An Introduction to PSyclone

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

ESIWACE2 Summer School, 24th August 2020
Overview

1. Motivation
2. PSyclone
   a. What it is and what it does
   b. Modes of Operation
   c. Levels of Abstraction
3. The LFRic Domain
4. The NEMO Domain
5. Other Features
Motivation

See previous talk on DSLs but essentially:

- 3P’s: **Performance**, **Portability** and **Productivity**
  - Maintainable high performance software
  - Single-source science code
  - Performance portability

- Complex parallel code + Complex parallel architectures + Complex compilers = Complex optimisation space => unlikely to be a single solution

- Single-source optimised code is unlikely to be possible

- So … **separate science specification/code from code optimisation**
A domain-specific compiler for embedded DSL(s)
  - Configurable: FD/FV NEMO, GOcean, FE LFRic
  - Currently Fortran -> Fortran/OpenCL
  - Supports distributed- and shared-memory parallelism
  - Supports code generation and code transformation

A tool for use by HPC experts
  - Hard to beat a human (debatable)
  - Work round limitations/bugs
  - Optimisations encoded as a ‘recipe’ rather than baked into the scientific source code
  - Different recipes for different computer architectures
  - Enables scriptable, whole-code optimisation
ESiWACE2 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823988
Fparser

- Pure Python Fortran parser
- Supports Fortran 2003 + some 2008
- Open source BSD3 licence
- Developed on GitHub
- Can fully parse UM, LFRic and NEMO source
- Work-in-progress to parse IFS source
- Used by PSyclone, Stylist, Loki

https://github.com/stfc/fparser
https://fparser.readthedocs.io/
> pip install fparser
PSyclone: Two Modes of Operation

Revolution

Process code written in a DSL.

Currently two Domains supported:

- **LFRic** - Mixed finite elements, mesh unstructured in horizontal, structured in vertical, embedded in Fortran
- **GOcean** - DSL for 2D, finite difference, stretched, structured grid, embedded in Fortran

Evolution

Process existing code that follows strict coding conventions.

Recognise certain code structures and construct higher-level Internal Representation.

Transformations applied to this IR.

In development for NEMO (plus associated models, e.g. SI3, MEDUSA). Also applied to ROMS.
Levels of Abstraction

Domain-specific: LFRic IR, NEMO IR, GOcean IR

Language-independent: PSyIR

Language-specific: Fortran, C, … OpenMP, OpenACC, MPI, …
The LFRic Domain

(Revolution)
LFRic: Separation of Concerns

PSyKAI: Separate the Natural Science from the Computational Science (performance)
LFRic DSL PSy Layer

- LFRic IR
- Transformations
- DSLs
- Not DSLs!

- Alg and Kern metadata
- Other languages
- Parallel Fortran code

- PSyIR

ESIWACE2 has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 823988
LFRic DSL: Algorithm Layer Example

```
type(field_type) :: hb_inv

type(field_type), private :: grad_p
```

Logically-global field objects

Specify kernels to execute using an `invoke()`
LFRic DSL: Kernel Metadata Example

```plaintext
type, public, extends(kernel_type) :: apply_variable_hx_kernel_type
private
  type(arg_type) :: meta_args(10) = (/ 
    arg_type(GH_FIELD,  GH_WRITE, W3), 
    arg_type(GH_FIELD,  GH_READ,  W2), 
    arg_type(GH_FIELD,  GH_READ,  ANY_SPACE_1), 
    arg_type(GH_FIELD,  GH_READ,  W3), 
    arg_type(GH_OPERATOR,  GH_READ, W3, W2), 
    arg_type(GH_OPERATOR,  GH_READ, W3, ANY_SPACE_1), 
    arg_type(GH_OPERATOR,  GH_READ, ANY_SPACE_1, W2), 
    arg_type(GH_OPERATOR,  GH_READ, W3, W3), 
    arg_type(GH_REAL,     GH_READ), 
    arg_type(GH_REAL,     GH_READ) /)

  integer :: iterates_over = CELLS
contains
  procedure, nopass :: apply_variable_hx_code
end type
```
LFRic DSL: Vanilla PSy-layer Code

```
DO df=1,undef_aspc1_grad_p
  grad_p_proxy%data(df) = 0.0_r_def
END DO
DO cell=1,grad_p_proxy%vspace%get_ncell()
  !
  CALL scaled_matrix_vector_code(nlayers, grad_p_proxy%data, p_proxy%data, div_star_proxy%data, hb_inv_proxy%data, ndf_aspc1_grad_p, undef_aspc1_grad_p, map_aspc1_grad_p(:,cell), ndf_aspc2_p, undef_aspc2_p, map_aspc2_p(:,cell), ndf_w3, undef_w3, map_w3(:,cell))
END DO
DO cell=1,grad_p_proxy%vspace%get_ncell()
  !
  CALL enforce_bc_code(nlayers, grad_p_proxy%data, ndf_aspc1_grad_p, undef_aspc1_grad_p, map_aspc1_grad_p(:,cell), boundary_dofs_grad_p)
END DO
```
LFRic Transformation Example
(psyclone/examples/lfric/eg3)

Consider a simpler example where an `invoke()` contains a single, user-supplied kernel. Algorithm code:

```fortran

module solver

  type(field_type), intent(inout) :: lhs
  type(field_type), intent(in) :: rhs
  type(mesh_type), intent(in) :: mesh
  type(field_type), intent(in) :: chi(3)
  integer(i_def), intent(in) :: solver_type
  type(quadrature_type), optional, intent(in) :: qr

  subroutine solve
    call invoke( w3_solver_kernel_type(lhs, rhs, chi, ascalar, qr) )
  end subroutine solve

end module solver
```

ESIWARE2 has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 823988
LFRic Transformation Example

Corresponding PSyIR representation:

```
InvokeSchedule[invoke='invoke_0_w3_solver_kernel_type', dm=False]

0: Loop[type=' ', field_space='w3', it_space='cells', upper_bound='ncells']
    Literal[value='NOT.Initialised', Scalar<INTEGER, UNDEFINED>]
    Literal[value='NOT.Initialised', Scalar<INTEGER, UNDEFINED>]
    Literal[value='1', Scalar<INTEGER, UNDEFINED>]
    Schedule[]

0: CodedKern solver_w3_code(lhs, rhs, chi, a_scalar) [module_inline=False]
```
Transformation script:

```python
def trans psy):
    """ PSyyclone transformation script for the dynamo0p3 api to apply
colouring and OpenMP generically."
    ctrans = Dynamo0p3ColourTrans()
otrans = DynamoOMPParallelLoopTrans()

    # Loop over all of the Invokes in the PSy object
    for invoke in psy.invokes.invoke_list:

        schedule = invoke.schedule

        # Colour all of the loops over cells unless they are on
        # discontinuous spaces
        cschedule = schedule
        for child in schedule.children:
            if isinstance(child, Loop) \ 
                and child.field_space.orig_name \ 
                not in FunctionSpace.VALID_DISCONTINUOUS_NAMES \ 
                and child.iteration_space == "cells":
                cschedule, _ = ctrans.apply (child)
        # Then apply OpenMP to each of the colour loops
        schedule = cschedule
        for child in schedule.children:
            if isinstance(child, Loop):
                if child.loop_type == "colours":
                    schedule, _ = otrans.apply (child.loop_body[0])
                else:
                    schedule, _ = otrans.apply (child)
```

ESI\WACE2 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823988.
LFRic Transformation Example

Transformed PSyIR representation:

```
InvokeSchedule[invoke='invoke_0_w3_solver_kernel_type', dm=False]
  0: Directive[OMP parallel do]
    Schedule[]
      0: Loop[type='', field_space='w3', it_space='cells', upper_bound='ncells']
        Literal[value='NOT_INITIALIZED', Scalar<INTEGER, UNDEFINED>]
        Literal[value='NOT_INITIALIZED', Scalar<INTEGER, UNDEFINED>]
        Literal[value='1', Scalar<INTEGER, UNDEFINED>]
      Schedule[]
        0: CodedKern solver_w3_code(lhs,rhs,chi,ascalar) [module_inline=False]
```
LFRic Transformation Example

Generated Fortran (PSy layer):

```fortran
!$omp parallel do default(shared), private(cell), schedule(static)
DO cell=1,lhs_proxy%vspace%get_ncell()
  !
  CALL solver_w3_code(nlayers, lhs_proxy%data, rhs_proxy%data, chi_proxy(1)%data,
  a, chi_proxy(2)%data, chi_proxy(3)%data, ascalar, ndf_w3, undf_w3, map_w3(:,cell),
  bsiw3 qr, ndf_wchi, undf_wchi, map_wchi(:,cell), diff_basis_wchi qr, np_xy qr, np_z qr,
  weights_xy qr, weights_z qr)
END DO
!$omp end parallel do
```

Transformed Algorithm code:

```fortran
CALL invoke_0_w3_solver_kernel_type(lhs, rhs, chi, ascalar, qr)
```
The NEMO Domain

(Evolution)
NEMO DSL

Construct high-level representation of existing source code:
NEMO Transformation Example

Original code (tral_ldf_iso routine):

```
! DO  jn = 1, kjpt
! !
! ! I - masked horizontal derivative
! !
! bug.... why (x,:,:)? (1,jpj,:) and (jpi,1,:) should be sufficient....
zdtt (1,:,:) = 0._wp    ;    zdtt (jpi,:,:) = 0._wp
zdjt (1,:,:) = 0._wp    ;    zdjt (jpi,:,:) = 0._wp
!end

! Horizontal tracer gradient
DO  jk = 1, jpkml
  DO  jj = 1, jpjml
    DO  ji = 1, jpmml    ! vector opt.
      zdtt(ji,jj,jk) = ( ptb(ji+1,jj   ,jk,jn) - ptb(ji,jj,jk,jn) ) * umask(ji,jj,jk)
      zdjt(ji,jj,jk) = ( ptb(ji   ,jj+1,jk,jn) - ptb(ji,jj,jk,jn) ) * vmask(ji,jj,jk)
    END DO
  END DO
END DO
```

IF ( ln_zps ) THEN
  ! bottom and surface ocean correction of the horizontal gradient
  DO  jj = 1, jpjml
    ! bottom correction (partial bottom cell)
PSyIR constructed by PSyclone:
NEMO Transformation Script

```python
def trans(psy):
    """ Transform a specific Schedule by making all loops 
    over levels OpenMP parallel. 
    
    :param psy: the object holding all information on the PSy layer 
    to be modified. 
    :type psy: :py:class:`psyclone.psyGen.PSy`
    
    :returns: the transformed PSy object 
    :rtype: :py:class:`psyclone.psyGen.PSy`
    
    ..."

    from psyclone.psyGen import TransInfo
    from psyclone.nemo import NemoKern

    # Get the Schedule of the target routine
    sched = psy.invokes.get('tra_ldf_isol').schedule

    # Get the transformation we will apply
    ompt = TransInfo().get_trans_name('OMPParallelLoopTrans')

    # Apply it to each loop over levels containing a kernel
    for loop in sched.loops():
        kernels = loop.walk(NemoKern)
        if kernels and loop.loop_type == "levels":
            sched, _ = ompt.apply(loop)

    # Return the modified psy object
    return psy
```
Transformed PSyIR:

4: Directive[OMP parallel do]
   Schedule[]
   0: Loop[type='levels', field_space='None', it_space='None']
      Literal[value='1', Scalar<INTEGER, UNDEFINED>]
      Reference[name='jpkm1']
      Literal[value='1', Scalar<INTEGER, UNDEFINED>]
      Schedule[]
      0: Loop[type='lat', field_space='None', it_space='None']
      Literal[value='1', Scalar<INTEGER, UNDEFINED>]
      Reference[name='jpjm1']
      Literal[value='1', Scalar<INTEGER, UNDEFINED>]
      Schedule[]
      0: Loop[type='lon', field_space='None', it_space='None']
      Literal[value='1', Scalar<INTEGER, UNDEFINED>]
      Reference[name='fs_jpjm1']
      Literal[value='1', Scalar<INTEGER, UNDEFINED>]
      Schedule[]
      0: InlinedKern[]
         Schedule[]
         0: Assignment[]}
Generated Fortran with OpenMP directives added
Other Features

Available transformations (loop fusion, OpenMP, OpenACC, OpenCL, asynchronous halo exchanges, redundant computation)

PSyData API - allows calipers to be inserted for e.g. profiling, debugging, validation, kernel (benchmark) extraction, on-line visualisation etc.

DAG view of PSy-layer Schedules
Summary

● PSyclone is a **Domain-Specific Compiler** for use with both DSLs and existing code
● Intended as a **tool for use by an HPC expert**
● Initially developed in support of the MO LFRic Model (revolution)
● Extended to tackle existing finite difference code (evolution)
● Constructs a **PSyclone Internal Representation** of supplied code
● User transforms this representation using **Python scripts**
● Generates Fortran (or OpenCL) for the transformed PSyIR
Thank you

User, Developer and Reference Guides are available:

psyclone[-dev,-ref].readthedocs.io

For more information please contact:

rupert.ford@stfc.ac.uk
andrew.porter@stfc.ac.uk
Extras