On using a DSL Approach Performance Portability of the LFRic Weather and Climate Model

Dr Chris Maynard

Scientific Software Engineer - Met Office

Associate Professor of Computer Science – University of Reading

www.aces.cs.reading.uk

Paper DOI

https://doi.org/10.1016/j.jpdc.2019.02.007

S.V. Adams, R. W. Ford, M. Hambley, J.M. Hobson, I. Kavcic, C. M. Maynard, T. Melvin, E.H. Mueller, S. Mullerworth, A. R. Porter, M. Rezny, B. Shipway, R. Wong

Related Talks

PSyclone

MS02: Kavcic: Wed 1300 –
PSyclone and its Use in LFRic
MS29: Ford: Thur 1515 – *PSyIR: the PSy Intermediate Representation*

LFRic

MS23: CMM: Thurs 1245 - *Scalable Linear Solvers for Next Generation Weather and Climate Models*
CSM07: Poster : *Building a Performance Portable Software System for the Met Office's Weather and Climate Model, LFRic*

Programming Model

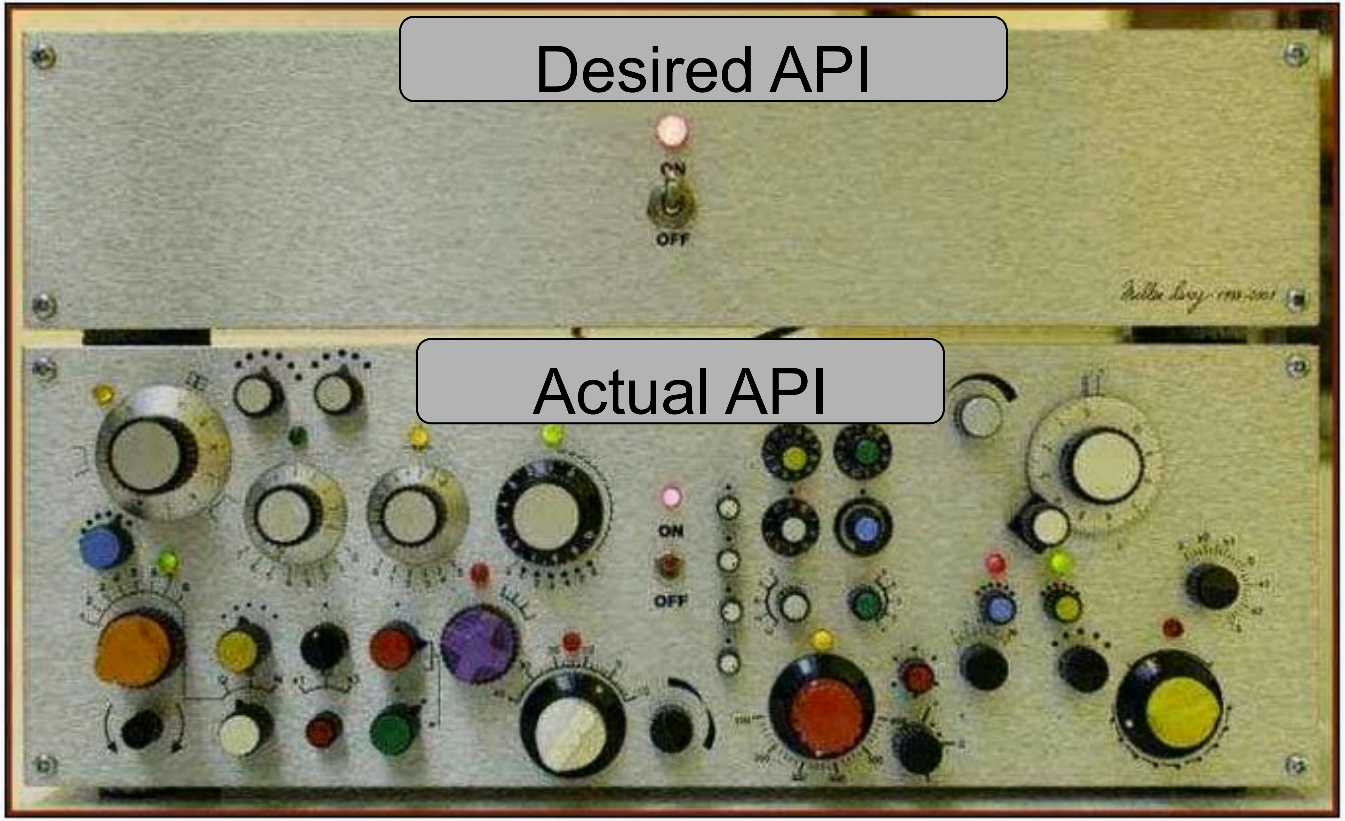
Fortran – high level language
Abstraction of the numerical mathematics
Implementation and architecture is hidden
Code – text which conforms to the semantics and syntax of the language definition
Compiler transforms code into

```
real(kind=r_def), dimension(nqp_h),     intent(in)    :: wqp_h
real(kind=r_def), dimension(nqp_v),     intent(in)    :: wqp_v

!Internal variables
integer                                               :: df, df2, k, ik
integer                                               :: qp1, qp2

real(kind=r_def),                                     :: chi1_e, chi2_e, chi3_e
real(kind=r_def)                                      :: integrand
real(kind=r_def), dime                                :: dj
real(kind=r_def), dimension(3,3,nqp_h,nqp_v)          :: jac


!loop over layers: Start from 1 as in this loop k is not an offset
do k = 1, nlayers
   ik = k + (cell-1)*nlayers

   ! indirect the chi coord field here
   do df = 1, ndf_chi
      chi1_e(df) = chi1(map_chi(df) + k - 1)
      chi2_e(df) = chi2(map_chi(df) + k - 1)
      chi3_e(df) = chi3(map_chi(df) + k - 1)
   end do

  call coordinate_jacobian(ndf_chi, nqp_h, nqp_v, chi1_e, chi2_e, chi3_e,  &
                           diff_basis_chi, jac, dj)
```

Separation of concerns

Abstraction is *broken* by parallel/performance/memory features exposed
Hacked back together with
MPI, OMP, Open ACC, OpenCL, CUDA, PGAS, SIMD, compiler directives
Libraries, languages (exts), directives and compiler (specific) directives

Programming Model II

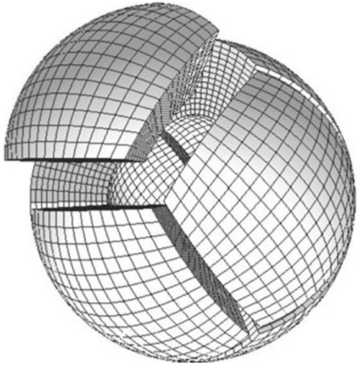


Separation of Concerns

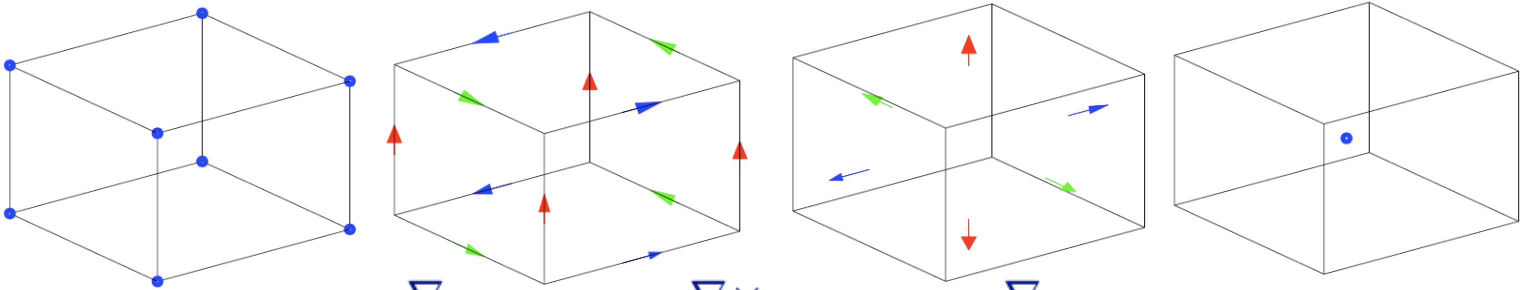
Scientific programming
Find numerical solution (and estimate of the uncertainty) to a (set of) mathematical equations which describe the action of a physical system

Parallel programming and optimisation are the methods by which large problems can be solved faster than real-time.

Gung Ho – new Dycore



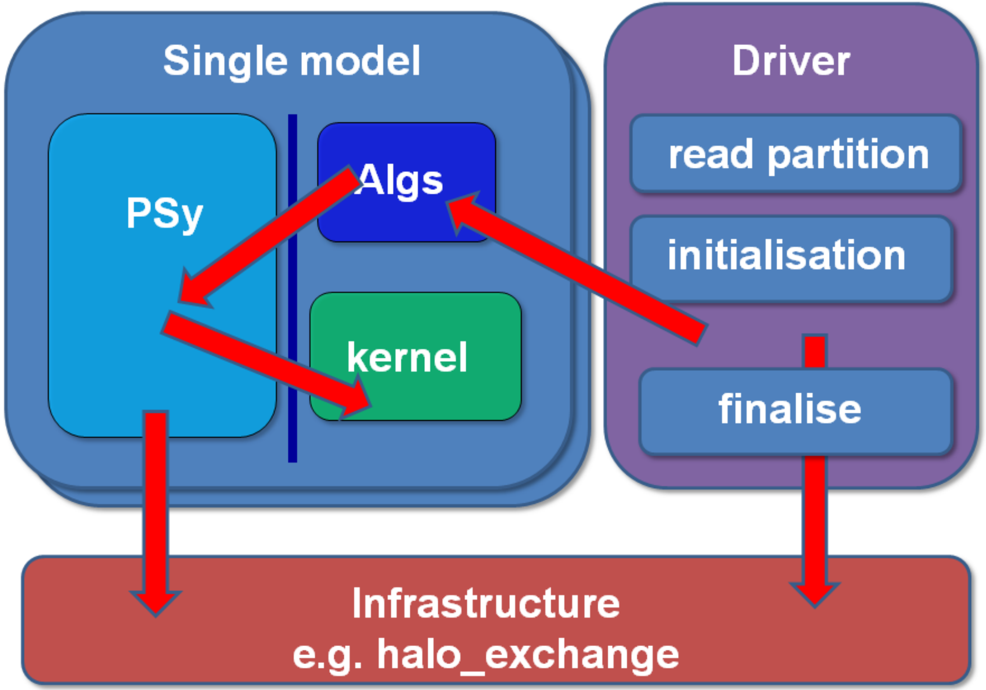
Cubed Sphere → no singular poles lon-lat
Unstructured mesh → can use other meshes
Mixed finite element scheme – *C-Grid*
Exterior calculus *mimetic* properties
Semi-implicit in time



$$
\begin{tikzcd}
\mathbb{W}_0 \arrow[r, shift left, "\nabla"] & \mathbb{W}_1 \arrow[l, dashed, shift left, "\tilde{\nabla}."] \arrow[r, shift left, "\nabla\times"] & \mathbb{W}_2 \arrow[l, dashed, shift left, "\tilde{\nabla}\times"] \arrow[r, shift left, "\nabla."] & \mathbb{W}_3 \arrow[l, dashed, shift left, "\tilde{\nabla}"]
\end{tikzcd}
$$

Layered architecture - PSyKAl

Alg layer – high level expression of operations on global fields
Kernel layer – low level Explicit operation on a single column of data
Code has to follow set of rules (PSyKAl API is DSL)
Parallelisation System
Horizontal looping and parallel code.
Can generate parallel code according to rules



Algorithm Layer

```
call invoke(                                          &
        held_suarez_kernel_type(                      &
               rhs_heldsuarez(igh_u),                 &
               rhs_heldsuarez(igh_t),                 &
               state_n(igh_u),                        &
               state_n(igh_t),                        &
               state_n(igh_d),                        &
               chi, qr) ,                             &

         enforce_bc_kernel_type(                      &
               rhs_heldsuarez(igh_u) )                &
         )
```

invoke() *Do this in parallel*

kernels *single column operations*

fields *data parallel global fields*

Multiple kernels in single invoke → scope of ordering/parallel communication, *etc*

Kernel Metadata

- Embed metadata as (compilable) Fortran, but it doesn't get executed
- Data Access descriptors
- Explicitly describe kernel arguments
- Richer information than Fortran itself

```
!-------------------------------------------------------------------------------
!> The type declaration for the kernel. Contains the metadata needed by the Psy layer
type, public, extends(kernel_type) :: exner_gradient_kernel_type
  private
  type(arg_type) :: meta_args(3) = (/                                 &
       arg_type(GH_FIELD,   GH_INC,  W2),                             &
       arg_type(GH_FIELD,   GH_READ, W3),                             &
       arg_type(GH_FIELD,   GH_READ, ANY_SPACE_9)                     &
       /)
  type(func_type) :: meta_funcs(3) = (/                               &
       func_type(W2, GH_BASIS, GH_DIFF_BASIS),                        &
       func_type(W3, GH_BASIS),                                       &
       func_type(ANY_SPACE_9, GH_BASIS, GH_DIFF_BASIS)                &
       /)
  integer :: iterates_over = CELLS
  integer :: gh_shape = GH_QUADRATURE_XYoZ
  ! gh_shape replaces evaluator_shape
  integer :: evaluator_shape = QUADRATURE_XYoZ
contains
  procedure, nopass ::exner_gradient_code
end type
```

PSyclone

Python code generator
Parser, transformations, generation
Controls parallel code (MPI/OpenMP and OpenACC)
Potentially other programming models
  e.g. OpenCL for FPGA

MS02: Kavcic: Wed 1300 –
PSyclone and its Use in LFRic
MS29: Ford: Thur 1515 – *PSyIR: the PSy Intermediate Representation*

Generated PSy layer

Update halos
YAXT→MPI

colouring from infrastructure

OpenMP workshare across cells in colour

```
!
IF (chi_proxy(3)%is_dirty(depth=1)) THEN
  CALL chi_proxy(3)%halo_exchange(depth=1)
END IF
!
CALL rhs_heldsuarez_proxy%vspace%get_colours(ncolour, ncp_colour, cmap)
!
DO colour=1,ncolour
  !$omp parallel default(shared), private(cell)
  !$omp do schedule(static)
  DO cell=1,ncp_colour(colour)
    !
     call held_suarez_code(nlayers, rhs_heldsuarez_proxy%data, &
           ! ...
           )
  END DO
  !$omp end do
  !$omp end parallel
END DO
!
! Set halos dirty for fields modified in the above loop
!
CALL rhs_heldsuarez_proxy%set_dirty()
```

kernel call for single column. Args are arrays and scalars

Psyclone transformations

Single kernel invoke

```
Transforming invoke 'invoke_26_rtheta_kernel_type' ...
Schedule[invoke='invoke_26_rtheta_kernel_type' dm=False]
    Loop[type='',field_space='w0',it_space='cells', upper_bound='ncells']
        KernCall rtheta_code(rtheta,theta,wind) [module_inline=False]
```

Apply distributed memory

```
Transforming invoke 'invoke_26_rtheta_kernel_type' ...
Schedule[invoke='invoke_26_rtheta_kernel_type' dm=True]
    HaloExchange[field='rtheta', type='region', depth=1, check_dirty=True]
    HaloExchange[field='theta', type='region', depth=1, check_dirty=True]
    HaloExchange[field='wind', type='region', depth=1, check_dirty=True]
    Loop[type='',field_space='w0',it_space='cells', upper_bound='cell_halo(1)']
        KernCall rtheta_code(rtheta,theta,wind) [module_inline=False]
```

Open MP

Simple python script to apply Open MP transformation

Can apply on whole model

Or as fine-grained as single file

```
from psyclone.transformations import Dynamo0p3ColourTrans, \
                                     Dynamo0p3OMPLoopTrans, \
                                     OMPParallelTrans

def trans(psy):
    ctrans = Dynamo0p3ColourTrans()
    otrans = Dynamo0p3OMPLoopTrans()
    oregtrans = OMPParallelTrans()

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

        print "Transforming invoke '{0}' ...".format(invoke.name)
        schedule = invoke.schedule

        # Colour loops unless they are on W3 or over dofs
        for loop in schedule.loops():
            if loop.iteration_space == "cells" and loop.field_space != "w3":
                 schedule, _ = ctrans.apply(loop)

        # Add OpenMP to loops unless they are over colours
        for loop in schedule.loops():
            if loop.loop_type != "colours":
                schedule, _ = oregtrans.apply(loop)
                schedule, _ = otrans.apply(loop, reprod=True)

        # take a look at what we've done
        schedule.view()

    return psy
```

Transformed Schedule

```
Transforming invoke 'invoke_26_rtheta_kernel_type' ...
Schedule[invoke='invoke_26_rtheta_kernel_type' dm=True]
    HaloExchange[field='rtheta', type='region', depth=1, check_dirty=True]
    HaloExchange[field='theta', type='region', depth=1, check_dirty=True]
    HaloExchange[field='wind', type='region', depth=1, check_dirty=True]
    Loop[type='colours',field_space='w0',it_space='cells', upper_bound='ncolours']
        Directive[OMP parallel]
            Directive[OMP do]
                Loop[type='colour',field_space='w0',it_space='cells', upper_bound='ncolour']
                    KernCall rtheta_code(rtheta,theta,wind) [module_inline=False]
```

Generated PSy layer

Update halos YAXT→MPI

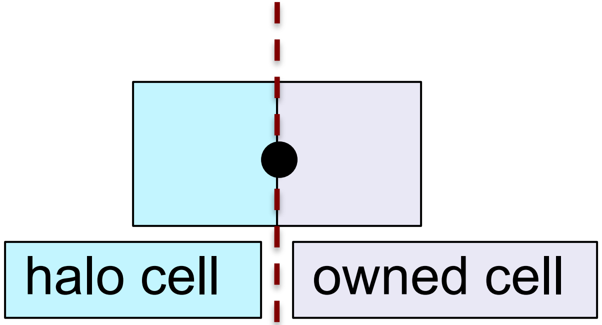
colouring from infrastructure

OpenMP workshare across cells in colour

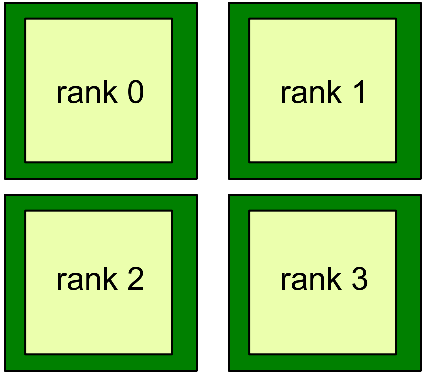
```
!
IF (chi_proxy(3)%is_dirty(depth=1)) THEN
  CALL chi_proxy(3)%halo_exchange(depth=1)
END IF
!
CALL rhs_heldsuarez_proxy%vspace%get_colours(ncolour, ncp_colour, cmap)
!
DO colour=1,ncolour
  !$omp parallel default(shared), private(cell)
  !$omp do schedule(static)
  DO cell=1,ncp_colour(colour)
    !
     call held_suarez_code(nlayers, rhs_heldsuarez_proxy%data, &
          ! ...
          )
  END DO
  !$omp end do
  !$omp end parallel
END DO
!
! Set halos dirty for fields modified in the above loop
!
CALL rhs_heldsuarez_proxy%set_dirty()
```

kernel call for single column. Args are arrays and scalars

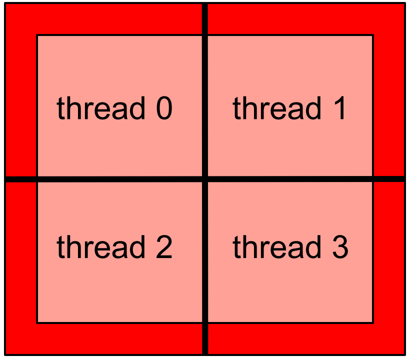
Redundant computation



Dof living on shared (partitioned) entity (edge).
Receive contribution from owned and halo cell.
Redundant compute contribution in halo to shared dof.
Less communication

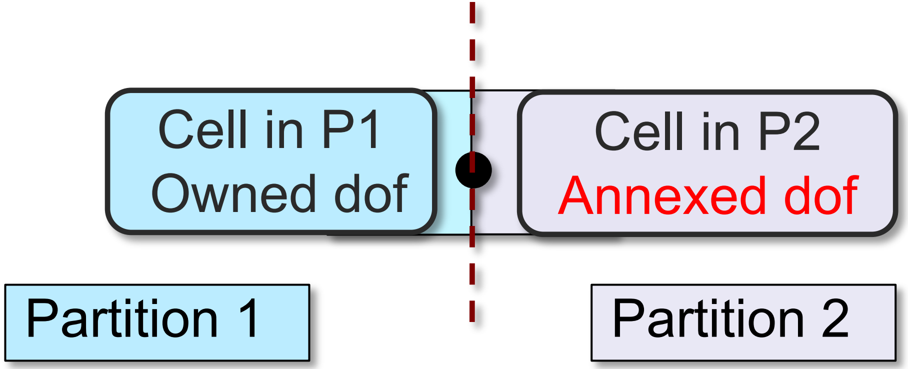


MPI only, 4 MPI ranks all have halos
Hybrid, 1 MPI task has a halo, 4 OpenMP threads share halo
boundary-to-area scaling
→ Less work for OpenMP threads



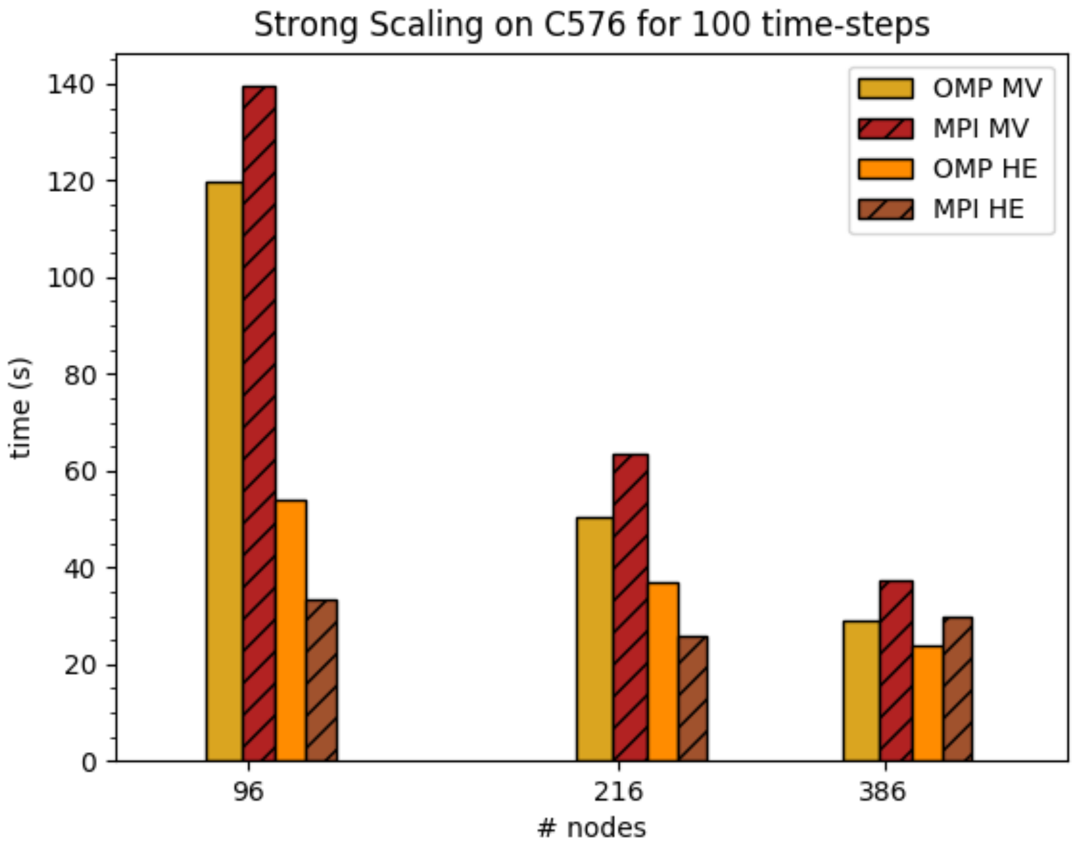
rank 0

Annexed dofs



Point-wise computations (e.g. set field to a scalar) loop over dofs
Looping to owned dofs ➔ halo exchange required for P2
Looping to annexed dofs is now transformation in Psyclone
Small increase in redundant computation
Large reduction in number of halo exchanges required

MPI and OMP



C576 is 576x576x6 cubed sphere ~ 17Km resolution
Intel 17, Cray XC40 dual socket, 18-core Broadwell
MPI is 36 MPI ranks per node
OMP is 6 MPI ranks / 6 omp threads
MV is matrix-vector kernel
HE is Halo-Exchange
Redundant computation

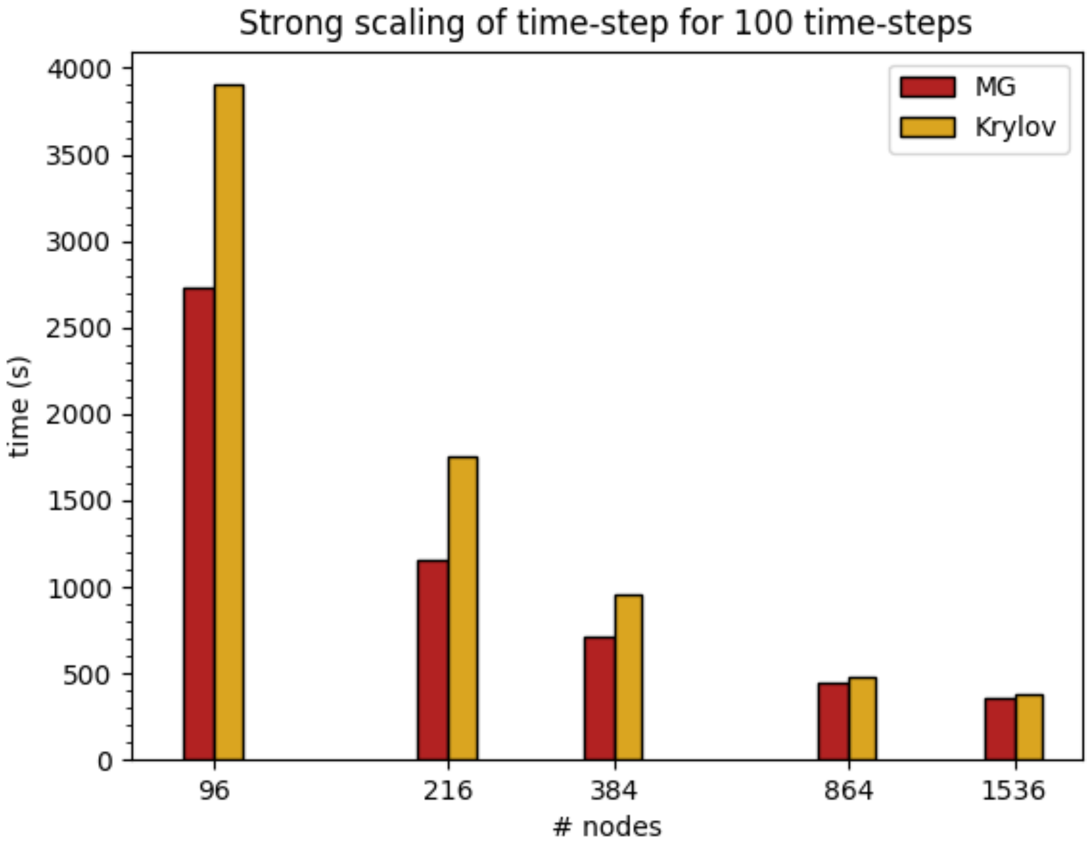
i. 7.5 x reduction in number of halo exchanges
ii. OMP has less work than MPI

The bad news …

Only compile and run code with Intel Compiler
Cray, PGI have problems with F2K3 OO code
Intel 17 OMP profile shows ~10% run-time is OMP synchronisation.
Grows with number of MPI ranks
`OMP_WAIT_POLICY=active` helps a bit
c.f. CCE8.5.8 and Intel 17 on single kernel code show 1-2% OMP synchronisation
Not clear what (and whose) the problem is?

Cray XC40 Aries network variability affects local comms (Halo exchange) and global comms (global sum)
Hard to measure performance without doing lots of runs
Is OMP faster than MPI. It can be!

Strong Scaling



C1152 cubed sphere ~ 9 Km
Comparing Multigrid and Krylov subspace solvers
Scales well (out to 55K cores) – mixed mode
LV at far right is 12x12 x 30L
For bigger problems cans scale to more nodes
Target is 1Km resolution
MS23: CMM: Thurs 1245 - *Scalable Linear Solvers for Next Generation Weather and Climate Models*

Programming models

Heterogeneous nodes with distinct memory spaces .e.g GPU
OpenACC and OpenMP4.5 (++) offload kernel for execution
PGI and Cray have problems compiling F2K3 OO code
PSyclone can generate OpenACC in PSy layer (GOcean)
Cannot yet annotate kernel code - ongoing

PSyclone Kernel Extractor (PSYKE) – dump out looping data in PSy layer
Dummy PSy layer (driver) can run a single kernel in isolation
LFRic-microbenchmark suite (On Github)
Can experiment with single kernel code

Met Office OpenACC

```
!$acc data copyin(ptheta_2_local_stencil,x_data,map_any_
!$acc& ncp_colour, nlayers,ncell_3d,ndf_any_space_1_thet
!$acc& ndf_any_space_2_x, undf_any_space_2_x) copy(theta

do colour = 1, ncolour
!$acc parallel loop private(cell, map1, map2), firstpriv
   do cell = 1, ncp_colour(colour)

      map1(:)=map_any_space_1_theta_adv_term(:,cmap(colo
      map2(:)=map_any_space_2_x(:,cmap(colour,cell))
      call matrix_vector_code(cell, nlayers, theta_adv_t
           ptheta_2_local_stencil, ndf_any_space_1_theta
           undf_any_space_1_theta_adv_term, &
           map1, &
           ndf_any_space_2_x, undf_any_space_2_x,&
           map2 )
   end do
   !$acc end parallel loop
end do
!$acc end data
```

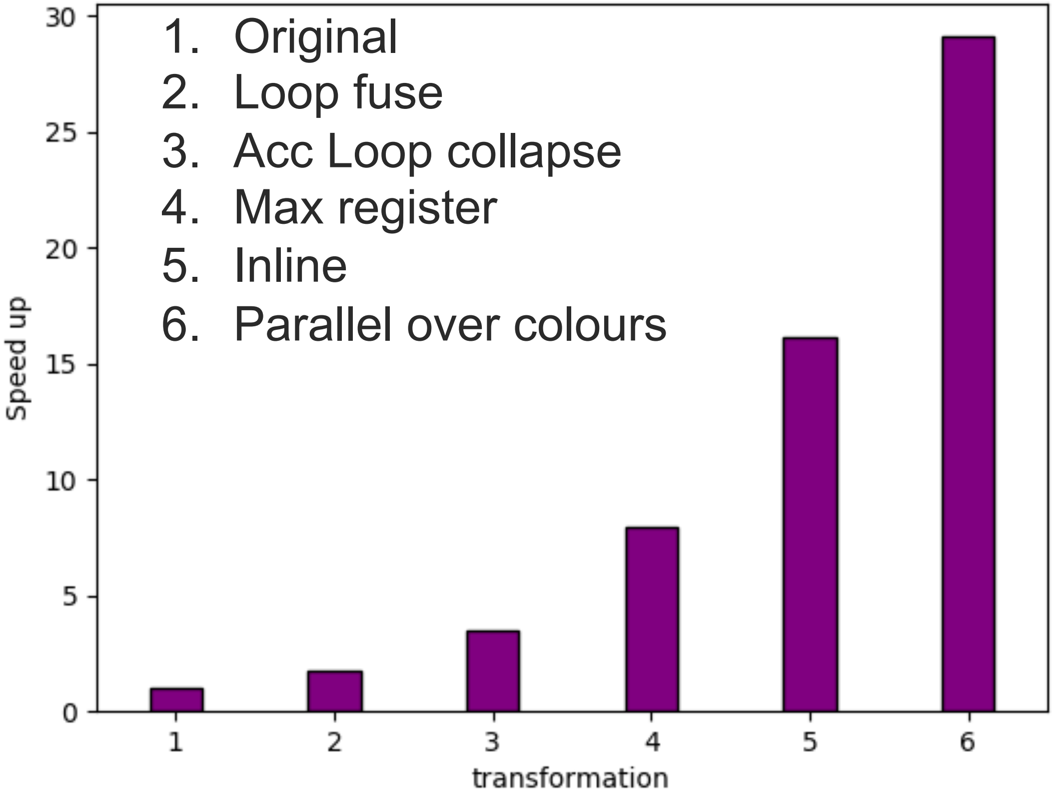
Offload data and kernel, same logic as OpenMP
Want bigger data regions

```
!$acc loop vector private(ik, df, df2, lhs_e, x_e)
 do k = 0, nlayers-1
   do df = 1, ndf2
     x_e(df) = x(map2(df)+k)
   end do
   lhs_e(:) = 0.0_r_def
   ik = (cell-1)*nlayers + k + 1
   do df = 1, ndf1
      do df2 = 1, ndf2
         lhs_e(df) = lhs_e(df) + matrix(df,df2,ik)*x_e(df2)
      end do
   end do
   do df = 1,ndf1
      !$acc atomic update
      lhs(map1(df)+k) = lhs(map1(df)+k) + lhs_e(df)
   end do
 end do
!$acc end loop
```

need to annotate kernel source SIMD (vector/warp) level parallelism

P9 + Volta GPU: MV-LMA

A. Gray (NVIDIA)
Cumulative speed up against original OpenACC code.
Problem size is too small for GPU.
Amortise cost of data movement by offloading multiple kernels



Summary

Separation of concerns is a powerful abstraction

High-level language + Optimising compiler is not sufficient in the age of parallelism

Lots of programming models – no single one is sufficient

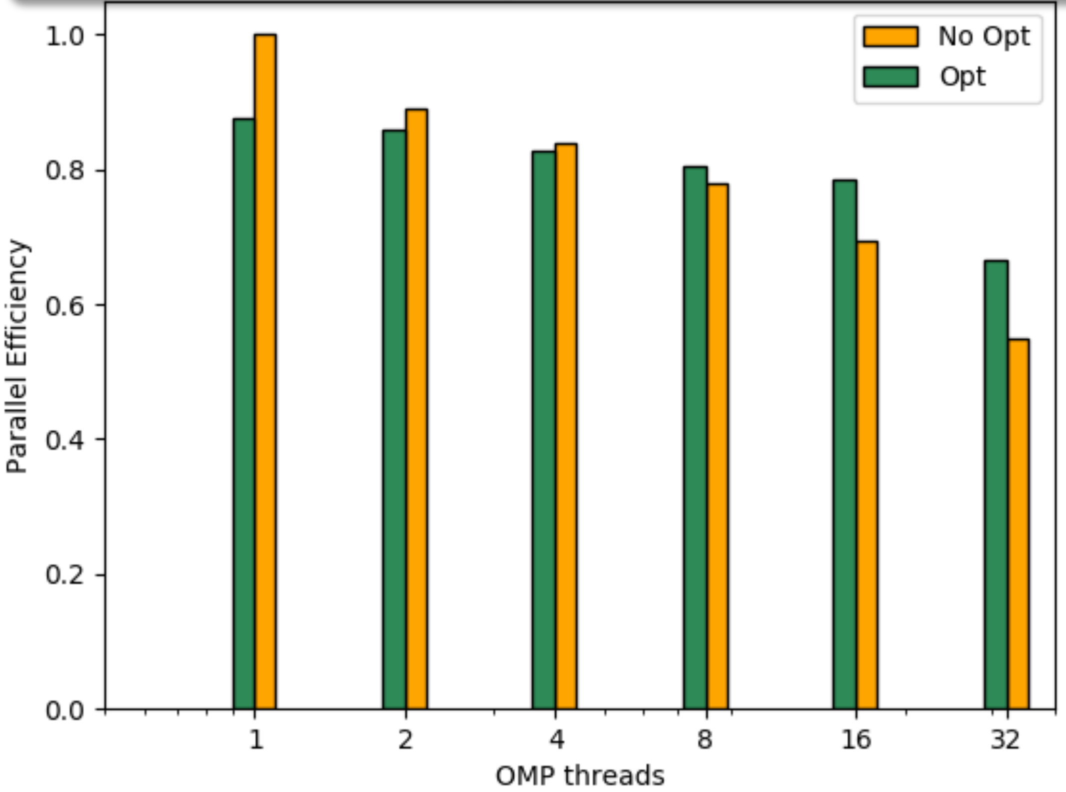
Developing DSL - PSyKAl API + PSyclone

Performance, Portability and Productivity

TX2 MV-LMA

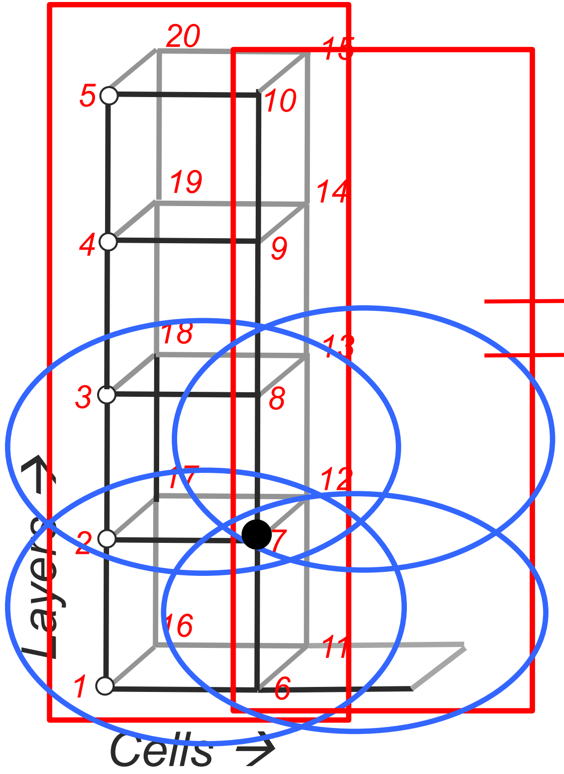
32 core per socket
Can over-subscribe
4 HDW threads
Cray Compiler
PE decrease due to small problem size

Parallel efficiency c.f. no opt 1 thread



Data layout, unstructured mesh

W_0 space (vertices)



1	2	3	4	5	6	7	8	9	10	…

Data array (1-d)

```
data(map(1,4) + 0)
data(map(1,3) + 1)
data(map(2,2) + 0)
data(map(2,1) + 1)
```

Dofmap 2-d array

1	2	6	7	11	12	16	17
6	7	21	22	26	27	11	12
…	…				…	…	…

ndof per cell

ncell

PSy layer

Kernel layer

Visit same dof more than once: loop over cells, levels, dofs
Mesh and dofmap form an ordered set
Change mesh topology (element), geometry (cubed sphere)
→Change to mesh generation and partition
→No change to science code