Science and Technology Facilities Council

Hartree Centre

PSyclone: A code generation and transformation system for weather and climate DSLs

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

Ateliers de Modélisation de l'Atmosphèr, 8th -10th June 2022

Science and Technology Facilities Council

Hartree Centre

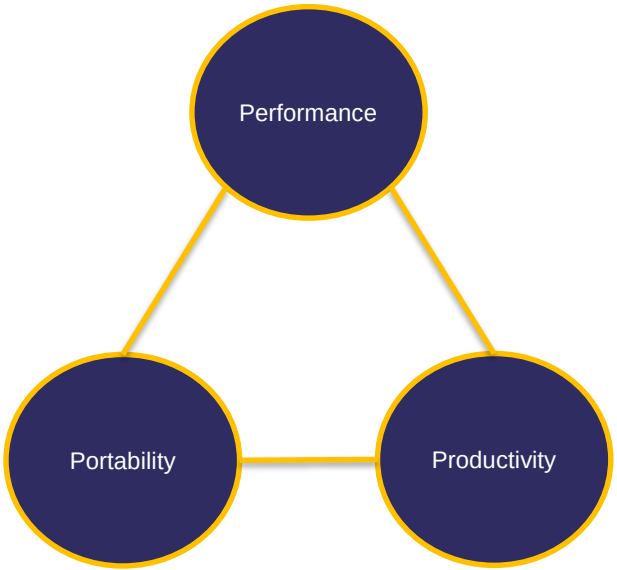
Met Office

Australian Government

Bureau of Meteorology

- Motivation
- PSyclone
- The LFRic Domain
- The NEMO Domain
- The GOcean Domain

The 3Ps of HPC software development



Difficult to compromise on one

Productivity: Maintainable software, many people contributing with different areas of expertise.

Single-source.

Performance Portability :

Risky to choose just one platform

Complex parallel code + Complex parallel architectures + Complex compilers = Complex optimisation space

⇒ Single-source optimised code is unlikely to be possible

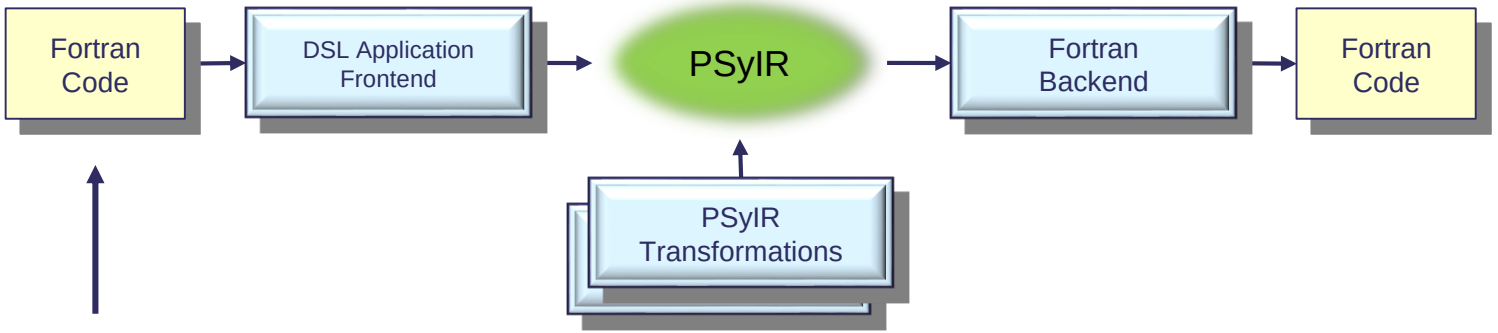
Separate science specification from code optimisation

PSyclone

PSyclone 2.3.0 BSD 3-clause
https://github.com/stfc/PSyclone
https://psyclone.readthedocs.io
```
> pip install psyclone
```

- A domain-specific compiler for embedded DSL(s)
  - Configurable: FD/FV NEMO, GOcean, FE LFRic
  - 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)
  - Optimisations encoded as a ‘recipe’ rather than baked into the scientific source code
  - Different recipes for different computer architectures
  - Enables scriptable, whole-code optimisation

PSyclone workflow

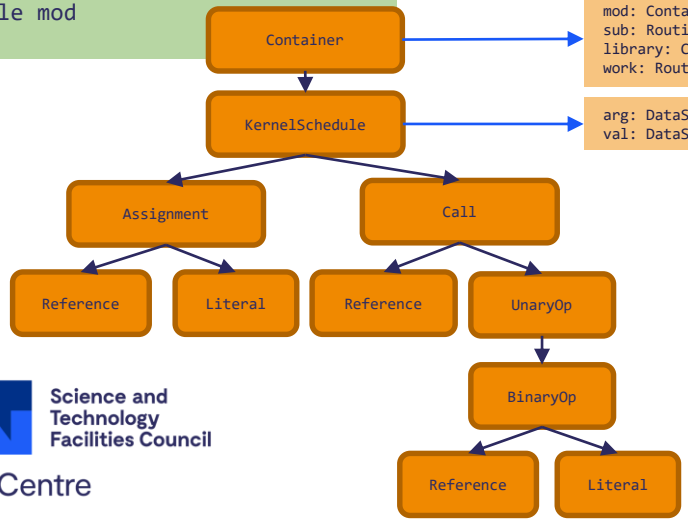


Embedded DSL with the scientific domain knowledge

Parallelisation and optimisation encoded as ‘recipes’ (Python scripts).

PSyIR :
Abstract Syntax Tree with Scoped Symbol Tables

```
module mod
  use library, only : sub
contains
  subroutine work(arg)
    real, intent(inout) :: arg
    integer :: val

    val = 3
    call sub(val, SIN(arg + 2))
  end subroutine work
end module mod
```




```
(Pdb) CONTAINER.symbol_table.view()
Symbol Table:
library: <not linked>
sub : RoutineSymbol
mod
work : RoutineSymbol

(Pdb) KERNEL_SCHEDULE.symbol_table.view()
Symbol Table:
arg: <Scalar<REAL, UNDEFINED>, Argument(pass-by-value=False)>
val: <Scalar<INTEGER, UNDEFINED>, Local>

(Pdb) CONTAINER.view()
Container[mod]
    KernelSchedule[name:'work']
        0: Assignment[]
            Reference[name:'val']
            Literal[value:'3', Scalar<INTEGER, SINGLE>]
        1: Call[name='sub']
            Reference[name:'val']
            UnaryOperation[operator:'SIN']
                BinaryOperation[operator:'ADD']
                    Reference[name:'val']
                    Literal[value:'2.0', Scalar<REAL, UNDEFINED>]
```

Science and Technology Facilities Council

Hartree Centre

Transformed with python scripts

```
GOInvokeSchedule[invoke='invoke_0', Constant loop bounds=False]
    0: Loop[type='outer', field_space='go_ct', it_space='go_internal_pts']
        Literal[value:'ssha_t%internal%ystart', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'ssha_t%internal%ystop', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: Loop[type='inner', field_space='go_ct', it_space='go_internal_pts']
                Literal[value:'ssha_t%internal%xstart', Scalar<INTEGER, UNDEFINED>]
                Literal[value:'ssha_t%internal%xstop', Scalar<INTEGER, UNDEFINED>]
                Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                Schedule[]
                    0: CodedKern continuity_code(ssha_t,sshn_t,sshn_u,sshn_v,hu,hv,un,vn,area_t)
    1: Loop[type='outer', field_space='go_cu', it_space='go_internal_pts']
        Literal[value:'ua%internal%ystart', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'ua%internal%ystop', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
```

```
def trans(psy):
    ''' Add OpenMP directives '''
    tinfo = TransInfo()
    loop_trans = tinfo.get_trans_name('GOceanOMPLoopTrans')
    parallel_trans = tinfo.get_trans_name('OMPParallelTrans')

    schedule = psy.invokes.get('invoke_0').schedule

    for loop in schedule.walk(Loop, stop_type=Loop):
        loop_trans.apply(loop)

    parallel_trans.apply(schedule.children)

    return psy
```

```
GOInvokeSchedule[invoke='invoke_0', Constant loop bounds=False]
    0: Directive[OMP parallel]
        Schedule[]
            0: Directive[OMP do]
                Schedule[]
                    0: Loop[type='outer', field_space='go_ct', it_space='go_internal_pts']
                        Literal[value:'ssha_t%internal%ystart', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'ssha_t%internal%ystop', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                        Schedule[]
                            0: Loop[type='inner', field_space='go_ct', it_space='go_internal_pts']
                                Literal[value:'ssha_t%internal%xstart', Scalar<INTEGER, UNDEFINED>]
                                Literal[value:'ssha_t%internal%xstop', Scalar<INTEGER, UNDEFINED>]
                                Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                                Schedule[]
                                    0: CodedKern continuity_code(ssha_t,sshn_t,sshn_u,sshn_v,hu,hv,un,vn,area_t)
            1: Directive[OMP do]
                Schedule[]
                    0: Loop[type='outer', field_space='go_cu', it_space='go_internal_pts']
                        Literal[value:'ua%internal%ystart', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'ua%internal%ystop', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                        Schedule[]
```

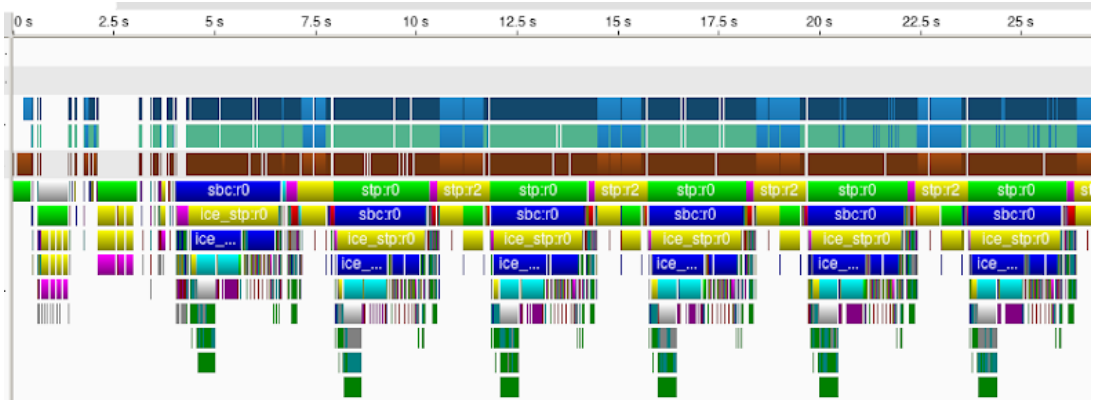
Science and Technology Facilities Council

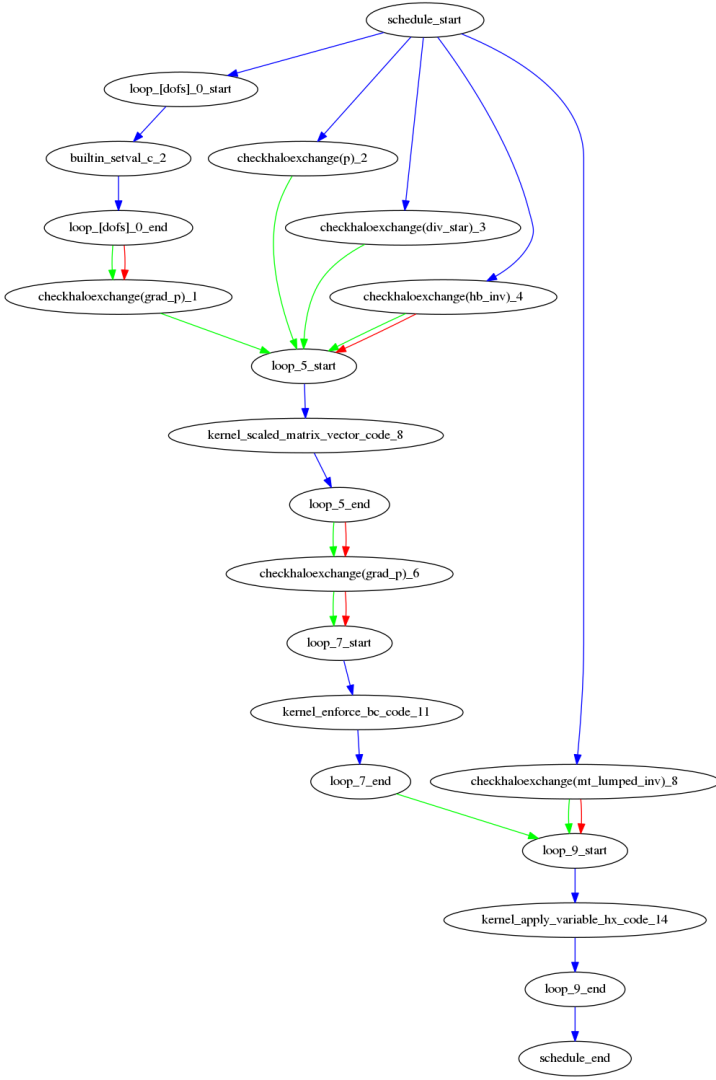
Hartree Centre

PSyclone utilities

DAG view of PSy-layer Schedules

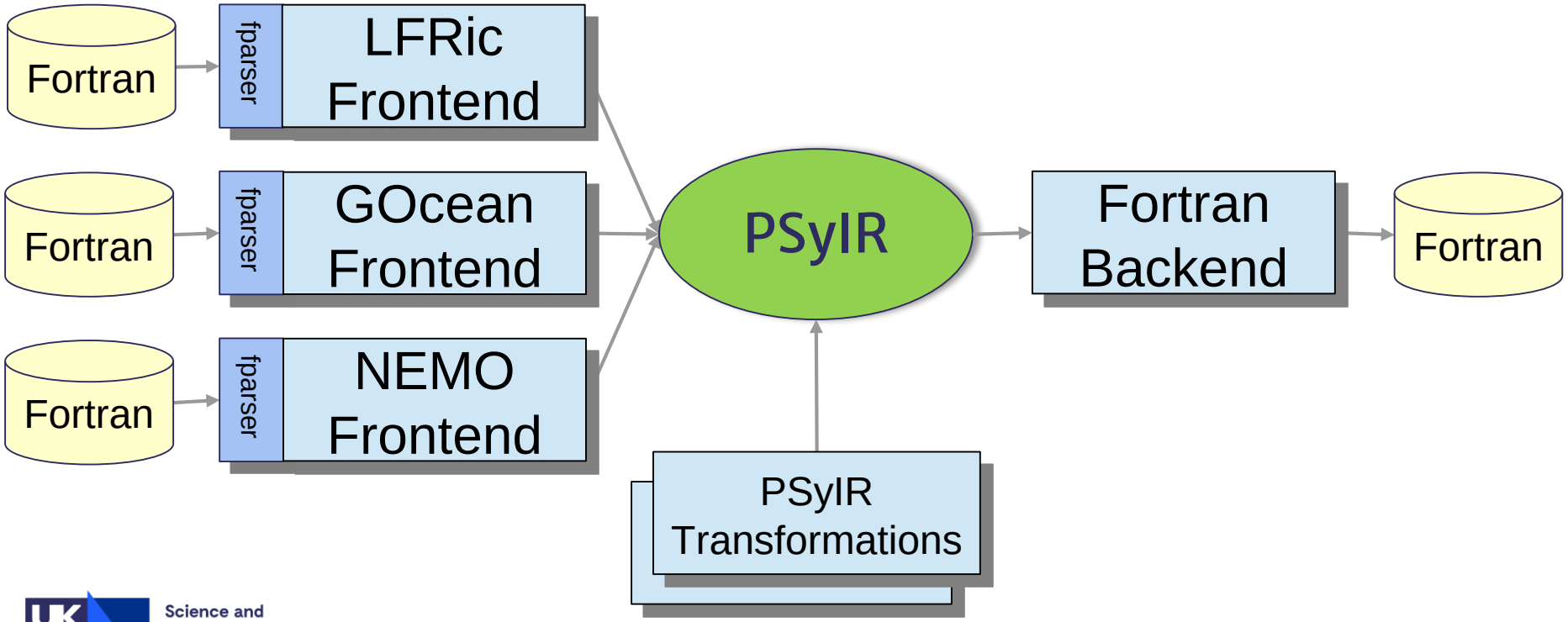
PSyData API - allows calipers to be inserted for e.g. profiling, debugging, validation, kernel (benchmark) extraction, on-line visualisation etc.





Science and Technology Facilities Council

Hartree Centre



Science and Technology Facilities Council

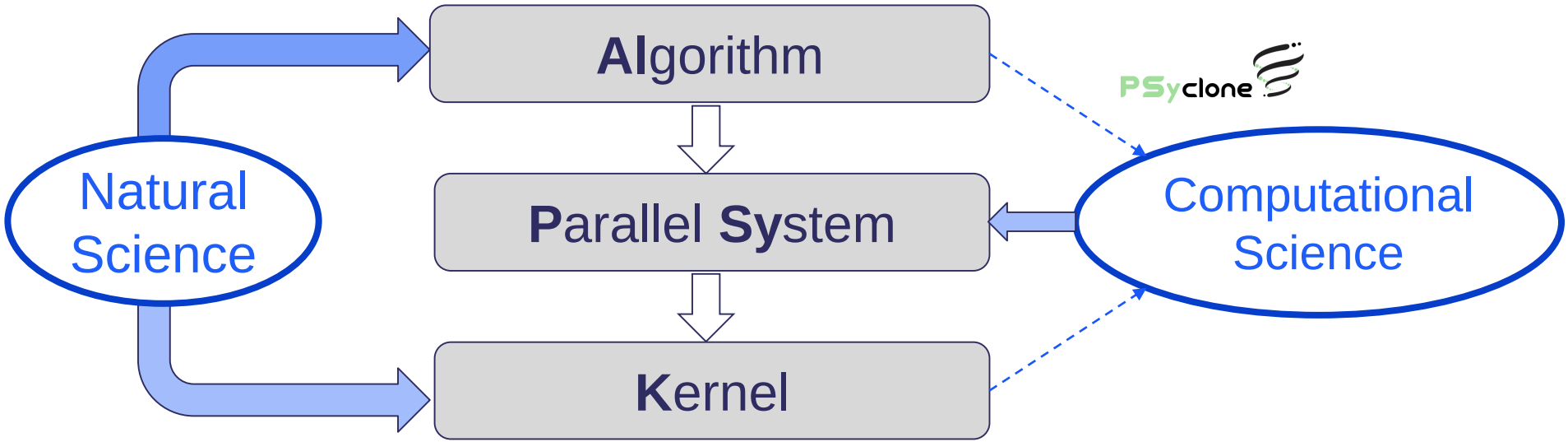
Hartree Centre

The LFRic Domain

Next generation atmospheric model of the UK MetOffice
PSyclone is used in the LFRic build system to provide parallelism and optimizations

Cubed-sphere mesh – unstructured
Mixed finite element method
Semi-implicit

PSyKAL: Separation of science from parallelisation



LFRic Algorithm-layer Example

From LFRic `norm_alg_mod.x90`

```
type(field_type),              intent(in) :: field
real(kind=r_def)                          :: l2_norm
type(operator_type),              pointer :: mass_matrix => null()
type(field_type)                          :: rhs_field
```

Logically-global field objects

```
call invoke( name = "l2_norm_calculation",                                  &
             setval_c(rhs_field, 0.0_r_def),                                &
             matrix_vector_kernel_type(rhs_field, field, mass_matrix), &
             X_innerproduct_Y(l2_norm, rhs_field, field) )
```

Specify kernels to execute using an invoke()

Kernel Example

- From LFRic `matrix_vector_kernel_mod.F90`
- Metadata for PSyclone

```
type, public, extends(kernel_type) :: matrix_vector_kernel_type
  private
  type(arg_type) :: meta_args(3) = (/                                    &
       arg_type(GH_FIELD,    GH_REAL, GH_INC,  ANY_SPACE_1),             &
       arg_type(GH_FIELD,    GH_REAL, GH_READ, ANY_SPACE_2),             &
       arg_type(GH_OPERATOR, GH_REAL, GH_READ, ANY_SPACE_1, ANY_SPACE_2) &
       /)
  integer :: operates_on = CELL_COLUMN
end type
```

```
do k = 0, nlayers-1
  do df = 1, ndf2
    x_e(df) = x(map2(df)+k)
  end do
  ik = (cell-1)*nlayers + k + 1
  lhs_e = matmul(matrix(:,:,ik),x_e)
  do df = 1,ndf1
     lhs(map1(df)+k) = lhs(map1(df)+k) + lhs_e(df)
  end do
end do
```

PSy-layer with distributed memory parallelism

```
InvokeSchedule[invoke='invoke_l2_norm_calculation', dm=True]
    0: Loop[type='dof', field_space='any_space_1', it_space='dof', upper_bound='ndofs']
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: BuiltIn setval_c(rhs_field,0.0_r_def)
    1: HaloExchange[field='rhs_field', type='region', depth=1, check_dirty=False]
    2: HaloExchange[field='field', type='region', depth=1, check_dirty=True]
    3: Loop[type='', field_space='any_space_1', it_space='cell_column', upper_bound='cell_halo(1)']
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: CodedKern matrix_vector_code(rhs_field,field,mass_matrix) [module_inline=False]
    4: Loop[type='dof', field_space='any_space_1', it_space='dof', upper_bound='ndofs']
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: BuiltIn x_innerproduct_y(l2_norm,rhs_field,field)
    5: GlobalSum[scalar='l2_norm']
```

Current LFRic optimisation script

```
# Make setval_* compute redundantly to the level 1 halo if it
# is in its own loop.
for loop in schedule.loops():
    if loop.iteration_space == "dof":
        if len(loop.kernels()) != 1:
            raise Exception(
                "Expecting loop to contain 1 call but found '{0}'".
                format(len(loop.kernels())))
        if loop.kernels()[0].name in ["setval_c", "setval_x"]:
            setval_count += 1
            rtrans.apply(loop, options={"depth": 1})

# Colour loops over cells unless they are on discontinuous
# spaces or over dofs
for loop in schedule.loops():
    if loop.iteration_space == "cell_column" \
        and loop.field_space.orig_name \
            not in const.VALID_DISCONTINUOUS_NAMES:
        ctrans.apply(loop)

# Add OpenMP to loops unless they are over colours or are null
for loop in schedule.loops():
    if loop.loop_type not in ["colours", "null"]:
        oregtrans.apply(loop)
        otrans.apply(loop, options={"reprod": True})
```

```
> psyclone -api dynamo0.3 -s ./script.py alg.x90
```

Example of generated code

```
! Set-up all of the loop bounds
!
loop0_start = 1
loop0_stop = rhs_field_proxy%vspace%get_last_dof_halo(1)
loop1_start = 1
loop1_stop = ncolour
loop2_start = 1
loop3_start = 1
loop3_stop = rhs_field_proxy%vspace%get_last_dof_owned()
!
! Call kernels and communication routines
!
!$omp parallel default(shared), private(df)
!$omp do schedule(static)
DO df=loop0_start,loop0_stop
  rhs_field_proxy%data(df) = 0.0_r_def
END DO
!$omp end do
!
! Set halos dirty/clean for fields modified in the above loop
!
!$omp master
CALL rhs_field_proxy%set_dirty()
CALL rhs_field_proxy%set_clean(1)
!$omp end master
!
!$omp end parallel
IF (field_proxy%is_dirty(depth=1)) THEN
  CALL field_proxy%halo_exchange(depth=1)
END IF
!
DO colour=loop1_start,loop1_stop
  !$omp parallel default(shared), private(cell)
  !$omp do schedule(static)
  DO cell=loop2_start,last_halo_cell_all_colours(colour,1)
    !
    CALL matrix_vector_code(cmap(colour, cell), nlayers, rhs_field_proxy%data, field_proxy%data,
                            mass_matrix_proxy%ncell_3d, mass_matrix_proxy%local_stencil, ndf_aspc1_rhs_field,
                            undf_aspc1_rhs_field, map_aspc1_rhs_field(:,cmap(colour, cell)), ndf_aspc2_field,
                            undf_aspc2_field, map_aspc2_field(:,cmap(colour, cell)))
  END DO
  !$omp end do
  !$omp end parallel
END DO
!
! Set halos dirty/clean for fields modified in the above loop
!
CALL rhs_field_proxy%set_dirty()
```

OpenMP parallelism

Distributed-memory parallelism

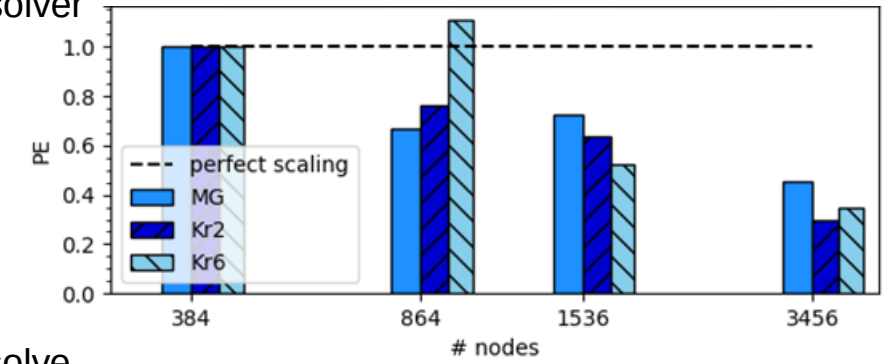
Colouring

C1152 mesh → 1152 X 1152 X 6 mesh with 30 levels – 9Km resolution
Dynamics only, (Baroclinic Wave)

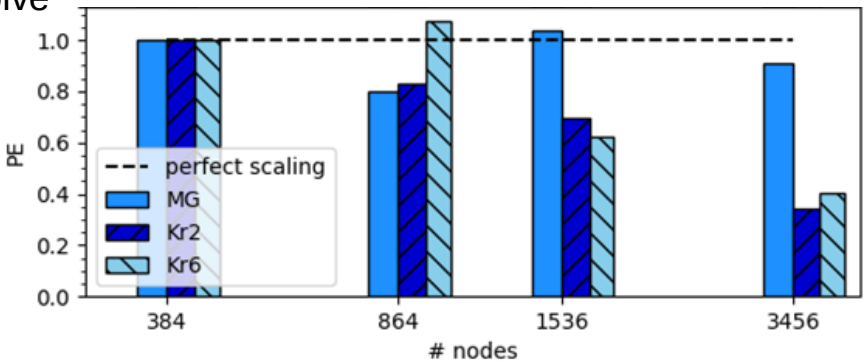
$$\mathrm{CFL_H} = \mathrm{c_s}\frac{\Delta \mathrm{t}}{\Delta \mathrm{x}} \quad c_s = 340 m/s$$

400 time-steps. **Δ**t = 205 s $\mathrm{CFL_H}$ (acoustic) ~ 8

Semi-implicit solver



Pressure solve



Strong Scalability
Intel 17 compiler
6 MPI ranks per node
6 OpenMP threads per rank
384 nodes (13824 cores)
3456 nodes (124416 cores)
Data per PE 24x24, 16x16, 12x12, 8x8

LFRic scalability study by Christopher Maynard

c.m.maynard@reading.ac.uk

Science and Technology Facilities Council

Hartree Centre

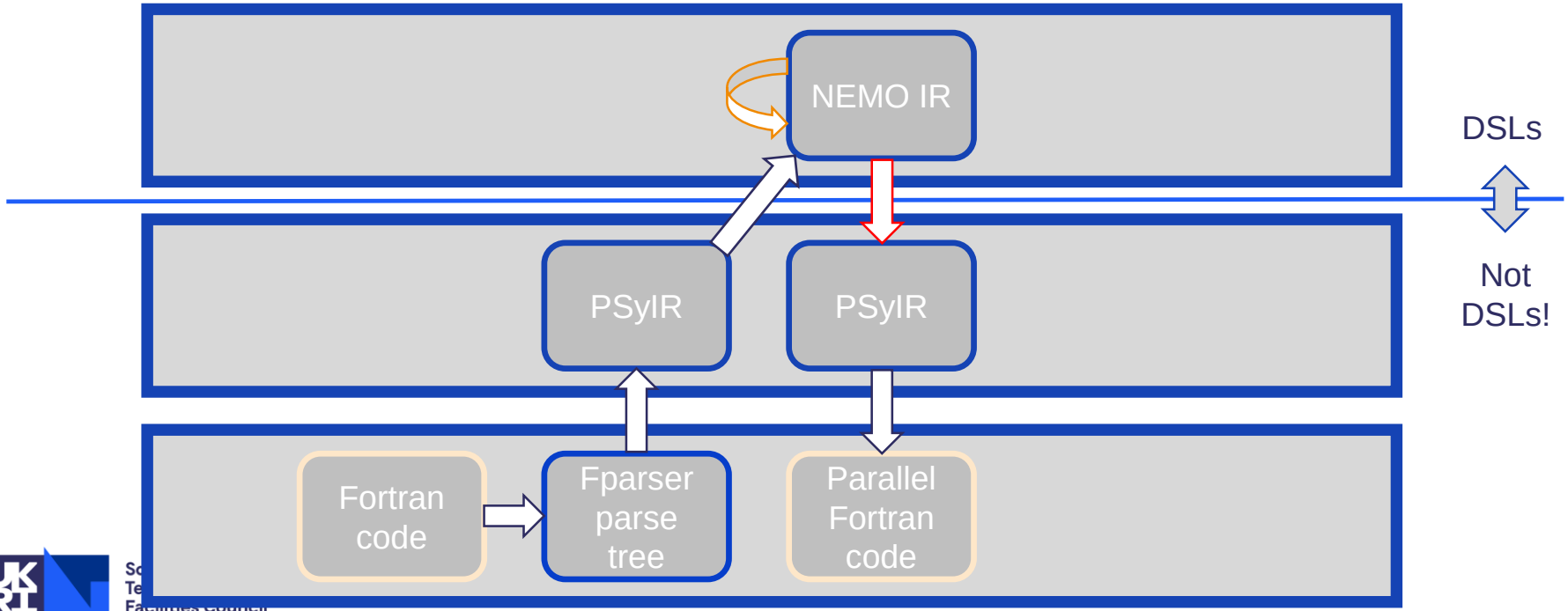
The NEMO Domain

(Evolution)

UKRI Science and Technology Facilities Council
Hartree Centre

NEMO DSL

Construct high-level representation of existing source code:



NEMO Transformation Example

(src/OCE/TRA/traadv.F90)

```
!                                      ==/==  effective transport  ==!
zun(:,:,jpk) = 0._wp
zvn(:,:,jpk) = 0._wp
zwn(:,:,jpk) = 0._wp
IF( ln_wave .AND. ln_sdw )  THEN
   DO jk = 1, jpkm1                                                    ! eulerian transport + Stokes Drift
      zun(:,:,jk) = e2u  (:,:) * e3u_n(:,:,jk) * ( un(:,:,jk) + usd(:,:,jk) )
      zvn(:,:,jk) = e1v  (:,:) * e3v_n(:,:,jk) * ( vn(:,:,jk) + vsd(:,:,jk) )
      zwn(:,:,jk) = e1e2t(:,:)                 * ( wn(:,:,jk) + wsd(:,:,jk) )
   END DO
ELSE
   DO jk = 1, jpkm1
      zun(:,:,jk) = e2u  (:,:) * e3u_n(:,:,jk) * un(:,:,jk)            ! eulerian transport only
      zvn(:,:,jk) = e1v  (:,:) * e3v_n(:,:,jk) * vn(:,:,jk)
      zwn(:,:,jk) = e1e2t(:,:)                 * wn(:,:,jk)
   END DO
ENDIF
!
IF( ln_vvl_ztilde .OR. ln_vvl_layer ) THEN                             ! add z-tilde and/or vvl corrections
   zun(:,:,:) = zun(:,:,:) + un_td(:,:,:)
   zvn(:,:,:) = zvn(:,:,:) + vn_td(:,:,:)
ENDIF
```

Uses Fortran array notation for a readable specification of the domain science

PSyIR constructed by PSyclone:

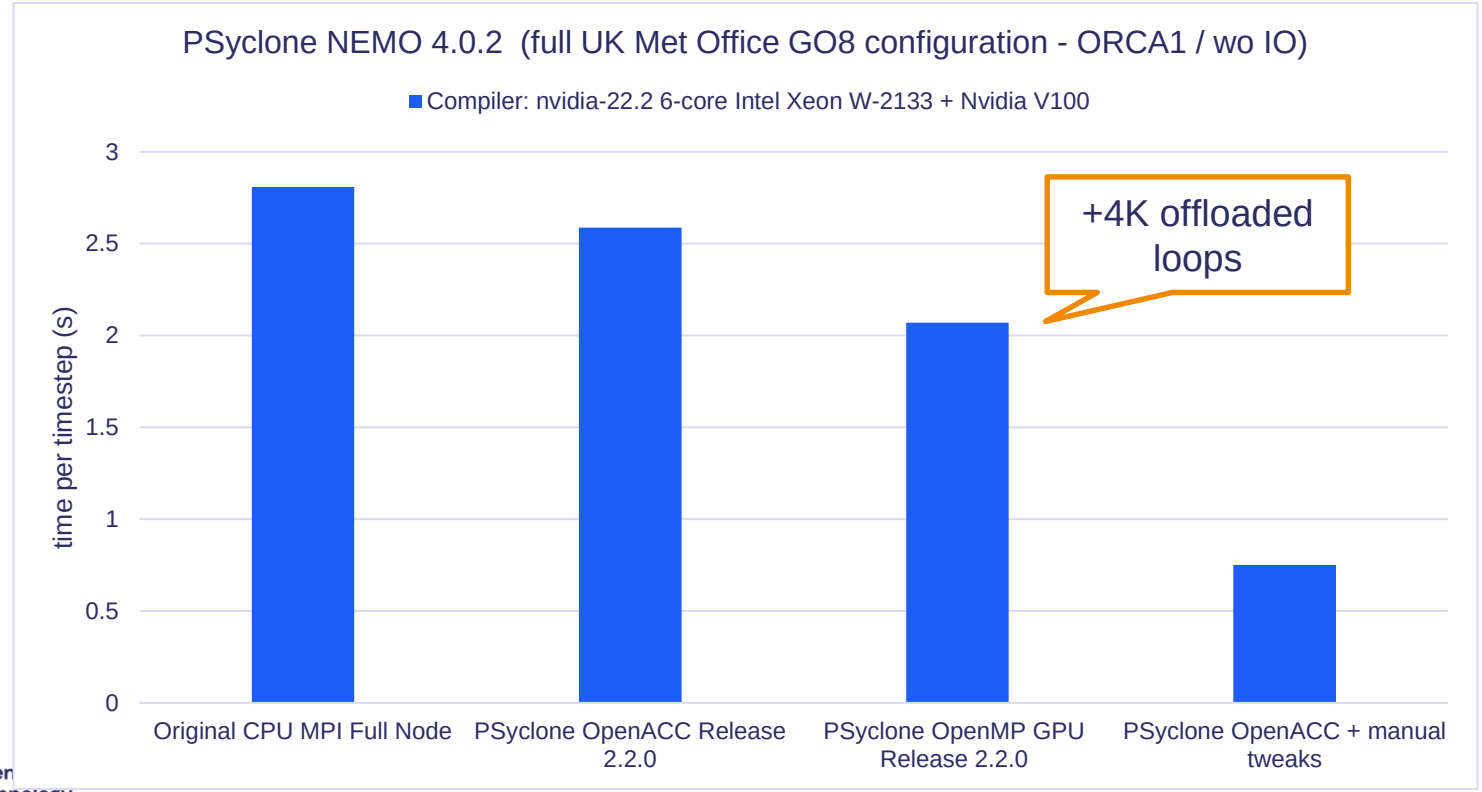
- Converts Fortran array notation to explicit loops
- Deduces domain specific knowledge from loop patterns and naming conventions of the code style-guide

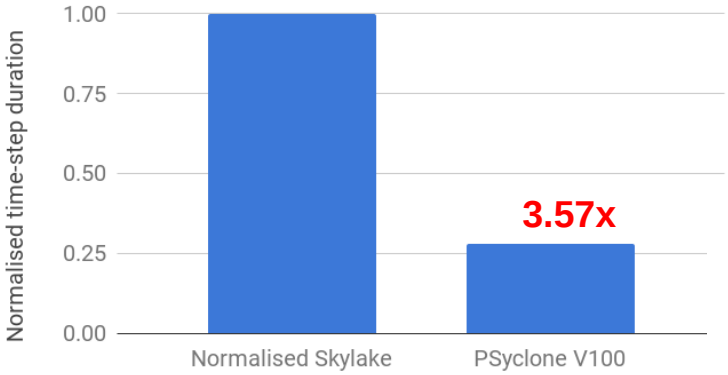
```
5: If[]
    BinaryOperation[operator:'AND']
        Reference[name:'ln_wave']
        Reference[name:'ln_sdw']
    Schedule[]
        0: Loop[type='levels', field_space='None', it_space='None']
            Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
            Reference[name:'jpkm1']
            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:'jpj']
                    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:'jpi']
                            Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                            Schedule[]
                                0: InlinedKern[]
                                    Schedule[]
                                        0: Assignment[]
                                            ArrayReference[name:'zun']
                                                Reference[name:'ji']
                                                Reference[name:'jj']
                                                Reference[name:'jk']
                                            BinaryOperation[operator:'MUL']
```

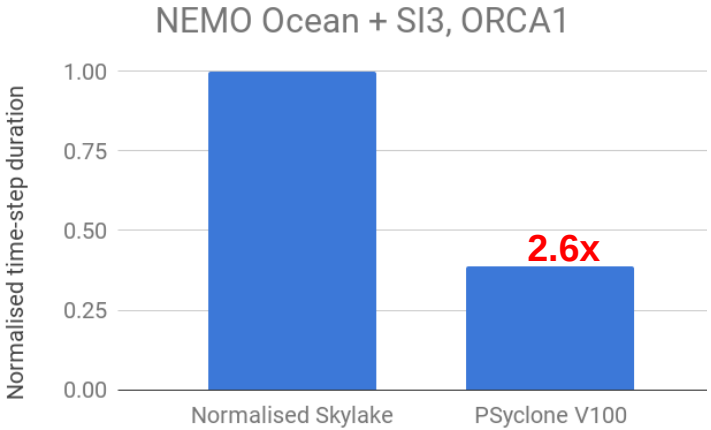
Science and Technology Facilities Council

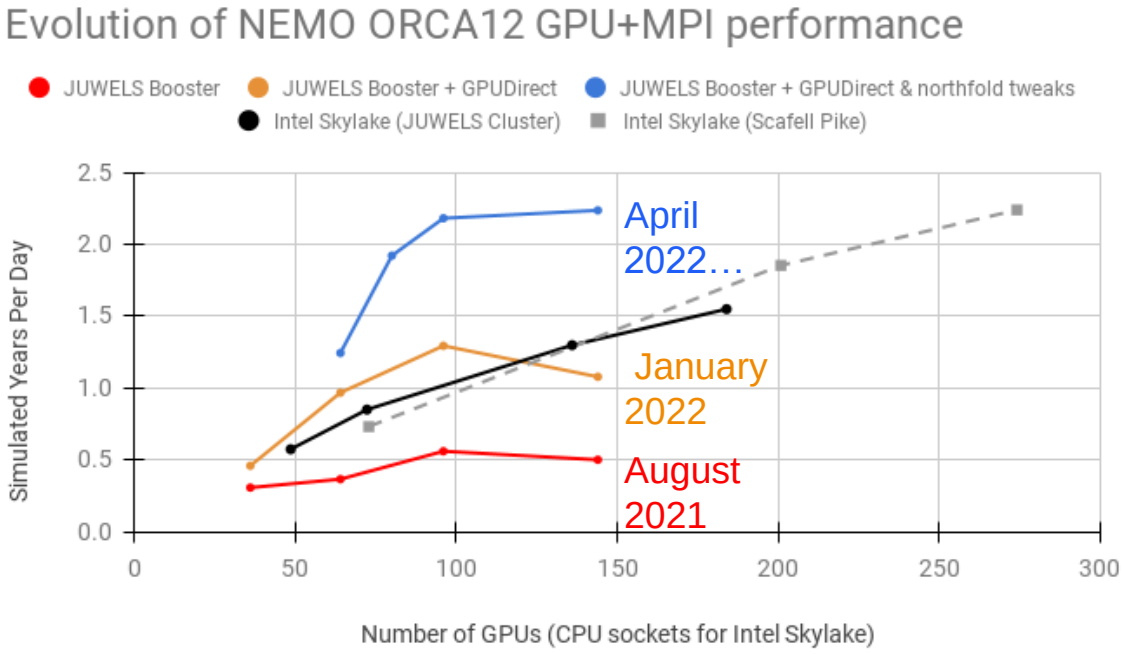
Hartree Centre

NEMO GPU offloading





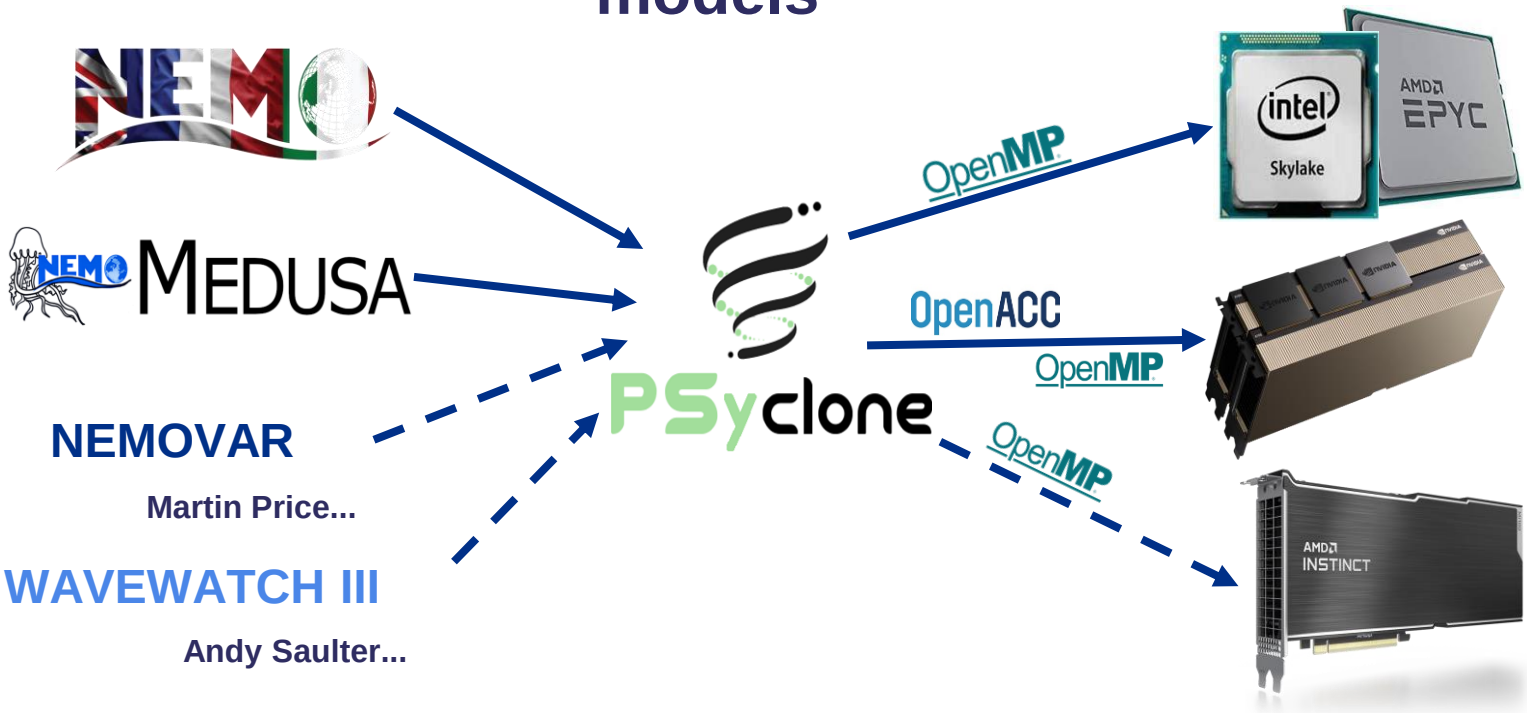




UKRI Science and Technology Facilities Council
Hartree Centre

Manual tweaks provided by Chris Dearden (chris.dearden@stfc.ac.uk)

Current status of PSyclone for various marine models



- Ongoing work to improve explicit memory management for GPU offloading
- Ongoing work to improve hybrid MPI/OpenACC offloading
- Bring remaining manual tweaks to PSyclone

The GOcean Domain

(Revolution)

Science and Technology Facilities Council
Hartree Centre

A PSyKAl-based API for Finite Difference

Directly-addressed on 2D, stretched, regular grid

Much simpler than LFRic

Used in 2 shallow-water benchmarks & a Tsunami code

Simplicity lends itself to prototyping work, e.g.:

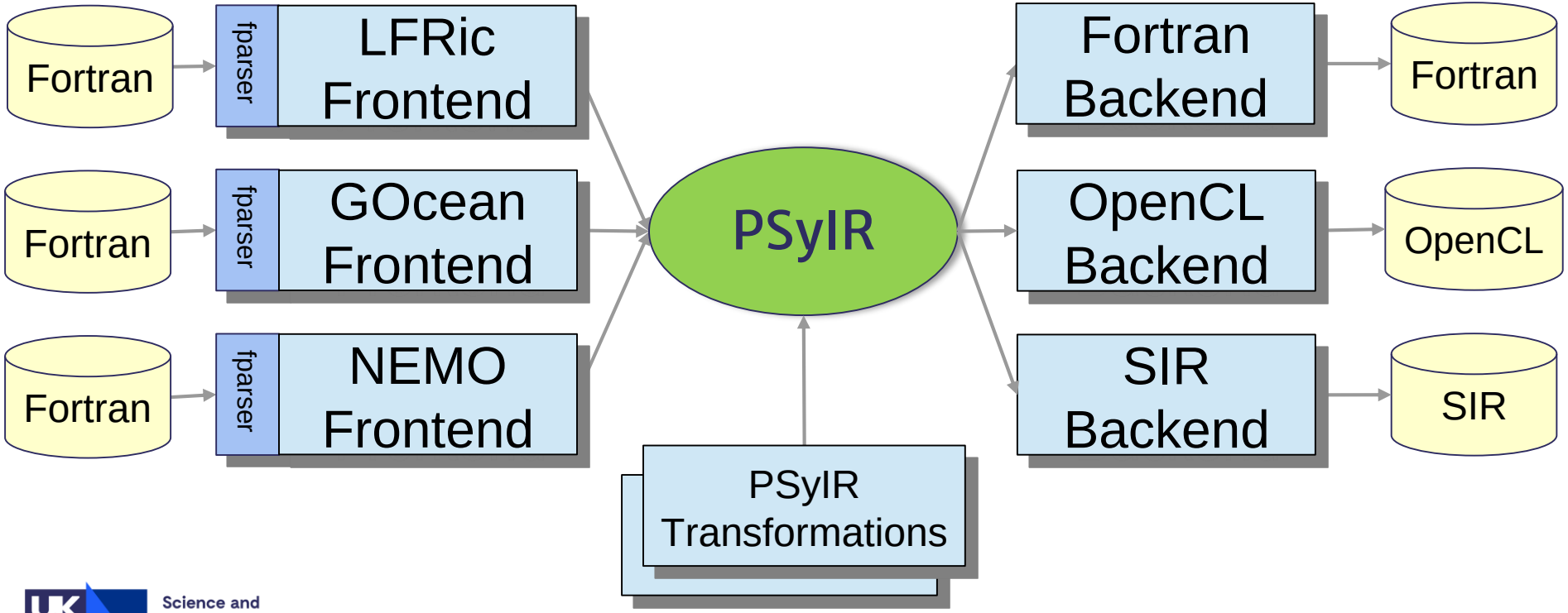
- OpenCL backends.
- kernel extraction with PSyData

See PSyclone/examples/gocean

```
type, extends(kernel_type) :: compute_cu
   type(go_arg), dimension(3) :: meta_args =    &
        (/ go_arg(GO_WRITE, GO_CU, GO_POINTWISE),            &
           go_arg(GO_READ,  GO_CT, GO_STENCIL(000,110,000)), &
           go_arg(GO_READ,  GO_CU, GO_POINTWISE)             &
         /)
   !> This kernel writes only to internal points of the
   !! simulation domain.
   integer :: ITERATES_OVER = GO_INTERNAL_PTS

   !> The U,V and F points that share the same index as a given
   !! are those immediately to the South and West of it.
   integer :: index_offset = GO_OFFSET_SW

contains
  procedure, nopass :: code => compute_cu
end type compute_cu
```

Point-wise kernels

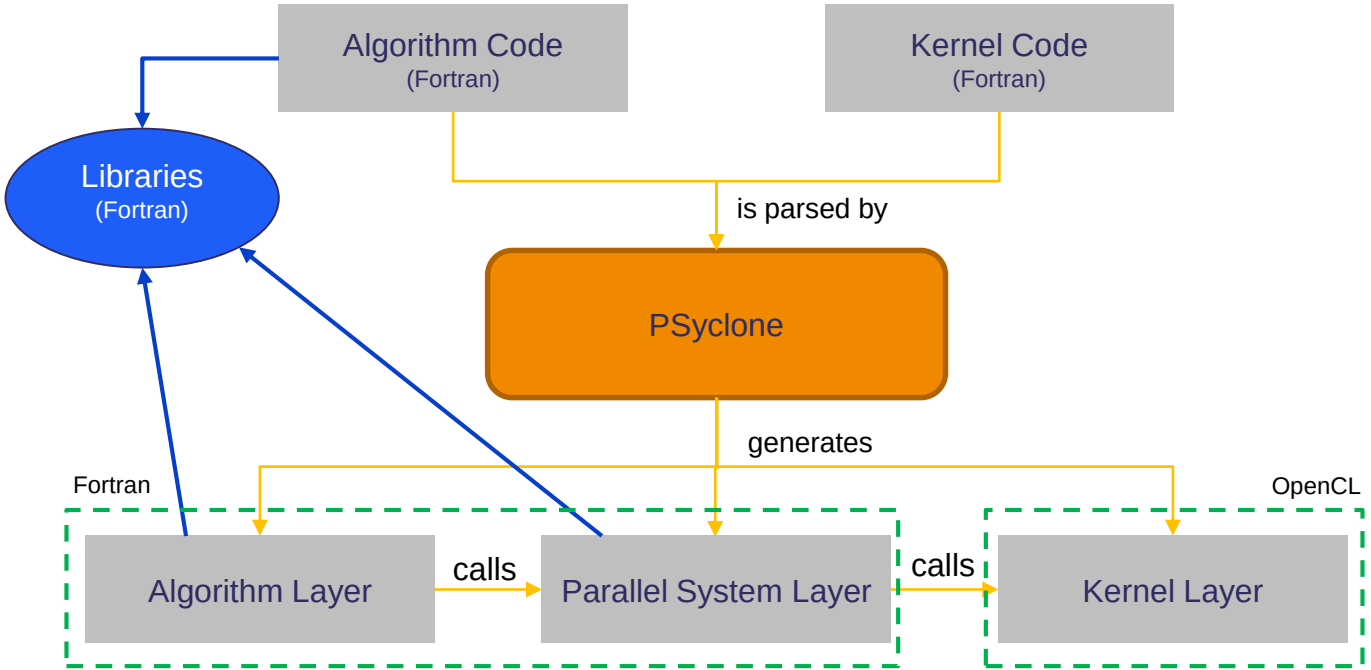
```
!> Compute the mass flux in the x direction at point (i,j)
subroutine compute_cu_code(i, j, cu, p, u)
  implicit none
  integer,  intent(in) :: I, J
  real(go_wp), intent(out), dimension(:,:) :: cu
  real(go_wp), intent(in),  dimension(:,:) :: p, u

  CU(I,J) = 0.5d0*(P(i,J)+P(I-1,J))*U(I,J)
```



Science and Technology Facilities Council

Hartree Centre

Fortran to OpenCL



Fortran to OpenCL

- OpenCL driver layer: PSyclone generates host code to control the execution of OpenCL kernels using the FortCL library (https://github.com/stfc/FortCL)
- OpenCL Kernels: PSyclone uses the PSyIR language-independent representation to generate an OpenCL version of each kernel.

```
Schedule[name:'compute_cu_code']
    Assignment[]
        ArrayReference[name:'cu']
            Reference[name:'i']
            Reference[name:'j']
        BinaryOperation[operator:'MUL']
            BinaryOperation[operator:'MUL']
                Literal[value:'0.5D0']
                BinaryOperation[operator:'ADD']
                    ArrayReference[name:'p']
                        Reference[name:'i']
                        Reference[name:'j']
                    ArrayReference[name:'p']
                        BinaryOperation[operator:'SUB']
                            Reference[name:'i']
                            Literal[value:'1']
                        Reference[name:'j']
            ArrayReference[name:'u']
                Reference[name:'i']
                Reference[name:'j']
```

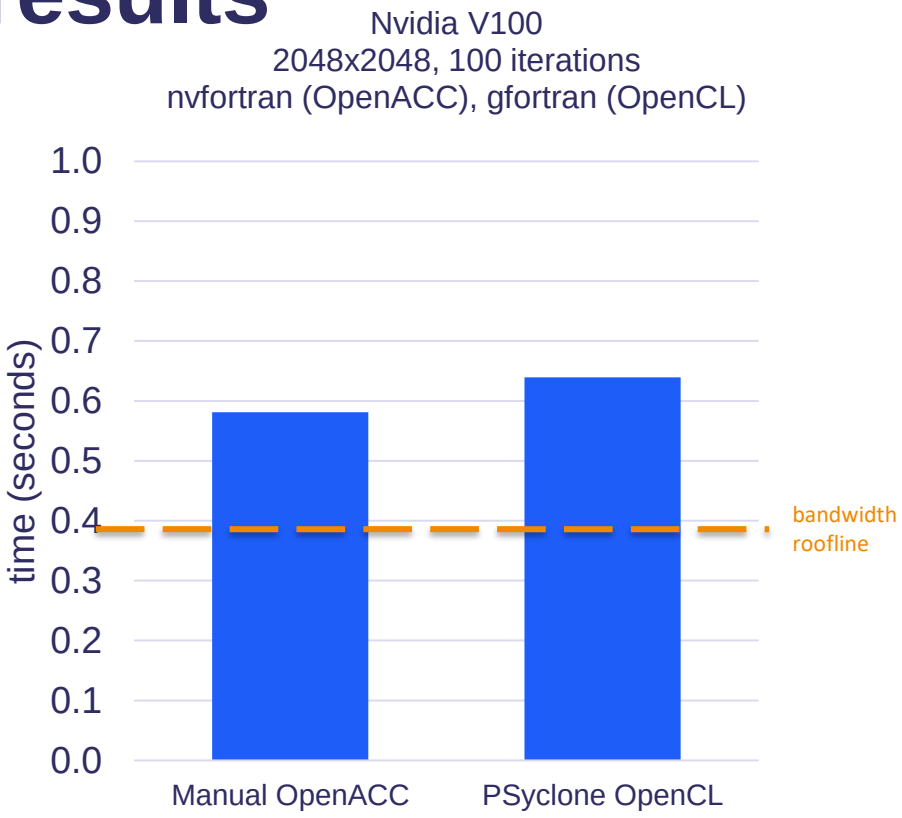
```
__attribute__((vec_type_hint(double)))
__attribute__((reqd_work_group_size(4, 1, 1)))
__kernel void compute_cu_code(
  __global double * restrict cu,
  __global double * restrict p,
  __global double * restrict u
  ){
  int cuLEN1 = get_global_size(0);
  int cuLEN2 = get_global_size(1);
  int pLEN1 = get_global_size(0);
  int pLEN2 = get_global_size(1);
  int uLEN1 = get_global_size(0);
  int uLEN2 = get_global_size(1);
  int i = get_global_id(0);
  int j = get_global_id(1);
  cu[j * cuLEN1 + i] = ((0.5e0 * (p[j * pLEN1 + i] + p[j * pLEN1 +
(i - 1)])) * u[j * uLEN1 + i]);
}
```

*Simplified view

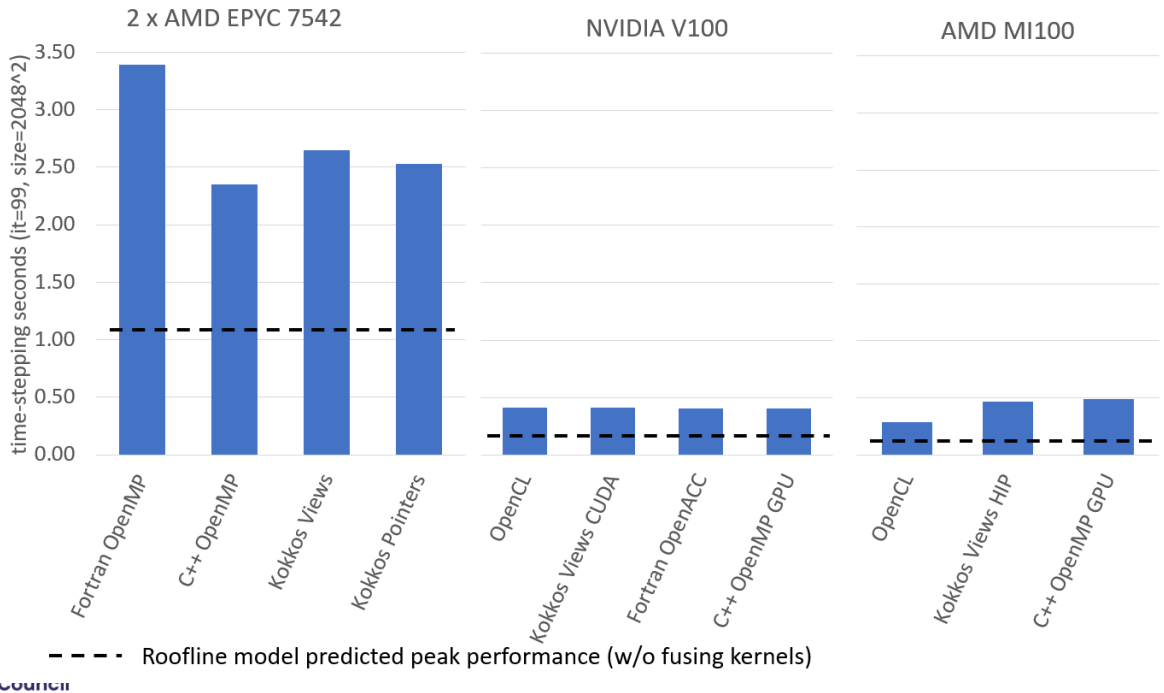
OpenCL backend results

```
$ psyclone -s ./ocl_trans.py nemolite2d.f90
```

```
global_trans = KernelGlobalsToArguments()
ocl_trans = OCLTrans()

# Get the code we want to transform
schedule = psy.invokes.get('invoke_0').schedule

# Remove global variables from kernels
for kern in schedule.kernels():
    global_trans.apply(kern)

# Transform the code to OpenCL
ocl_trans.apply(schedule)

# Specify OpenCL local size to improve
# performance
for kern in schedule.kernels():
    kern.set_opencl_options({'local_size': 64})
```



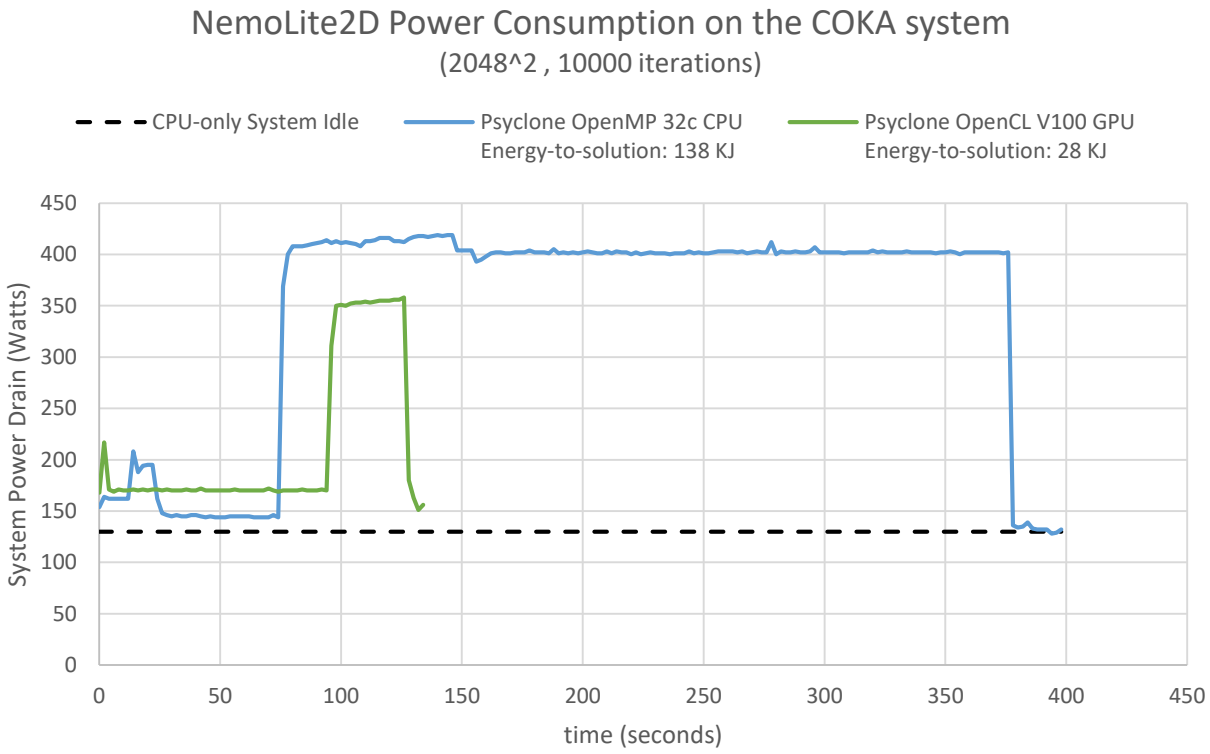
Exploring feasibility of C++ backends: e.g. Kokkos and SYCL



Science a
Technolo
Facilities Council
Hartree Centre

C++ OpenMP GPU offloading provided by Victoria Atkinson (victoria.atkinson@stfc.ac.uk)

Energy-to-Solution



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
- Used with production/full configurations:
  - LFRic (parallelises next UK MetOffice atmospheric model)
  - NEMO (being integrated in the build system as an optional prepressing step)
- Constructs a PSyclone Internal Representation of supplied code
- User transforms this representation using Python scripts
- Provides error checking and analysis functionality
- 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
sergi.siso@stfc.ac.uk

https://github.com/stfc/PSyclone

ESiWACE2 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823988

Extras...

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
```

```
PROGRAM copy_stencil
  IMPLICIT NONE
  INTEGER, PARAMETER :: n = 10, np1 = 11
  INTEGER :: i, j, k
  REAL, DIMENSION(np1, n, n) :: out, in
  DO k = 1, n
    DO j = 1, n
      DO i = 1, n
        out(i, j, k) = in(i + 1, j, k)
```

```
child type =  <class 'fparser.two.Fortran2003.Execution_Part'>
  child type =  <class 'fparser.two.Fortran2003.Block_Nonlabel_Do_Construct'>
    child type =  <class 'fparser.two.Fortran2003.Nonlabel_Do_Stmt'>
      child type =  <class 'str'> 'DO'
      child type =  <class 'fparser.two.Fortran2003.Loop_Control'>
        child type =  <class 'NoneType'>
        child type =  <class 'tuple'>
        child type =  <class 'NoneType'>
    child type =  <class 'fparser.two.Fortran2003.Block_Nonlabel_Do_Construct'>
      child type =  <class 'fparser.two.Fortran2003.Nonlabel_Do_Stmt'>
        child type =  <class 'str'> 'DO'
        child type =  <class 'fparser.two.Fortran2003.Loop_Control'>
          child type =  <class 'NoneType'>
          child type =  <class 'tuple'>
          child type =  <class 'NoneType'>
      child type =  <class 'fparser.two.Fortran2003.Block_Nonlabel_Do_Construct'>
        child type =  <class 'fparser.two.Fortran2003.Nonlabel_Do_Stmt'>
          child type =  <class 'str'> 'DO'
```

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

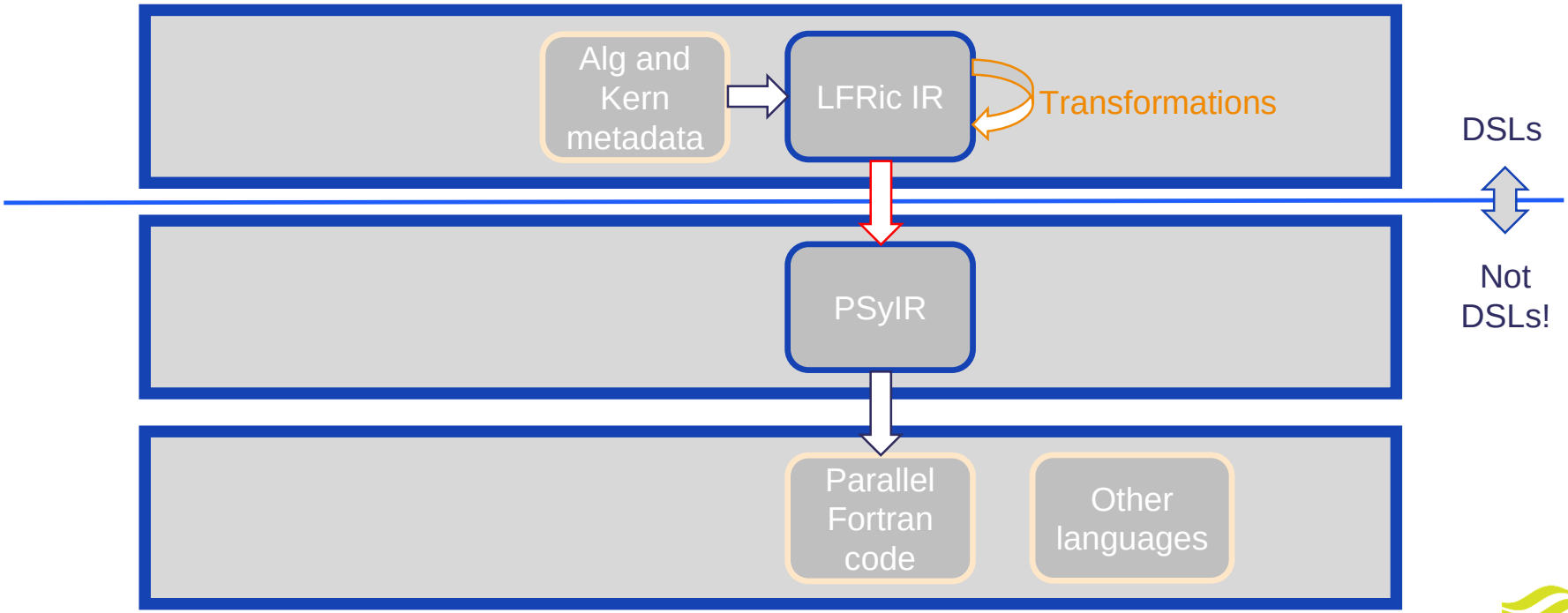
Recognise certain code structures and construct higher-level Internal Representation

Transformations applied to this IR

In development for NEMO (plus associated models, e.g. SI^3, MEDUSA). Also applied to ROMS.

Levels of Abstraction

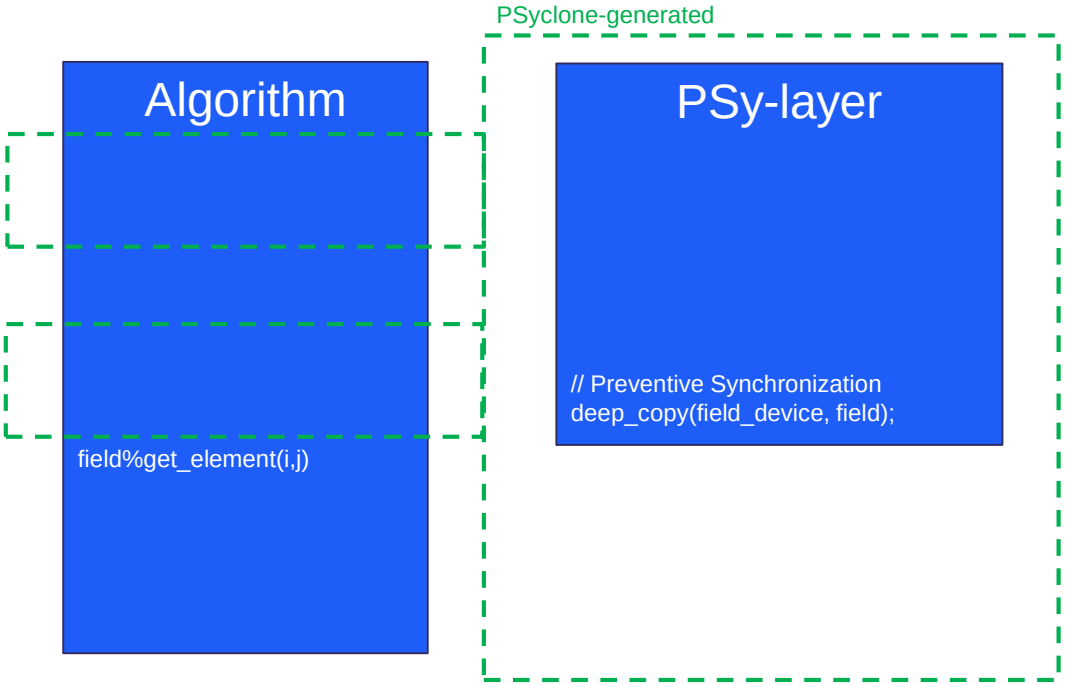
LFRic DSL PSy Layer



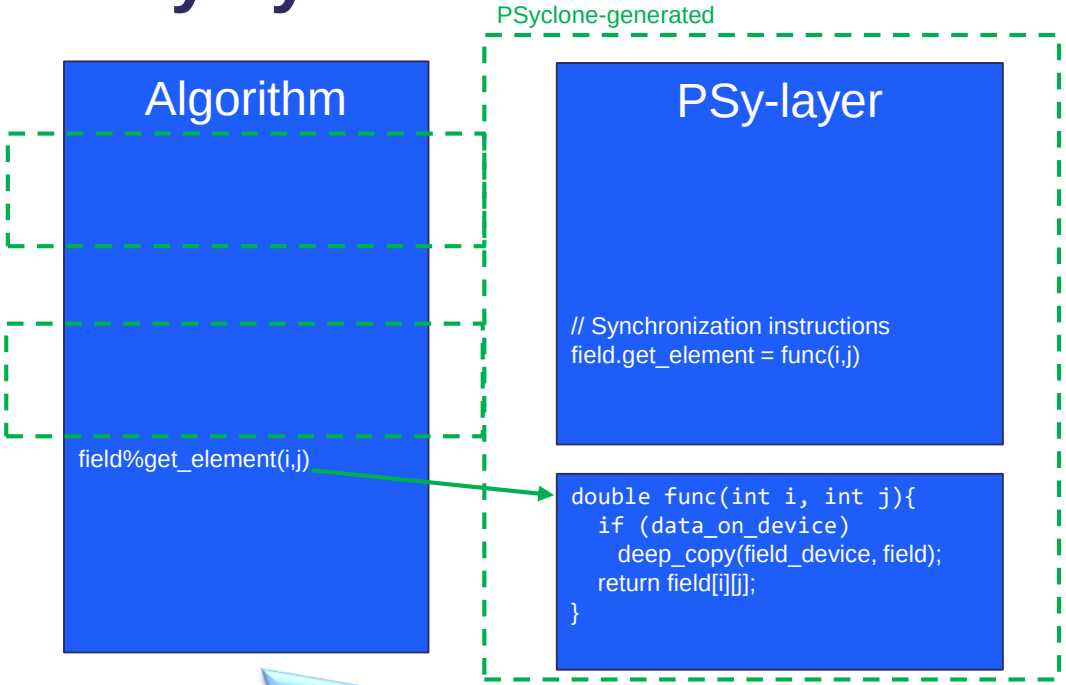
Science and Technology Facilities Council
Hartree Centre

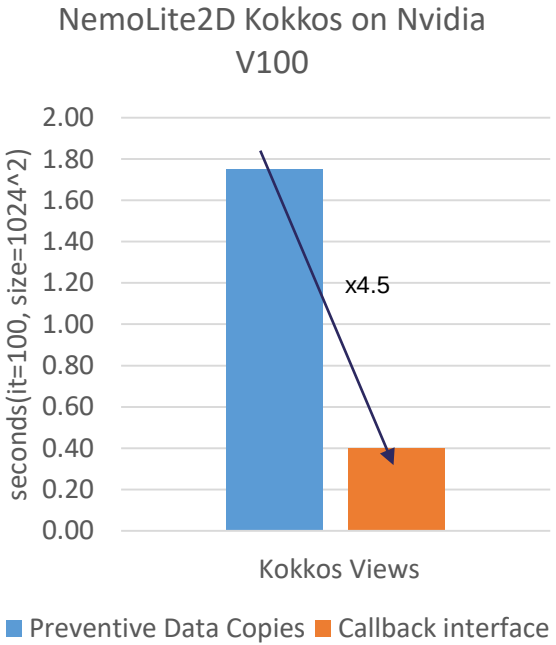
Met Office

What happens outside of PSyclone domain: Unnecessary preventive data copies



What happens outside of PSyclone domain: Lazy synchronisation to reduce data copies





The callback function makes the Algorithm parallel implementation agnostic
Tested with OpenACC, OpenCL, Kokkos and SYCL

More complex callbacks

```
read_from_device_c_interface(from, to, startx, starty, nx, ny, blocking)
write_to_device_c_interface(from, to, startx, starty, nx, ny, blocking)
```

- Support for transfer of sub-regions of data.
- Support non-blocking requests for asynchronous data transfers.

- Initially tested in the PSyclone-generated MPI+OpenCL. But we still need to explore this optimisation space.