of `a1` except some system variables which are replaced with the results of calculations. Variable `a2` can be passed to a high-level interface of another code as the input model; in this way one can organize data flow between codes. A user can arbitrarily change parameters of a model, e.g. to describe a relaxation scheme.

This framework structure allows to develop independently code back ends (i.e. low-level and high-level classes for a code), once the structure of the classes representing general model is defined.

III. Example: a simplified pin model

A simplified pin model is considered to illustrate steps necessary to perform neutronics calculations. First, a geometry of a model is described using general model classes, and second, MCNP-specific date are specified with the help of the MCNP high-level interface.

1. Calculation geometry

The simplified pin model consists of two regions: a vertical cylinder (fuel) is immersed into a rectangular parallelepiped (moderator). The following Python script defines this model:

```
from hpmc import Box, Cylinder

b = Box(X=1.2, Y=1.2, Z=110)
c = Cylinder(R=0.5, Z=100)
b.insert(0, c)

b.material = 'water'
c.material = 'fuel'

b.dens.set_grid([1, 1])
b.dens.set_values(1.)

c.temp.set_grid([1]*3)
c.temp.set_values([300, 500, 350])

c.heat.set_grid([1]*10)
```

The `Cylinder` and `Box` classes are imported from the `hpmc` module provided by PIRS. These classes are used to set the geometrical elements – constituents of the pin model. Variable `b` is an instance of the `Box` class; it describes a rectangular parallelepiped with sides 1.2 x 1.2 x 110 cm. Variable `c` is an instance of the `Cylinder` class and represents a vertical cylinder with radius 0.5 cm and height 100 cm. The `insert` method of variable `b` is used to specify positional relationship between solids `b` and `c`: cylinder `c` is inserted into box `b`. Coordinates of `c` with respect to `b` are not given explicitly and by default `c` is centered with respect to `b`.

A material name can be given to each model's element, by means of the `material` attribute. This attribute is set by default to `'void'` and can take any string value. Actual meaning of the material name must be defined for each code using correspondent high-level interface.

Next, meshes to represent system variables are defined. Attributes `dens`, `temp` and `heat` represent density, temperature and heat deposition axial distributions for each model's element, respectively. In the example above, density of the element `b` is represented by an axial mesh with two equidistant mesh elements; in both mesh elements density is set to 1. Temperature of the element `c` is given by an axial mesh with three equidistant mesh elements, temperatures are 300 K in the lower mesh element, 500 K in the middle element and 350 K in the upper mesh element. Additionally we specified that heat deposition should be represented by an axial mesh with 10 equidistant elements, but no values are given explicitly.

2. high-level interface to MCNP

The above script describes only geometry of the model. MCNP-specific data are specified by means of the `MCNPInterface` class from the `hpmc` module. This includes specification of material isotopic compositions, boundary conditions and neutron source for criticality calculations.

```
from hpmc import McnpInterface
from mcnp import Material

m = McnpInterface(b)

u = Material((92235,  0.5, 2),
             (92238, 95.5, 2))

o = Material('O')
h = Material('H')

f = u + 2*o
w = h*2 + o
w.thermal = 'lwtr'

f.sdict[8018] = 8016
w.sdict[8018] = 8016

m.materials['fuel'] = f
m.materials['water'] = w
```

```
m.bc['radial'] = '*'

m.adc.append('ksrc 0 0 0')
m.adc.append('kcode 500 1. 20 100')

m.run('P')

r = m.run('R')
```

The `m` variable is an instance of the `McnpInterface` class; its constructor takes optionally a variable representing general model (in this example, the variable `b` defined earlier). Any geometry described using `Cylinder` and `Box` classes and their arbitrary combinations can be handled (understood) by the MCNP interface.

The `Material` class imported from the `mcnp` module (this module describes low-level interface to MCNP) is used to define material compositions. Variables `u`, `o` and `h` represent isotopic compositions of chemical elements uranium, oxygen and hydrogen. Uranium isotopic composition is given in weight fractions, for oxygen and hydrogen natural abundance data[9] are used. Arithmetical operations for instances of the `Material` class are defined in the way that can be used to define new mixtures. In the example, variable `f` is a mixture of 1 atomic part of uranium and two atomic parts of oxygen and represents uranium dioxide. The `w` variable represents water. Optionally, one can specify a string pattern that will be used to find cross-section data for thermal scattering, see the `thermal` attribute. In the example, thermal data set for water will be chosen among thermal data sets with names containing `'lwtr'` string with the closest temperature.

Materials `f` and `w` contain all stable isotopes of oxygen, among them isotope 8018. In the cross-section data set applied implicitly in this example, there is no data for this isotope. To handle such situations, there is the `sdict` attribute of the dictionary class that defines a substitution table: if for a given isotope (in our example 8018) there is no cross-section, it is substituted with another one (in our example 8016).

Correspondence between material compositions and general model material names is defined in the attribute `materials`. It is a dictionary with material names as keys and instances of the `Material` class as values.

Boundary conditions are set up using the `bc` attribute. It is an instance of a dictionary-based class that admits only two keys, `radial` and `axial`. By default, the values are empty strings meaning no reflection on radial and axial general model outer boundaries. One can set them alternatively to `'*'` or `'+'` which means reflective or white boundary conditions, respectively (similar to MCNP syntax). In the example we specify reflective radial boundary conditions (for our model, the radial boundaries are the boundaries of the water box that are perpendicular to the vertical axis) and leave default axial boundary conditions.

Currently, there is no high-level interface to specify neutron source parameters for an MCNP problem. As a workaround, a user can use possibility to specify directly lines for the input file, provided by the low-level MCNP interface. Attributes `adc`, `asc`, `acc` and `amc` are lists of lines that will be appended to the data cards block, surface cards block, cell cards block and message cards block, respectively. In the example we append two lines to the data cards block, specifying initial spatial distribution of fission neutrons and parameters of criticality calculations.

The `run` method generates the MCNP input file and starts the code. The first call to the `run` method, with `'P'` argument, starts MCNP in the geometry plotter mode. In the second call with `'R'` argument, MCNP is started in the particle transport mode. In this mode the interface waits until MCNP finishes, reads heat deposition computed by mesh tallies and returns a copy of the general model with heat attributes containing MCNP results. Heat deposition mesh tallies are computed for each element of the general model with non-trivial (non-zero or with more than 1 mesh element) `heat` attribute. In the first example we specified ten equidistant mesh elements for the `heat` attribute of the fuel cylinder while the `heat` attribute of the box is by default trivial. Thus, heat deposition is computed in the fuel cylinder only and the cylindrical element of the returned general model, `r`, will contain the results.

The MCNP input file generated during the first call to the `run` method is listed below, and figure 2 shows a plot produced by MCNP.

```
MESSAGE:  datapath=/home/data/mcnp/all_jeff

c title
1 0  -3 4 -5 6 -2 1 fill=1 imp:n=1
2 0  -7 fill=2 imp:n=1 u=1
3 1 -1.0 -8 imp:n=1 tmp=2.585203e-08 u=2
4 2 -1.0 8 -9 imp:n=1 tmp=4.308671e-08 u=2
5 3 -1.0 9 imp:n=1 tmp=3.016070e-08 u=2
6 4 -1.0 -10 7 imp:n=1 tmp=2.585203e-08 u=1
7 4 -1.0 10 7 imp:n=1 tmp=2.585203e-08 u=1
8 0  11 (3:-4:5:-6:2:-1) imp:n=0 tmp=2.585203e-08

c surfaces
1 pz  -55.0
2 pz  55.0
*3 px  0.6
*4 px  -0.6
*5 py  0.6
*6 py  -0.6
7 rcc  0.0  0.0  -50.0  0.0  0.0  100.0  0.5
8 pz  -16.666666667
9 pz  16.666666667
10 pz  0.0
11 pz  -1055.01817881

c data cards
c materials
m1                    $ mixture U-O at 300 K
      92235.31c 5.0000000e-01
      92238.31c 9.5500000e+01
       8016.31c 1.9951400e+00
       8017.31c 7.6000000e-04
       8016.31c 4.1000000e-03
m2                    $ mixture U-O at 500 K
      92235.31c 3.9962042e-01  92235.40c 1.0037958e-01
      92238.31c 7.6327500e+01  92238.40c 1.9172500e+01
       8016.31c 1.5945974e+00   8016.40c 4.0054263e-01
       8017.31c 6.0742304e-04   8017.40c 1.5257696e-04
       8016.31c 3.2768874e-03   8016.40c 8.2311256e-04
m3                    $ mixture U-O at 350 K
      92235.31c 5.0000000e-01
      92238.31c 9.5500000e+01
       8016.31c 1.9951400e+00
       8017.31c 7.6000000e-04
       8016.31c 4.1000000e-03
m4                    $ 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
mt4 lwtr01.31t          $ thermal data at 293.606K
c tallies
fmesh14:n               $ heat in ('/', 0)
     geom=cyl
     origin=0.0 0.0 -50.0
     axs=0.0 0.0 1.0
     vec=1.0 0.0 0.0
     imesh= 0.5
     jmesh= 10.0 20.0 30.0 40.0 50.0 60.0
            70.0 80.0 90.0 100.0
     kmesh= 1.0
fm14 -1 0 -6 -8
prdmp j j 1             $ write mctal file
ksrc 0 0 0
kcode 500 1. 20 100
```

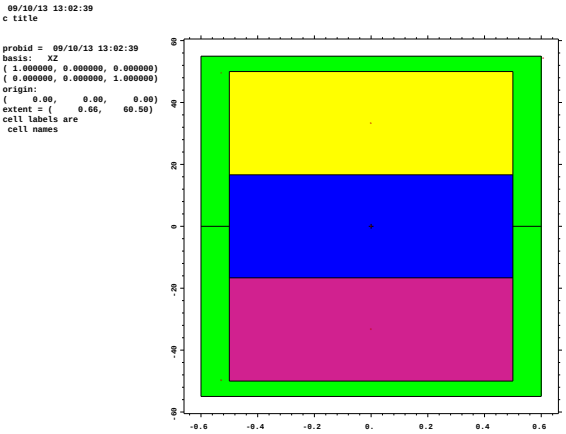

Figure 2: Vertical section of MCNP model, plotted with MCNP.

Cell and surface numbers used in the input file are assigned automatically by the MCNP interface, so that a user does not need to invent them and check their uniqueness.

Density and temperature axial distributions are represented as a set of cells with different density iand filled with different materials. One can see three U-O mixtures, `m1`, `m2` and `m3` that represent the same fuel isotopic composition but use different data suffixes. If there is no cross-section data set prepared for a given temperature, it is interpolated using two data sets with temperatures below and above the specified value, see material `m2`. This technique is reffered to as pseudo-material cross-section interpolation and is used for different type of reactors.[10,11] Note that we didn't specify cross-section suffixes in the code above. Their choice depends on the `$DATAPATH` environmental variable (must be set before using PIRS), name of the xsdir file (`xsdir` by default) and by material temperatures. When a material is specified to the MCNP interface, it looks in the xsdir files for the cross-sections with the closest temperature and correspondingly chooses the suffix.

Mesh tallies are used to compute axial distribution of heat. Mesh tally parameters are defined by the geometry element dimensions and its `heat` attribute. In the example, the mesh tally covers a region that coincides with the fuel cylinder and has ten axial equidistant mesh elements.

3. High-level interface to SCF

The SCF code interface can handle only particular types of geometries described by the general model classes. This limitation is due to the origin of problems solved with SCF. Particularly, the general model defined above, `b`, cannot be handled with the current implementation of the SCF interface, since it needs a model representing at least one rod – a set of two or three coaxial cylinders representing clad, gap and fuel regions of a rod. Thus the usage of the SCF interface will be shown only schematically:

```
from hpmc import ScfInterface

s = ScfInterface(r)

# ...

s.inlet_temperature = 560 # K
s.total_power = 3e4 # W
s.inlet_flow_rate = 400 # g/s
s.exit_pressure = 15.5e6 # Pa

p = s.run('R')
```

The usage of the SCF interface is similar to the usage of the MCNP interface. First, an instance of the `ScfInterface` is created , its constructor takes a general model as an argument. Additionally one needs to specify parts of the general model representing fuel rods and optionally – the part representing a container (coolant channel wrapper).

In the current implementation, there are rather simplified possibilities to specify boundary conditions. One can specify inlet temperature, total power, inlet flow rate, exit pressure and some other (the SCF code has more possibilities to specify boundary and initial conditions, for example one can set inlet temperature to each subchannel).

Similar to the MCNP interface, the SCF code is started by calling the `run` method of the interface. The SCF input file is generated, based on the geometry information from the general model and SCF-specific data provided to the SCF interface, the code is executed, and results – fuel temperature, coolant temperature and densities – are read and returned in the general model copy.

IV. Example results for PWR assembly

1. Assembly model

At the current stage, the MCNP and SCF interfaces allow to model a assembly-like geometries. Test calculations have been performed for a PWR assembly model based on the PWR benchmark specifications.[12]

The assembly contains 264 UOX fuel pins and 25 guide tubes, positioned in a square 17x17 lattice with 1.26 cm pitch; the model's horizontal cross-section is a square with sides 21.42 cm. The model's height of 408.6 cm is defined by the active length of fuel pins, plus lower and upper water reflectors, each

21.42 cm thick. There are two types of fuel pins: usual pins and pins with IFBA (integrated fuel burnup absorber).

The coolant inlet temperature is 560 K. This temperature and correspondent density (0.752 g/cm3 at 15.5 MPa) is used for the lower reflector. Temperature and density of water in guide tubes is 580 K and 0.712 g/cm3, respectively. The fuel and coolant in the active length region are divided into 11 axial layers of varying thickness to represent the fuel temperature, coolant density and coolant temperature axial distributions. In the upper reflector, the coolant properties are heterogeneous and correspond to the channel exit parameters, as calculated by SCF.

All fuel pins have the same fuel composition: UO2 with U enriched to 4.2 % (in mass). The pin cladding and guide tube wall are represented by zircaloy-2.[12] Coolant is pure water. Gaps in fuel pins are filled with oxygen. The IFBA pins have additional cylindrical layer of zirconium diboride (ZrB2).

In the MCNP models, in the active length region and in the upper reflector, the coolant temperature and density as well as the fuel temperature are defined by the SCF calculations. In the SCF models, the power profile is based on the MCNP results.

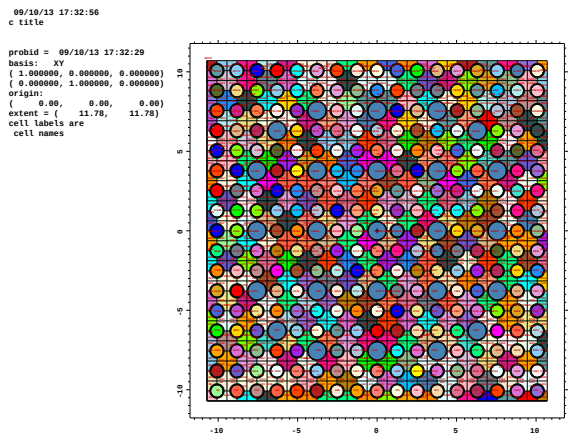

Figure 3: Horizontal section of MCNP model at iteration 62. Different colors correspond to materials with different temperatures.

Figures 3 and 4 show MCNP plots for the model. The plots illustrate some capabilities of the MCNP and SCF interfaces:

- the active length region can be split into axial layers arbitrarily; in the assembly model we apply a non-uniform pattern with thiner layers at the bottom and top of the model. The non-uniform pattern should represent a cosine-like distribution of power density axial profile more accurately.
- Subchannels in SCF are "coolant-centered". The subchannels are converted into the MCNP model exactly, thus no interpolation or averaging of coolant properties is necessary.
- Bundle of pins is represented in the MCNP model using the lattice cell. Another way to represent bundles of rods, without lattice cells, is also implemented in the MCNP interface, but cannot be applied to a model with 17x17 elements.

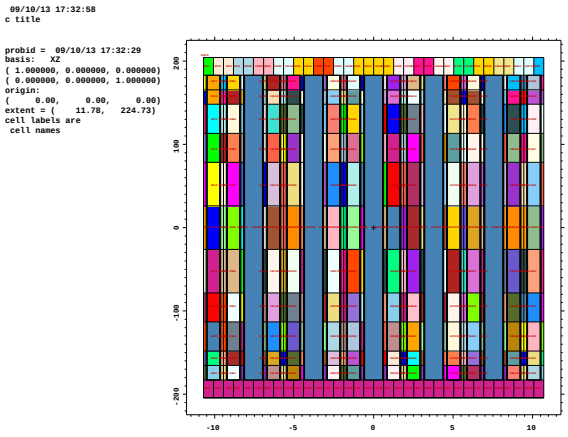

Figure 4: Vertical section of MCNP model at iteration 62.

The following boundary conditions are specified for SCF models: the coolant inlet temperature is 560 K in all subchannels. Assembly thermal power is 18.47 MW; this value is obtained by dividing the core thermal power to the number of assemblies in the core. The inlet mass flow rate is 82.12 kg/s; this value is obtained also by dividing the average core mass flow rate to the number of assemblies. The exit pressure is set to 15.5 MPa.

Thermo-physical properties of cladding and fuel pellets, specified in the benchmark, are hard-coded in the SCF code under the name `'benpwr'`. These properties are applied in the SCF calculations to compute fuel temperature.

The relaxation scheme[13] is applied for the coupled calculations. This scheme assumes an increase of statistical precision of the MCNP results on each iteration. This is implemented by increasing the number of cycles. On the first iteration, MCNP sampled 50 cycles with 1000 histories per cycle and skipped first 10 cycles. On the 32-th iteration, MCNP sampled 835 cycles and on the 62-nd iteration, MCNP sampled 1673 cycles.

For the demonstration calculations we have chosen an xsdir with cross-sections sets for only two temperatures, 300 K and 1800 K (suffixes 31c and 40c, respectively). Thus, any fuel temperature is represented by a mixture of only these two data sets, that minimizes the initialization time for MCNP. To perform calculations with a finer cross-section temperature mesh, one needs only to provide an xsdir with more data sets; no modifications in the scripts describing the coupled calculations are needed.

The fuel temperature passed from SCF to MCNP is the radially-averaged temperature, reported in the SCF output file in tables with rod results in the column named `tfuave`.

The subchannel coolant temperature and density is reported by SCF at the boundaries of each axial layer. Coolant parameters for MCNP models must correspond to the layer interior, they are estimated as a mean of the boundary values.

2. Results

Results are shown for the 62-nd iteration. The coupled calculation has been started with the convergence criterion set to 50 pcm, and should iterate until the standard deviation of the value

of Keff computed with MCNP, and the difference between two values of Keff computed on subsequent iterations, become below this value. At the 62-nd iteration, the convergence criterion is not meet yet, iterations continue, but the intermediate results can be analyzed since all necessary data is dumped after each iteration and can be processed without stopping the iterations.

Figure 5 shows Keff obtained on each cycle. In the first iteration, the coolant properties correspond to an uniform power profile and the initial neutron source is axially positioned in the center of the assembly. These two aspects lead to a larger Keff, as compared to results obtained on subsequent iterations. The statistical precision of Keff is improved with iterations, since the number of sampled cycles in increased.

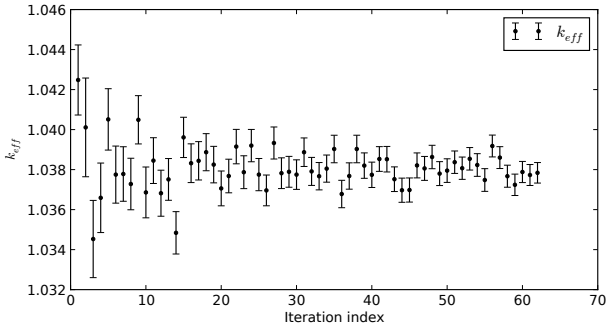

Figure 5: Value of Keff versus iteration number.

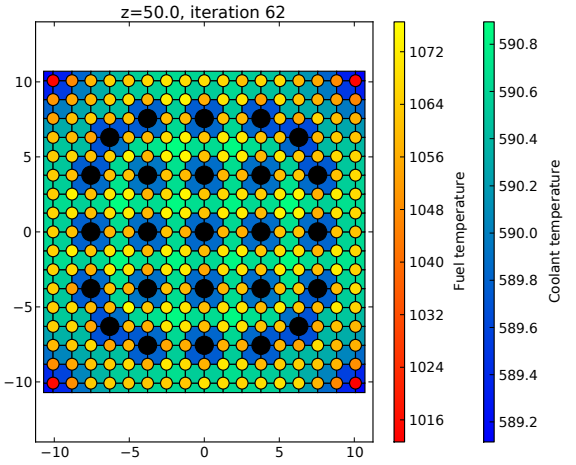

Figure 6: Coolant and fuel pellets temperature map in axial layer where maximal fuel temperature is reached.

The map on figure 6 shows radial temperature distribution of fuel pellets and coolant. Although radial temperature deviations are small, one can clearly see the effect of moderator channels to the coolant temperature in adjacent channels.

The fuel temperature and power density axial distributions are shown on figure 7 for the fuel pin where the maximal power density and maximal fuel temperature are reached. In the upper plot, the yellow line shows results of the MCNP calculation on the 62-nd iteration and the black line shows distribution of the relaxed power density. The later is used to get fuel temperature axial distribution, which is shown on the lower plot on figure 7.

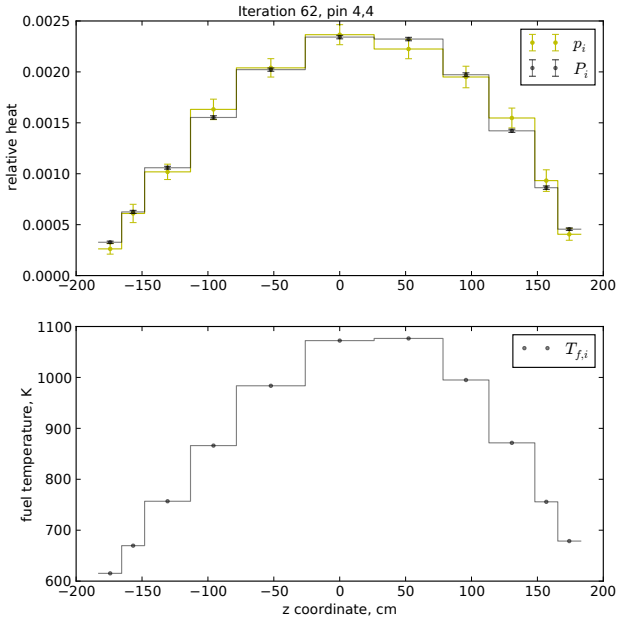

Figure 7: Axial distribution of power density and fuel temperature in the pin with maximal power density.

V. Current state and outlook

At the current stage of development, PIRS provides a convenient way to setup MCNP and SCF calculations for pin- and assembly-like geometries. The example considered in the paper illustrates that description of a neutronics and TH model using PIRS classes is more compact as compared to the correspondent input files but still allows to set all relevant parameters.

The concept of PIRS – independent description of geometry, and code interfaces that understand this description – establishes basis for coupled calculations. So far a code interface is added to PIRS, it can be used to organize data transfer between this code and already integrated into PIRS.

Python runs on Windows, Linux/Unix and other OS.[6] Currently, PIRS has been tested on desktop computers running Windows and Linux and on the JUQUEEN supercomputer.[14] Most of the PIRS code is OS-independent; for OS-dependent parts of the code, Python has well-documented libraries.

Packages provided by the Python community simplify to large extent post-processing of results obtained with PIRS. The code examples shown in this paper illustrate only calculations-relevant part of PIRS. There are also auxiliary classes and functions to dump calculation results for later use, to generate plots etc. This allows to setup calculation work-flow starting from input data specifications and up to generation of plots with results in a single Python script.

PIRS documentation is written in parallel with the code itself, with a delay necessary to avoid documenting of experimental stuff. The Sphinx[15] system is used. It provides environment, where code examples – very important part of software documentation – is simple to add and to keep updated with permanently developing software.

The presented framework, although already can be used to perform coupled simulations of a pin- or assembly model, is under further development. The nearest plans are dictated to

large extent by the goals of the HPMC project. Among them is ability to communicate with the job submission system of JUQUEEN, which will allow to choose MCNP job parameters dependent on current loading of the supercomputer. We also plan an interface to the SERPENT code.[16] Futher plans might include interfaces to the KANEXT[17] system developed at our institute; this however lies beyond the HPMC project.

VI. Acknowledgment

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) "HPMC An FP7 Euroatom Project," http://www.fp7-hpmc.eu.
2) E. Hoogenboom, C. Diop, A. Ivanov, and V. Sanchez, "Development of coupling methodologies for TRIPOLI and FLICA in the frame of NURISP Project," *Proc. 2011 International Conference on Mathematics and Computational Methods applied to nuclear science and engineering (M&C 2011)*, Rio De Janeiro, RJ, Brazil, May, 2011.
3) C. Chauliac et al., "NURESIM: a eurupean platform for nuclear reactor simulation," *Proc. ICONE-14 Workshop on advanced LWR*, Miami, USA, July, 2006.
4) "NURESIM," http://www.nuresim.com.
5) "SALOME: open source integration platform for numerical simulation," http://www.salome-platform.org.
6) "Python Programming Language – Official Website," http://www.python.org.
7) X-5 Monte Carlo team, *MCNP – A general N-Particle transport code, Version 5 – Volume I: overview and Theory*, Los Alamos National Laboratory, 2003.
8) U. Imke and V. Sanchez, "Validation of the subchannel code SUBCHANFLOW using the NUPEC PWR test (PSBT)," *Science and Technology of Nuclear Installations*, **2012**, 12 (2012).
9) M. Berglund and M. E. Wieser, *Pure and Applied Chemistry*, **83**, 397–410 (2011).
10) F. B. Brown, W. R. Martin, and R. D. Mosteller, "Monte Carlo Advances and Challenges," *Proc. PHYSOR'08 – International Conference of Physics of Reactors*, Interlaken, Switzerland, September, 2008.
11) A. Al-Hamry and V. Sanchez, "Development of a coupling scheme between MCNP and COBRA-TF for the prediction of the pin power of a PWR fuel assembly," *Proc. 2009 International Conference on Mathematics, Computational Methods and Reactor Physics (M&C 2009)*, New York, USA, May, 2009.
12) T. Kozlowksi and T. J. Downar, "PWR MOX/UO2 Core Transient Benchmark, Final Report," NEA.
13) J. Dufek and W. Gudowski, "Stochastic Approximation for Monte Carlo Calculation of Steady-State Conditions in Thermal Reactors," *Nucl.Sci.Eng.*, **152**, 274 (2006).
14) "JUQUEEN – Juelich BLue Gene/Q," http://www.fz-juelich.de/ias/jsc/juqueen.
15) "Sphinx documentation," http://sphinx-doc.rog.
16) "SERPENT 2, a Countinuous-energy Monte Carlo Reactor Physics Burnup Calculation Code," http://montecarlo.vtt.fi.
17) M. Becker, S. Criekingen, and C. Broeders, "KArlsruhe PROgram System KAPROS and its successor KArlsruhe Neutronic EXtentable Tool KANEXT," http://inrwww.webarchiv.kit.edu/kanext.html, last accessed September 2013.