PSyclone: a code generation and optimisation system for FE and FD

R. W. Ford, A. R. Porter, M. Ashworth

STFC Daresbury Laboratory
U.K.

This work was funded by the Hartree Centre for GungHo and by NERC for GOcean

16th September 2015, NCAR Multi-core Workshop
What is PSyclone (Google)?

- Legal high
- Canadian theme park ride
- Music festival
What is PSyclone (Wikipedia)?

“When I made Psyclone, I was at the height of my alcoholism and addiction, I was literally staring into the abyss"
What is PSyclone (Brikipedia)?

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

http://lego.wikia.com/wiki/Psyclone
What is PSyclone?

- Configurable domain specific compiler for domain specific languages using the PSyKAl approach
  - PSyKAl developed in the GungHo project
  - Configurable to related API's – implementations for unstructured FE (dynamo api's) and structured FD (gocean api's)
PSyKAI

Science

Algorithm

Performance

PSy

Kernels
PSyclone Code Generation

• Generate the PSy layer, modify the algorithm layer
• Code generation requires, Alg API, Kern API, inf API, Kernel metadata
• Code generation can help with
  – Generation of optimised code – labourious and error prone by hand
  – Flexible implementation of optimisations (not hard coded)
    • Single code base
    • Portable performance
  – changes in interfaces
PSyclone Code Generation

- Taking a user-specified optimisation approach to support the expert
- Compile time optimisation (static analysis)
- Generates correct sequential code for dynamo0.1, dynamo0.3, gocean0.1 and gocean1.0 API's
- Loop fusion and OpenMP parallel transformations supported
- 7,072 lines of Python code
  - 280 dynamo 0.1 API, 1479 lines dynamo0.3 API, 290 gocean0.1 API, 665 gocean 1.0 API, 545 transformations, 3813 generic
- 176 tests, 3009 lines
Using PSyclone

```
$ python generator.py [-h][-oalg ALG][-opsy PSY][-api API][-s SCRIPT][-d DIR] file
```

```python
>>> from generator import generate
>>> psy, alg = generate("example.f90", api="dynamo0.3", kernel_path="path" script_name="scr.py")
>>> print psy.gen
>>> print alg.gen
```

```python
>>> from algGen import Alg
>>> from parser import parse
>>> from psyGen import PSyFactory
>>> ast, info = parse("example.f90", api="dynamo0.3", kernel_path="path")
>>> psy = PSyFactory("dynamo0.3").create(info)
>>> alg = Alg(ast,psy)
>>> print psy.gen
>>> print alg.gen
```
Algorithm code example

```fortran
... call invoke(
    continuity(ssha_t, sshn_t, sshn_u, sshn_v, &
    hu, hv, un, vn, rdt), &
    momentum_u(ua, un, vn, hu, hv, ht, &
    ssha_u, sshn_t, sshn_u, sshn_v), &
    ...
    copy(sshn_t, ssha_t), &
    next_sshu(sshn_u, sshn_t), &
    next_sshv(sshn_v, sshn_t)
)
...

USE psy_gocean2d, ONLY: invoke_0
...
CALL invoke_0(ssha_t, sshn_t, sshn_u, sshn_v, hu, hv, un, vn,
    rdt, ua, ht, ssha_u, va, ssha_v, istp)
...
```
Generated vanilla PSy code example (gocean1.0 api)

```plaintext
MODULE psy_gocean2d
  ...
  SUBROUTINE invoke_0(ssha_t, sshn_t, sshn_u, sshn_v, hu, hv, un, vn, rdt, ua, ht, ssha_u, va, ssha_v, istp)
    ...
    DO j=ssha_t%internal%ystart,ssha_t%internal%ystop
      DO i=ssha_t%internal%xstart,ssha_t%internal%xstop
        CALL continuity_code(i, j, ssha_t%data, sshn_t%data, sshn_u%data, sshn_v%data, hu%data, hv%data, un%data, vn%data, rdt, sshn_t%grid%area_t)
        END DO
      END DO
    END DO
  END SUBROUTINE invoke_0
END MODULE psy_gocean2d
```
module continuity_mod

    type, extends(kernel_type) :: continuity
        type(arg), dimension(10) :: meta_args = &
        (/ arg(WRITE, CT, POINTWISE), & ! ssha
            ... arg(READ, CU, POINTWISE), & ! un
            arg(READ, CV, POINTWISE), & ! vn
            arg(READ, R_SCALAR, POINTWISE), & ! Time-step
            arg(READ, GRID_AREA_T) &
        /)

        integer :: ITERATES_OVER = INTERNAL_PTS
        integer :: index_offset = OFFSET_NE
    contains
        procedure, nopass :: code => continuity_code
    end type continuity

... subroutine continuity_code(ji, jj, &
    ssha, sshn, sshn_u, sshn_v, &
    hu, hv, un, vn, rdt, e12t)

... end subroutine continuity_code
end module continuity_mod
>>> psy = PSyFactory("dynamo0.3").create(info)
>>> invokes = psy.invokes
>>> invokes.names
>>> invoke = invokes.get("name")
>>> schedule = invoke.schedule
>>> schedule.view()
Schedule internal tree representation

... call invoke(
    continuity(ssha_t, sshn_t, sshn_u, sshn_v, &
    hu, hv, un, vn, rdt),
    momentum_u(ua, un, vn, hu, hv, ht, &
    ssha_u, sshn_t, sshn_u, sshn_v), &
...
    copy(sshn_t, ssha_t), &
    next_sshu(sshn_u, sshn_t), &
    next_sshv(sshn_v, sshn_t)
)
...

schedule

loop
  kern

loop
  kern

loop
  kern

loop
  kern

loop
  kern

loop
  kern
PSyclone performance results

- Earlier work: compared hand-coded PSyKAI compliant code to hand-coded original (shallow benchmark)
  - Executive summary: we are able to obtain the same performance as hand tuned code + tuned code for one compiler/architecture is not the same as for another
  - Upcoming PARCO paper
- This work: Compare hand-coded PSyKAI implementation to PSyclone generated implementation
- NEMOLite2D benchmark (developed by NOC in the GOcean project)
- Archer (Intel Ivy Bridge) and Intel 15.0.0.9, Cray 8.3.3 and Gnu 4.9.1 compilers
- Concentrate on OpenMP performance (because this is implemented!)
- Chosen a Static schedule with default block size in all cases (as that gave the best performance for the hand-optimised version).
- Codes the same except: Field copy subroutine vs field copy kernel
>>> api = "gocean1.0"
>>> _, invokeInfo = parse("nemolite2d_alg.f90",api=api)
>>> psy = PSyFactory(api).create(invokeInfo)
>>> schedule = psy.invokes.get('invoke_0').schedule
>>> schedule.view()
Schedule[invoke='invoke_0']
  Loop[type='outer',field_space='ct',it_space='internal_pts']
    Loop[type='inner',field_space='ct',it_space='internal_pts']
      Call continuity_code(ssha_t,sshn_t,sshn_u,..)
    Loop[type='outer',field_space='cu',it_space='internal_pts']
      Loop[type='inner',field_space='cu',it_space='internal_pts']
        Call momentum_u_code(ua,un,vn,hu,hv,ht,..)
    Loop[type='outer',field_space='cv',it_space='internal_pts']
      Loop[type='inner',field_space='cv',it_space='internal_pts']
        Call momentum_v_code(va,un,vn,hu,hv,ht,..)
  ...

>>> print psy.gen
...

NEMOLite2D – parallel do's interactive API vanilla code
NEMOLite2D – parallel do's interactive API transformed code

```python
from psyGen import TransInfo

t = TransInfo()
ltrans = t.get_trans_name('GOceanOMPParallelLoopTrans')

# Apply to *every* loop in the schedule
for child in schedule.children:
    newschedule, _ = ltrans.apply(child)
    schedule = newschedule

schedule.view()

Schedule[invoke='invoke_0']
    Directive[OMP parallel do]
        Loop[type='outer',field_space='ct',it_space='internal_pts']
            Loop[type='inner',field_space='ct',it_space='internal_pts']
                Call continuity_code(ssha_t,sshn_t,sshn_u,...)
        Directive[OMP parallel do]
            Loop[type='outer',field_space='cu',it_space='internal_pts']
                Loop[type='inner',field_space='cu',it_space='internal_pts']
                    Call momentum_u_code(ua,un,vn,hu,hv,ht,...)

>>> psy.invokes.get('invoke_0').schedule = schedule
>>> print psy.gen
```

---

**Science & Technology**

**Facilities Council**
NEMOLite2D – parallel do's script

```
$ python ${PSYCLONEHOME}/src/generator.py -api gocean1.0 -oalg alg.f90 -opsy psy.f90 nemolite2d_alg.f90

$ cat ./script.py
def trans(psy):
    schedule = psy.invokes.get('invoke_0').schedule
    ltrans = GOceanOMPParallelLoopTrans()
    for child in schedule.children:
        newschedule, _ = ltrans.apply(child)
        schedule = newschedule
    psy.invokes.get('invoke_0').schedule = schedule
    return psy

$ python ${PSYCLONEHOME}/src/generator.py -api gocean1.0 -oalg alg.f90 -opsy psy.f90 -s ./script.py nemolite2d_alg.f90
```
NEMOLite2D – parallel do's
NEMOLite2D – single parallel region
script + internal representation

```
$ cat ./script2.py
def trans(psy):
    schedule = psy.invokes.get('invoke_0').schedule
    ltrans = GOceanOMPLoopTrans()
    rtrans = OMPParallelTrans()
    for child in schedule.children:
        newschedule, _ = ltrans.apply(child)
        schedule = newschedule
        newschedule, _ = rtrans.apply(schedule.children)
    psy.invokes.get('invoke_0').schedule = newschedule
    return psy

$ python ${PSYCLONEHOME}/src/generator.py -api gocean1.0 -oalg alg.f90 -opsy psy.f90 -s ./script2.py nemolite2d_alg.f90
```

Schedule[invoke='invoke_0']
Directive[OMP Parallel]
Directive[OMP do]
Loop[type='outer',field_space='ct',it_space='internal_pts']
Loop[type='inner',field_space='ct',it_space='internal_pts']
Call continuity_code(ssha_t,sshn_t,sshn_u,...)
...

Science & Technology
Facilities Council
NEMOLite2D – single parallel region

Graphs showing performance (MPa) vs. number of OpenMP threads for different compilers and configurations.
Summary and Future work

- PSyclone is a code generation and optimisation system for the PSyKAI separation of concerns.
- For the NEMOLite2D benchmark there is no overhead using PSyclone generated OpenMP code compared with hand written OpenMP code.
- PSyclone is now part of the Dynamo build system.
  
- Continue to support Met Office Dynamo API extensions including MPI parallel.
- Extend scalar transformations and add OpenACC transformation support.
- Extend benchmarks (dynamo test cases) and architectures (Xeon Phi and GPU performance analysis).
- Search the optimisation space.