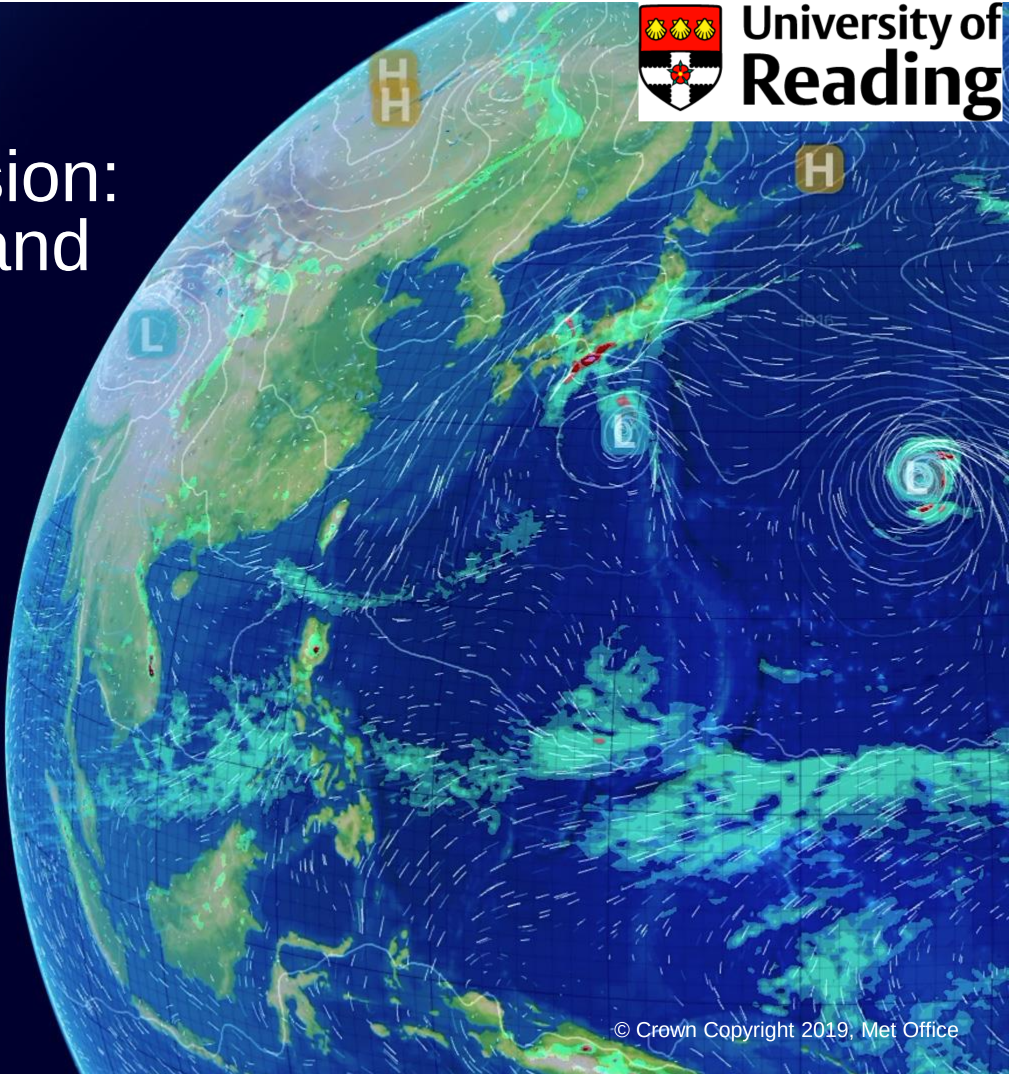
Met Office

University of Reading

Multigrid and Mixed-Precision: Fast Solvers for Weather and Climate Models.



Christopher Maynard

- Scientific Software Engineer
  Met Office, Exeter, UK
- Associate Professor of Computer Science.
  University of Reading

Abstract

Semi-implicit time-integration schemes, commonly used in Numerical Weather Prediction and Climate models, require a global matrix inversion of some kind. The linear solvers employed to do so must be fast and capable of running on highly parallel and complex supercomputers. Consequently there is a complex interplay between the algorithm and its implementation. In this presentation the use of mixed-precision arithmetic and a Geometric Multigrid Algorithm in the Met Office's Unified Model and LFRic Model are described and performance analysed.

A tale of two solvers

It was the best of times, it was the worst of times …

Apologies to Charles Dickens …

Mixed-precision arithmetic in the ENDGame dynamical core of the Unified Model, a numerical weather prediction and climate model code
C.M. Maynard and D.N. Walters. Comp. Phys. Comm. V244 Nov 2019 69--75

Performance of multigrid solvers for the mixed finite element dynamical core, LFRic
C.M. Maynard, T. Melvin, E.H. Müller in Prep.

Accuracy and Precision

Numerical algorithms have a defined accuracy. How fast they to the converge to continuous differential equations

Computers use floating-point arithmetic
Variable accuracy *c.f.* to real numbers
Not associative
Accumulated round-off error
More precision → bigger data type

π = 3.14000000001 Precise but not accurate
3 < **π** < 4 Accurate but not precise (John Gustafason)
Most scientific applications, especially weather and climate use 64-bit arithmetic
Is this necessary? 32-bit faster (memory/cache CPU, GPU etc)

Semi-implicit schemes

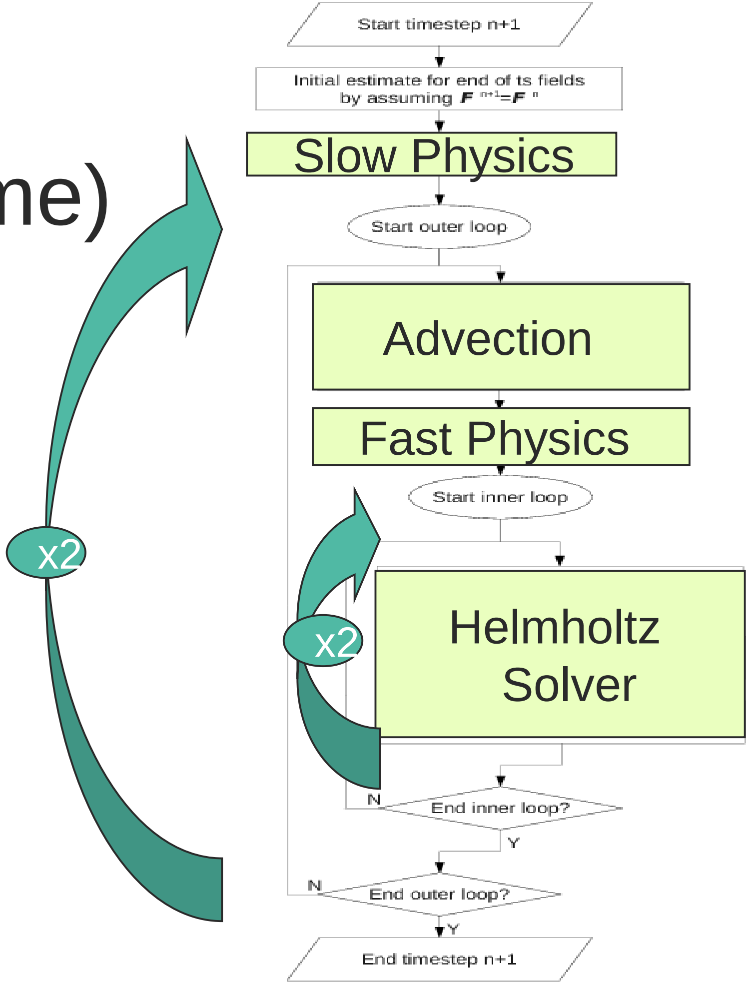
Lon-Lat grid → polar singularity
Near poles grid points very close together
Explicit time-stepping scheme unfeasibly short time-step for NWP
Semi-implicit schemes treat fast acoustic-gravity modes implicitly
In combination with semi-Lagrangian advection, SI allows stable integration around pole
Long, but computationally expensive time-steps
Global matrix inversion

Met Office

UM timestep (ENDGame)

- 1x Slow physics
- 2x Advection
- 2x Fast Physics
- 4x Helmholtz solver
- 5x Dynamics residuals

SI … expensive, but can take long timesteps



Met Office Helmholtz solve:Pressure correction

University of Reading

Equation takes the form

$$A \cdot \mathbf{x} = \mathbf{b}$$

Where A is a large, sparse matrix

b contains forcing terms

N1280 Lon-Lat mesh

~10Km resolution at mid-latitudes

$$(2 \times 1280)\frac{\times(3 \times 1280)}{2} \times 70 \approx 350M$$

For Semi-implicit time-stepping scheme, solver is part of a larger, non-linear system solution procedure

Accuracy of the solve is dictated **stability** of time-stepping scheme

FD ~ ∇p ~ 2^{nd} order → limit to effect of accuracy of solve on pressure

Once solver error is sufficiently small, discretisation errors dominate

Post-conditioned BiCGStab

Inputs: $\boldsymbol{x}$, $\boldsymbol{r}$, ϵ_{tol}, δ_{min}
$\delta = \max(||\boldsymbol{r}||, \delta_{min})$;
$\boldsymbol{p} = \boldsymbol{v} = 0$;
$\boldsymbol{r} = \boldsymbol{r} - A\boldsymbol{x}$; $\boldsymbol{r}_0 = \boldsymbol{r}$;
$\alpha = \omega = n = 1$;
for $k = 1,2,\ldots$ **do**
  $\rho = (\boldsymbol{r}, \boldsymbol{r}_0)$; $\beta = \alpha\rho/(n\omega)$;
  $\boldsymbol{t} = \boldsymbol{r} - \beta\omega\boldsymbol{v}$; $\boldsymbol{s} = C\boldsymbol{t}$;
  $\boldsymbol{p} = \boldsymbol{s} + \beta\boldsymbol{p}$; $\boldsymbol{v} = A\boldsymbol{p}$;
  $n = (\boldsymbol{v}, \boldsymbol{r}_0)$; $\alpha = \rho/n$;
  if $\omega < 10^{-12}$ **then**
    Convergence problem ω too small.
  end
  $\boldsymbol{s} = \boldsymbol{r} - \alpha\boldsymbol{v}$; $\tilde{\boldsymbol{s}} = C\boldsymbol{s}$; $\boldsymbol{t} = A\tilde{\boldsymbol{s}}$;
  $\omega = (\boldsymbol{t}, \boldsymbol{s})/||\boldsymbol{t}||^2$;
  $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{p} + \omega\tilde{\boldsymbol{s}}$; $\boldsymbol{r} = \boldsymbol{s} - \omega\boldsymbol{t}$;
  $n = \rho$; $\epsilon = ||\boldsymbol{r}||/\delta$;
  If $\epsilon < \epsilon_{tol}$ exit;

Halting criterion: norm of residual vector

$$||\mathbf{r}|| = ||A \cdot \mathbf{x}_i - \mathbf{b}||$$

Stop when $\epsilon = \dfrac{||r||}{\delta} < \epsilon_{tol}$

If $\epsilon_{tol} \gg \epsilon_{32}$
Where 32-bit Unit-of-least-precision (ULP) is $\sim 1.0 \times 10^{-7}$
Then 32-bit arithmetic is sufficient.
64-bit arithmetic won't improve *accuracy* of solution

$$\epsilon_{tol} = \{10^{-3}, 10^{-4}\}$$

Idealised solver

Examine effect of precision on convergence

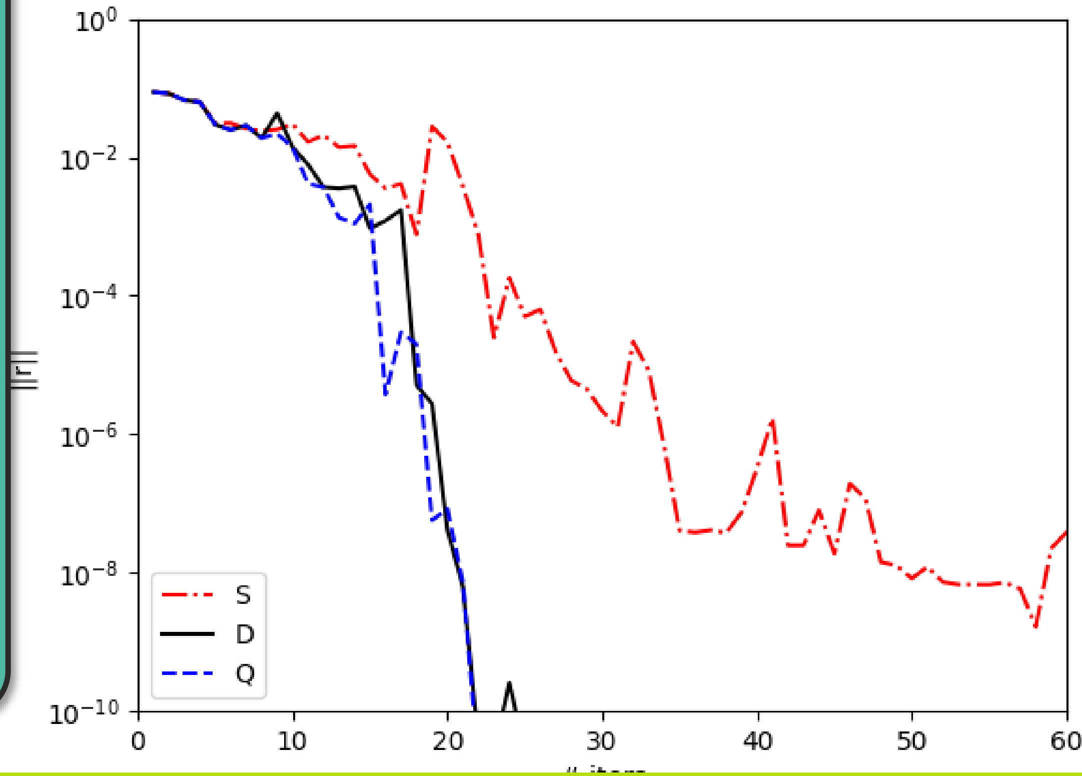
$$||\mathbf{r}|| = ||A \cdot \mathbf{x}_i - \mathbf{b}||$$

c.f. 32- 64- and 128-bit arithmetic

32-bit takes more iterations for residual fall

Iteration gap

Still converges



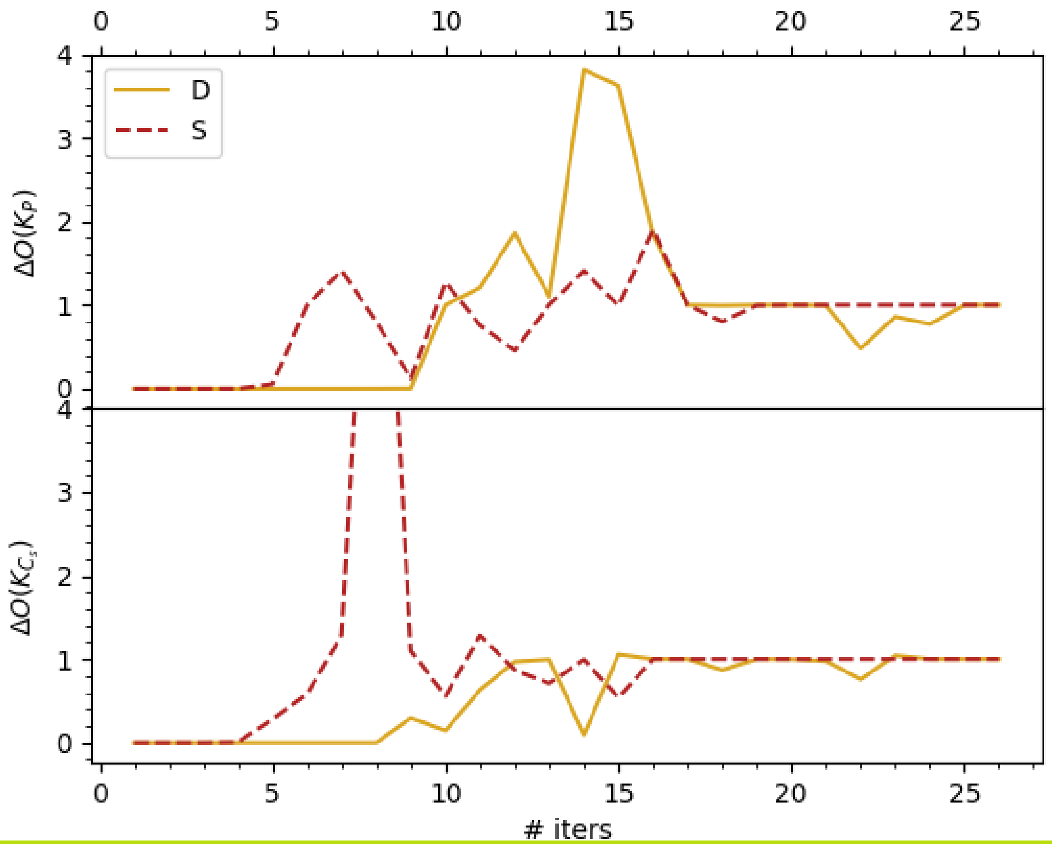
Orthogonality

$$\mathcal{K}_p = \text{span}\{p, Ap, \cdots\}$$

$$\mathcal{O}(\mathcal{K}_p) = p \cdot Ap$$

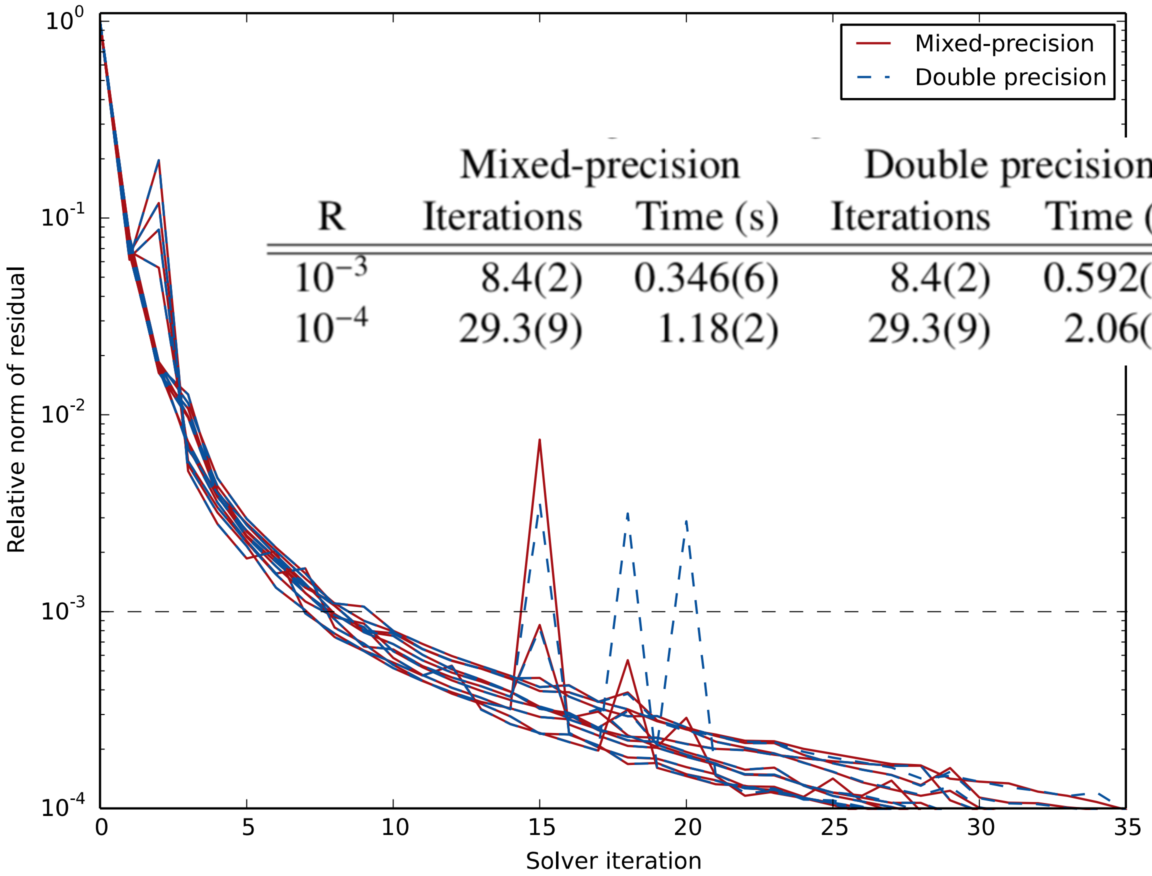
$$\Delta O_P = \frac{\text{abs}(\text{O}_\text{P} - \text{O}_{128})}{\text{abs}(\text{O}_\text{P} + \text{O}_{128})}$$

Zero → orthogonal
One → no orthooinality
32-bit (S) diverges earlier than 64-bit (D)



Mixed-precision in the UM

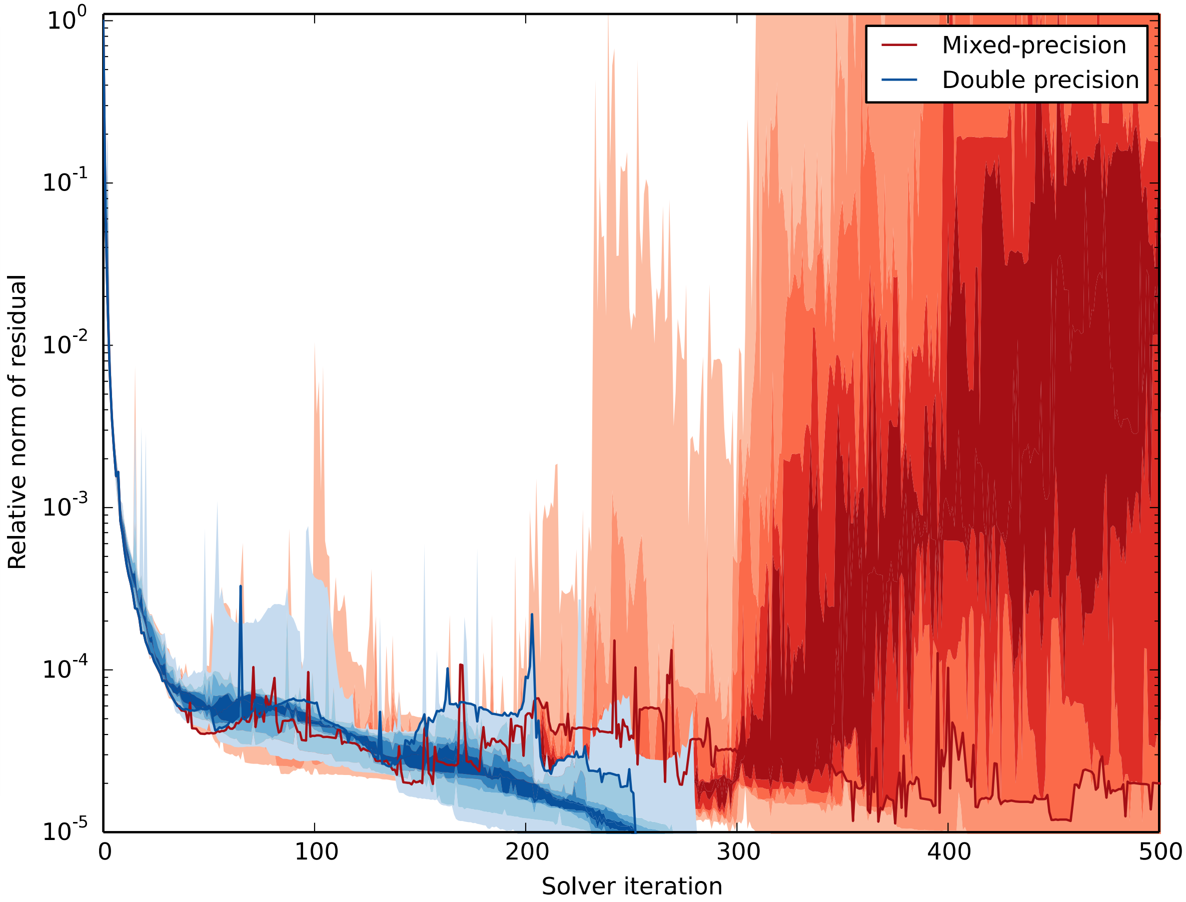
- Solver implemented as *mixed* precision
- Pressure field was kept as 64-bit
- 32-bit increments
- Ease of interfacing to model
- 11 N1280 operational cfgs
- First time-step, first solve
- 96 nodes Cray XC40
- 12 MPI ranks/3 OMP threads



	Mixed-precision		Double precision	
R	Iterations	Time (s)	Iterations	Time (s)
10^{-3}	8.4(2)	0.346(6)	8.4(2)	0.592(9)
10^{-4}	29.3(9)	1.18(2)	29.3(9)	2.06(3)

When does it break?

- Tighten tol to 10^{-5}
- Slow convergence – hundreds of iters
- BiCGstab does not guarantee monotonic convergence
- Jumps in value of residual
- BiCGStab is breaking down
- Mixed-precision fares worse – sometimes fails



Problems and solutions

Occasional problems at 10^{-4}
Slow convergence (hundreds of iters) – or even failures (divide by near zero)
Scalars ➔zero symptomatic of algorithm failing
In Mixed-precision global sums reverted to 64-bit arithmetic
Negligible cost (global sum is latency bound – sum is for single scalar)
Prevents failure, but slow convergence remains
In operations fixed iteration count limit imposed with full restart of solver

Ill conditioned problem arises from issues with "noise" in horizontal wind fields near poles
Original cfgs run with 10^{-3} tol, but problems in other parts of model
Tighter solver convergence helps but has its own problems
Solutions? i) Polar cap (transport across the poles)
ii) Multigrid (see later)

Conclusions I

Efficiency (speed), accuracy and stability are all important considerations

Reduced precision can provide significant performance benefits (almost 2x for 32-bit versus 64-bit)

UM operations at Met Office runs in mixed-precision

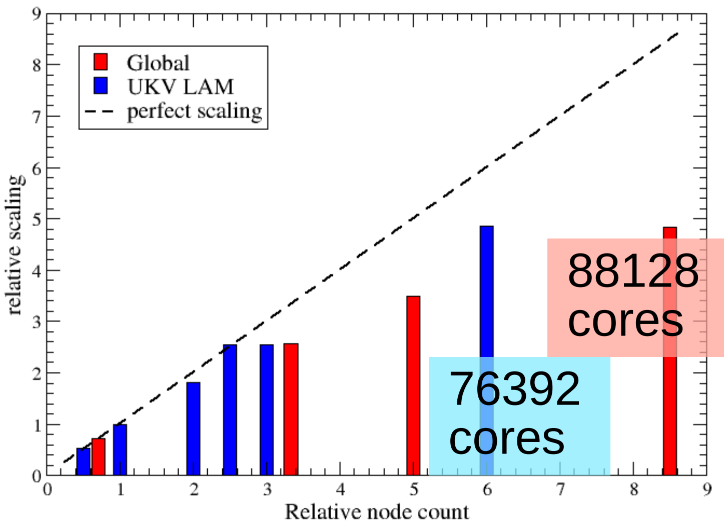
Care is needed as complex interplay between round-off and other numerical errors

Especially where Numerical algorithms experience other problems

The Unified Model

Uses Lon-Lat grid
Scientifically very good
Good computational performance

Very High Resolution scaling
6.5 Km resolution

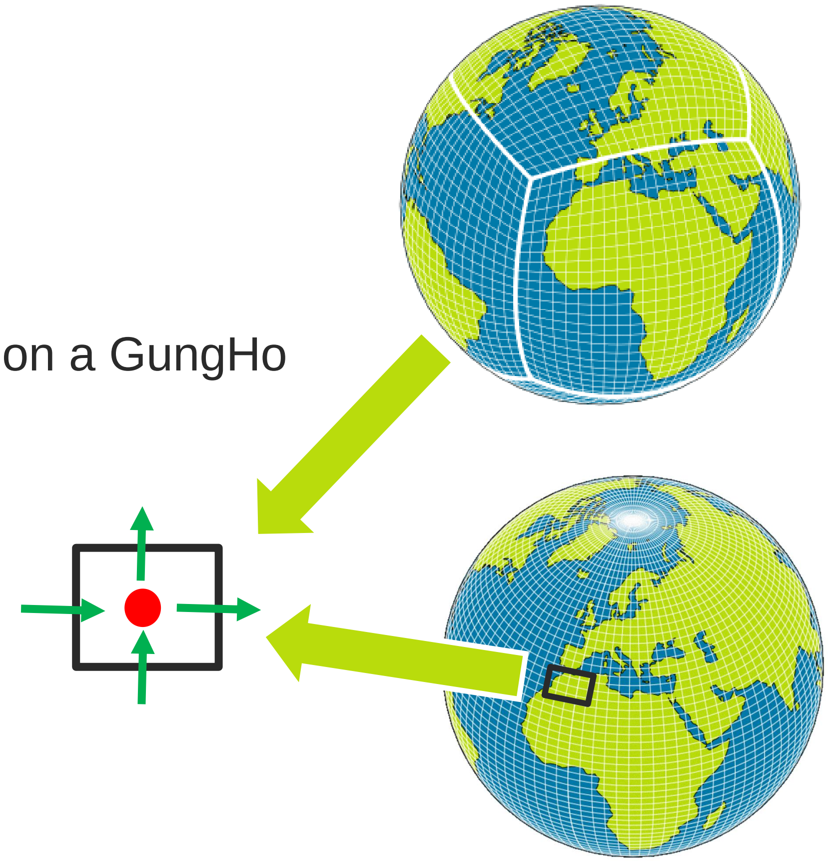


The finger of blame …
Lon-lat grid is preventing scaling
10km resolution (mid-latitudes) → 10m at poles

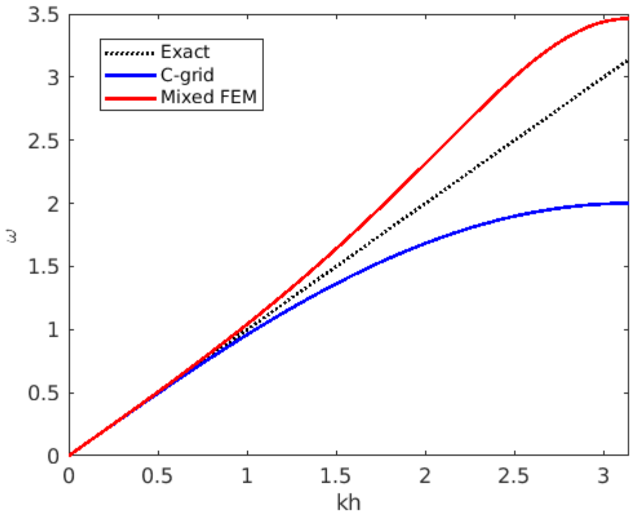
Met Office

GungHo Issues

- How to maintain accuracy of current model on a GungHo grid?
- Principal points about current grid are:
  - ➢ Orthogonal, Quadrilateral, C-grid
- Mixed Finite elements
  - ➢ Same layout as current C-grid
  - ➢ Not orthogonality condition
  - ➢ Gives consistent discretisation

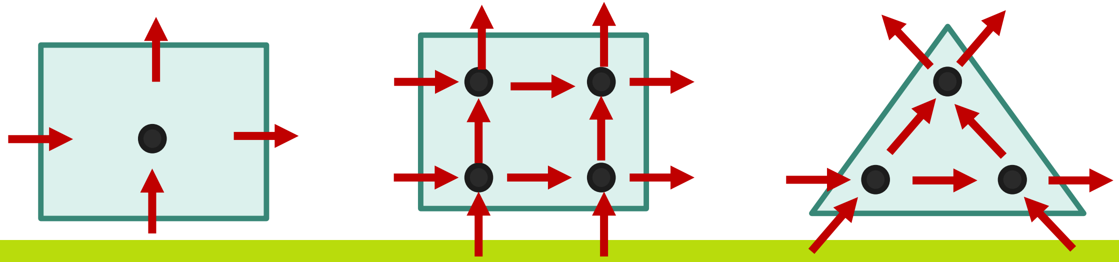


Mixed Finite Elements



Mixed Finite Element method gives

- Compatibility: $\nabla \times \nabla \varphi = 0, \nabla \cdot \nabla \times \boldsymbol{v} = 0$
- Accurate balance and adjustment properties
- No orthogonality constraints on the mesh
- Flexibility of choice mesh (quads, triangles) and accuracy (polynomial order)



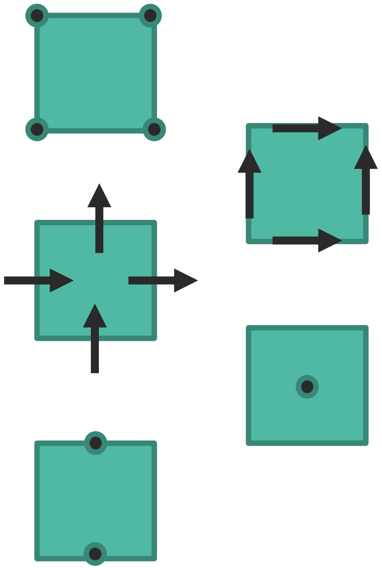
Met Office

University of Reading

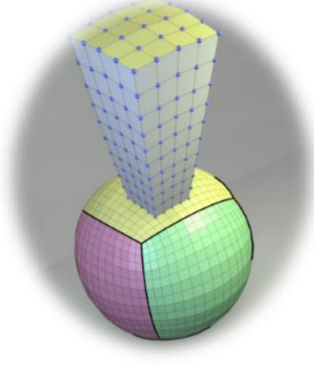
Mixed Finite Element Method

$$\mathbb{W}_0 \xrightarrow{\nabla} \mathbb{W}_1 \xrightarrow{\nabla\times} \mathbb{W}_2 \xrightarrow{\nabla\cdot} \mathbb{W}_3.$$

$\mathbb{W}_0$	Pointwise scalars	
$\mathbb{W}_1$	Circulation Vectors	Vorticity
$\mathbb{W}_2$	Flux Vectors	Velocity
$\mathbb{W}_3$	Volume integrated Scalars	Pressure, Density
$\mathbb{W}_\theta$	Pointwise scalars	Potential Temperature



Gungho Discretisation



Inspired by iterative-semi-implicit semi-Lagrangian scheme used in UM

Scalar transport uses high-order, upwind, explicit Eulerain FV scheme

Wave dynamics (and momentum transport) use iterative-semi-implicit, lowest order mixed finite element method (equivalent to C-grid/Charney-Phillips staggering)

$$\delta_t \mathbf{u} = -\overline{(2\Omega + \nabla \times \mathbf{u}) \times \mathbf{u} + \nabla (K + \Phi) + c_p \theta \nabla \Pi}^{\alpha}$$

$$\delta_t \rho = -\nabla \cdot \left[\mathcal{F}\left(\rho^n, \overline{\mathbf{u}}^{1/2}\right)\right]$$

$$\delta_t \theta = -\mathcal{A}\left(\theta^n, \overline{\mathbf{u}}^{1/2}\right)$$

$$\overline{F}^{\alpha} \equiv \alpha F^{n+1} + (1 - \alpha) F^n$$

Time-stepping

Quasi-Newton Method: $\mathcal{L}\left(\mathbf{x}^{*}\right)\mathbf{x}' = -\mathcal{R}\left(\mathbf{x}^{(k)}\right).$

Linearized around reference state (previous time-step state) $\boldsymbol{x}^{*} \equiv \boldsymbol{x}^{n}$

Solve for increments on latest state: $\boldsymbol{x}' \equiv \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$

Semi-Implicit system contains terms needed for acoustic and buoyancy terms

$$\mathcal{L}\left(\mathbf{x}^{*}_{\text{phys}}\right)\mathbf{x}'_{\text{phys}} = \begin{cases} \mathbf{u}' - \mu\left(\dfrac{\mathbf{n}_{\text{b}}\cdot\mathbf{u}'}{\mathbf{n}_{\text{b}}\cdot\mathbf{z}_{\text{b}}}\right)\mathbf{z}_{\text{b}} \\ \qquad +\tau_u\Delta t c_p\left(\theta'\nabla\Pi^{*} + \theta^{*}\nabla\Pi'\right), \\ \rho' + \tau_\rho\Delta t\nabla\cdot\left(\rho^{*}\mathbf{u}'\right), \\ \theta' + \tau_\theta\Delta t\mathbf{u}'\cdot\nabla\theta^{*}, \\ \dfrac{1-\kappa}{\kappa}\dfrac{\Pi'}{\Pi^{*}} - \dfrac{\rho'}{\rho^{*}} - \dfrac{\theta'}{\theta^{*}}, \end{cases}$$

Time-stepping II

Solver Outer system with Iterative (GCR) solver

$$\begin{pmatrix} M_2^{\mu,C} & & -P_{2\theta}^{\Pi^*} & -G^{\theta^*} \\ D^{\rho^*} & M_3 & & \\ P_{\theta 2}^{\theta^*} & & M_\theta & \\ & -M_3^{\rho^*} & -P_{3\theta}^{*} & M_3^{\Pi^*} \end{pmatrix} \begin{pmatrix} \widetilde{u}' \\ \widetilde{\rho}' \\ \widetilde{\theta}' \\ \widetilde{\Pi}' \end{pmatrix} = \begin{pmatrix} -\mathcal{R}_u \\ -\mathcal{R}_\rho \\ -\mathcal{R}_\theta \\ -\mathcal{R}_\Pi \end{pmatrix}$$

- Contains all couplings
- Preconditioned by approximate Schur complement for the pressure increment
- Velocity and potential temperature mass matrices are lumped

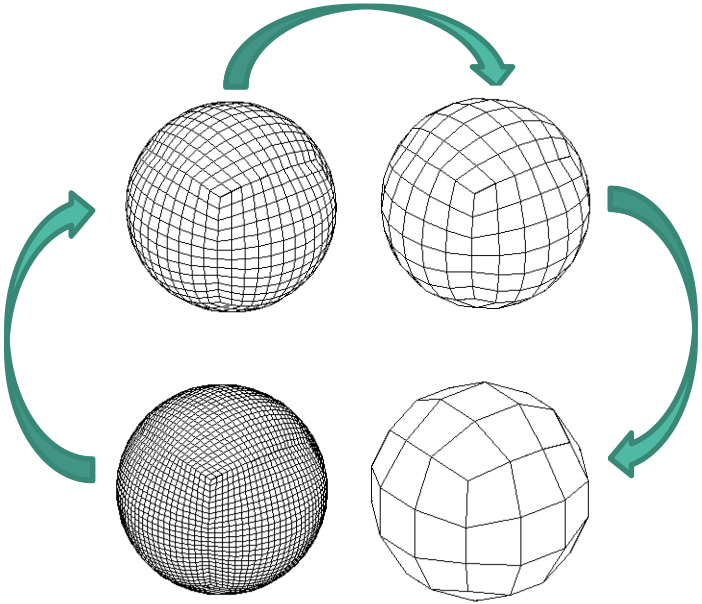
Multigrid

- Helmholtz system $H\Pi' = R$ solved using a single Geometric-Multi-Grid V-cycle with block-Jacobi smoother

$$H = M_3^{\Pi^*} + \left(P_{3\theta}^* \mathring{M}_\theta^{-1} P_{\theta 2}^{\theta^*,z} + M_3^{\rho^*} M_3^{-1} D^{\rho^*}\right)\left(\mathring{M}_2^{\mu,C}\right)^{-1} G^{\theta^*}.$$

- Block-Jacobi smoother with small number (2) of iterations on each level
- Exact (tridiagonal) vertical solve: $\widehat{H}_z^{-1}$

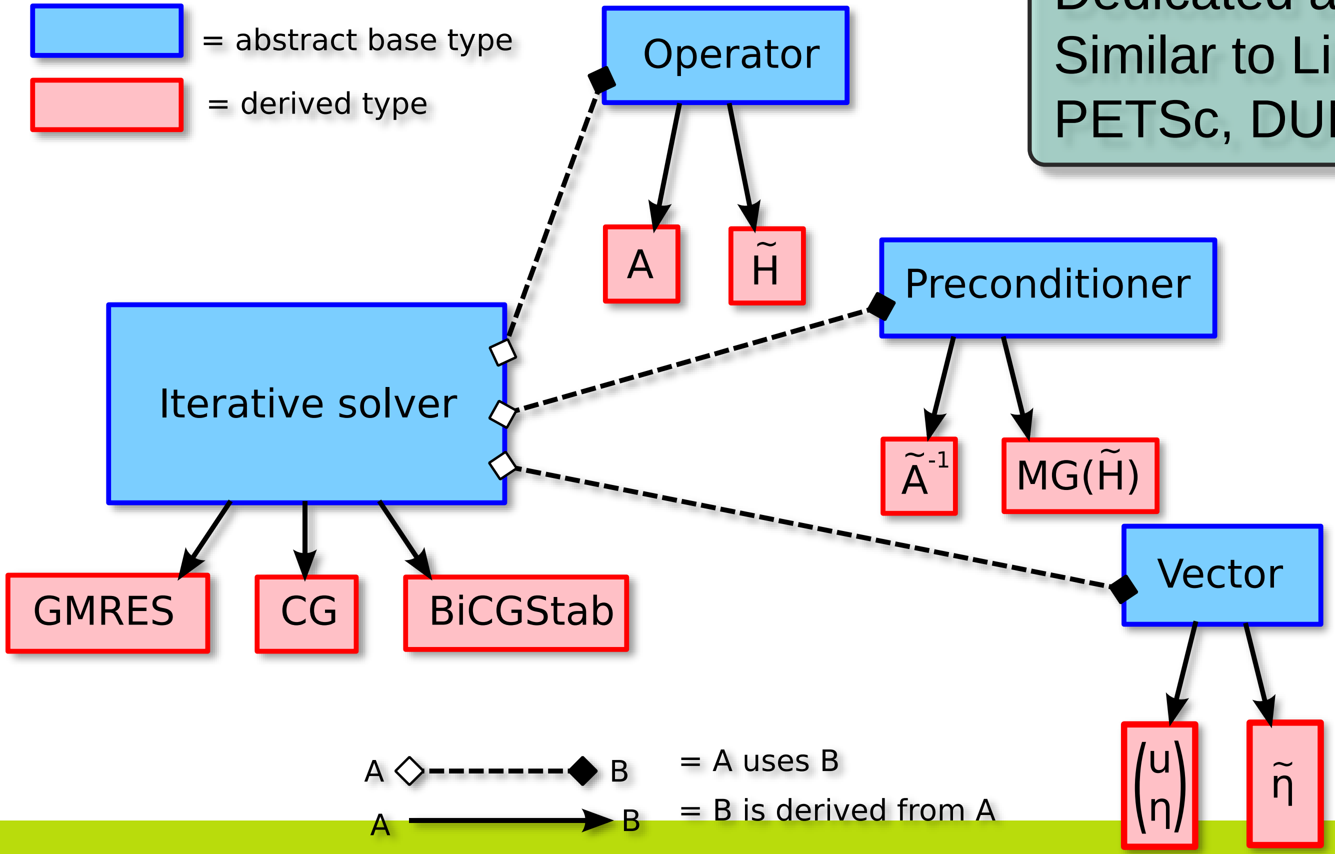
$$\widetilde{\Pi}' \leftarrow \widetilde{\Pi}' + \omega \widehat{H}_z^{-1}\left(\mathcal{B} - H\widetilde{\Pi}'\right)$$



Met Office University of Reading

The solver

Dedicated abstraction in F2K3 OO
Similar to Lin. Alg Libs e.g. PETSc, DUNE-ISTL, Trillinios

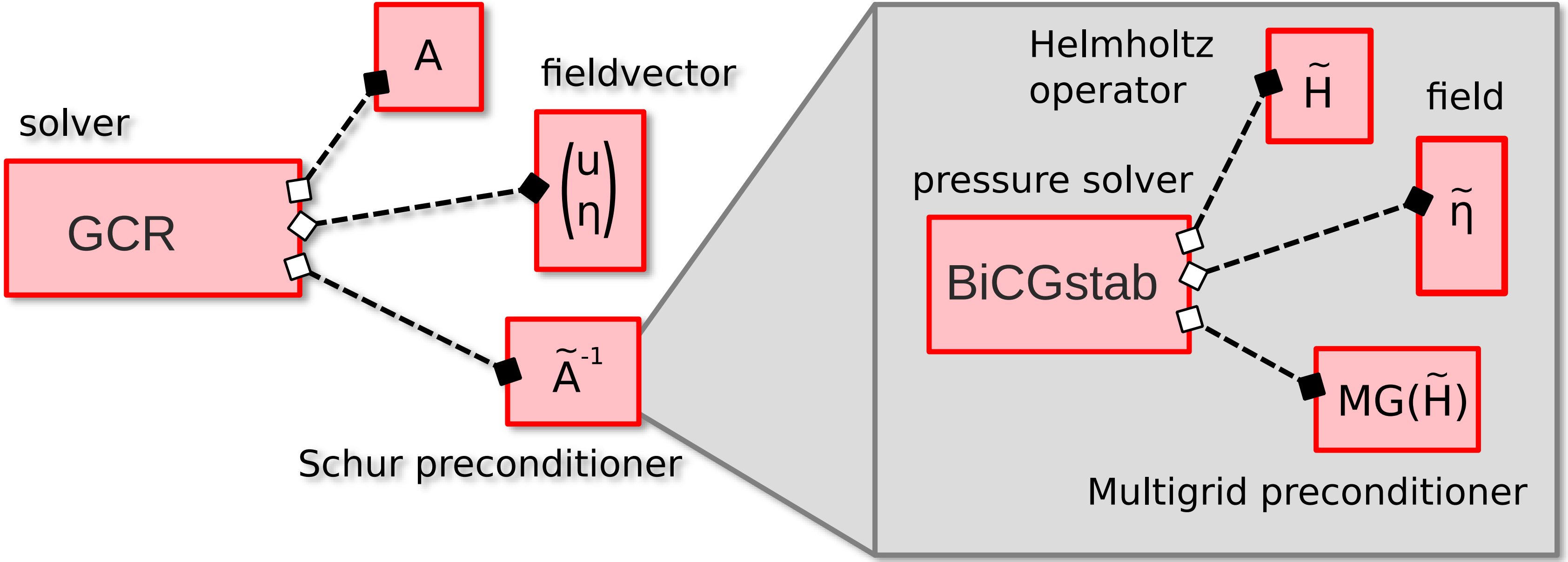


www.metoffice.gov.uk

Met Office

The solver II

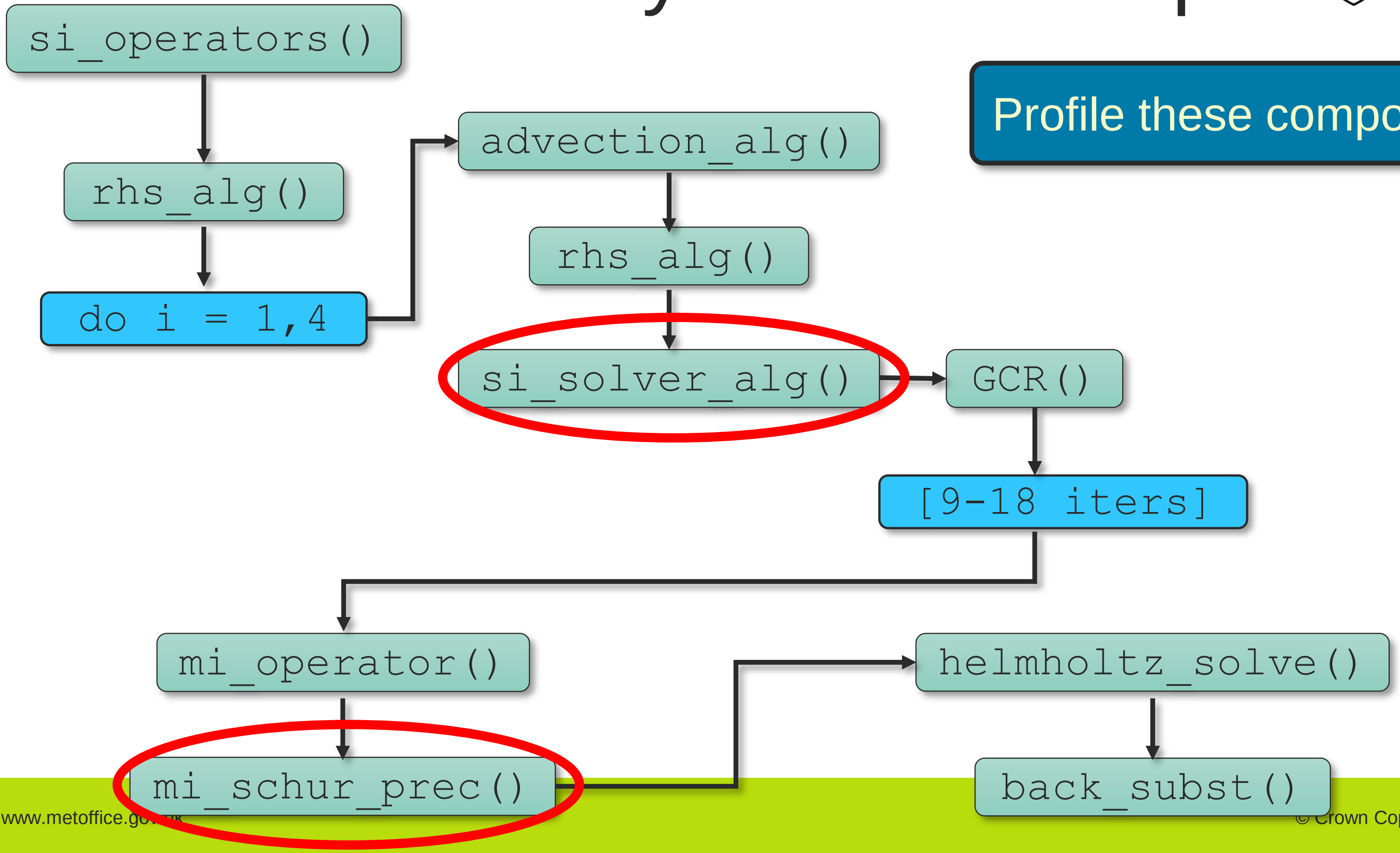
University of Reading



Allows for easy implementation of sophisticated nested solver
Multigrid preconditioner - reduce work for iterative solver
- faster and less global sums (better scaling)

Anatomy of a time-step

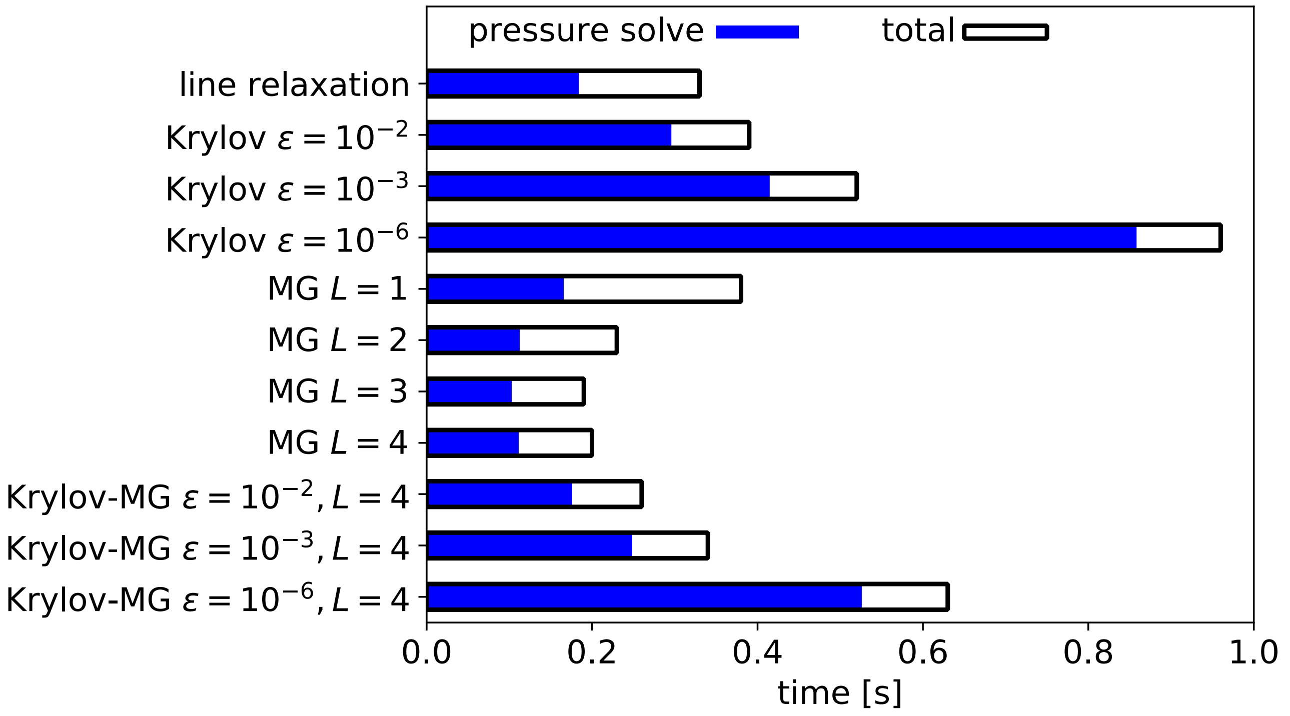
Profile these components



Initial Results

C192 cubed sphere with 30 L (~50Km) Baroclinic wave test Met Office Cray XC40 64 nodes (2304 cores) Mixed mode 6 MPI/6 OMP threads

c.f. $||r|| = ||\mathbf{A}x - b||$ Of Krylov 10^{-2}
Before and after MG
3-level V-cycle



Time-step and scaling

SI → long time-step as possible
Stability is limited by *vertical* stability.
C192 ~ 50Km, $\Delta t = 1200$

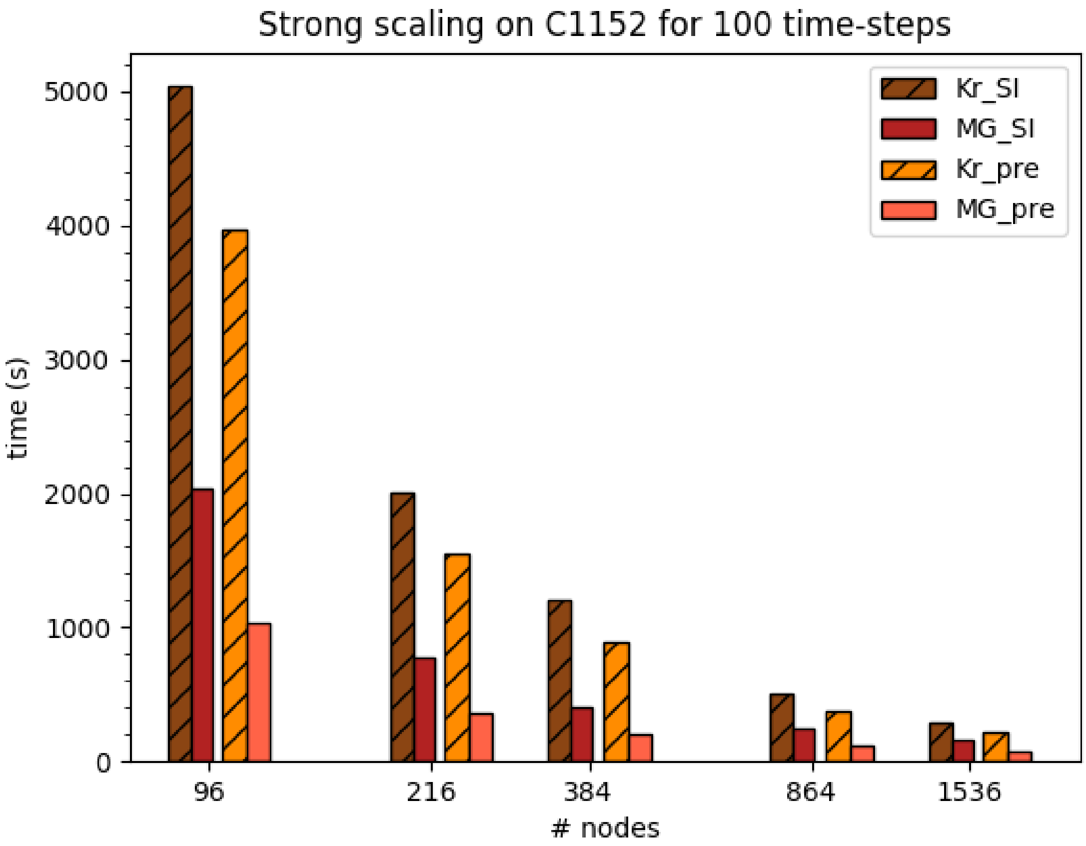
$$\mathrm{CFL_H} = \mathrm{c_s}\frac{\Delta \mathrm{t}}{\Delta \mathrm{x}} \quad c_s = 340m/s$$

$\mathrm{CFL_H} \sim 8$

C1152 ~ 9Km and $\Delta t = 205s$ → CFL ~ 8
Baroclinic wave test (Again 30L)
Kr 10^{-2} *cf* 3-level MG
Up to 1536 *nodes*

Strong scaling

Lower is better
MG is at least 2x faster than Kr
Both show good scaling
X-axis is logarithmic
96 : 1536 ~ 16x
55296 cores
LV = {48, 32, 24, 16, 12}^{2}

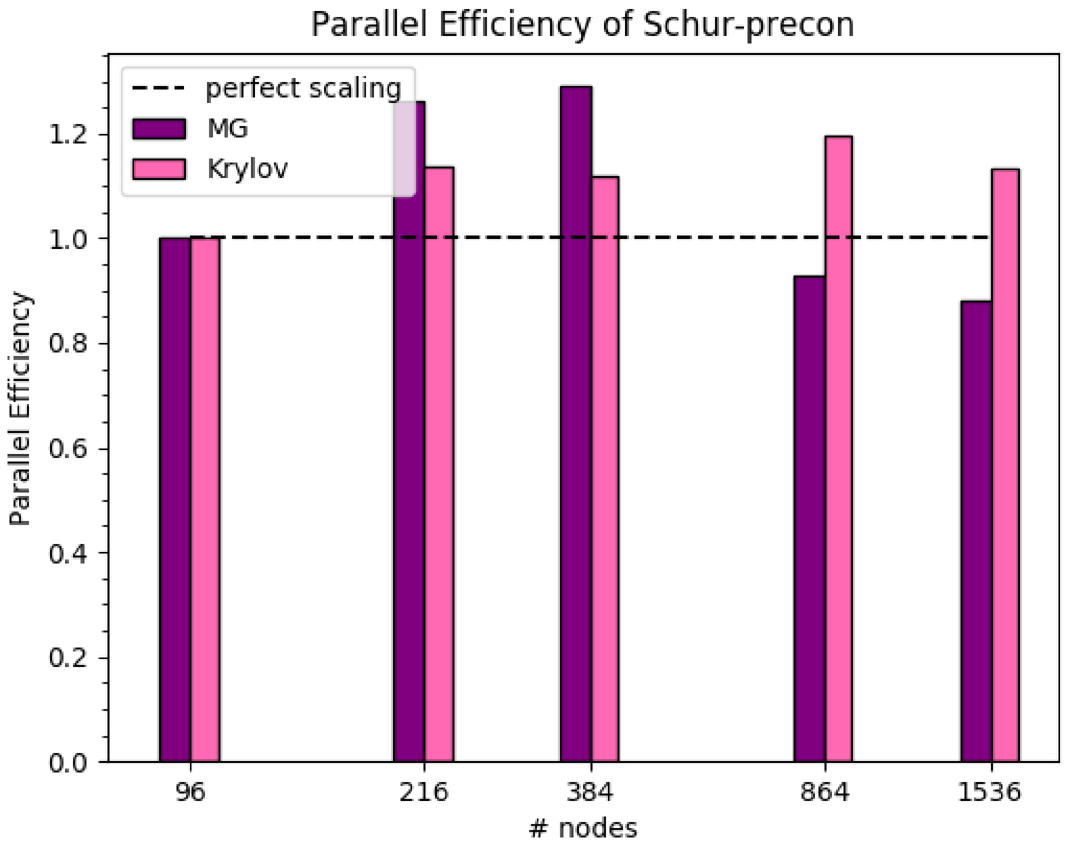


Met Office

Parallel Efficiency

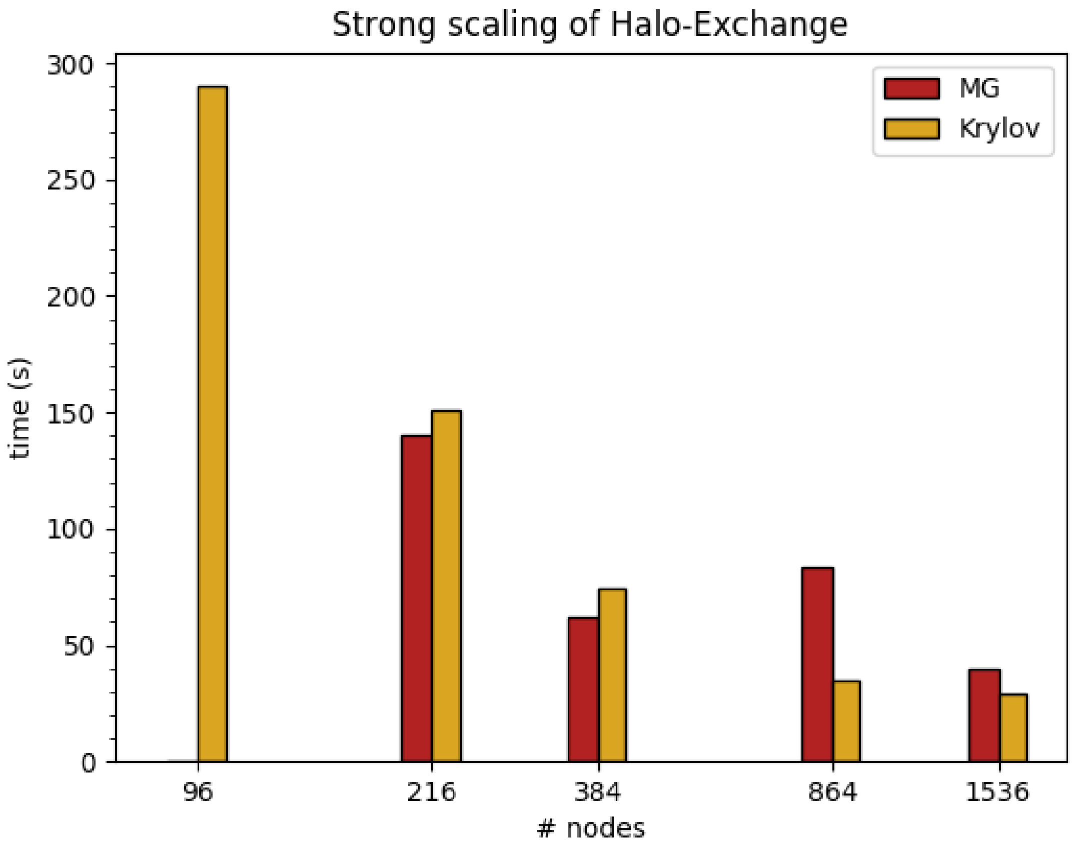
University of Reading

Higher is better
Scaled from 96 nodes
Both show good scaling
KR is better because 96 node is slow!



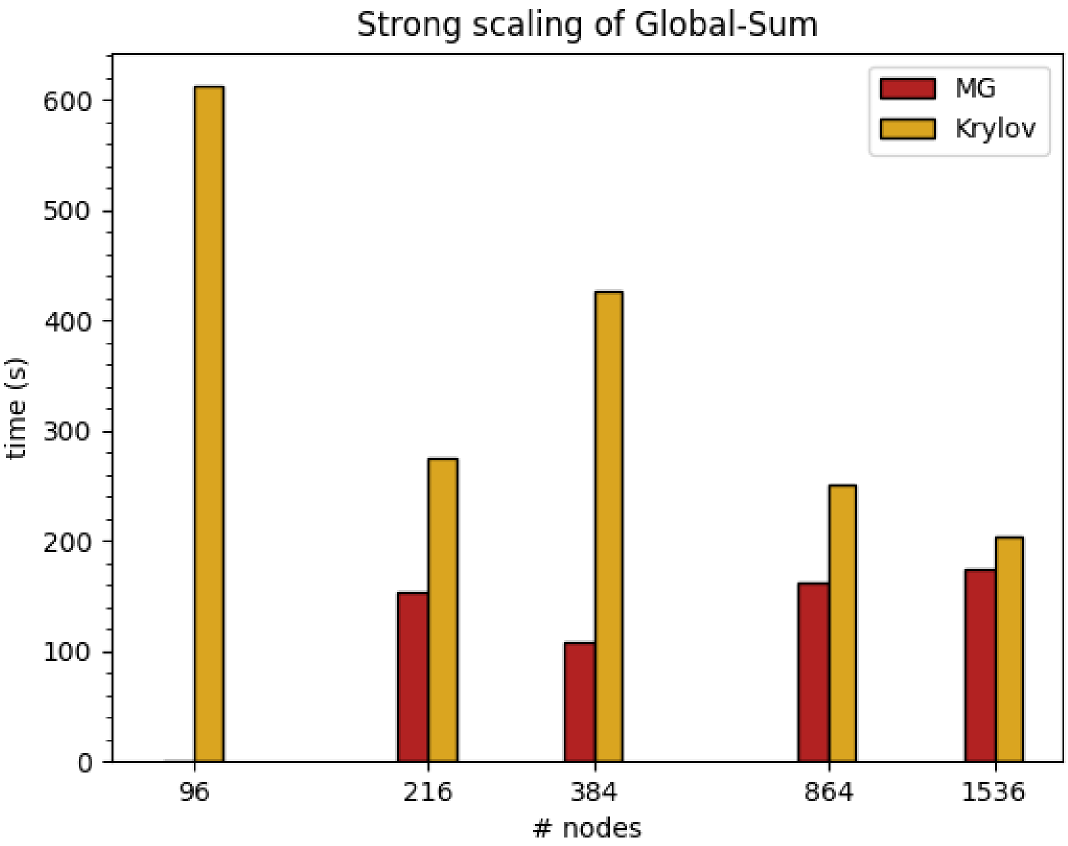
Halo-Exchange

Lower is better
Data produced by CrayPAT
96 node MG runs our of memory
Less comms for MG
Large variation due to Aries adaptive routing



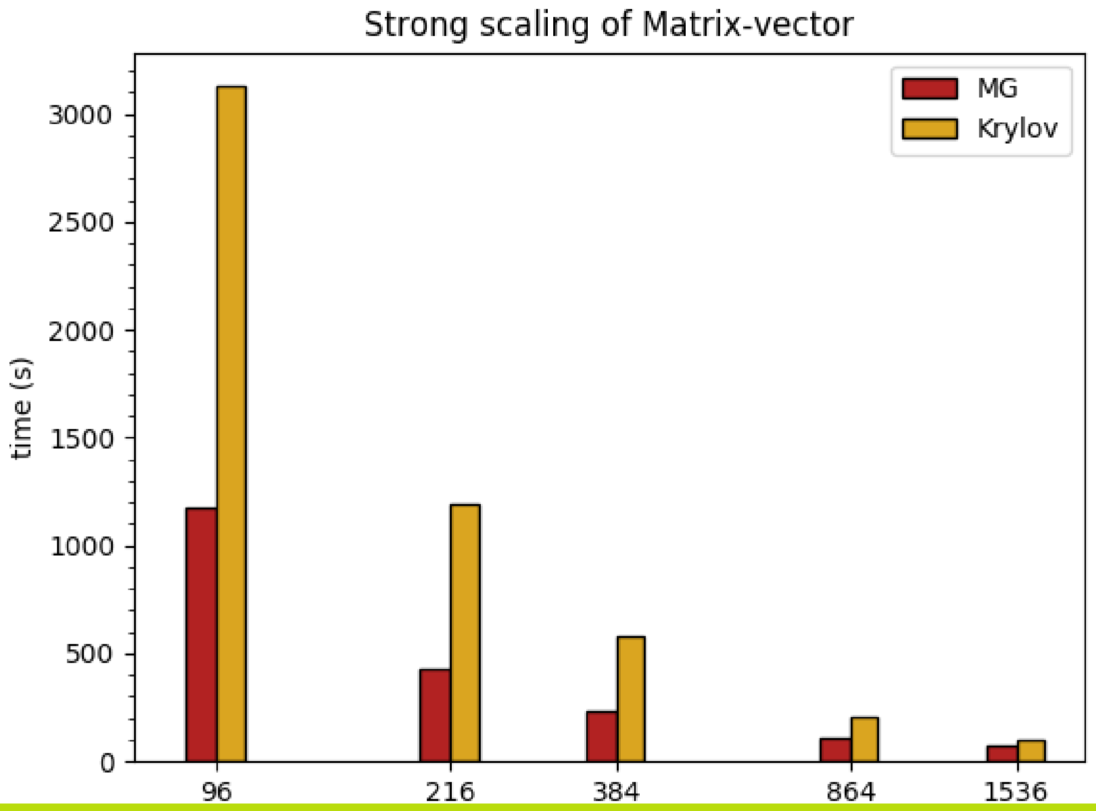
Global comms

Lower is better
Both algs have global sums in outer solve, plus limited diagnostic
Kr still has GS for inner solver
10^{-2} → only a few iterations.
Very large variation due to Aries adaptive routing



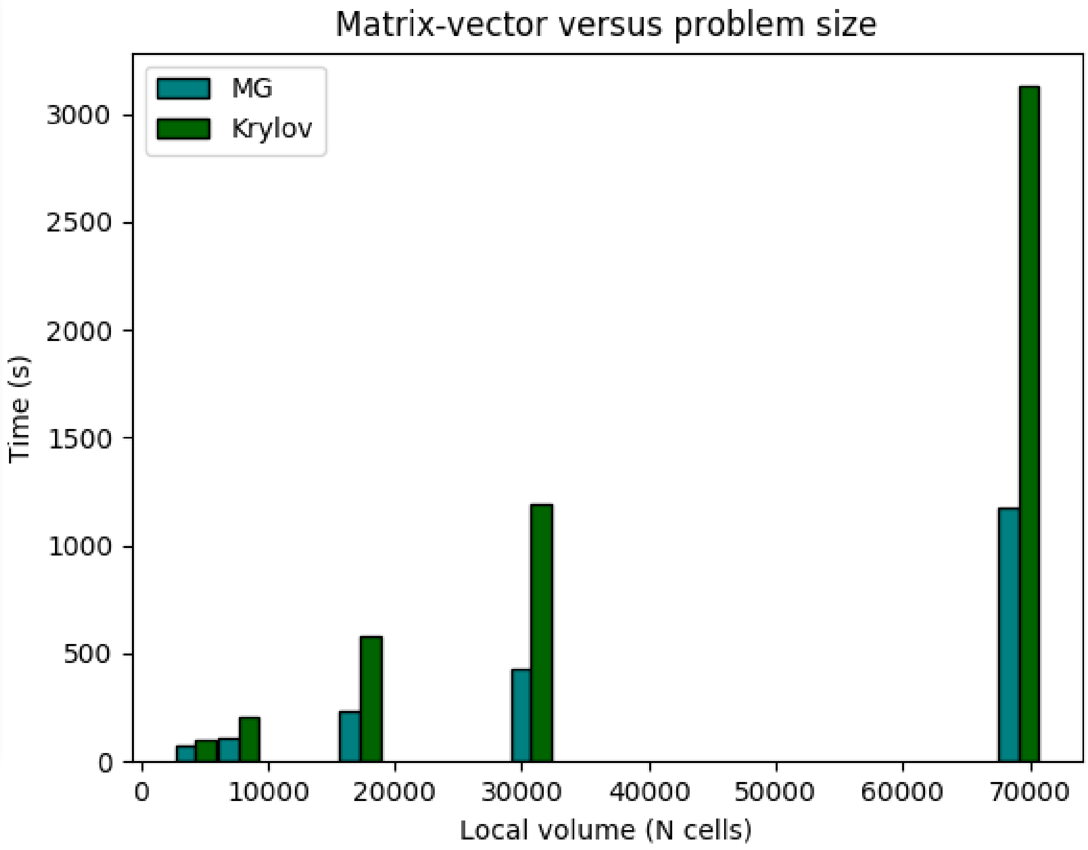
Matrix-vector

Lower is better
MG is much more efficient
Much less work
Used Schur-precon scaling to estimate MG 96 node cost
No comms, hence good scaling

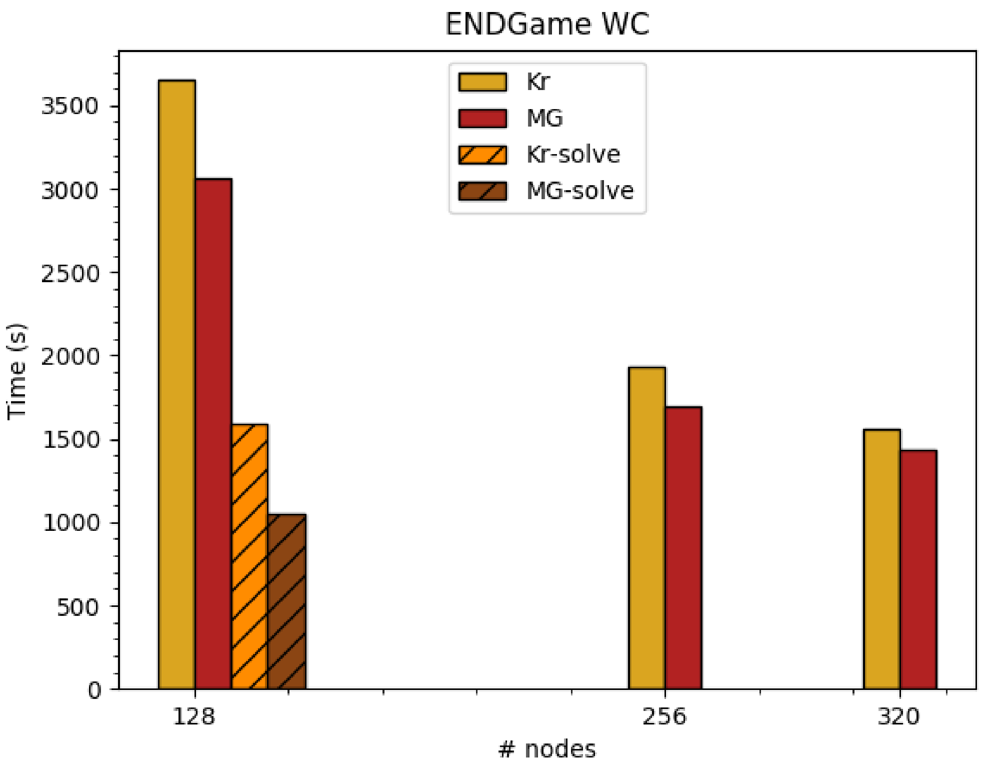


Matrix-vector II

Lower is better
X-axis is linear
Data are reversed
Shows cost of computation
Scales linearly with problem size
Smallest problem size not much work *c.f.* with comms
Fischer *et al*, suggests strong scaling limit is around LV~10000 (my interpretation)
doi 10.2514/6.2015-3049



Multigrid & Mixed-precision



UM ENDGame N1280
Multigrid for Helmholtz solve.
Faster than Krylov (BiCGStab)
Big effect on solver
Polar singularity (Advection) spoils scaling
Lower precision helps both equally
MG can help avoid convergence issues

LFRic solver, currently 64-bit
Mixed-precision solver planned
Coarse grids can be done in lower precision, especially comms (8-bit)

Met Office

Conclusions

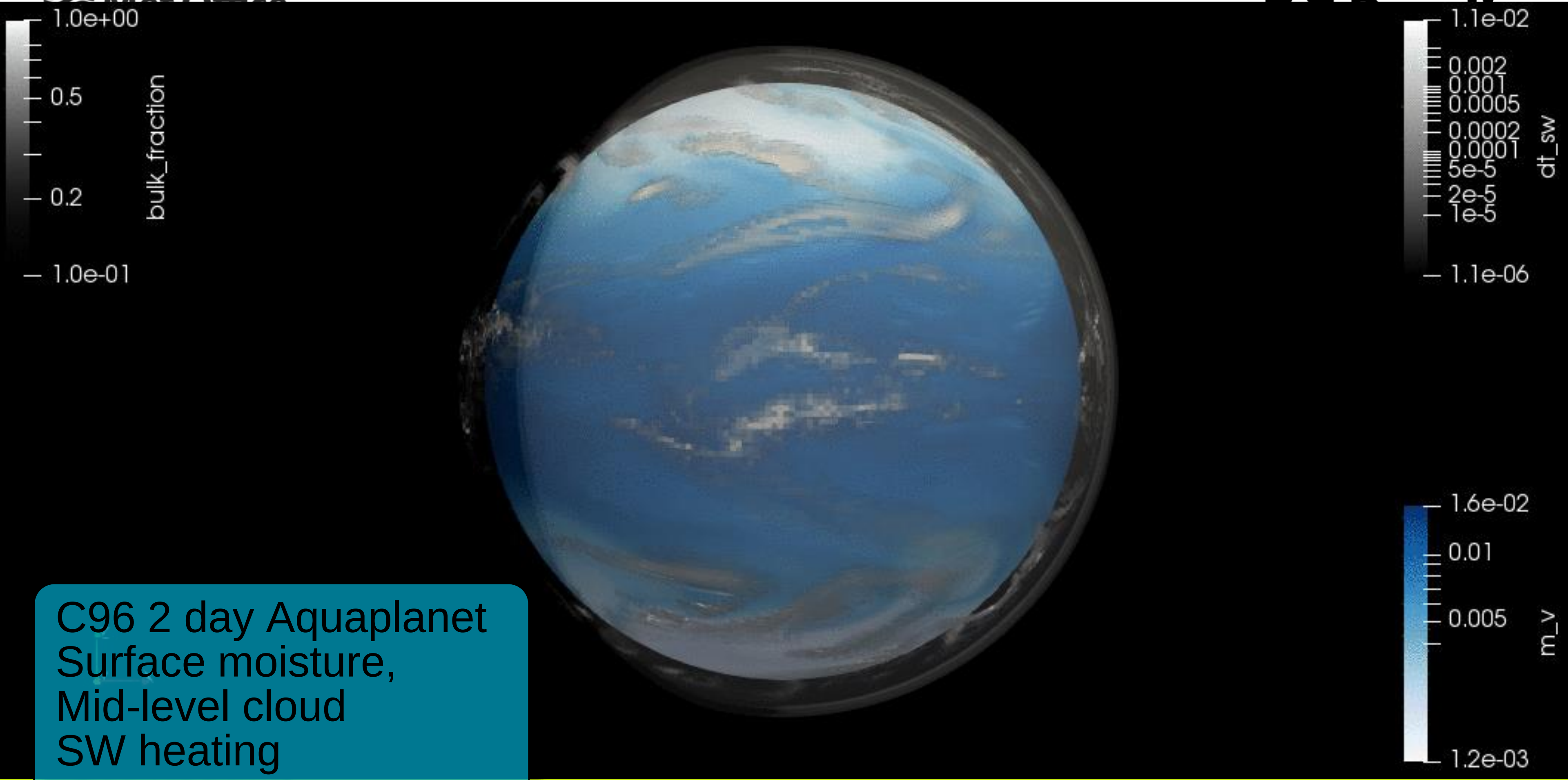
University of Reading

Complex interplay between accuracy, efficiency, algorithm and implementation

64-bit arithmetic is expensive. Lower precision can, with care be used without compromising accuracy – depending on algorithm and implementation

Choice of algorithm, such as Multigrid to avoid global sums or Redundant computation to reduce communication are in some some being deployed to exploit architectural features - scaling

Accelerator architectures will require specific algorithmic choices



C96 2 day Aquaplanet
Surface moisture,
Mid-level cloud
SW heating

Met Office

Some names

University of Reading

工合 Gungho: Mixed finite element dynamical core

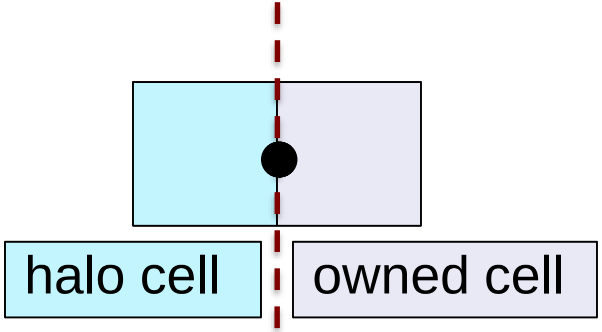
LFRic: Model infrastructure for next generation modelling

PSyclone: Parallel Systems code generation used in LFRic and Gungho

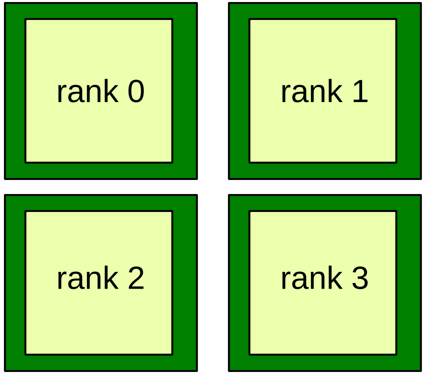
Unified Model

UM: Current modelling environment (UM parametrisations are being reused in LFRic

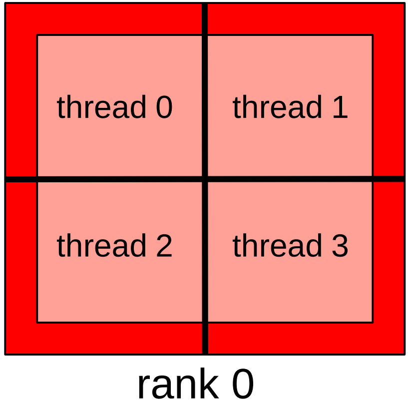
Redundant computation



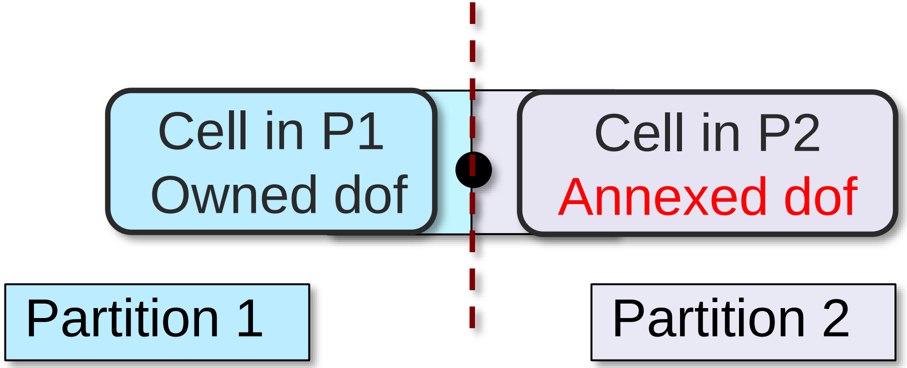
Dof living on shared (partitioned) entity (edge).
Receive contribution from owned and halo cell.
Redundant compute contribution in halo to shared dof.
Less communication



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

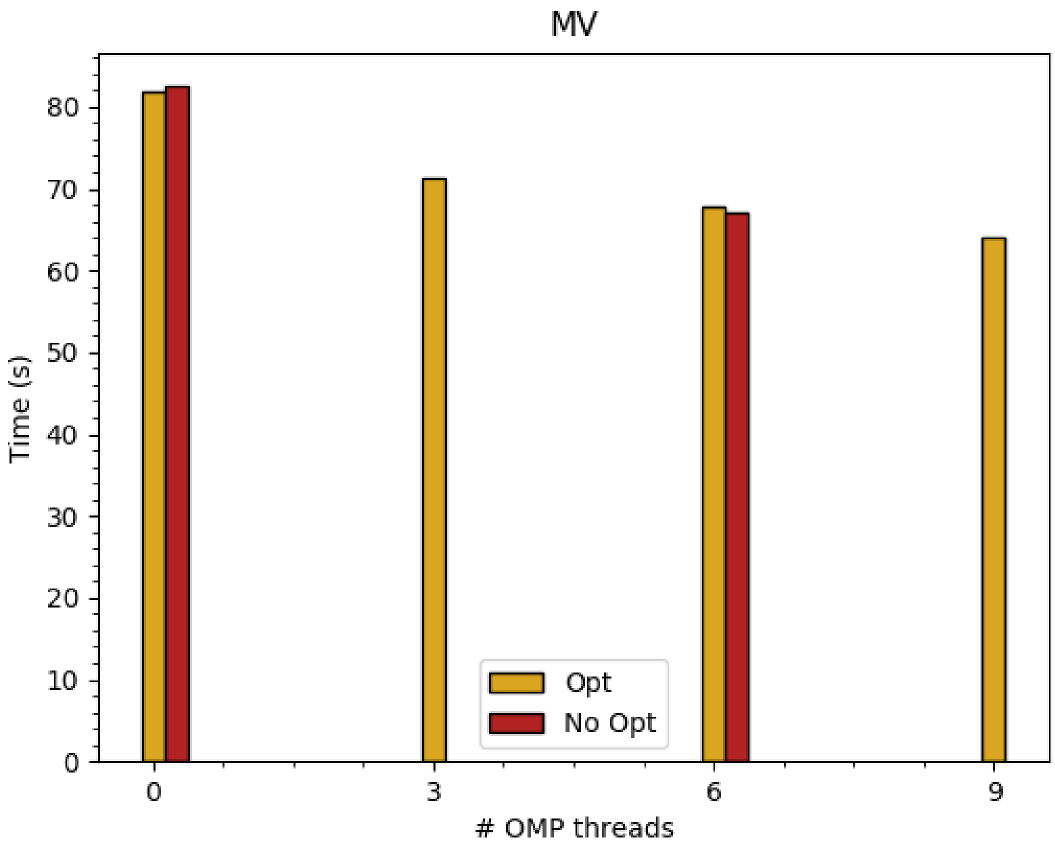


Annexed dofs



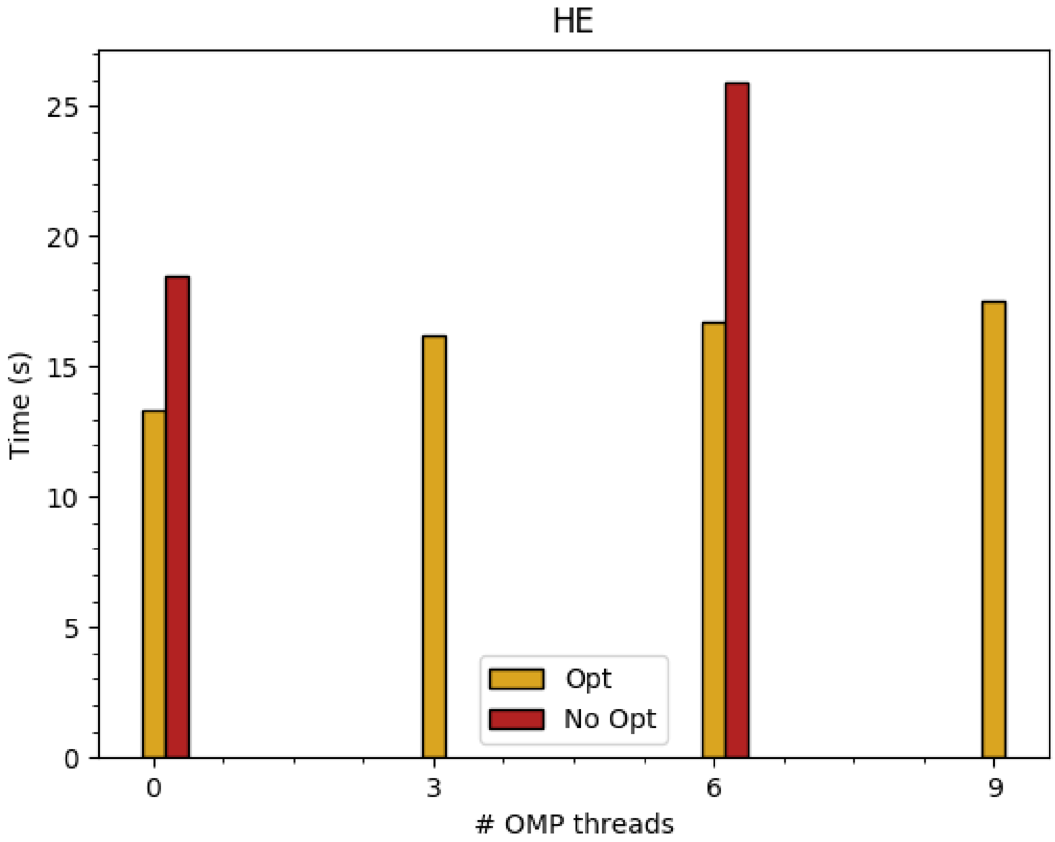
Point-wise computations (e.g. set field to a scalar) loop over dofs
Looping to owned dofs → halo exchange required for P2
Looping to annexed dofs is now transformation in Psyclone
Small increase in redundant computation
Large reduction in number of halo exchanges required

Redundant computation



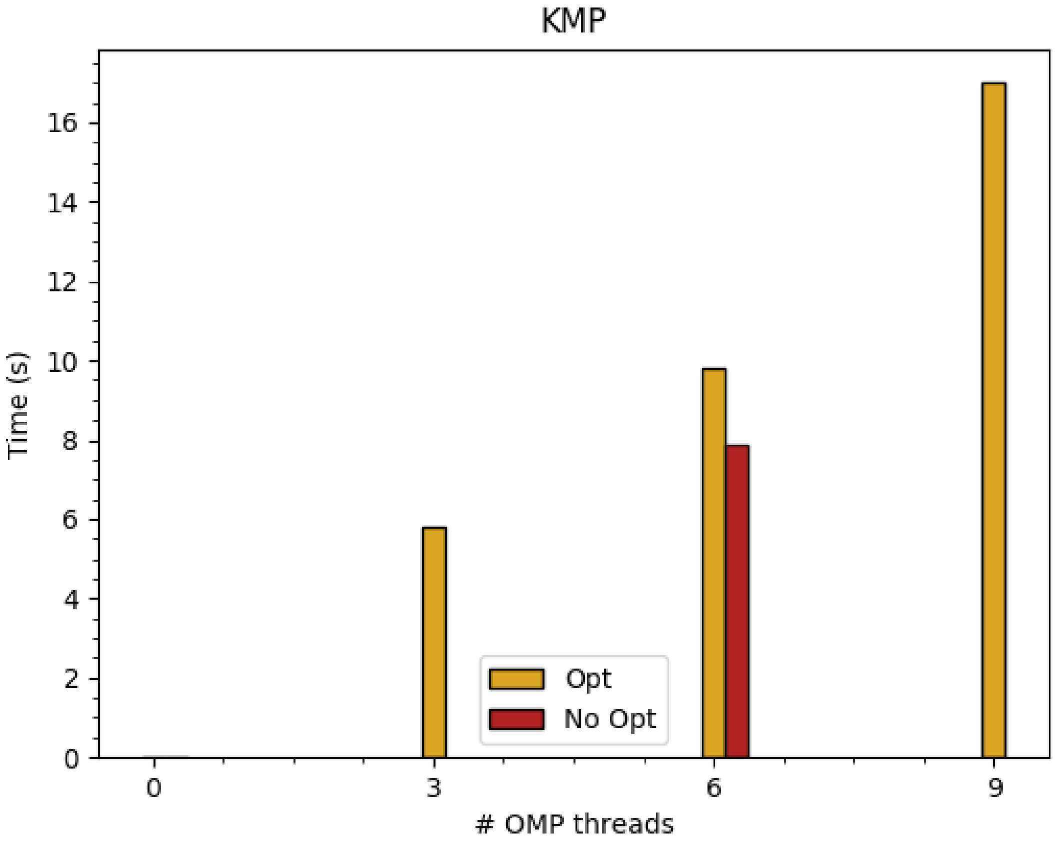
C288 running on 96 nodes
"0" thread is 36 MPI ranks per node
3/12, 6/6 and 9/4 (Dual socket
Profile by CrayPAT
Pure computation
OMP is faster as it has less work

Local comms



More threads → *fewer* MPI ranks send/receive *bigger* messages
Import to tune Rendevous/Eager protocol limit (larger)

OMP synchronisation



Intel 17 compiler
This seems very large to me
Can't compiler F2K3 OO objects with Cray or PGI
Single kernel results suggest Cray is better
OMP_WAIT_POLICY=active

Programming Model

Met Office

University of Reading

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), dim                          :: 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

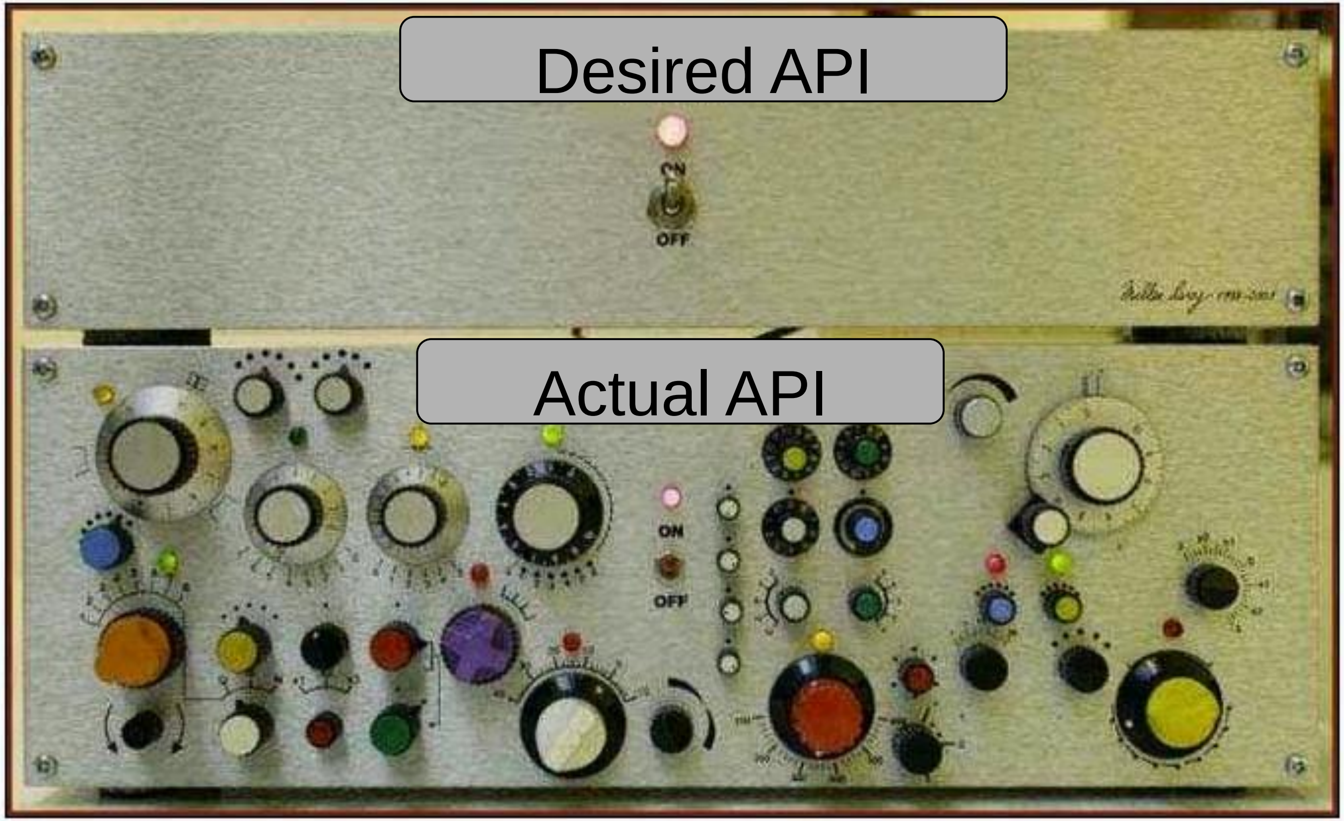
Abstraction is *broken* by parallel/performance/memory features exposed

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



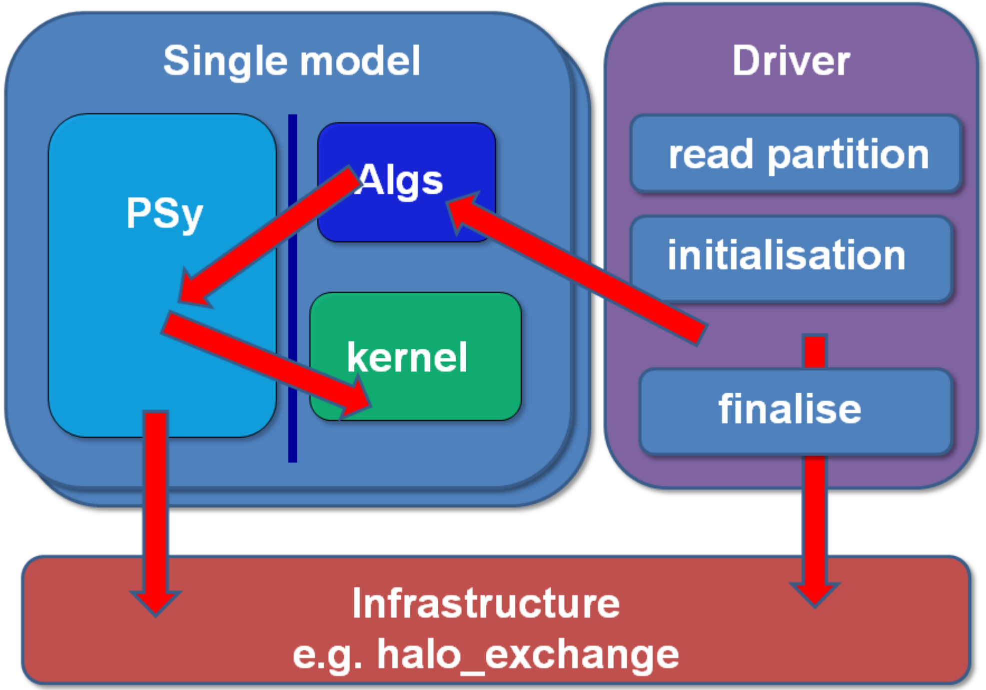
Separation of Concerns

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.

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*

Met Office

Kernel Metadata

University of Reading

- 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

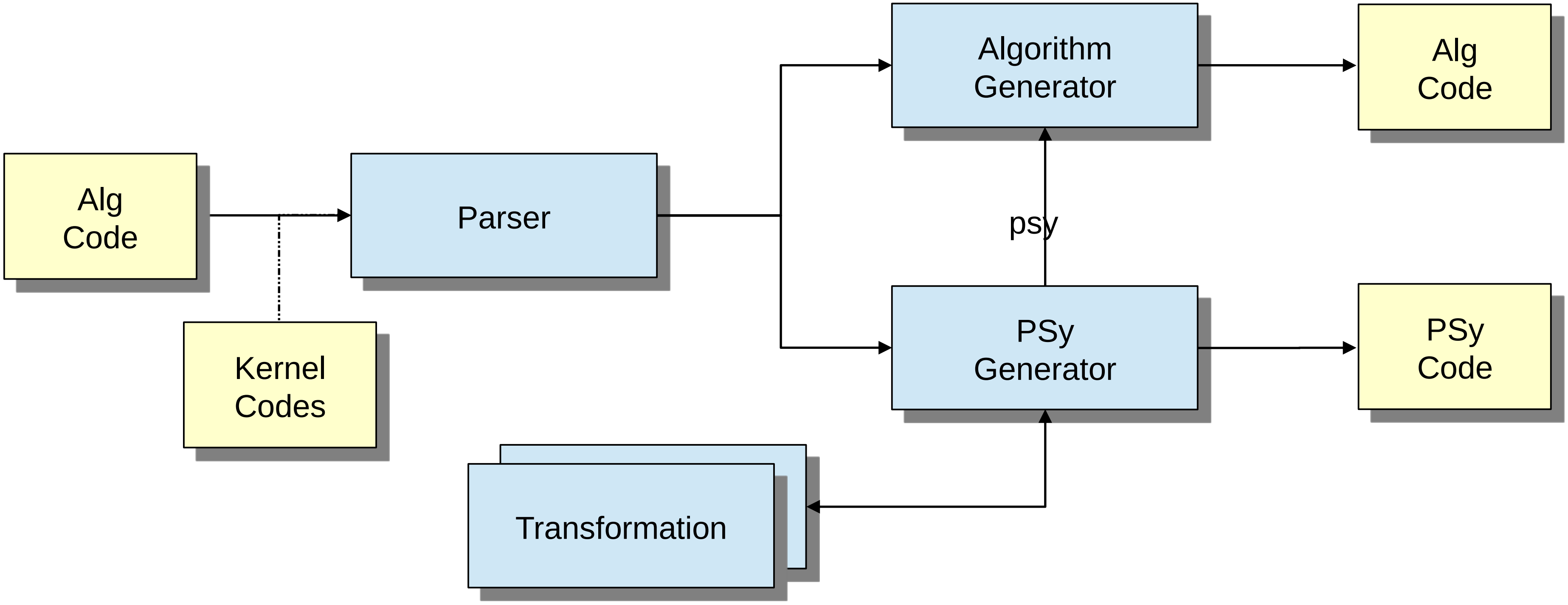
What is PSyclone (Brikipedia)?

"a former weathercaster turned crazy bad-guy with a craving for destruction"

http://lego.wikia.com/wiki/Psyclone

Developed at STFC Hartree
R. Ford, A. Porter, S. Siso
J. Henrichs, BoM
I Kavcic, M Hambley, CMM (MO)
Works with PSyKAl API

PSyclone



Alg
Code
Parser
Kernel
Codes
Algorithm
Generator
psy
PSy
Generator
Alg
Code
PSy
Code
Transformation

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]
```

Met Office

Generated PSy layer

University of Reading

Update halos
YAXT→MPI

colouring from infrastructure

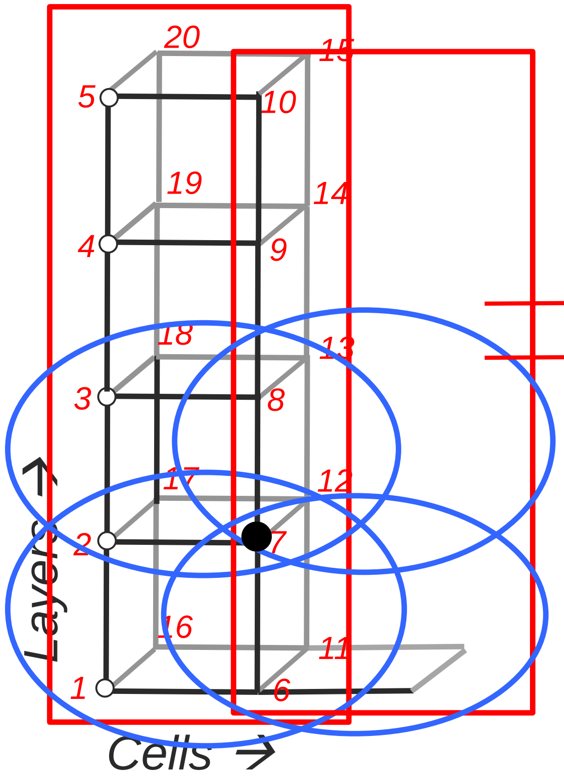
OpenMP workshare across cells in colour

```
!
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

Data layout, unstructured mesh

W_0 space (vertices)



1	2	3	4	5	6	7	8	9	10	…

Data array (1-d)

Dofmap 2-d array

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

ndof per cell

ncell

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

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