PSyclone – Separation of Concerns for HPC Codes

Dr. Joerg Henrichs
Computational Science Manager
Bureau of Meteorology
PSyclone

- PSyclone developed by The Hartree Centre STFC Daresbury Laboratory, UK (since 2014)
  - Main developers:
    • Rupert Ford and Andy Porter
- In close cooperation with UK Met Office's Exascale Project
  - Used for the next generation Unified Model development (LFRic)
    • Expected to become operational in 2023
  - Based on 3d mixed finite elements
- International cooperation with UM Collaboration Partners
  - Including BOM
- Open source python program (BSD 3 clause)
  https://github.com/stfc/PSyclone
call HaloExchange(da, ...)  
$OMP PARALLEL DO default(none) private(j) ...  
do j=2,nx-1  
  dw(1:ny)=da(j,1:ny)  
  call swlat(uw,qw,vw,dw,ny)  
enddo  
!$OMP END PARALLEL  
...  
  call swlon(...)  
...

subroutine swlat(...)  
do i=2, ny-1  
  d2=GA*t*((dw(ip)-dw(i))/h(i)-(dw(i)-dw(im))/h(im))  
  u1(i)=uw(i)+rl-tl*(3.*uw(im)+qw(im)+3.*uw(ip)
call swlat(uw,qw,vw,dw,ny)

... call swlon(....)

...

subroutine swlat(....)

d2=GA*t*((dw(ip)-dw(i))/h(i)-(dw(i)-dw(im))/h(im))
ul(i)=uw(i)+rl-t1*((3.*uw(im)+qw(im)+3.*uw(ip)
Computational Science: Optimisation and Parallelisation

call HaloExchange(da, …)
$OMP PARALLEL DO default(none) private(j) …
do j=2,nx-1
  dw(1:ny)=da(j,1:ny)
  call swlat(uw,qw,vw,dw,ny)
enddo
!$OMP END PARALLEL
...
call swlon(...)
...

subroutine swlat(...)  
do i=2, ny-1
  d2=GA*t*((dw(ip)-dw(i))/h(i)-(dw(i)-dw(im))/h(im))
  u1(i)=uw(i)+rl-t1*((3.*uw(im)+qw(im)+3.*uw(ip))
HPC Code Analysis

• Typical HPC code combines skills from natural and computational science
  - Each one has its own special knowledge and experience
  - Highly interdisciplinary

• Optimised code looks different for different hardware:
  - OpenMP parallel, MPI, hybrid, GPU, Xeon Phi
  - What will future architectures look like?

• How can we manage and future-proof source code under these circumstances?
Separation of Concerns

Algorithm

Parallel System

Kernel

"PSyKAI"

Natural Science

Computational Science
What is PSyclone?

• "Domain Specific Language" or a code generator
  - Reads high level algorithm and kernel
  - Creates the parallel system layer
    • i.e. the (potential parallel) loop structure
  - But it is plain Fortran

• Restricts the way you can program somewhat
  - You need to think of 'kernels'

• Helps with the separation of concerns:
  - Natural scientists can write code using/thinking of whole arrays and single elements
  - Computational scientists can work on optimisation and parallelisation independently
Example: Algorithm Layer (High Level)

use compute_cu_mod, only: compute_cu
type(r2d_field) :: cu_fld, p_fld, u_fld, ...

call invoke( copy(u_fld, uold_fld) )

call invoke( name="update_fields",
            compute_cu(CU_fld, p_fld, u_fld), &
            compute_cv(CV_fld, p_fld, v_fld) )

call invoke( time_smooth(u_fld, UNEW_fld, UOLD_fld), &
            time_smooth(v_fld, VNEW_fld, VOLD_fld), &
            time_smooth(p_fld, PNEW_fld, POLD_fld) )
PSyclone Rewrites Algorithm Layer

use compute_cu_mod, only: compute_cu

type(r2d_field) :: cu_fld, p_fld, u_fld

! call invoke( copy(u_fld, uold_fld) )
call invoke_0(u_fld, uold_fld)

! call invoke( name="update_fields",
! compute_cu(CU_fld, p_fld, u_fld), &
! compute_cv(CV_fld, p_fld, v_fld) )

call invoke_update_fields(CU_fld, p_fld, u_fld, &
CV_fld, v_fld)
Pointwise Kernel Layer

subroutine copy_code(i, j, source, dest)
    implicit none
    integer, intent(in) :: i, j
    real(wp), intent(out), dimension(:, :) :: dest
    real(wp), intent(in), dimension(:, :) :: source

    dest(i, j) = source(i, j)
end subroutine copy_code

subroutine compute_cu_code(i, j, cu, p, u)
    ...
    cu(i,j) = 0.5d0*(p(i,j)+p(i-1,j))*u(i,j)
end subroutine compute_cu_code
Subroutine `invoke_update_fields(cu_fld, p_fld, u_fld, &
  cv_fld, v_fld)`

USE `compute_cu_mod`, ONLY: `compute_cu_code`

`istop = cu_fld%grid%simulation_domain%xstop`
`jstop = cu_fld%grid%simulation_domain%ystop`

! In case of MPI: possible halo exchange for `p_fld`
DO `j=2, jstop`
  DO `i=2, istop`
    CALL `compute_cu_code(i, j, cu_fld%data,`
    `p_fld%data, u_fld%data)`
  END DO
END DO

DO `j=2, jstop`
  DO `i=2, istop`
    CALL `compute_cv_code(i, j, cv_fld%data,`
    `p_fld%data, v_fld%data)`
  END DO
END DO
Kernels Need to Define Kernel Type

- Defines the iteration space and stencil operations:

```plaintext
type, extends(kernel_type) :: compute_cu
  type(arg), dimension(3) :: meta_args = &
  (/ arg(WRITE, FIELD, POINTWISE), & ! Cu
   arg(READ, FIELD, STENCIL(010, & ! P
      111, &
      010)), &
   arg(READ, FIELD, POINTWISE) & ! U
   /)
  integer :: ITERATES_OVER = INTERNAL_PTS
contains
  procedure, nopass :: code => compute_cu_code
end type compute_cu
```

- The type is defined only for PSyclone, not used anywhere else
PSyclone Schedule

- Internal tree representation of loops and kernel invocations

GOSchedule[invoke='invoke_1',Constant loop bounds=True]
  Loop[type='outer',field_space='cu',it_space='internal_pts']
    Loop[type='inner',field_space='cu',it_space='internal_...']
      KernCall compute_cu_code(cu_fld,p_fld,u_fld)
  Loop[type='outer',field_space='cu',it_space='internal_pts']
    Loop[type='inner',field_space='cu',it_space='intern..']
      KernCall compute_cv_code(cv_fld,p_fld,v_fld)
Transformations Act on Schedule

- Python scripts can act as transformations
- Transformations can be developed by computational scientist independent of Fortran code
- They can modify the schedule, e.g. adding halo exchanges, OMP directive nodes, fuse loops etc.

```python
from psyclone.transformations import LoopFuse

def trans(psy):
    schedule = psy.invokes.get('invoke_compute_fields').schedule
    schedule.view()
    fuse = LoopFuse()  # Create fuse transform
    s1 = fuse.apply(schedule.children[0],
                     schedule.children[1])
```
Result of Loop Fuse Transformation

GOSchedule[invoke='invoke_1', Constant loop bounds=True]
Loop[type='outer', field_space='cu', it_space=...]
  Loop[type='inner', field_space='cu', it_space=...]
    KernCall compute_cu_code(cu_fld,p_fld,u_fld)
  Loop[type='inner', field_space='cu', it_space=...]
    KernCall compute_cv_code(cv_fld,p_fld,v_fld)

DO j=2,jstop
  DO i=2,istop
    CALL compute_cu_code(i, j, cu_fld%data, ...)
  END DO
  i=2,istop
  CALL compute_cv_code(i, j, cv_fld%data, ...)
END DO
END DO
Transformations

- Transformations together with PSyclone can:
  - Optimise code (e.g. loop fuse, loop blocking/tiling, …)
  - Parallelise code using OpenMP or MPI
  - Generate code for different hardware architectures (OpenMP, OpenACC, …)
  - Generate different code for different compilers

- They are mostly independent of the actual code
  - To a certain degree 😊
PSyclone Infrastructure Library

- Driver/runtime library needed for setting up parallel environment
  - LFRic: Earth System Modelling Framework, now:
  - YAXT: Yet Another eXchange Tool
  - GOcean: GOcean Library (GOLib v.1.0)
- Supplies field types etc.
- Support for 2d finite differences, and 3d finite elements
Results: MOST

- Method of Splitting Tsunamis
  - Developed by the NOAA Pacific Marine Environmental Laboratory (PMEL)
- Optimised by stand-alone script to fuse loops and apply OpenMP statements
  - For GNU Fortran: additional module inlining support
- Compared with a manual developed and optimised C version

<table>
<thead>
<tr>
<th></th>
<th>Intel 17</th>
<th>Cray 8.4.5</th>
<th>GNU Fortran 5.1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original code</td>
<td>111.0</td>
<td>115.6</td>
<td>102.4</td>
</tr>
<tr>
<td>PSyclone</td>
<td>48.5</td>
<td>31.4</td>
<td>88.7 → 70.4</td>
</tr>
<tr>
<td>Optimised C version</td>
<td>49.0</td>
<td>47.0</td>
<td>-</td>
</tr>
</tbody>
</table>
OpenMP Scaling – Added OpenMP Optimisations

![Graph showing runtime vs number of threads for different OpenMP optimisations. The graph compares Intel PSyclone, Intel Optimised PSyclone, Intel Optimised C version, Cray PSyclone, and Cray Optimised PSyclone.]
Summary

- PSyclone helps with the separation of concern
  - Separate natural science code from computational science code
  - Natural science can be developed and maintained independent of optimisations (to a certain degree)
- Originally developed for 3d FEM, it can be used with other methods
- Domain-specific language 'in Fortran'
- Can automatically parallelise code, e.g. MPI, OpenMP
  - But of course restricts code development to 'kernels'
- Can create hardware-specific code
  - E.g. CPU, GPU, Xeon Phi
- Tools under development:
  - Profiler support, automatic test-case creation, perhaps automatic performance optimisation?
Thank you

Joerg.Henrichs@bom.gov.au

Thanks to GDJ and oksmith from openclipart.org for the images of scientists
PSyclone Architecture

Algorithm Generator

Parser

PSy Generator

Algorithm Code

PSy Code

Kernel Codes

Transformations

Parser

AST

PSy

info

schedule

Alg Code

API

f90

f90
<table>
<thead>
<tr>
<th># Threads</th>
<th>Intel PSyclone Fortran</th>
<th>Cray PSyclone Fortran</th>
<th>Intel C, hand-tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>195.45</td>
<td>145.65</td>
<td>152.4</td>
</tr>
<tr>
<td>2</td>
<td>116.38</td>
<td>91.49</td>
<td>89.7</td>
</tr>
<tr>
<td>4</td>
<td>71.78</td>
<td>60.93</td>
<td>59.1</td>
</tr>
<tr>
<td>8</td>
<td>48.42</td>
<td>43.24</td>
<td>39.0</td>
</tr>
<tr>
<td>12</td>
<td>43.70</td>
<td>38.12</td>
<td>31.5</td>
</tr>
<tr>
<td>24</td>
<td>37.33</td>
<td>35.88</td>
<td>35.3</td>
</tr>
</tbody>
</table>