Science & Technology Facilities Council

Towards Performance Portability in GungHo and GOcean

M. Ashworth, **R. Ford**, M. Glover, D. Ham, M. Hobson, J. Holt, H. Liu, C. Maynard, T. Melvin, L. Mitchell, E. Mueller, S. Pickles, A. Porter, M. Rezny, G. Riley, P. Slavin, N. Wood

工合

29th October 2014, ECMWF HPC Workhop

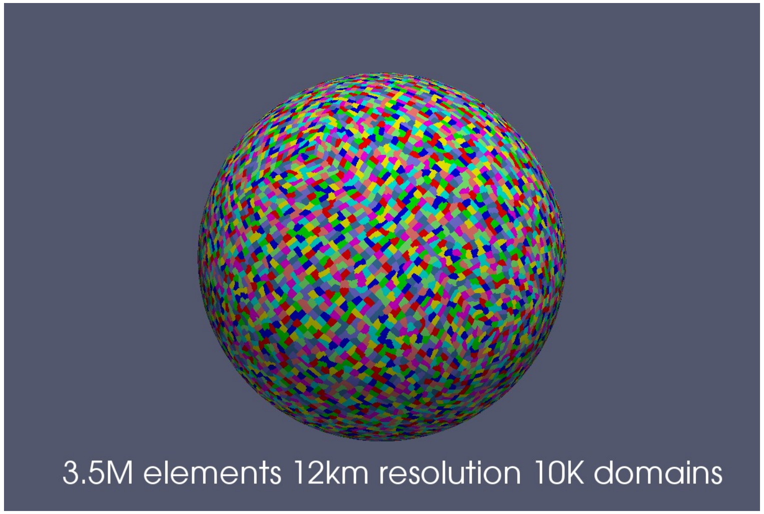
工合海洋

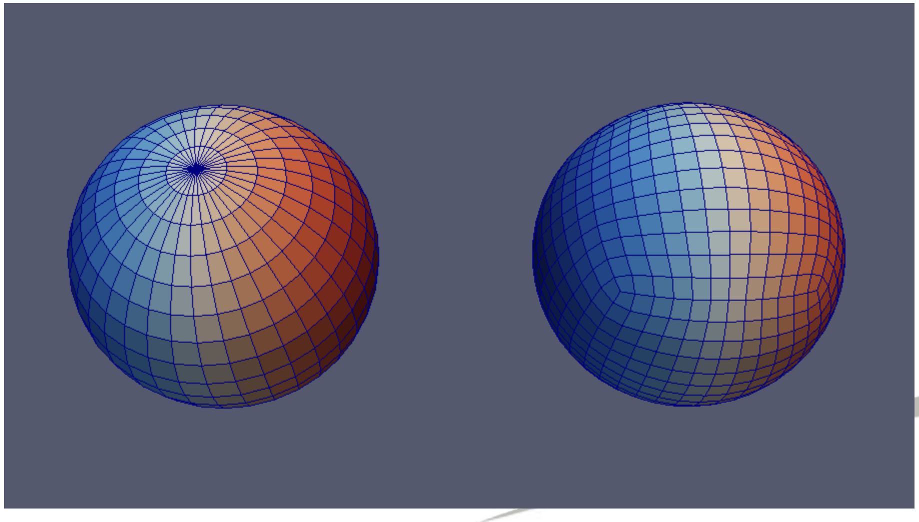
GungHo

- 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, NERC funding Universities of: Bath, Exeter, Imperial, Leeds, Manchester, Reading

GungHo Status

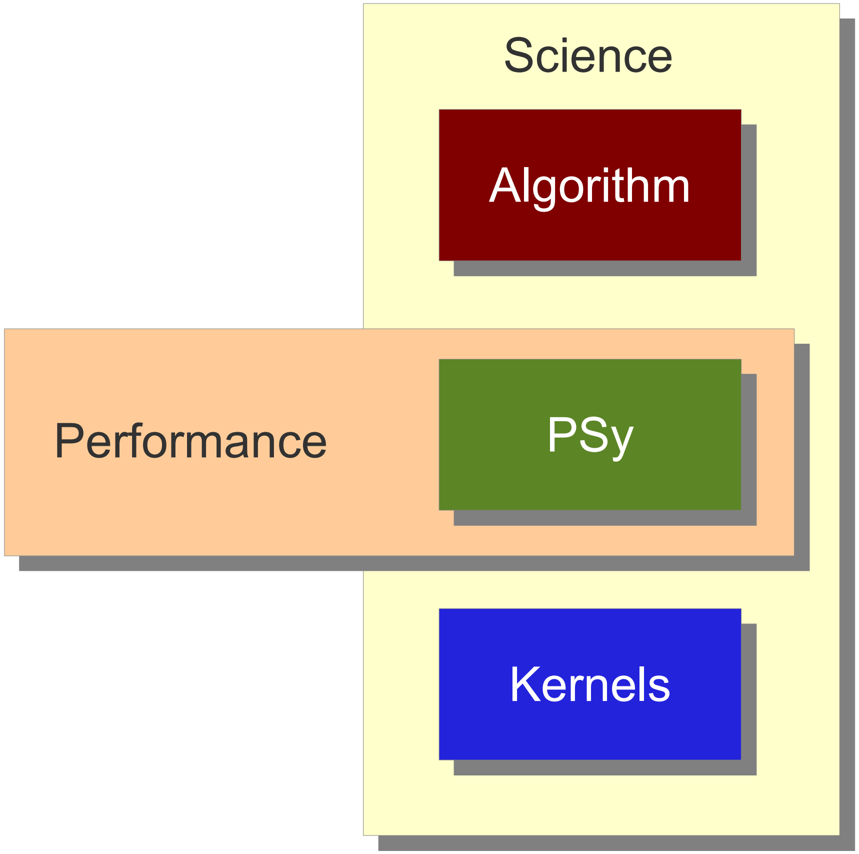
- (Most likely) Cubed Sphere
  - Extruded (columnar) mesh (2d+1d)
- Multi-grid
- Finite element approach
- Dynamo implementation





Science & Technology Facilities Council

PSyKAl



GH Algorithm

- Hand written (conforming to Fortran 2003)
- Use Field objects
- Logically operate on global fields
- Invoke approach
  - Algorithm layer **specifies** what the PSy layer has to do
  - Algorithm layer “specifications” will be pre-processed to specific calls which replace original
  - Invocation can take a 'list' of kernel specs

```
call invoke(kern(field1,field2,field3)...)
```

Science & Technology Facilities Council

Illustration : Alg

```
...
call invoke(&
    rhs_v3_type(rhs)&
    )
...
```

```
...
use psy, only: invoke_rhs_v3
...
call invoke_rhs_v3(rhs)
...
```

Multi-function Illustration : Alg

```
...
call invoke(&
    set(res_norm, 0.0), &
    galerkin_action(x, Mu, u), &
    galerkin_matrix_free_update(u, Mu, b, M_l, res_norm) &
    )
...
```

```
...
USE psy, ONLY: invoke_2
...
CALL invoke_2(b, m_l, mu, u, x, res_norm)
...
```

GH Kernel

- Hand written conforming to the GungHo PSyKAl API
- Scientific
  - There will also be library routines e.g. linear algebra
- Column based (written assuming k-inner)
- Access raw data (not field objects)
- Associated metadata (required for code generation) e.g.
  - Intents (extending fortran's in and out)
  - The function space a field is on (v0, v1, v2, ...)
  - what the kernel iterates over (cells, edges, ...)

Illustration GH0.1 API

```
module rhs_v3_mod
…
type, public, extends(kernel_type) :: rhs_v3_type
  private
  type(arg_type) :: meta_args(1) = [ &
       arg_type(gh_rw,v3,fe,.true.,.false.,.true.) &
       ]
  integer :: iterates_over = cells
contains
  procedure, nopass :: rhs_v3_code
end type
...
subroutine rhs_v3_code(nlayers,ndf,map,v3_basis,x,gq)
    ...
end subroutine rhs_v3_code
end module rhs_v3_mod
```

Science & Technology Facilities Council

GH PSy

- The Optimised PSy may be generated
  - Manual “reference/vanilla” version
  - Should be easily debuggable
- 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
  - Optimise for particular architectures → portable performance
  - Threading: OpenMP, OpenACC, …, Kernel re-ordering, Fusion, Inlining, ...

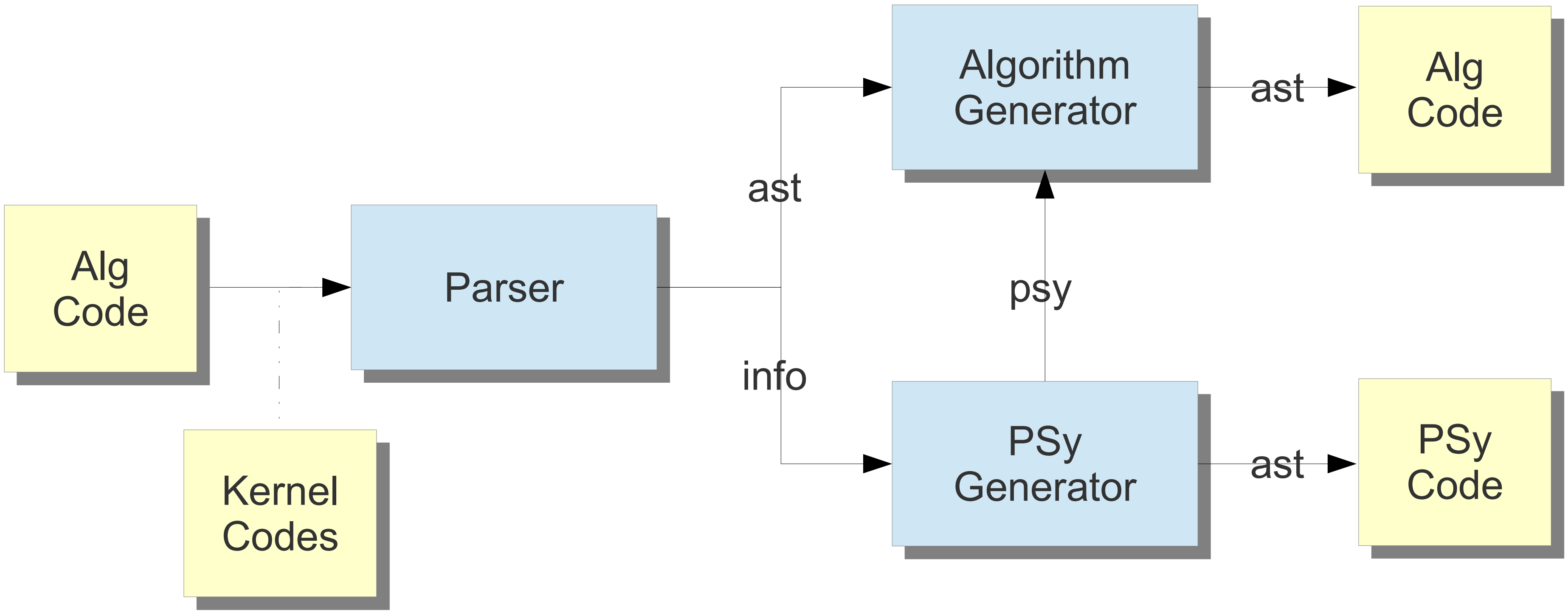
Illustration GH0.1 API

```
module psy
...
  subroutine invoke_rhs_v3(rhs)
    use rhs_v3_mod, only : rhs_v3_code
    ...
    nlayers=rhs%get_nlayers()
    ndf = rhs%vspace%get_ndf()
    call rhs%vspace%get_basis(v3_basis)
    do cell = 1, rhs%get_ncell()
       call rhs%vspace%get_cell_dofmap(cell,map)
       call rhs_v3_code(nlayers,ndf,map,v3_basis,rhs%data,rhs%gaussian_quadrature)
    end do
  end subroutine invoke_rhs_v3
…
end module psy
```

PSyclone Code Generation

- Code generation requires, Alg API, Kern API, Kernel metadata
- Code generation can help with
  - optimisation – labourious and error prone by hand
  - changes in interfaces
- PSyclone:
  - Taking an interactive optimisation approach to support the expert
  - Could also offer full automation option at a later date
  - Generates correct sequential code for GH 0.1 API
  - 4,113 lines of Python code
  - Following optimisations are available:
    - Loop fusion
    - OMP loop parallelisation
    - Loop colouring
    - Kernel inlining

PSyclone



Alg
Code
Kernel
Codes
Parser
ast
info
Algorithm
Generator
psy
PSy
Generator
ast
Alg
Code
ast
PSy
Code

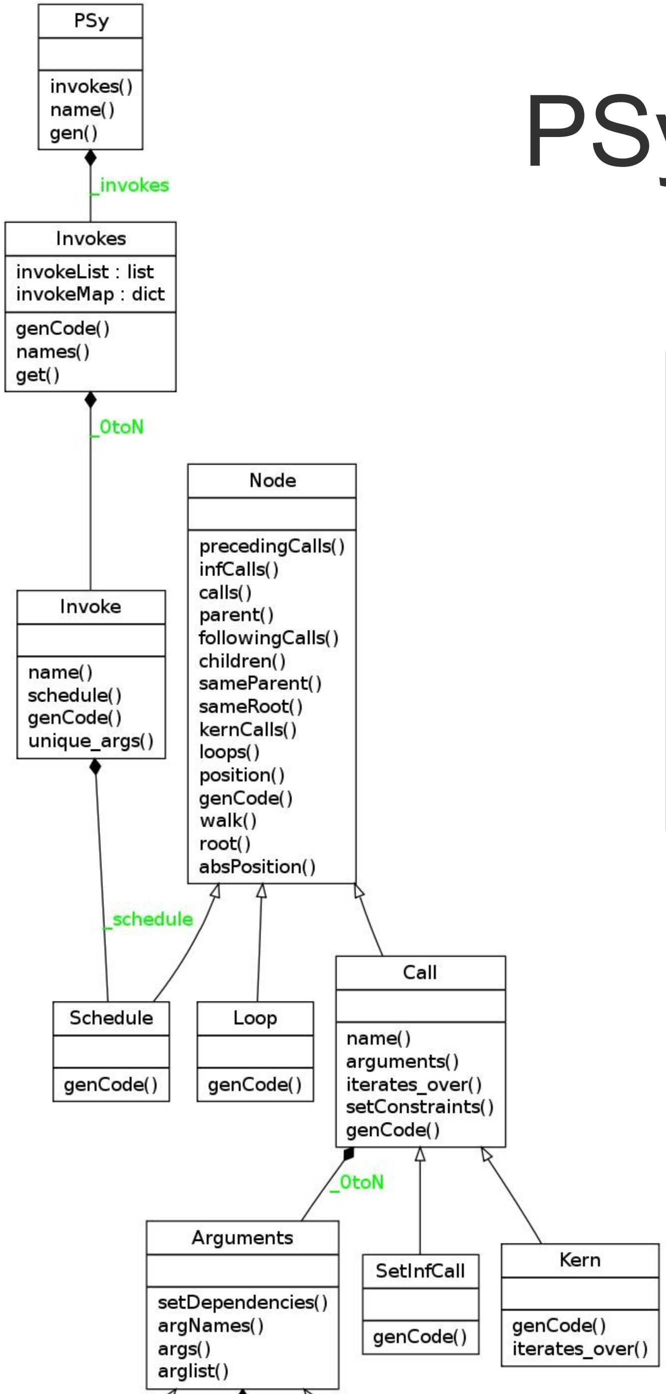
PSyclone

```
> python generate.py -oalg alg.f90 -opsy psy.f90 -api dynamo0.1 example.f90
```

```
>>> from generator import generate
>>> psy, alg = generate("example.f90", api=”dynamo0.1”)
>>> print str(psy.gen)
>>> print str(alg.gen)
```

```
>>> from algGen import Alg
>>> from parser import parse
>>> from psyGen import PSyFactory
>>> ast, info = parse(“example.f90”, api=”dynamo0.1”)
>>> psy = PSyFactory(“dynamo0.1”).create(info)
>>> alg = Alg(ast,psy)
>>> print str(psy.gen)
>>> print str(alg.gen)
```

PSy Schedule

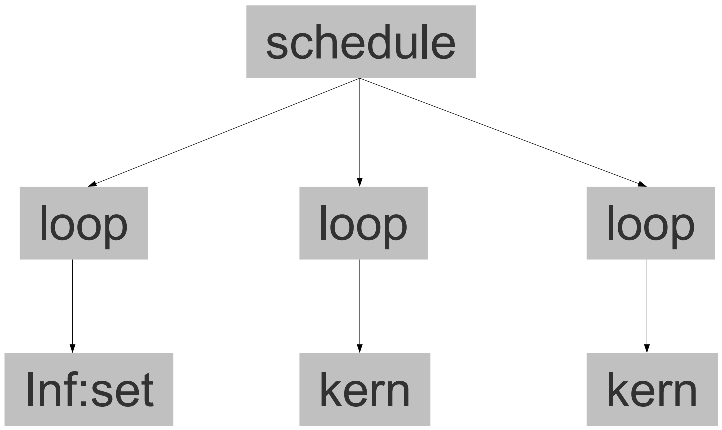


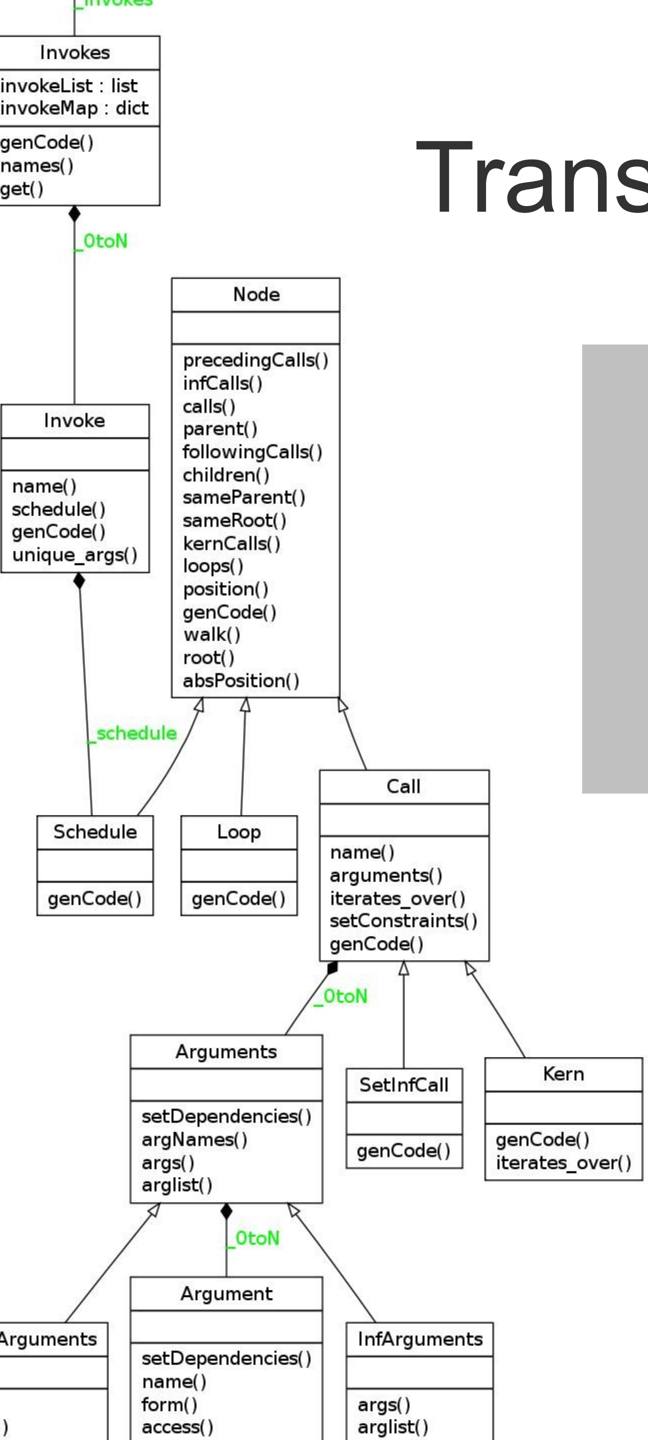

```
>>> psy = PSyFactory("dynamo0.1").create(info)
>>> invokes = psy.invokes
>>> invokes.names
>>> invoke = invokes.get("name")
>>> schedule = invoke.schedule
>>> schedule.view()
```

Science & Technology Facilities Council

Schedule Illustration

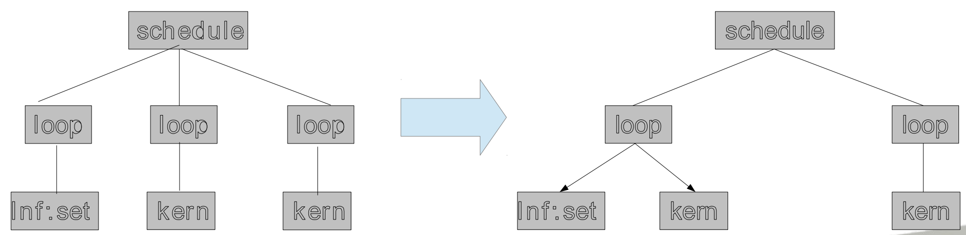
```
...
call invoke(&
    set(res_norm, 0.0), &
    galerkin_action(x, Mu, u), &
    galerkin_matrix_free_update(u, Mu, b, M_l, res_norm)&
    )
...
```





Transform PSy Schedule

```
>>> lf = LoopFuseTrans()
>>> loop1 = schedule.children[0]
>>> loop2 = schedule.children[1]
>>> new_schedule, memento = lf.apply(loop1, loop2)
>>> invoke._schedule = new_schedule
```



GOcean

- NERC funded Proof-of-principle 1 year project
- STFC & NOC + advice from GungHo colleagues
- Can GungHo PSyKAl approach be applied to Ocean Models?
- Finite Difference
- 2+1 test codes: shallow, NEMO-lite-2D, NEMO-lite-3D

GOcean

- T, P, U, V metadata to describe staggering
- API: Point-wise kernels, direct addressing
- Manual shallow and NEMO-lite-2D using GOcean PSyKAl
- PSyclone correct sequential code for 0.1 API
- Loop fusion, OpenMP loop parallel, inlining supported
- Nearly 90% of PSyclone is common

Science & Technology Facilities Council

Original Shallow

```
...
      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
...
```

Science & Technology Facilities Council

Shallow Alg

```
...
call invoke( compute_cu_type(CU, P, U), &
             compute_cv_type(CV, P, V), &
             compute_z_type(Z, P, U, V), &
             compute_h_type(H, P, U, V) )
...
```

```
...
USE psy_shallow, ONLY: invoke_0
...
CALL invoke_0(cu, p, u, cv, v, z, h)
...
```

Shallow Kern

```
module compute_cu_mod
  use kind_params_mod
  ...
  type, extends(kernel_type) :: compute_cu_type
     type(arg), dimension(3) :: meta_args =    &
          (/ arg(WRITE, CU, POINTWISE),        & ! cu
             arg(READ,  CT, POINTWISE),        & ! p
             arg(READ,  CU, POINTWISE)         & ! u
           /)
     integer :: ITERATES_OVER = DOFS
  contains
    procedure, nopass :: code => compute_cu_code
  end type compute_cu_type
  ...
  subroutine compute_cu_code(i, j, cu, p, u)
    ...
    CU(I,J) = .5*(P(I,J)+P(I-1,J))*U(I,J)
  end subroutine compute_cu_code
end module compute_cu_mod
```

Example

```
SUBROUTINE invoke_0(cu_1, p, u, cv_1, v, z, h)
  ...
  DO j=cu%jstart,cu%jstop
    DO i=cu%istart,cu%istop
      CALL compute_cu_code(i, j, cu_1, p, u)
    END DO
  END DO
  DO j=cv%jstart,cv%jstop
    DO i=cv%istart,cv%istop
      CALL compute_cv_code(i, j, cv_1, p, v)
    END DO
  END DO
  ...
END SUBROUTINE invoke_0
```

GOcean shallow 128

Compiler:	Cray 8.3.3	Intel 14.0.1	Gnu 4.8.2	Intel 14.0.0
Hardware:	IvyBridge	IvyBridge	Haswell	Haswell
Original	0.29	0.40	0.37	0.37
Vanilla	0.41	0.49	6.30	0.42
Explicit bounds	0.34	0.47	6.34	0.43
In-lined kernels	0.35	0.47	0.55	0.42
Loop fused	0.34	0.43	0.53	0.39
In-lined copy	0.34	0.43	0.54	0.39
Fused copy	0.31	0.51	0.54	0.45
Fastest	0.31	0.43	0.53	0.39
% slower	4.26	7.30	42.25	5.43

Science & Technology Facilities Council

GOcean shallow sizes with Cray compiler

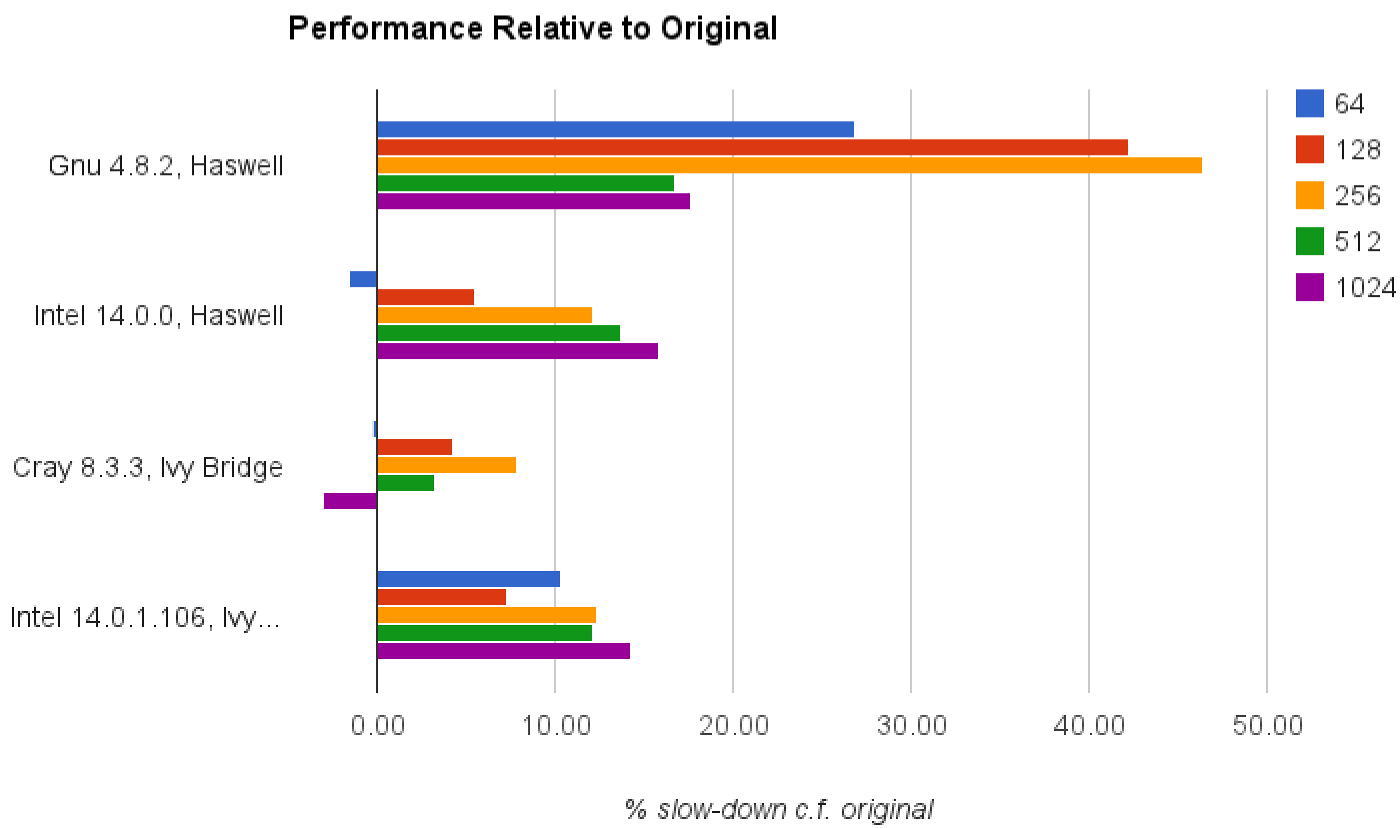
Problem size	64	128	256	512	1024
Original	0.008	0.29	1.21	5.70	44.12
Fastest	0.008	0.31	1.3	5.88	42.77
% slower	-.23	4.26	7.78	3.18	-3.06

Science & Technology Facilities Council

Summary

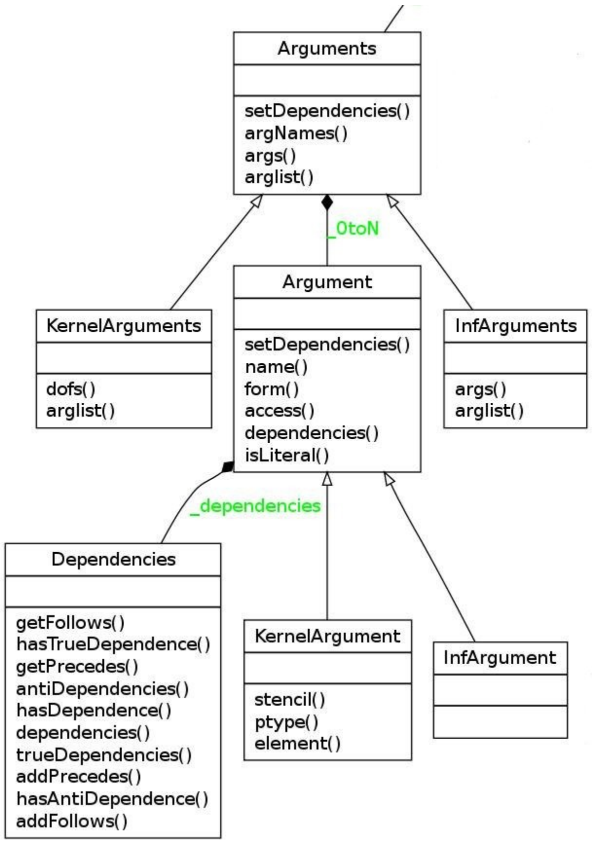
- PSyKAl approach shows promise
- Code Generation shows promise
- Initial results show promise
- Can promise become practice?

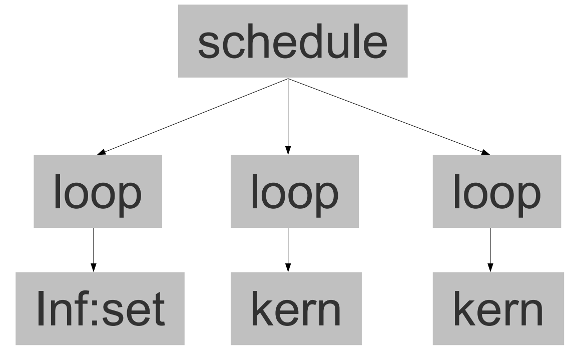
Gocean shallow



Science & Technology Facilities Council

Dependencies





set(res_norm, 0.0)

galerkin_action(x, Mu, u)

galerkin_matrix_free_update(u, Mu, b, M_l, res_norm)