An Introduction to the LFRic Project

Mike Hobson

Acknowledgements: LFRic Project

Met Office:

Sam Adams, Tommaso Benacchio, Matthew Hambley, Mike Hobson, Chris Maynard, Tom Melvin, Steve Mullerworth, Stephen Pring, Steve Sandbach, Ben Shipway, Ricky Wong.

STFC, Daresbury Labs:

Rupert Ford, Andy Porter, Karthee Sivalingam.

University of Manchester:

Graham Riley, Paul Slavin.

University of Bath:

Eike Mueller.

Monash University, Australia:

Mike Rezny.

Project History

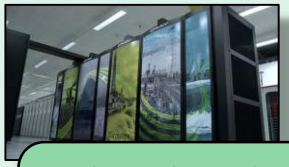
Some very worthy people had some serious thoughts about the future…

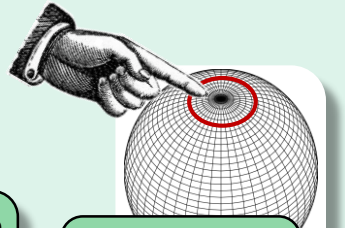
Diverse Future HPC:
MPI? OpenMP?
Accelerators?
GPUs? ARM? …?

Exascale

GungHo Project 工合

Met Office | NERC SCIENCE OF THE ENVIRONMENT | Science & Technology Facilities Council





Need to make porting the codes from machine to machine easier:
Flexible Implementation

Need a more scalable dynamical core

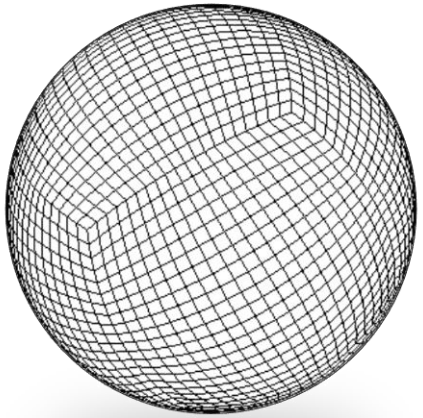
Over the lifespan of an NWP model, all we really know is that we don't know very much.

Scalability Problems

GungHo

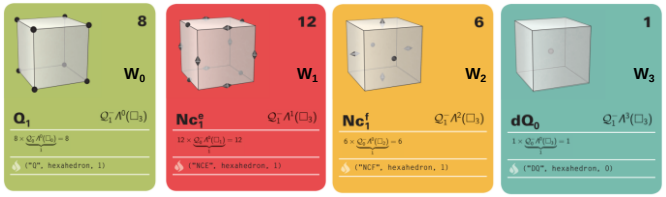
- Project ran from 2011 to 2016
- Collaboration between: Met Office, STFC Daresbury and various Universities through NERC
- Split into two activities:
  - Natural Science: new dynamical core
  - Computational Science: new infrastructure

GungHo: Natural Science



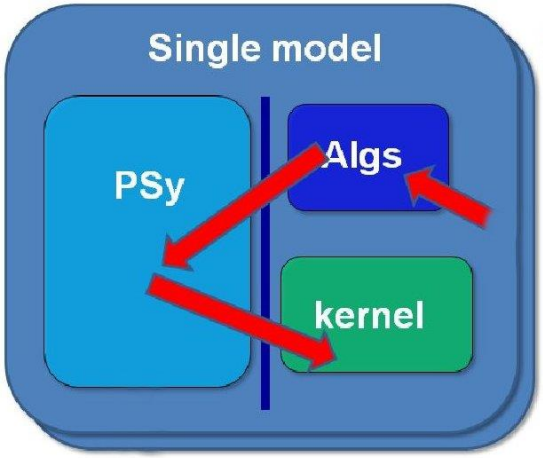
- Mesh choice: No singularities at poles
  - Current choice: cubed-sphere
  - Horizontal adjacency lost
  - Vertically adjacent cells contiguous in memory
- Science choices –
  Staniforth & Thuburn (2012) came up with
  "Ten essential and desirable properties of a dynamical core"
- Mixed finite elements

1. Mass conservation
2. Accurate representation of balanced flow and adjustment
3. Computational modes should be absent or well controlled
4. Geopotential and pressure gradients should produce no unphysical source of vorticity
   ⇒ $\nabla\times\nabla p = \nabla\times\nabla\Phi = 0$
5. Terms involving the pressure should be energy conserving ⇒ $u\cdot\nabla p + p\nabla\cdot u = \nabla\cdot(pu)$
6. Coriolis terms should be energy conserving
   ⇒ $u\cdot(\Omega\times u) = 0$
7. There should be no spurious fast propagation of Rossby modes; geostrophic balance should not spontaneously break down
8. Axial angular momentum should be conserved
9. Accuracy approaching second order
10. Minimal grid imprinting



GungHo: Computational Science

- Need to be able to mitigate against an uncertain future
- So it was decided to separate out the natural science code
  (Single Source Science)
  from the system infrastructure, parallelisation and optimisation
  (Separation of Concerns)
- Infrastructure and optimisations provided by a code generator
- Introduced a layered, "single-model" structure
- Object-orientated Fortran 2003



Spawning the LFRic Project

- Continue the work from GungHo.
- But develop the code from just a dynamical core into a full weather and climate model
- Named after **Lewis Fry Richardson**
  1922: *Weather Prediction by Numerical Process*
- Develop the infrastructure further
- Bring in Physics parameterisations
  - Reuse of UM code where possible
  - Couple these finite-different codes to the new finite-element core

PSyKAl Infrastructure:

Parallel Systems, Kernels, Algorithms

Scientific code doesn't need to be changed for different HPC architectures

Algorithm layer

Scientists write in a domain-specific language aligned with the written equations

Fortran-like DSL

Scientist-written

- Refers to kernels that do the work
- All operations are on whole fields
- No optimisations

Parallel-Systems (PSy) layer

Aims to optimise for different hardware

Optimisation script

Python

Input

PSyclone code generator

Input

Input

Generate

Generate

Kernel layer

Kernel code

Metadata describes how to unpack data

Scientist-written Fortran

Algorithm code

Code generated from the DSL

Generated Fortran

Fortran call

PSy-layer code

- Breaks fields down into columns of data
- Calls kernels each column
- Shared and distributed memory parallelism and other optimisations

Generated Fortran

Fortran call

Science code for a column

LFRic: Algorithm Code (Fortran-like DSL)

Written by Scientists

Some (abridged) Algorithm layer code:

```
module rk_alg_timestep_mod
use pressure_gradient_kernel_mod, only: pressure_gradient_kernel_type
 subroutine rk_alg_step( … result, rho, theta, … )
 implicit none
 type(field_type), intent(inout) :: result, rho, theta
…
  do stage = 1,num_rk_stage
…
   if( wtheta_off ) then
    call invoke( pressure_grad_kernel_type(result, rho, theta) )
   end if
…
  end do
…
 end subroutine
end module
```

PSyKAl Infrastructure:

Parallel Systems, Kernels, Algorithms

Scientific code doesn't need to be changed for different HPC architectures

Algorithm layer

Scientists write in a domain-specific language aligned with the written equations

```
subroutine iterate_alg(rho,theta, u, ...)
...loops, if-blocks etc...
  call invoke(
    pressure_grad_kernel(rho,theta),
    enforce_bc_kernel(u,coords)
  )
...more invoke calls...
end subroutine
```

Fortran-like DSL

Scientist-written

- Refers to kernels that do the work
- All operations are on whole fields
- No optimisations

Parallel-Systems (PSy) layer

Aims to optimise for different hardware

Optimisation script

Python

Input

PSyclone code generator

Input

Input

Generate

Generate

Kernel layer

Kernel code

```
module pressure_grad_kernel_mod

  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, W0)  &
    /)
  type(func_type) :: meta_funcs(3) = (/ &
    func_type(W2, GH_BASIS),          &
    func_type(W3, GH_BASIS),          &
    func_type(W0, GH_BASIS, GH_DIFF_BASIS) &
    /)
  integer :: iterates_over = CELLS
end type
```

Metadata describes how to unpack data

Algorithm code

```
call invoke_1(result, rho, theta, coords)
```

Code generated from the DSL

Generated Fortran

Fortran call

PSy-layer code

- Breaks fields down into columns of data
- Calls kernels each column
- Shared and distributed memory parallelism and other optimisations

Generated Fortran

Fortran call

```
subroutine pressure_gradient_code( ... )
do k = 0, nlayers-1
  ...
  end do
end do
end subroutine
end module
```

Science code for a column

Scientist-written Fortran

LFRic: Kernel Code (Fortran)

Written by Scientists

Some (abridged) Kernel layer code.

Metadata tells PSyclone how to unpack data:

```
module pressure_grad_kernel_mod
 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, W0)        &
   /)
 type(func_type) :: meta_funcs(3) = (/     &
   func_type(W2, GH_BASIS, GH_DIFF_BASIS),&
   func_type(W3, GH_BASIS),                &
   func_type(W0, GH_BASIS, GH_DIFF_BASIS) &
   /)
  integer :: iterates_over = CELLS
end type
…
```

LFRic: Kernel Code (Fortran)

Written by Scientists

Some (abridged) Kernel layer code.

Science code (for a column of nlayers levels):

```
…
  subroutine pressure_gradient_code( … result, rho, theta, &
                                …sizes, maps, basis functions for all function spaces  )

    real, intent(inout) :: result( ndf_w2 )
    real, intent(in)    :: rho( ndf_w3 )
    real, intent(in)    :: theta( ndf_w0 )
…
    do k = 1, nlayers
      do df = 1, num_dofs_per_cell
        result(map(df)+k)=theta(map(df)+k) * …
      end do
    end do
…
  end subroutine
end module
```

PSyKAl Infrastructure:

Parallel Systems, Kernels, Algorithms

Scientific code doesn't need to be changed for different HPC architectures

Algorithm layer

Scientists write in a domain-specific language aligned with the written equations

```
subroutine iterate_alg(rho,theta, u, ...)
...loops, if-blocks etc...
  call invoke(
    pressure_gradient_kernel(rho,theta),
    enforce_bc_kernel(u,coords)
  )
...more invoke calls...
end subroutine
```

Fortran-like DSL

Scientist-written

- Refers to kernels that do the work
- All operations are on whole fields
- No optimisations

Parallel-Systems (PSy) layer

Aims to optimise for different hardware

Optimisation script

Python

Input

PSyclone code generator

Input

Input

Generate

Generate

Kernel layer

Kernel code

```
module pressure_grad_kernel_mod

  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, W0)  &
    /)
  type(func_type) :: meta_funcs(3) = (/ &
    func_type(W2, GH_BASIS),          &
    func_type(W3, GH_BASIS),          &
    func_type(W0, GH_BASIS, GH_DIFF_BASIS) &
    /)
  integer :: iterates_over = CELLS
end type
```

Metadata describes how to unpack data

Algorithm code

```
call invoke_1(r_solver, rho, theta, coords)
```

Code generated from the DSL

Generated Fortran

Fortran call

PSy-layer code

- Breaks fields down into columns of data
- Calls kernels each column
- Shared and distributed memory parallelism and other optimisations

Generated Fortran

Fortran call

```
subroutine pressure_gradient_code( ... )
  do k = 0, nlayers-1
    ...
  end do
end do
end subroutine

end module
```

Science code for a column

Scientist-written Fortran

LFRic: PSy Code (Generated Fortran)

Written by PSyclone

Some (abridged) PSy layer code:

```
MODULE psy_rk_alg_timestep_mod
 SUBROUTINE invoke_2_pressure_gradient_kernel_type(result, rho, theta, …)
 TYPE(field_type), intent(inout) :: result, rho, theta
 TYPE(field_proxy_type) result_proxy, rho_proxy, theta_proxy
 result_proxy = result%get_proxy()
 rho_proxy = rho%get_proxy()
 theta_proxy = theta%get_proxy()
…
 DO cell=1,mesh%get_last_halo_cell(1)
  map_w2 => result_proxy%funct_space%get_cell_dofmap(cell)
  map_w3 => rho_proxy%funct_space%get_cell_dofmap(cell)
  map_w0 => theta_proxy%funct_space%get_cell_dofmap(cell)
  CALL pressure_gradient_code( … result_proxy%data, rho_proxy%data, theta_proxy%data, &
                                …sizes, maps, basis functions for all function spaces )
 END DO
…
END SUBROUTINE
END MODULE
```

LFRic: PSy Code (Generated Fortran)

Written by PSyclone

Addition of code to support distributed memory parallelism:

```
…
IF (result_proxy%is_dirty(depth=1)) CALL result_proxy%halo_exchange(depth=1)
IF (rho_proxy%is_dirty(depth=1))    CALL rho_proxy%halo_exchange(depth=1)
IF (theta_proxy%is_dirty(depth=1))  CALL theta_proxy%halo_exchange(depth=1)
DO cell=1,mesh%get_last_halo_cell(1)
 map_w2 => result_proxy%funct_space%get_cell_dofmap(cell)
 map_w3 => rho_proxy%funct_space%get_cell_dofmap(cell)
 map_w0 => theta_proxy%funct_space%get_cell_dofmap(cell)
 CALL pressure_gradient_code( … result_proxy%data, rho_proxy%data, theta_proxy%data, &
                              sizes, maps, basis functions for all function spaces )
END DO
CALL result_proxy%set_dirty()
…
```

LFRic: PSy Code (Generated Fortran)

Written by PSyclone

Addition of code to support OpenMP parallelism:

```
…
DO colour=1,ncolour
 !$omp parallel do default(shared), private(cell,map_w2,map_w3,map_w0), schedule(static)
 DO cell=1,ncp_colour(colour)
  map_w2 => result_proxy%funct_space%get_cell_dofmap(cmap(colour, cell))
  map_w3 => rho_proxy%funct_space%get_cell_dofmap(cmap(colour, cell))
  map_w0 => theta_proxy%funct_space%get_cell_dofmap(cmap(colour, cell))
  CALL pressure_gradient_code( … result_proxy%data, rho_proxy%data, theta_proxy%data, &
                               sizes, maps, basis functions for all function spaces )
 END DO
 !$omp end parallel do
END DO
…
```

PSyKAl Infrastructure:

Parallel Systems, Kernels, Algorithms

Scientific code doesn't need to be changed for different HPC architectures

Algorithm layer

Scientists write in a domain-specific language aligned with the written equations

Fortran-like DSL

Scientist-written

- Refers to kernels that do the work
- All operations are on whole fields
- No optimisations

Input

Parallel-Systems (PSy) layer

Aims to optimise for different hardware

Optimisation script

Python

Input

PSyclone code generator

Input

Kernel layer

Kernel code

Metadata describes how to unpack data

Scientist-written Fortran

Generate

Generate

Algorithm code

Code generated from the DSL

Generated Fortran

Fortran call

PSy-layer code

- Breaks fields down into columns of data
- Calls kernels each column
- Shared and distributed memory parallelism and other optimisations

Generated Fortran

Fortran call

Science code for a column

LFRic: Algorithm Code (Code Generated from DSL)

Written by PSyclone

Some (abridged) Algorithm layer code:

```
module rk_alg_timestep_mod
use pressure_gradient_kernel_mod, only: pressure_gradient_kernel_type
 subroutine rk_alg_step( … u, rho, theta, … )
 implicit none
 type(field_type), intent(inout) :: u, rho, theta
…
  do stage = 1,num_rk_stage
…
   if( wtheta_off ) then
    call invoke_2_pressure_gradient_kernel_type(result, rho, theta)
   end if
…
  end do
…
 end subroutine
end module
```

PSyKAl Infrastructure:

Parallel Systems, Kernels, Algorithms

Scientific code doesn't need to be changed for different HPC architectures

Algorithm layer

Scientists write in a domain-specific language aligned with the written equations

```
subroutine iterate_alg(rho, theta, u, ...)
  ...loops, if-blocks etc...
  call invoke(
    ...
  )
  ...more invoke calls...
end subroutine
```

Fortran-like DSL

Scientist-written

- Refers to kernels that do the work
- All operations are on whole fields
- No optimisations

Parallel-Systems (PSy) layer

Aims to optimise for different hardware

Optimisation script

Python

PSyclone code generator

Kernel layer

Kernel code

```
module pressure_grad_kernel_mod
  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, W0) &
    /)
  type(func_type) :: meta_funcs(3) = (/ &
    func_type(W2, GH_BASIS), &
    func_type(W3, GH_BASIS), &
    func_type(W0, GH_BASIS, GH_DIFF_BASIS) &
    /)
  integer :: iterates_over = CELLS
end type
```

Metadata describes how to unpack data

Input

Input

Input

Generate

Generate

Algorithm code

```
call invoke_1(result, rho, theta, coords)
```

Code generated from the DSL

Generated Fortran

Fortran call

PSy-layer code

- Breaks fields down into columns of data
- Calls kernels each column
- Shared and distributed memory parallelism and other optimisations

Generated Fortran

Fortran call

```
subroutine pressure_gradient_code( ... )
  do k = 0, nlayers-1
    ...
  end do
end do
end subroutine
end module
```

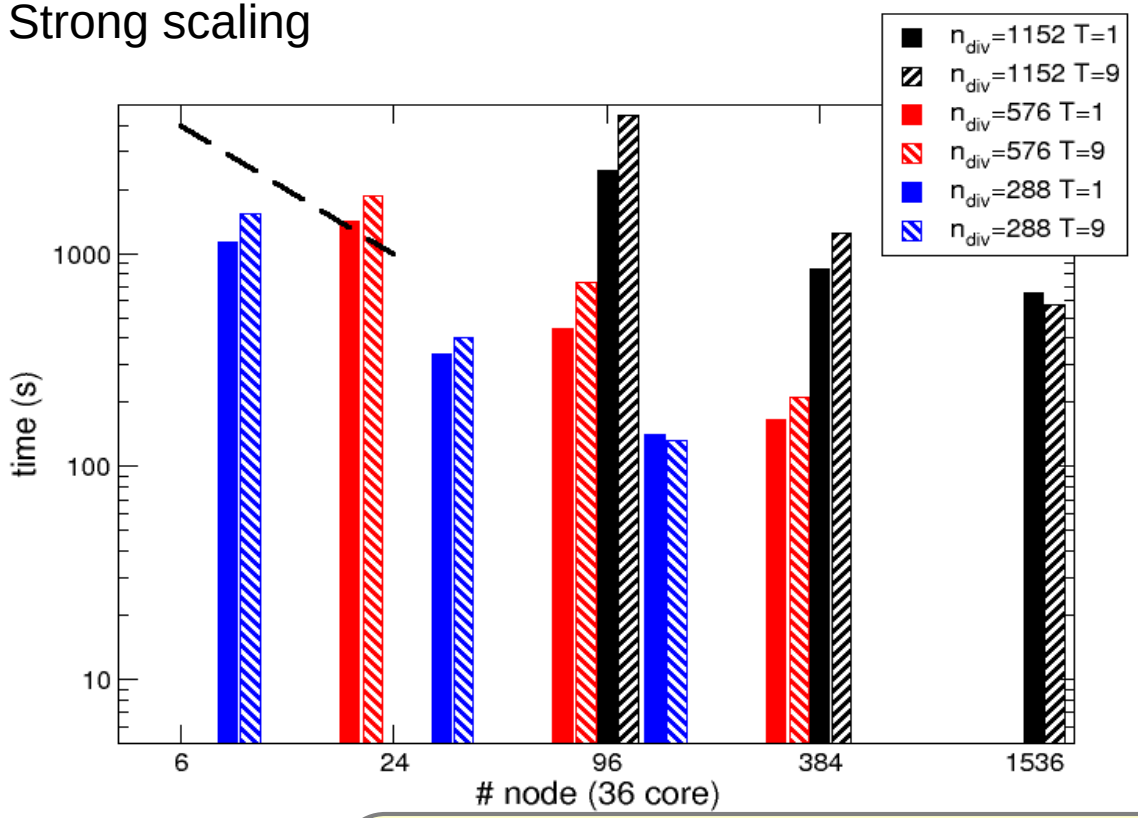
Science code for a column

Scientist-written Fortran

Results

Total job size remains constant, so work per processor reduces as processor count increases.
For perfect scaling, the bars for a particular problem size should reduce in height following the slope of the dashed line.

Strong scaling



Solid bars – parallelism achieved through MPI (distributed memory)

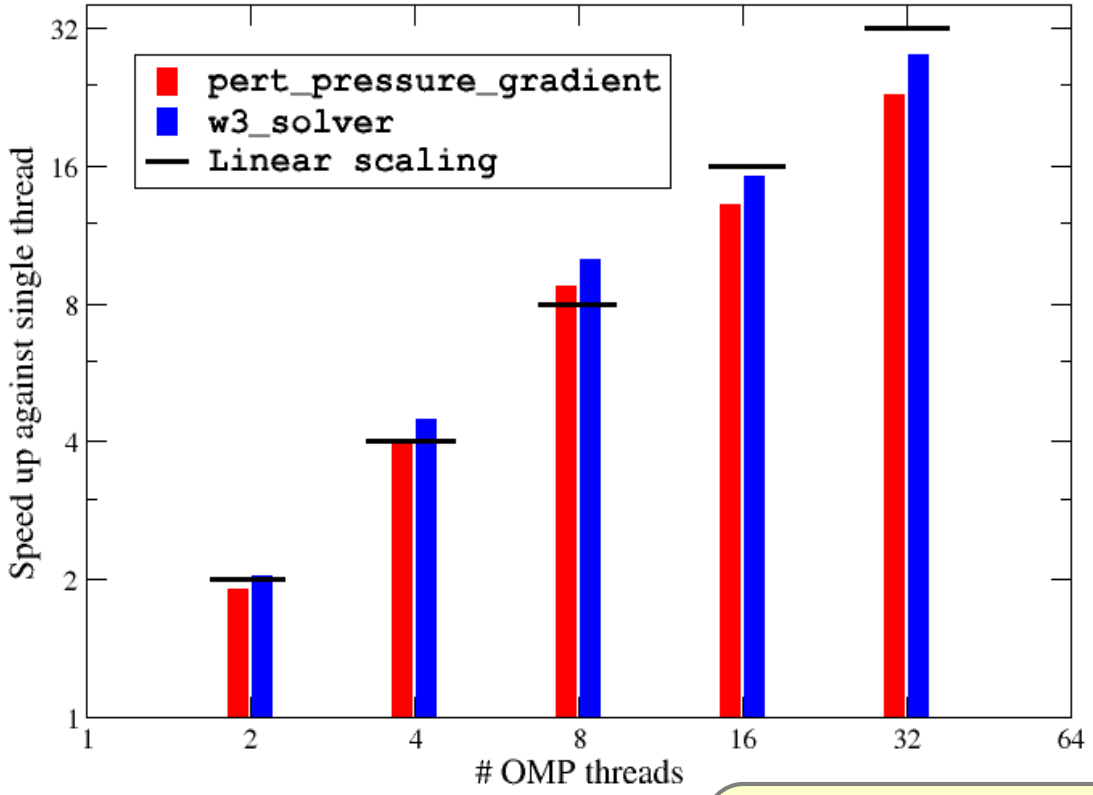
Hatched bars – parallelism achieved through OpenMP (shared memory)

Full model run (on 18-core socket Broadwell)
Gravity wave test on a cubed-sphere global mesh with 20 vertical levels.
Running with a scaled 1/10 size Earth at lowest order for 20 time steps.
Naïve solver preconditioner → short time-step (Δt=10s).
Up to 8 million cells per level (9 km resolution on a full sized Earth).

Results

Each thread (cores) has an L2 cache, so for fixed problem size, more threads means more L2 cache in total.
Between 2 and 8 threads, vertical columns fit into total L2 cache resulting in super-linear scaling.

Kernel performance



Individual kernel scaling
Single node (16-core socket Haswell).
Kernel speed up *c.f.* single OpenMP thread.
For two example kernels.