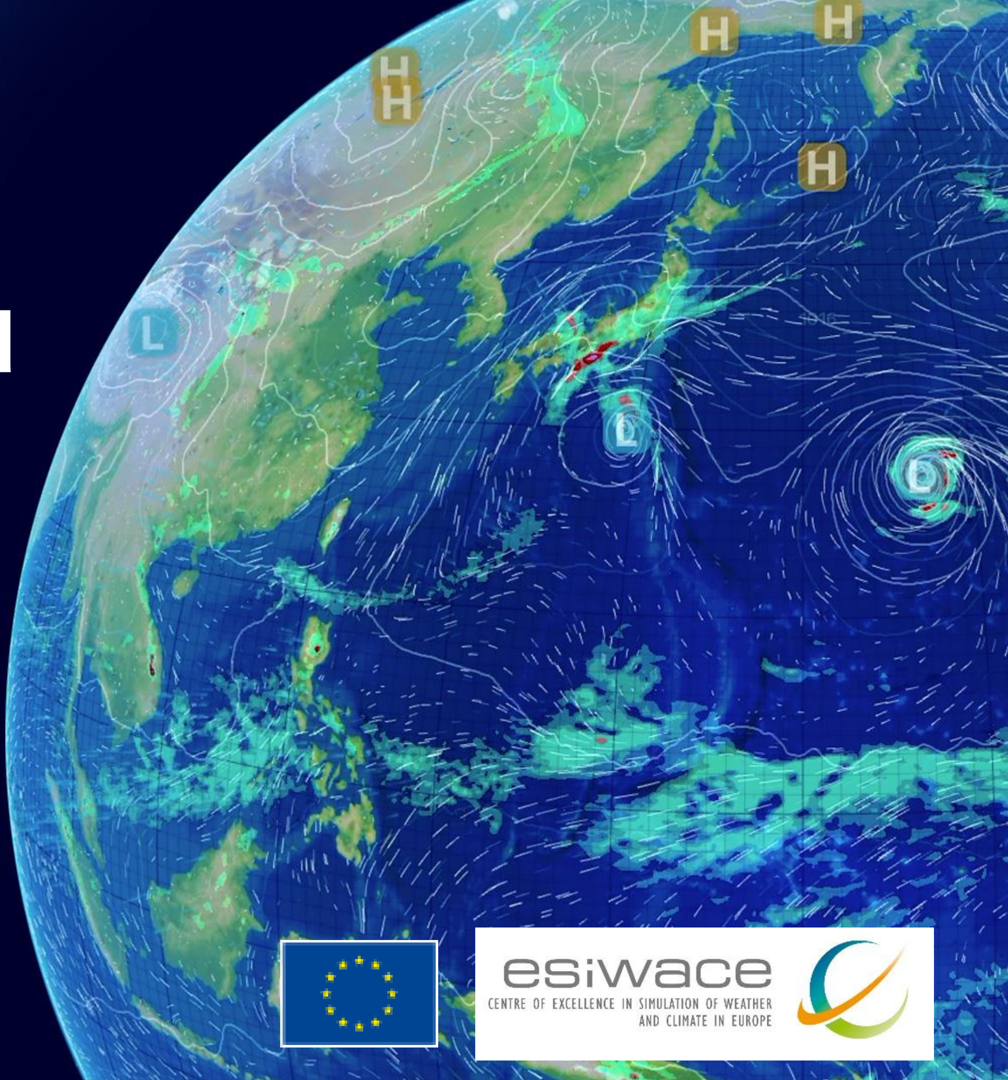
Met Office

PSyclone in Met Office: Evolution and revolution

Iva Kavcic, Met Office, UK &

Rupert Ford, **Andrew Porter**, Sergi Siso (STFC, UK); Joerg Henrichs (BOM, AU); LFRic Team, Marine Systems Team (Met Office, UK)….

7th ENES HPC Workshop, 9-11 May 2022



esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

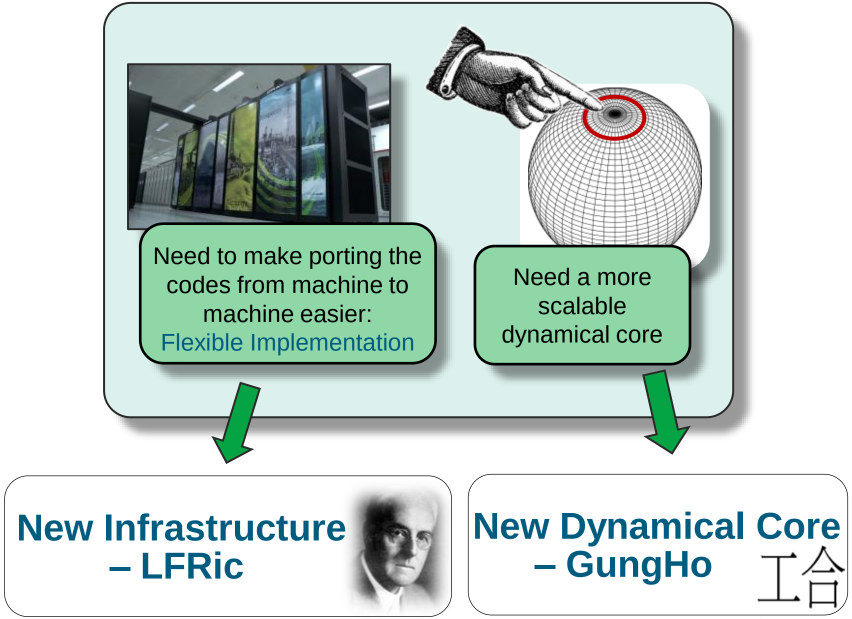
Overview

PSyclone: Motivation and intro

Revolution: LFRic API

Evolution: NEMO API

Motivation (GungHo project)

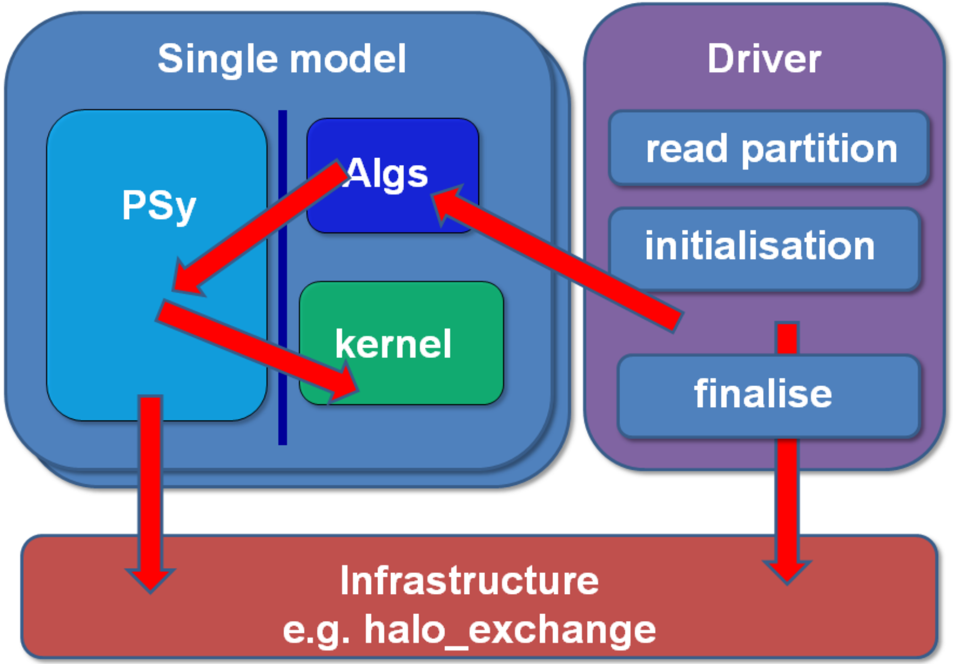


- Increased resolution → exascale computation
- Future architectures - MPI? OpenMP? Accelerators? GPUs? ARM? …?
- Scientific code can be ~ 10^6 lines (of Fortran).
- Complex parallel code + Complex parallel architectures + Complex compilers = Complex optimisation space => Single-source optimised code unlikely to be possible
- 3P's : **Performance**, **Portability** and **Productivity**
  - Maintainable high performance software
  - Single-source science code
  - Performance portability

Separation of Concerns: Science and code optimisation

PSyKAl

- **Parallel-Systems:** Computational Science, applies optimisations – ***generated code***
- **Kernels:** Natural Science, operations on (columns of) data points
- **Algorithms:** Natural Science, operations on whole **data structures** (e.g. fields)



PSyclone 2.2.0
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
  - 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

Met Office

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
```

```fortran
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)
```

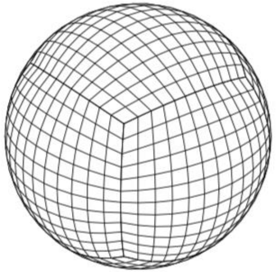
```
child type =  <class 'fparser.two.Fortran2003.Execution_Part'>
  child type =  <class 'fparser.two.Fortran2003.Block_Nonlabel_Do_Construct'>
    child type =  <class 'fparser.two.Fortran2003.Nonlabel_Do_Stmt'>
      child type =  <class 'str'> 'DO'
      child type =  <class 'fparser.two.Fortran2003.Loop_Control'>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

Revolution: LFRic API*

- Process code written in a DSL embedded in Fortran
- PSyKAl code structure
- Generated PSy layer

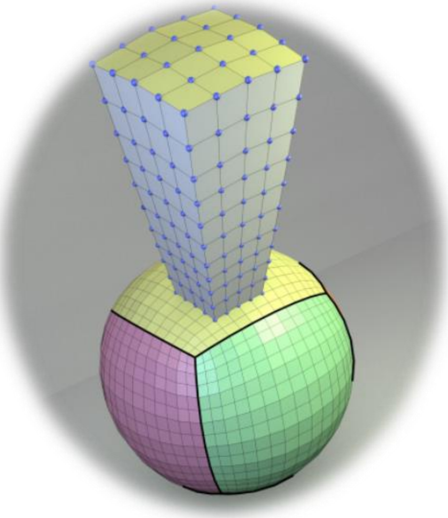
** Also GOcean API (FD, 2D structured grid)*

LFRic system & GungHo dynamical core



2D cubed-sphere mesh extruded into 3D levels

*→ **Mesh layout needs to be stored***



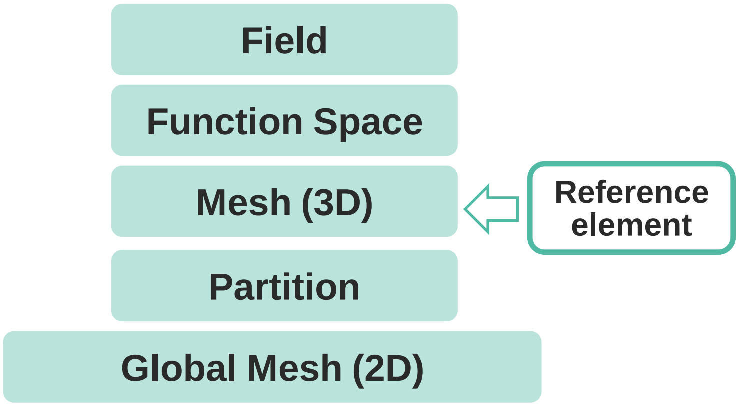
- **Mixed Finite Element Method**
- **Horizontally unstructured**, vertically structured quasi-uniform mesh
- **Generated optimisations**

- Horizontal adjacency lost
- **Vertically** adjacent cells **contiguous** in memory → operate on **columns** of data

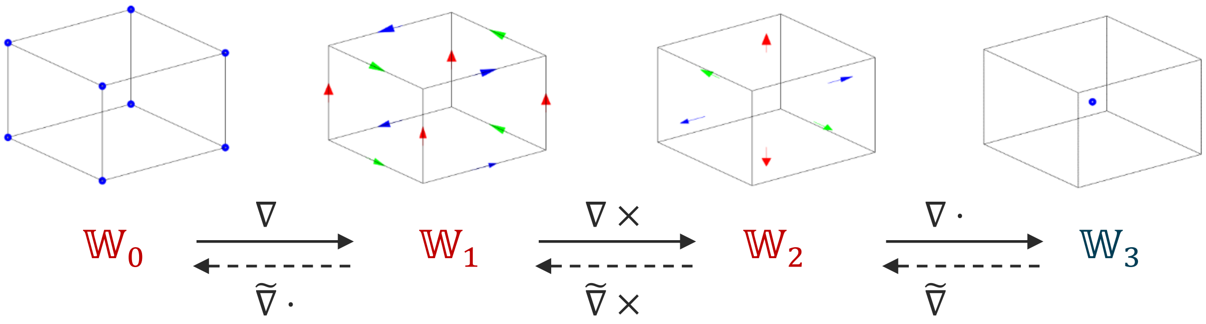
LFRic infrastructure (Object-orientated Fortran 2003)

- **LFRic: data classes**
  - Storing prognostic and diagnostic quantities: **field**;
  - Mathematical operations: **operator** (FEM matrices/FS mappings), **scalar** (global reductions).
- **Data classes** for supporting objects, e.g. mesh, reference element, function space
- **Challenge: Compiler support for OO F2003** is mixed → “compiler league table” (communication with vendors)

LFRic infrastructure: Hierarchy of objects



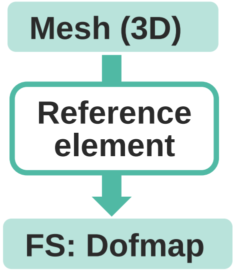
Mixed Finite Element Method (continuous + discontinuous)



Lowest order (k = 0) reference element

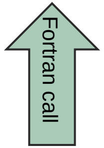
Shared DoFs (degrees of freedom): W_0 (all), W_1 (tangential components) & W_2 (normal components)

No shared DoFs: W_3



LFRic infrastructure supports two main modes of updating fields:

- Looping over **cell columns**,
- Looping over **DoFs**.

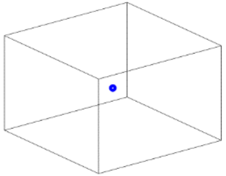


PSyclone generates calls to LFRic infrastructure support (PSy layer).

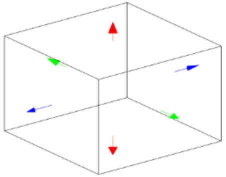
Met Office

DSL embedded in Fortran: Kernel metadata (how to **access** and **update data**; *kernel code written by scientists, Fortran 90*)

```
module rrho_kernel_mod
…
  type, public, extends(kernel_type) :: rrho_kernel_type
    private
    type(arg_type) :: meta_args(4) = (/                           &
         arg_type(GH_FIELD, GH_REAL, GH_READWRITE, W3),           &
         arg_type(GH_FIELD, GH_REAL, GH_READ,                     &
                                     ANY_DISCONTINUOUS_SPACE_1),  &
         arg_type(GH_FIELD, GH_REAL, GH_INC,       W2),           &
         arg_type(GH_FIELD, GH_REAL, GH_READ,      ANY_SPACE_1),  &
         /)
    integer :: operates_on = CELL_COLUMN
  contains
    procedure, nopass :: rrho_code
  end type
…
```



Discontinuous *function spaces: no shared DoFs*



Continuous *function spaces: shared DoFs*

DSL embedded in Fortran: Algorithm code (operations on whole fields; *written by scientists, Fortran 2003)*

```fortran
module rhs_rho_alg_mod
…
 subroutine on_the_fly_rhs_alg(rhs, state, ref_state, … )
   use rrho_kernel_mod,          only: rrho_kernel_type
   use matrix_vector_kernel_mod, only: matrix_vector_kernel_type
   implicit none
   type(field_type), target, intent(in) :: state(bundle_size)
   type(field_type), target, intent(inout) :: rhs(bundle_size)
   …
   call invoke( name = "compute_rhs_rho",                                &
                rtheta_kernel_type( rhs_tmp(igh_t), rho_ref, u, u_ref ), &
                matrix_vector_kernel_type( rhs(igh_t), theta, mm_rho ),  &
                inc_X_plus_bY( rhs(igh_t), tau_t_dt, rhs_tmp(igh_t) ) )
  …
  end subroutine on_the_fly_rhs_alg
end module rhs_rho_alg_mod
```

***Global fields:** data layout hidden*

Kernel (LFRic)

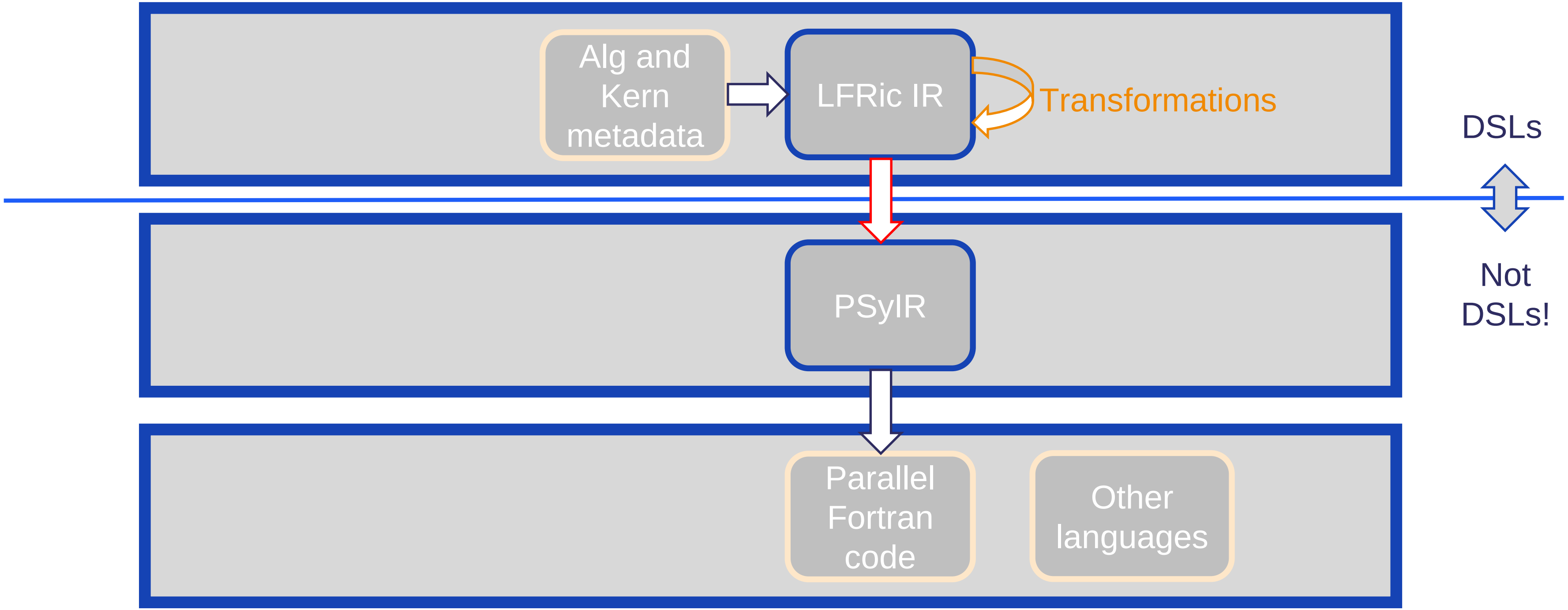
- PSy-layer loop over **columns of cells**

Built-in (PSyclone)

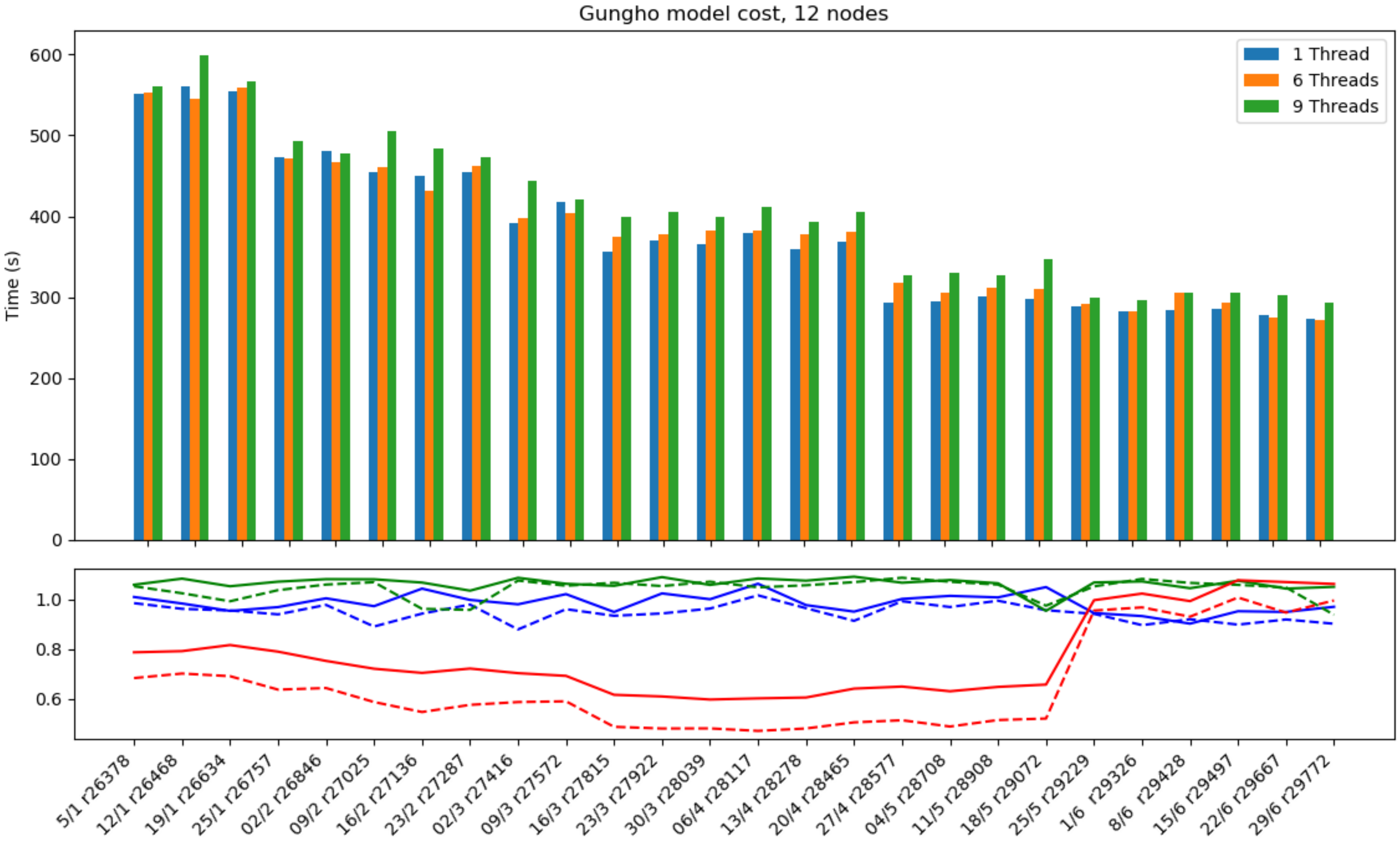
- PSy-layer loop over all field **DoFs** (arithmetic operations)

Met Office

LFRic DSL PSy Layer



Met Office



GungHo dynamical core, Held-Suarez test case:

- C192 MG ($6*192^2$ columns ≈ 50 km hor
- L30 DCMIP vert
- dt = 1200 s (SI)
- Local volume: 32x16 (1 OMP), 64x48 (6 OMP), 96x48 (9 OMP)
- Cray XC40, 2x18-core Broadwell

*Courtesy of **Tom Melvin**, Dynamics Research, MO*

Evolution: NEMO API*

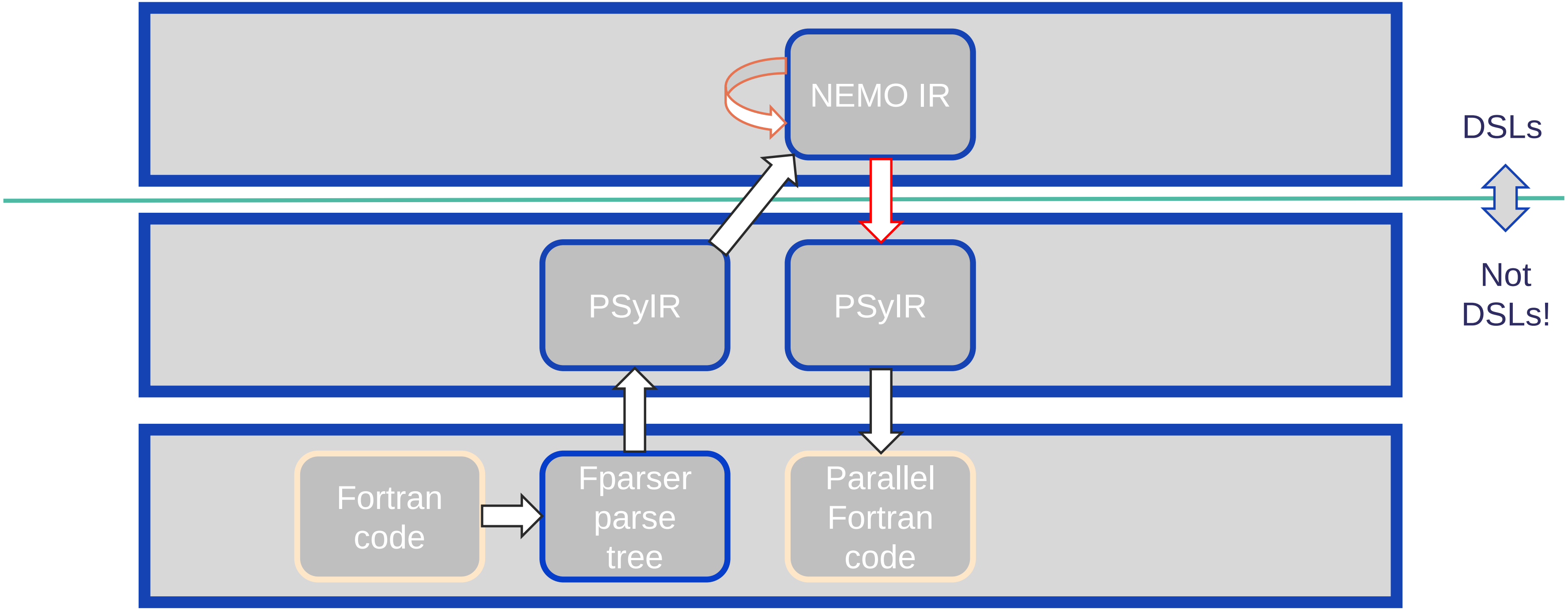
- Process existing code that follows strict coding conventions
- Recognise certain code structures and construct higher-level Internal Representation
- Transformations applied to this IR

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

Met Office

NEMO DSL

Construct high-level representation of existing source code:



Met Office Science & Technology Facilities Council esiwace CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

NEMO transformation example

(PSyclone/examples/nemo/eg2)

Original NEMO source code (`tral_ldf_iso` routine):

Source code treated as a "manually written" PSy layer with all kernels in-lined (NEMO coding conventions)

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

   ! Horizontal tracer gradient
   DO jk = 1, jpkm1
      DO jj = 1, jpjm1
         DO ji = 1, jpim1   ! vector opt.
            zdit(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      ! botton and surface ocean correction of the horizontal gradient
      DO jj = 1, jpjm1            ! bottom correction (partial bottom cell)
```

Met Office

PSyIR constructed by PSyclone:

```
4: 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_jpim1']
                    Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                    Schedule[]
                        0: InlinedKern[]
                            Schedule[]
                                0: Assignment[]
                                    ArrayReference[name:'zdit']
                                        Reference[name:'ji']
                                        Reference[name:'jj']
                                        Reference[name:'jk']
                                    BinaryOperation[operator:'MUL']
```

Hands-on on Binder:
https://github.com/stfc/PSyclone

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_jpim1']
                            Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                            Schedule[]
                                0: InlinedKern[]
                                    Schedule[]
                                        0: Assignment[]
```

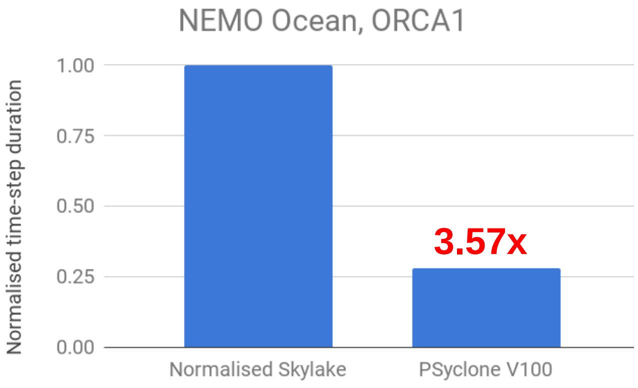
Generated Fortran with OpenMP directives added

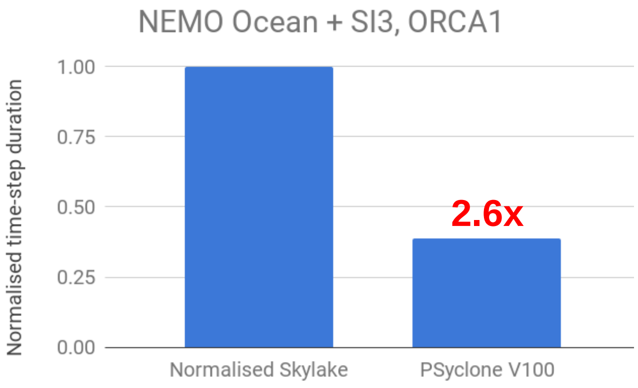
```
DO jn = 1, kjpt
  zdit(1, :, :) = 0._wp
  zdit(jpi, :, :) = 0._wp
  zdjt(1, :, :) = 0._wp
  zdjt(jpi, :, :) = 0._wp
  !$OMP parallel do default(shared), private(ji,jj,jk), schedule(static)
  DO jk = 1, jpkm1
    DO jj = 1, jpjm1
      DO ji = 1, fs_jpim1
         zdit(ji, jj, jk) = (ptb(ji + 1, jj, jk, jn) - ptb(ji, jj, jk, jn)) &
                * umask(ji, jj, jk)
         zdjt(ji, jj, jk) = (ptb(ji, jj + 1, jk, jn) - ptb(ji, jj, jk, jn)) &
                * vmask(ji, jj, jk)
      END DO
    END DO
  END DO
  !$OMP end parallel do
```

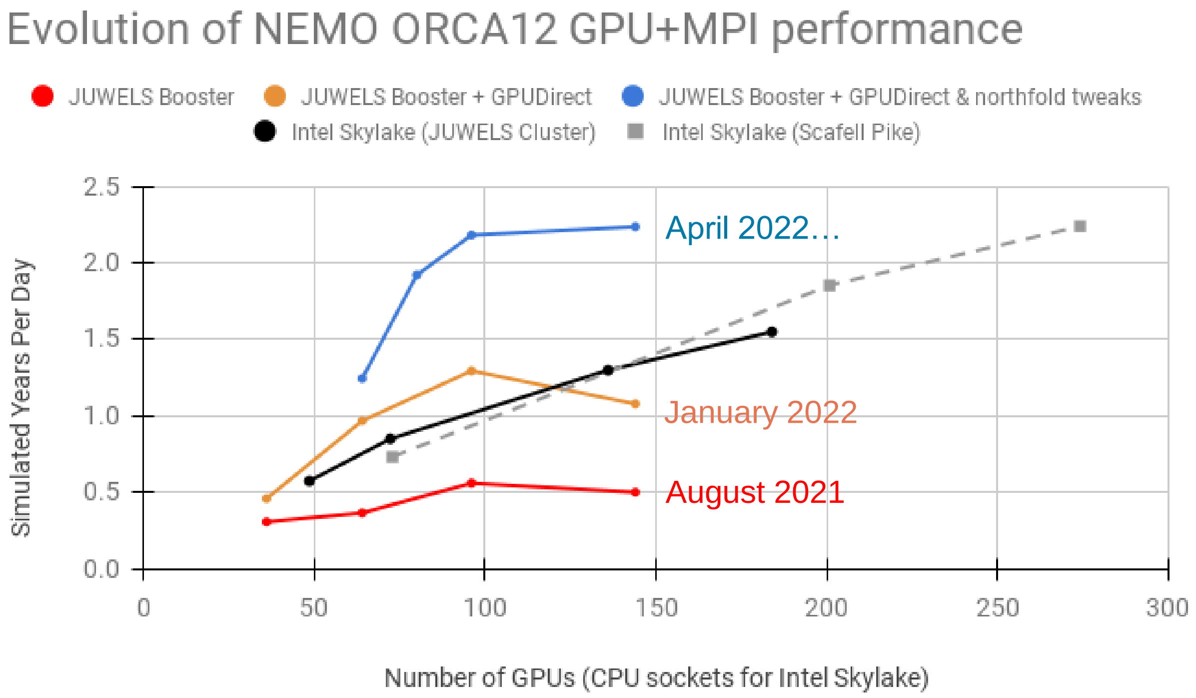
All loops parallelised over vertical levels using OpenMP

NEMO performance (May 2022)

(*Courtesy of **Chris Dearden**, STFC Hartree Centre*)







Summary (revolution meets evolution)

- Separation of concerns → flexible optimisations
- Flexible optimisations + API knowledge → performance improvements
- “Knowledge sharing” between PSyclone APIs in MO
  - NEMO API → Run LFRic on GPU (OpenACC in LFRic API)
  - PSyclone in LFRic build system → Incorporate PSyclone in building NEMO
  - Management of PSyclone in MO

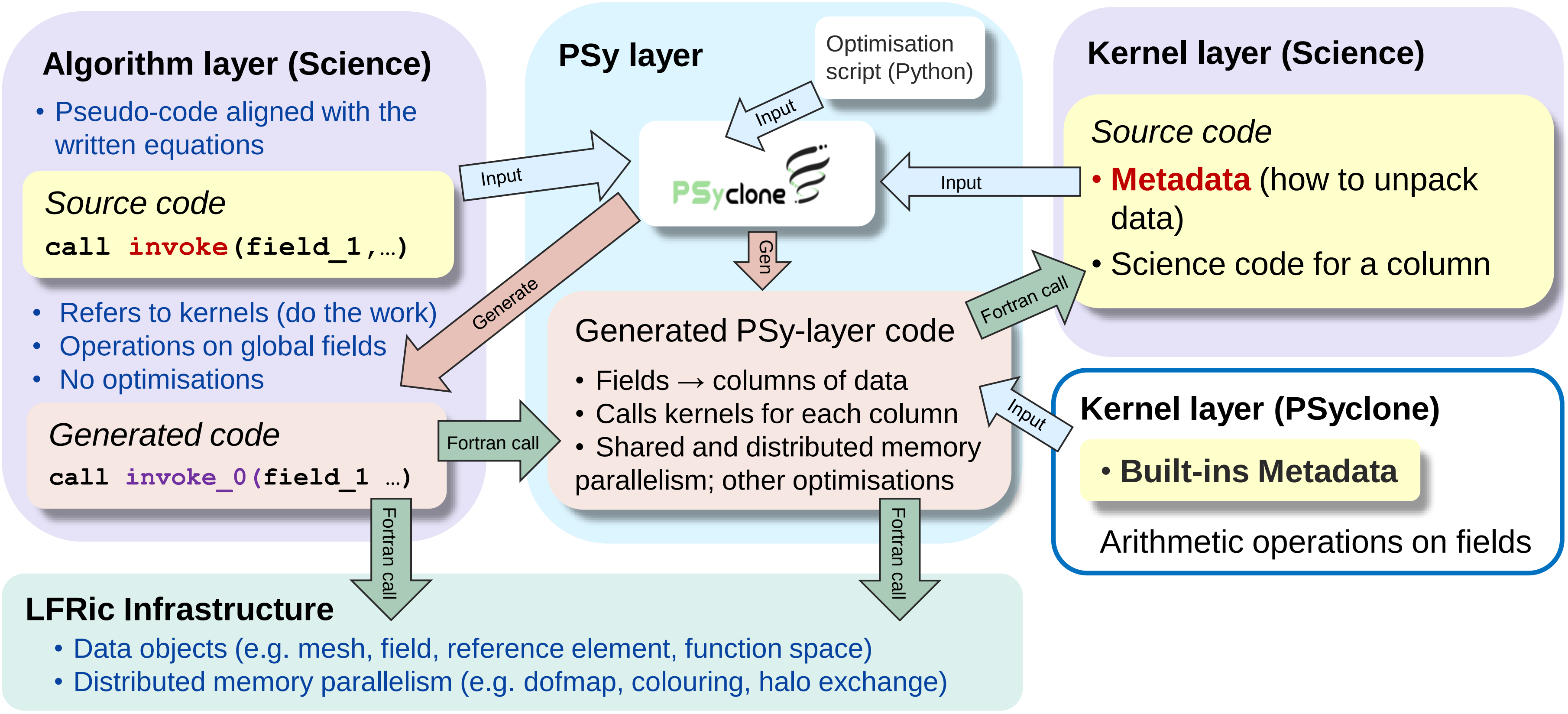
Questions?

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

Links and references

- PSyclone and fparser
  - https://github.com/stfc/PSyclone
  - https://psyclone.readthedocs.io
  - https://github.com/stfc/fparser
  - https://fparser.readthedocs.io
  - Hands-on on Binder: https://github.com/stfc/PSyclone
- LFRic: https://code.metoffice.gov.uk/trac/lfric/wiki
- PSyclone in LFRic: https://code.metoffice.gov.uk/trac/lfric/wiki/PSycloneTool
- NEMO Coding Conventions. 2013. URL: https://forge.ipsl.jussieu.fr/nemo/attachment/wiki/Documentation/NEMO_coding.conv_v3.pdf.
- Stylist: https://github.com/MetOffice/stylist
- Adams et al. (2019), *LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models*, Journal of Parallel and Distributed Computing,132, 383-396

PSyKAl Infrastructure in LFRic: Parallel-Systems, Kernels, Algorithms



Met Office

Rules of engagement: LFRic ⟷ PSyclone LFRic API

LFRic

- **Rules** (e.g. writing algorithms & kernels, data properties)
- Kernel **metadata**
- **Infrastructure support** (e.g. dofmap, colouring, MPI)

⟺

LFRic API

- Flexible **optimisations**
- Support for data structures & properties following **API rules**
- Development mandated by requirements

PSyKAl-lite

Required functionality (hand-written)

Distributed + shared memory parallelism

PSyclone transformations

- Dynamo0p3ColourTrans
- Dynamo0p3OMPLoopTrans
- OMPParallelTrans
- Dynamo0p3RedundantComputationTrans

⇩

LFRic support

- Halo exchange
- Colouring (cell & halo) for fields on continuous FS

…

NEMO transformation script

```
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_iso').schedule
    # Get the transformation we will apply
    ompt = TransInfo().get_trans_name('OMPParallelLoopTrans'
    # Apply it to each loop over levels containing a ker
    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
```

Parallelises all loops over vertical levels using OpenMP