Australian Government
Bureau of Meteorology

PSyclone's PSyData API – Or: Tools Tools Tools

Rupert Ford, Andy Porter, Sergi Siso, STFC Hartree Centre
Iva Kavcic, Chris Maynard, Andrew Coughtrie, UK Met Office
Joerg Henrichs, Australian Bureau of Meteorology

ESIWACE2 training course on Domain-specific Languages in Weather and Climate, 23rd-27th November 2020

Overview

- PSyclone transformations (~ 20 minutes)
- What is the PSyData interface? (~ 25 minutes)
    - How to use them
    - Which tools are available and planned
- Hands-on session (~45 minutes)

PSyclone Transformations

- Transformation allow to modify the PSyIR tree
  - Modify existing code
  - Add new code
- Crucial for separation of concerns
  - Transformations can be independent of the code
  - Change code for
    - Optimisation
    - Parallelisation
    - Add debugging feature
- Transformations are Python scripts
  - Using PSyclone objects

Australian Government
Bureau of Meteorology

Transformation vs Transformation Script

- A **transformation** is a instance of a class that modifies the PSyIR tree and has two methods:
  - `validate()` – verify that the transform can be applied
  - `apply()` – apply the transform
    - Calls `validate()` first
  - Needs to be instantiated
- A **transformation script** has
  - A single functions ``` `trans()` ``` to modify the PSyIR tree
  - Is specified on the command line using `-s`
    - `trans()` is called by PSyclone once per file
  - Typically uses one or more transformations

Australian Government
Bureau of Meteorology

Applying a Transformation Script

- A Python transformation script can be specified on the PSyclone command line:

```
psyclone … -s ./my_script.py \
           -opsy psy.f90       \
           -oalg alg.f90 file.x90
```

- The script must contain the `trans()` function, which is called once per file
  - After distributed memory has been added to PSyIR (see presentation later)
  - OpenMP is added using this kind of script (see presentation later)

Australian Government
Bureau of Meteorology

A Transformation Script

```
def trans(psy):
    '''
    Take the supplied psy object, and transform
    it

    :param psy: the PSy layer to transform.
    :type psy: :py:class:`psyclone.psyGen.PSy`

    :returns: the transformed PSy object.
    :rtype: :py:class:`psyclone.psyGen.PSy`
    '''
```

- Parameter: the PSyclone top level PSy object, gives access to PSyIR

The PSy Object

- Contains a list of invokes
  - Subroutine call to PSy-layer
- Each invoke:
  - (nested) loop structure (depending on API)
  - Kernel computation
  - Kernel can have various complexities
    - LFRic: loop over column
    - Other APIs: call to a single element, or 'computations'

Australian Government
Bureau of Meteorology

Getting the Invokes

- By name for a single invoke:

```
psy.invokes.get("invoke_name")
     # ‘name’ is specified in invoke
```

- By index:

```
psy.invokes.invoke_list[n]
```

- Loop over names:

```
for name in psy.invokes.names:
    invoke = psy.invokes.get(name)
```

Australian Government
Bureau of Meteorology

Schedules

- A schedule stores a sequence of statements
- Each invoke has one schedule:

```
schedule = invoke.schedule
```

- Example:

```
InvokeSchedule[invoke='invoke_propagate_pertur
                       bation', dm=False]
0: Loop[type='', field_space='w3', …]
        Literal [...] …
        Schedule[]
            0: CodedKern prop_perturb(
                           perturbation,chi,t_tot)
```

Australian Government
Bureau of Meteorology

Using a Transformation

- Import it:

```
from psyclone.psyir.transformations \
      import SomeTrans
```

Transformation are currently being refactored

- Create an instance:

```
some_trans = SomeTrans()
```

- Apply the transformation to a single node, list of nodes, or schedule:

```
some_trans.apply(some_psyir_nodes)
```

Need for Transformation Options

- Some transformations need additional parameters:
  - Loop collapse: How many nested loop to collapse
  - OMP reductions: if reproductions should be reproducible (i.e. independent of # processes….)
  - OMP parallel: whether to check for allowed node types (e.g. a write-statement would not be allowed)

  …

Parameters for Transformations

- Both `validate()` and `apply()` take one optional parameter called 'options'
- This is a Python dictionary that specifies additional, transformation-specific parameters:

```
options = {“node-type-check”: False}
some_trans.apply(some_psyir_nodes,
                 options)
```

- Note: at this stage an invalid key in the option dictionary is not flagged as an error

```
some_trans.apply(nodes,
        {‘nodde-type-check': False})
```

Putting it Together

```
def trans(psy):
    from psyclone.psyir.transformations \
         import ExtractTrans
    extract = ExtractTrans()

    invoke   = psy.invokes.get("invoke_name")
    schedule = invoke.schedule

    schedule.view()
    # Enclose everything in an extract region
    options  = {}
    extract.apply(schedule, options)

    schedule.view()
    return psy
```

Australian Government
Bureau of Meteorology

Outcome: One Node Inserted

```
InvokeSchedule[invoke='invoke_propagate_perturbation’,
               dm=False]
    0: Loop[type='', field_space='w3’, it_space=…]
         Schedule[]
            0: CodedKern prop_pert_code(pert,chi)
```

```
InvokeSchedule[invoke='invoke_propagate_perturbation’,
               dm=False]
    0: Extract[]
        Schedule[]
            0: Loop[type='', field_space='w3', it_space=…]
               Schedule[]
                    0: CodedKern prop_pert_code(pert,chi)
```

Summary

- Defined Transformation and Transformation Script
- Showed how a transformation script can use transformations
- Looked at common PSyclone functions to use

- More about transformations in next sessions

Questions (1)

- Anything so far?

PSyData API

- An interface for an object-oriented Fortran library
- A set of transformations that insert calls to this library into a PSyclone application
- The Fortran library can either be:
  - A stand-alone application
  - Use an existing third party library

Example Application (1)

- Profiling
  - The PSyData transformations specify a profiling region
  - Typically call to third-party profiling library
    - E.g. DrHook, NVIDIA, dl_timer, …
    - One simple stand-alone timer library
  - While profiling tools often provide their functions to define regions (and automatically instrument functions)
    - In some APIs a kernel computes one element
    - User can independent of tool specify region to measure

Example Application (2)

- Verify kernel parameter
  - Make sure no input parameter is NAN/infinte
  - Flag if a result of a kernel contains a NAN/infinite
  - Internally a dependency analysis determines input/output
  - Future: check if value is within a specified range
- Verify read-only parameters are not changed
  - Compiler only avoids explicit overwrites
  - This will not catch out-of-bounds array accesses (which are very expensive to test for)
  - Using checksum for fields

Example Application (3)

- Extract kernel parameter (PSyKE)
  - Write all input parameters to a file
  - Call kernel
  - Write all output parameters to the file
- Create driver that reads file and calls kernel
  - Maybe then compare results
- Useful for
  - Tuning science
  - Tuning performance
  - Creating unit tests

Planned Applications (4+5)

- In-situ visualisation
  - Plot fields while the application is running
- Compute diagnostic
  - CFL number of field

Intended Use for PSyData API

- It is mostly intended for tools
  - To help natural scientists and computer scientists to do their job
- Not as replacement for e.g. a proper IO library, or important diagnostics constantly used
  - If something is part of the application, it should not be added via PSyData

Some Details

- This slide is not actually required to use the transformations or any PSyData library
- But might give you an idea of what can be done with this interface
  - Develop your own tools
  - Suggest new tools to us
- Full details in PSyclone’s developer’s guide

Australian Government
Bureau of Meteorology

The PSyData Calls

```
type(PSyData_type), save :: psy_data_var
call psy_data_var%PreStart("mod","region", …)
call psy_data_var%DeclareVariable("a", a)
call psy_data_var%DeclareVariable(“b", b)
…
call psy_data_var%PreEndDeclaration()
call psy_data_var%ProvideVariable("a", a)
…
call psy_data_var%PreEnd()
! Execute kernel
call psy_data_var%PostStart()
call psy_data_var%ProvideVariable("b", b)
…
call psy_data_var%PostEnd()
```

The PSyData API

- Some functions are optional
  - If a transformation does not provide variables, no `Declare()` or `Provide()` functions will be used
  - E.g. profiling does only insert `PreStart()` and `PostEnd()` calls
- Many transformation will automatically select the variables
  - Based on variable usage analysis
  - Or they might take a list of parameters

Australian Government
Bureau of Meteorology

Example Script (repeated ☺)

```
def trans(psy):
    from psyclone.psyir.transformations \
         import ExtractTrans
    extract = ExtractTrans()

    invoke   = psy.invokes.get("invoke_name")
    schedule = invoke.schedule
    schedule.view()
    # Enclose everything in an extract region
    options  = {}
    extract.apply(schedule, options)

    schedule.view()
    return psy
```

Australian Government
Bureau of Meteorology

The PSyData Libraries

- Use Fortran generic interface
- Implemented using template language Jinja
  - Significantly reduces code duplication
- The PSyData libraries must be compiled and accessible when compiling the application
  - Needs the .mod files from the PSyData library
- Must be linked in with the application, before infrastructure libraries

Third Party Dependencies

- NAN-testing, read-only testing
  - No dependencies
- Kernel Extraction:
  - NetCDF
- Profiling
  - NVIDIA, DrHook, dl_timer, …
  - Simple_timing stand-alone
- In all cases the safe link order is:
  - Application, PSyData, third-party, infrastructure

Australian Government
Bureau of Meteorology

Classes of PSyData Applications

- The following PSyData classes are defined:
  - Profile, extract, read_only_verify, nan_test
- You can apply transformation from more than one class at the same time
  - Same file, even same region
- You cannot apply different transformations of the same class
  - Profiling with DrHook and NVIDIA at the same time

Australian Government
Bureau of Meteorology

Existing PSyData Transformations

- Profile:
```
from psyclone.psyir.transformations
     import ProfileTrans
```
- NAN:
```
from psyclone.psyir.transformations
     import NanTestTrans
```
- ReadOnly
```
from psyclone.psyir.transformations
     import ReadOnlyVerifyTrans
```
- Kernel Extraction
```
from psyclone.psyir.transformations
     import ExtractTrans
```

Caveats

- Installation of PSyData libraries not done yet
  - You need to install git, and pre-compile the required libraries yourself
- We will design a better way of creating transformations
  - ATM you need to know if there is an API-specific implementation or a generic one
- Some transformations are work in progress
  - Driver that reads in kernel extraction not working yet

Australian Government
Bureau of Meteorology

Summary

- Explained what PSyData is
  - Fortran object-oriented library
  - Set of transformations to insert calls into PSyclone code
- How to use the current PSyData tools

Hands-on

- Use the LFRic example from yesterday's session
  - Compileable and runnable version
- Apply kernel extraction to one kernel
- Apply kernel extraction to all kernels
- Apply NAN-checking and/or Read-only testing

- See directory tutorial/practicals/LFRic/building/4_psydata

Australian Government
Bureau of Meteorology

Thank you

Joerg Henrichs, BOM
joerg.henrichs@bom.gov.au