Science & Technology Facilities Council

GOcean

Mike Ashworth
Rupert Ford
Stephen Pickles
Andy Porter
Graham Riley

STFC Daresbury

工合shun

Overview

- Intro to GungHo
- Current status of GungHo
- Example
- Adding a FD structured code to GungHo
- Adding a FV unstructured code to GungHo
  - Just use existing approach?

GungHo

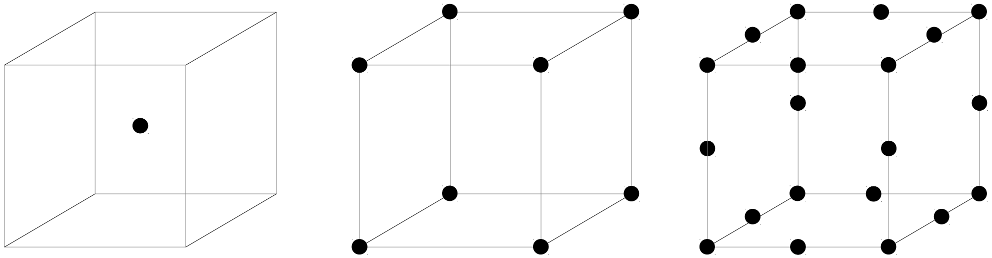
- Globally Uniform, Next Generation, Highly Optimised
- “To research, design and develop a new dynamical core suitable for operational, global and regional, weather and climate simulation on massively parallel computers of the size envisaged over the coming 20 years.”
- Remove the pole problem (replace lat-lon grid)
- aimed at massively parallel computers – 10^6 way parallel → petaflop
- Split into two phases:
  - 2 years “research” (2011-13)
  - 3 years “development” (2013-2016)
- Met Office, STFC, Universities of: Bath, Exeter, Imperial, Leeds, Manchester, Reading, Warwick

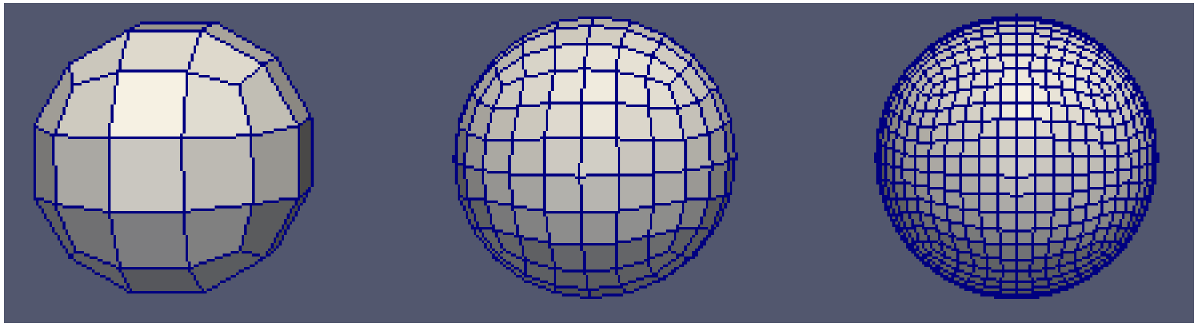
Current Status of GungHo

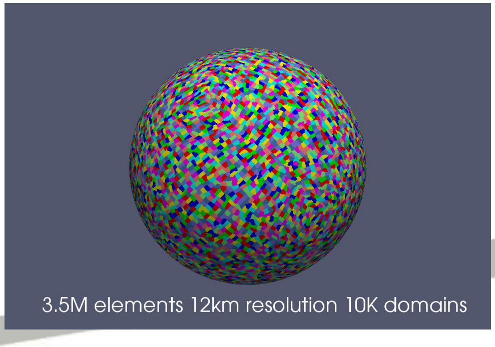
- Numerics (Nigel Wood + ...)
- Algorithm and Kernel (David Ham + ...)
- Distributed Memory (Mike Hobson + …)
- Code Generation and Optimisation (Psy Layer)
  - Rupert Ford, Chris Maynard, Graham Riley, Lawrence Mitchell

Current Status of GungHo

- (Most likely) Cubed Sphere
  - Extruded (columnar) mesh (2d+1d)
- Low order finite element approach
  - rt0, rt1, bdfm
  - Dofs





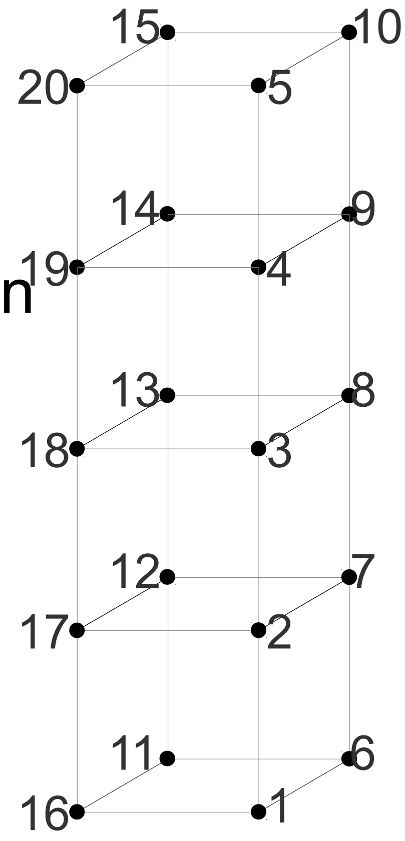


工合shun

Science & Technology
Facilities Council

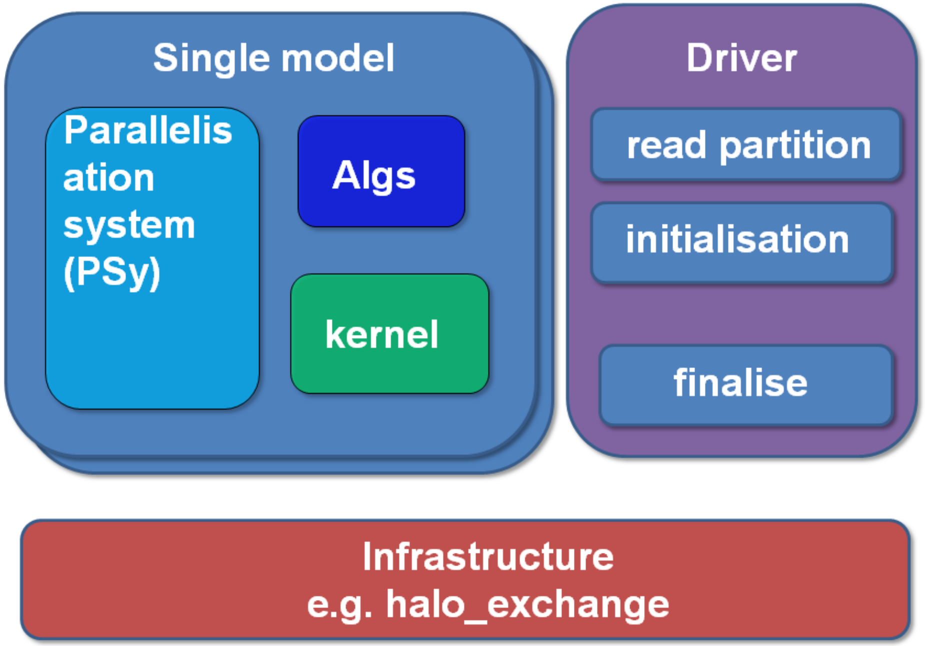
Data model

- Unstructured mesh format in horizontal, direct in the vertical
- “K-inner” for performance
- Separate mesh and location of data on a mesh (function space)
- Vertically aligned degrees of freedom are contiguous
- Dofmap references base element (1,2,6,7,11,12,16,17)
- Designed to support different mesh and element types



Separation of Concerns

- PSy-Kern-Alg-Inf separation (PSyKAI)
- Manual and/or Generated PSy layer



© Crown copyright Met Office

Kernels

- Kernels will be hand written conforming to some standard
- Kernels will be scientific
  - There will also be library routines e.g. linear algebra
- Kernels will have metadata associated with them e.g.
  - Intents (extending fortran's in and out)
  - Halo access information
  - what the kernel iterates over
- Kernels are column based : caveat, iterate over all dofs
  - Work with single columns to start with

Algorithm

- Algorithm layer will be hand written and conform to fortran 2003
- Invoke approach
  - Algorithm layer will (fully?) **specify** what the PSy layer has to do
    - Should be in a way that is “obvious” to the programmer
  - Algorithm layer engine “specifications” will be pre-processed to specific calls which replace original
  - Invocation should take a 'list' of kernel specs

```
call invoke(func(arg1,arg2,arg3),...)
```

PSy

- The Optimised PSy may be generated
  - Manual “reference” version
  - Should be easily debuggable (and modifiable??)
- Functional responsibility
  - iterating over columns
  - Mapping of algorithm fields types/objects to data required by kernel
    - Number of arguments may not be the same (e.g. dof information)
  - Halo exchange
- Performance responsibility
  - Threading
  - Kernel re-ordering (including halo replacement)
  - Fusion
  - Inlining
  - ...

Alg to PSy API : single call

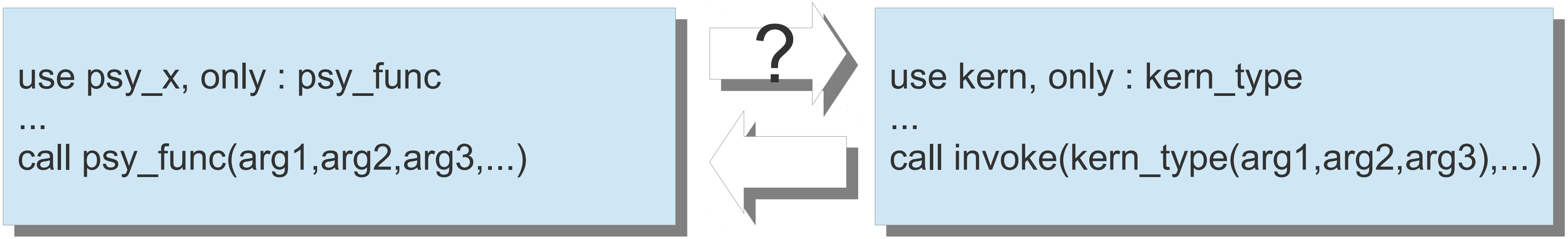
- What do scientists write?
- Specific or Generic call
  - Compute engine or not?
    - Dynamo

```
call psy_func(arg1,arg2,arg3,...)
```

```
call invoke(kern_type(arg1,arg2,arg3),...)
```

Alg to PSy API : single call

- Code transformation from generic to specific (or vice versa?)



```
use psy_x, only : psy_func
...
call psy_func(arg1,arg2,arg3,...)
```

```
use kern, only : kern_type
...
call invoke(kern_type(arg1,arg2,arg3),...)
```

- extended f2py parser

```
tree=fparser.api.parse(fileName,ignore_comments=False)
for stmt, depth in api.walk(tree, -1):
   if isinstance(stmt,fparser.statements.Call):
     use=getuse(stmt.parent,onlyname=stmt.designator)
     if use.name==options.psy:
        name=stmt.designator
        stmt.designator=options.engine
        stmt.items.insert(0,name)
        appenduse(use,options.engine)
        removeuse(use,name)
        kernuse=getuse(use.parent,usename=options.kernel)
        if kernuse==None:
          adduse(options.kernel,use.parent,only=True,funcnames=[name])
        else:
          appenduse(kernuse,name)
```

Example 1: Algorithm code content

```
...
type(ConstantFunctionSpace_type), pointer :: R_space
type(state_type) :: state
type(field_type), pointer :: integral, x
integer :: i

call read_triangle(state, "../data/unitsquare.1",&
     & layer_heights=[real(dp) :: (i/1000., i=0,1000)])

x => state%extract_field("Coordinate")

R_space => new_ConstantFunctionSpace("R_space")
integral => new_Field("integral", R_space)

call invoke(integrate_one_kernel(x, integral))
...
```

Example 1: Transformed Algorithm

```
program main
  ...
  use integrate_one_module, only : integrate_one_kernel
  ...
  call invoke(integrate_one_kernel(x, integral))
  ...
end program main
```

```
PROGRAM main
  ...
  USE psy_main, ONLY: invoke_integrate_one_kernel
  ...
  CALL invoke_integrate_one_kernel(x, integral)
  …
 END PROGRAM main
```

Example 1: Kernel code content

```
subroutine integrate_one_code(layers, p1dofm, X, R)
  integer, intent(in) :: layers
  integer, intent(in) :: p1dofm(6)
  real(dp), intent(in) :: X(3,*)
  real(dp), intent(inout) :: R

  real(dp) :: dx1(2), dx2(2), area
  integer :: k
  dx1 = X(1:2, p1dofm(3))-X(1:2, p1dofm(1))
  dx2 = X(1:2, p1dofm(5))-X(1:2, p1dofm(1))
  area = 0.5*abs(dx1(2)*dx2(1)-dx1(1)*dx2(2))
  do k = 0, layers-1
     R=R + area*(X(3, p1dofm(2) + k)-X(3, p1dofm(1) + k))
  end do
end subroutine integrate_one_code
```

Example 1: Kernel code

```
module integrate_one_module
  use kernel_mod
  implicit none
  private
  public integrate_one_kernel
  public integrate_one_code
  type, extends(kernel_type) :: integrate_one_kernel
    type(arg) :: meta_args(2) = (/&
        arg(READ, (CG(1)*CG(1))**3, FE), &
        arg(SUM, R, FE)/)
    integer :: ITERATES_OVER = CELLS
   contains
    procedure, nopass :: code => integrate_one_code
  end type integrate_one_kernel
contains
  subroutine integrate_one_code(layers, p1dofm, X, R)
  ...
```

Example 1: Generated PSy

```
  MODULE psy_main
    USE integrate_one_module, ONLY: integrate_one_code
    USE lfric
    IMPLICIT NONE
    CONTAINS
    SUBROUTINE invoke_integrate_one_kernel(x, integral)
      ...
      SELECT TYPE ( x_space=>x%function_space )
        TYPE IS ( FunctionSpace_type )
        topology => x_space%topology
        nlayers = topology%layer_count()
        p1dofmap => x_space%dof_map(cells, fe)
      END SELECT
      DO column=1,topology%entity_counts(cells)
        CALL integrate_one_code(nLayers, p1dofmap(:,column), x%data, integral
%data(1))
      END DO
    END SUBROUTINE invoke_integrate_one_kernel
  END MODULE psy_main
```

Support for finite difference in GungHo

- Could use current indirect addressing model as-is
- Investigate extending GungHo to support direct addressing
  - Simpler, more intuitive kernels?
  - Better performance?

Shallow example

```
 ...
 DO J=1,N
   DO I=1,M
     CU(I+1,J) = .5*(P(I+1,J)+P(I,J))*U(I+1,J)
     CV(I,J+1) = .5*(P(I,J+1)+P(I,J))*V(I,J+1)
     Z(I+1,J+1) =(FSDX*(V(I+1,J+1)-V(I,J+1))-FSDY*(U(I+1,J+1) &
         -U(I+1,J)))/(P(I,J)+P(I+1,J)+P(I+1,J+1)+P(I,J+1))
     H(I,J) = P(I,J)+.25*(U(I+1,J)*U(I+1,J)+U(I,J)*U(I,J)     &
         +V(I,J+1)*V(I,J+1)+V(I,J)*V(I,J))
   END DO
END DO
...
```

Algorithm layer : extreme splitting

```
...
call invoke(calcCU(cu,p,u), &
            calcCV(cv,p,v), &
            calcZ(z,v,u,p), &
            calcH(h,p,u,v))
...
```

Vanilla PSy layer : extreme splitting

```
module psy
contains
subroutine invoke_0(cu,cv,p,u,v,...)
    Do j=1,n
        Do i=1,m
          Call calcCUKern(K,i,j,cu%data,p%data,u%data)
        End do
    End do
    Do j=1,n
        Do i=1,m
          Call calcCVKern(K,i,j,cv%data,p%data,v%data)
        End do
    End do
    ...
```

Vanilla PSy layer : extreme splitting

```
module calcCUmod
contains
subroutine calcCUKern(K,i,j,cu,p,u)
    ! metadata about arguments and kernel implementation
    ...
    CU(I,J) = .5*(P(I,J)+P(I-1,J))*U(I,J)
End subroutine calcCUKern
End module calcCUmod
```