Université Grenoble Alpes

Inria

LABORATOIRE JEAN KUNTZMANN
MATHÉMATIQUES APPLIQUÉES - INFORMATIQUE

Croco & PSyClone

What it is / 07-2023

Sébastien Valat

Manual way for GPU support

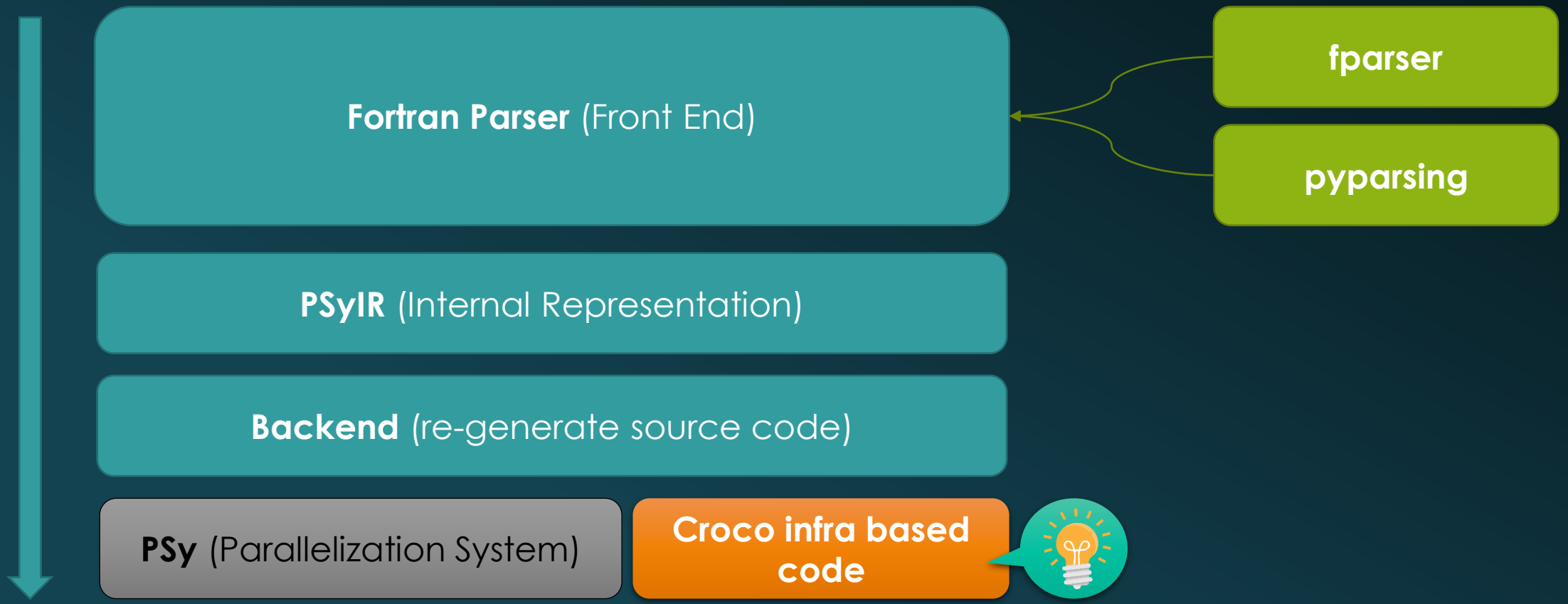
- Using OpenACC
- Adding directives to the code
- Easier than re-writing CUDA code
- But still….

```
real, allocatable :: a(:), b(:)
...
allocate(a(n),b(n))
...
!$acc data copy(a,b)
call process( a, b, n )
!$acc end data
...
subroutine process( a, b, n )
  real :: a(:), b(:)
  integer :: n, i
  !$acc parallel loop
  do i = 1, n
    b(i) = exp(sin(a(i)))
  enddo
end subroutine
```

What is PSyclone

- **Psyclone** is a **source-to-source** compiler for Fortran
- Implemented in **Python**
- Authors : **STFC & Met Office**
- Sources : https://github.com/stfc/PSyclone/
- Doc : https://psyclone.readthedocs.io

Reminder on **PSyClone**

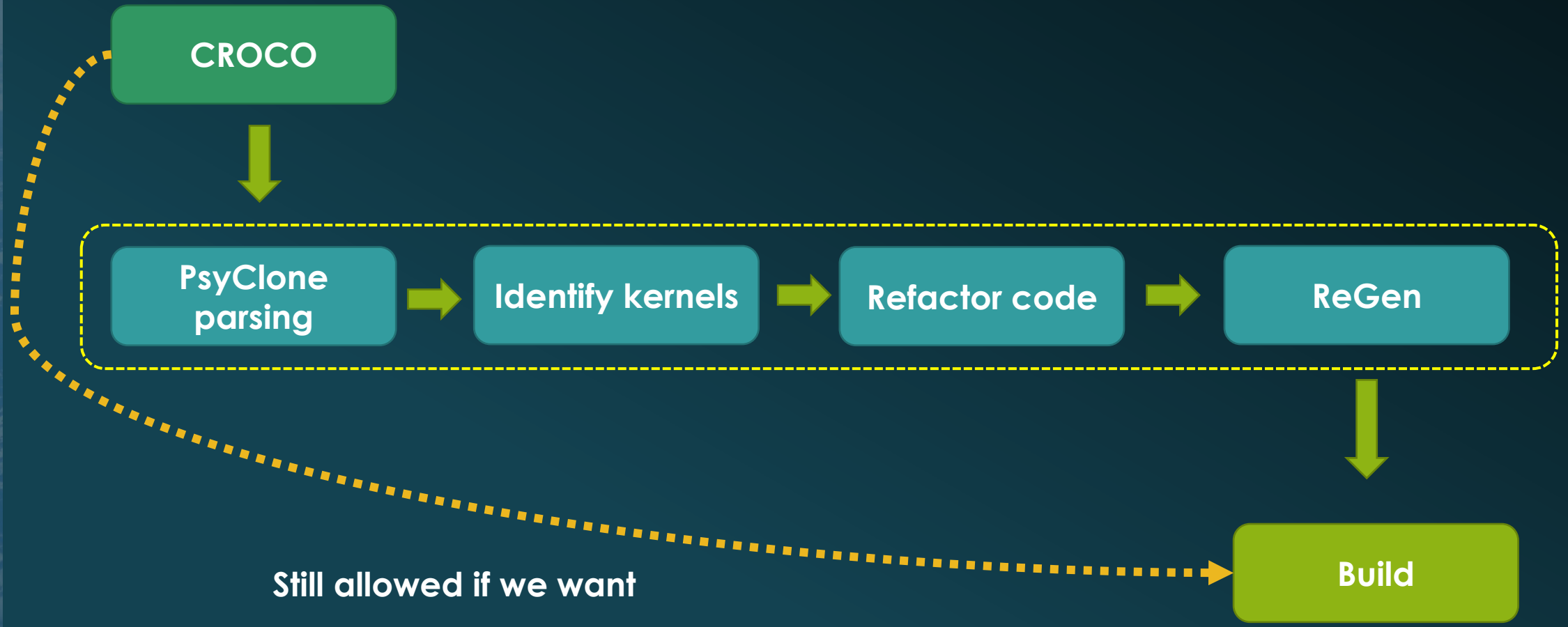


Has several support exploration

- **LFRic** : A fortran runtime to handle simulation
- **GOcean 1.0 :** For an ocean simulation library
- **Nemo** : to transform NEMO source code

=> I derived from this one

In other words



First look

- Need to see if **PSyClone** can **parse CROCO** code !


```
from psyclone.psyir.frontend.fortran import FortranReader
from psyclone.psyir.backend.fortran import FortranWriter

# parse
reader = FortranReader()
psyir_tree = reader.psyir_from_file(args.source_file, free_form = True)

# regen
writer = FortranWriter()
result = writer(psyir_tree)

f = open(args.output_file, "wb")
f.write(result.encode())
```

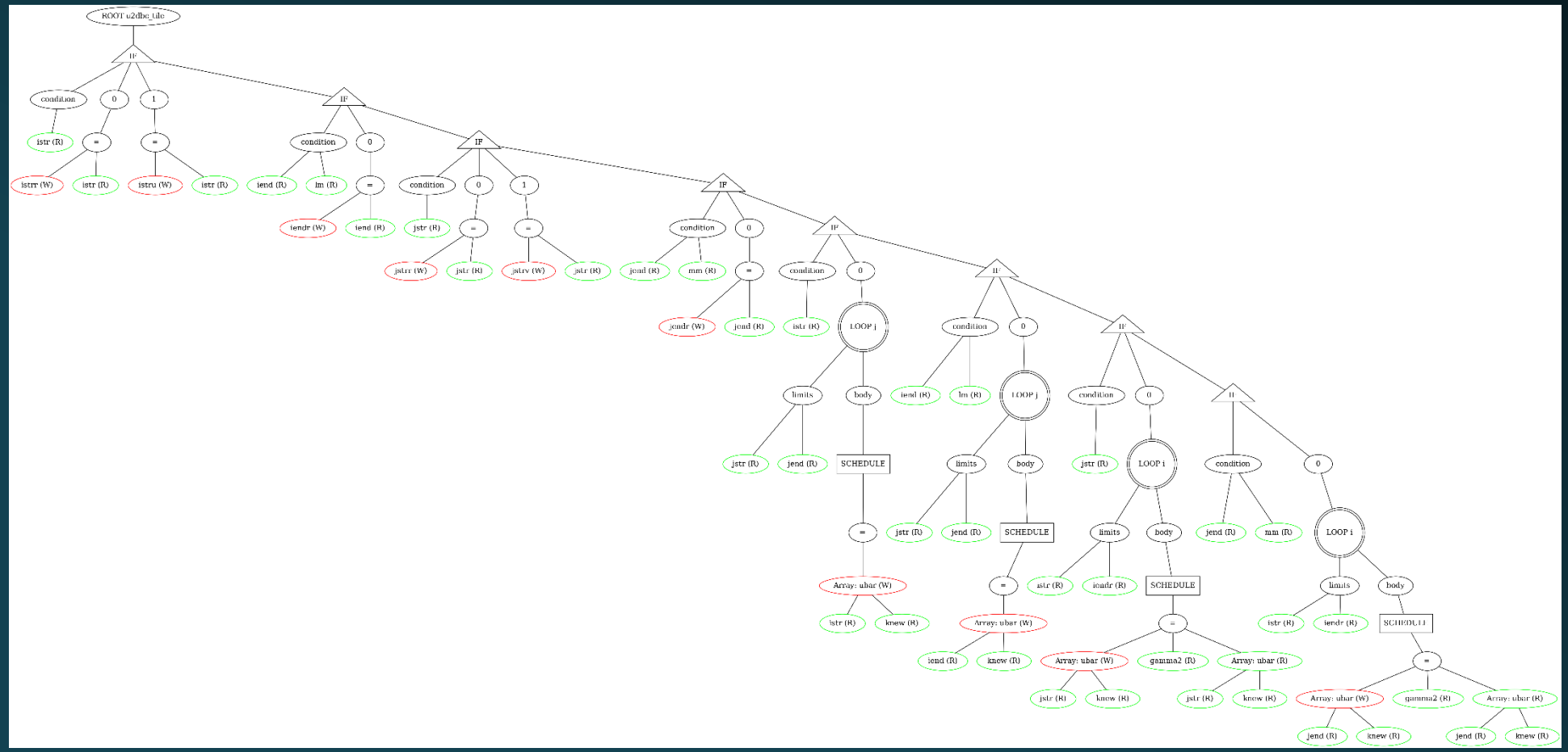
We can now also look on the IR

- Everything is a node
- Quickly dump via :

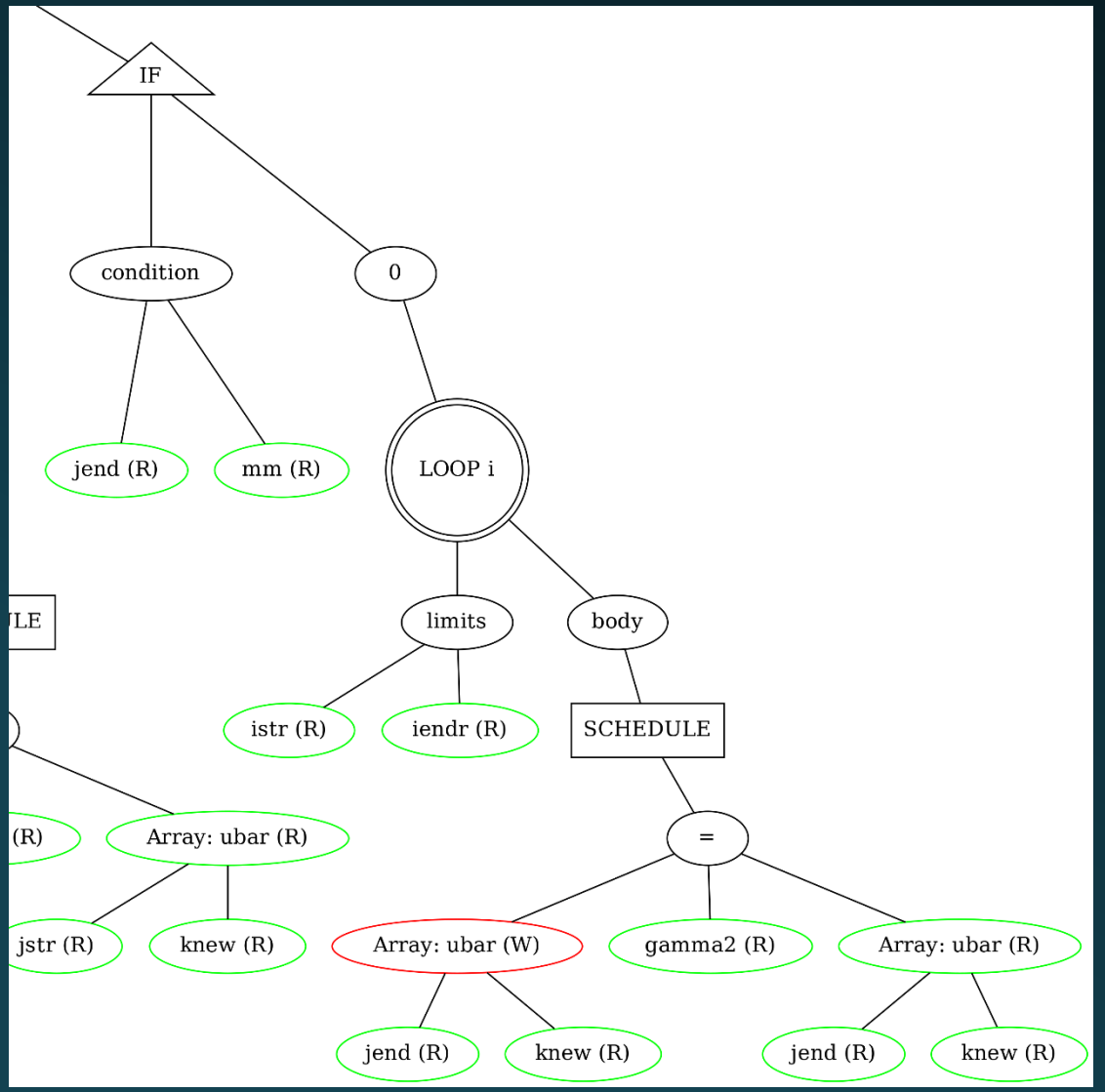
```
print(psyir_tree.view())
```

```
FileContainer[]
    Routine[name:'step2d']
        0: Assignment[]
            Reference[name:'chunk_size_x']
            BinaryOperation[operator:'DIV']
                BinaryOperation[operator:'SUB']
                    BinaryOperation[operator:'ADD']
                        Reference[name:'lm']
                        Reference[name:'nsub_x']
                    Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                Reference[name:'nsub_x']
```

First look on graph



First look on graph



Kernel extraction

- Extract **2D loops** with their **stencil masks**
- Eg.:

```
================= KERNEL ==================
OUT   : duon, urhs
IN    : ubar, drhs, urhs, on_u
MASKS : (i, j), (i - 1, j)
------------------ CODE --------------------
do j = jstr - 1, jend + 1, 1
  do i = imin + 1, imax, 1
    urhs(i,j) = cff1 * ubar(i,j,kstp) + cff2 * ubar(i,j,kbak)
              + cff3 * ubar(i,j,kold)
    duon(i,j) = 0.5d0 * (drhs(i,j) + drhs(i - 1,j)) * on_u(i,j)
              * urhs(i,j)
  enddo
enddo
--------------------------------------------
```

Way of calling

- Calling psyclone on a file:

```
psyclone \
    -api nemo \
    -l output \
    -s your_psyclone_script.py \
    -opsy output_file.f \
    input_file.f
```

- We need to provide:
  - An **input source** file
  - A **transformation script**

Very basic script

- Introduce ACC kernel directives automatically
- **Caution** : the **magic** is in **ACCKnerlsTrans** !
- **Caution** : I **removed the checkings** in the example !

```
from psyclone.transformations import ACCKernelsTrans

def trans(psy):
    for invoke in psy.invokes.invoke_list:
        ACCKernelsTrans().apply(invoke.schedule.children, {
            "default_present": default_present,
            "async_queue": async_queue
        })
```

Patching loops

- **Some loops** need **to be swapped**
- To **perform efficiently on GPU**
- Need to **change** usage of some **temp variables**.

Very basic script

- Example apply **1D scratch** vars

```
def patch_scratch_1d_arrays(top_loop: Loop) -> None:
    for ref in top_loop.walk(ArrayReference):
        if ref.name.lower() in ['fc', 'cf', 'dc']:
            new_name = ref.name.lower()+'1d'
            ref.symbol = Symbol(new_name)
            ref.indices.pop(0)

def trans(psy):
    for top_loop in routine.walk(Loop, stop_type=Loop):
        patch_scratch_1d_arrays(top_loop)
```

- Do :

```
cff = 2.0 * FC(i, k – 1)
```

```
cff = 2.0 * FC1D(k – 1)
```

Can be unit tested

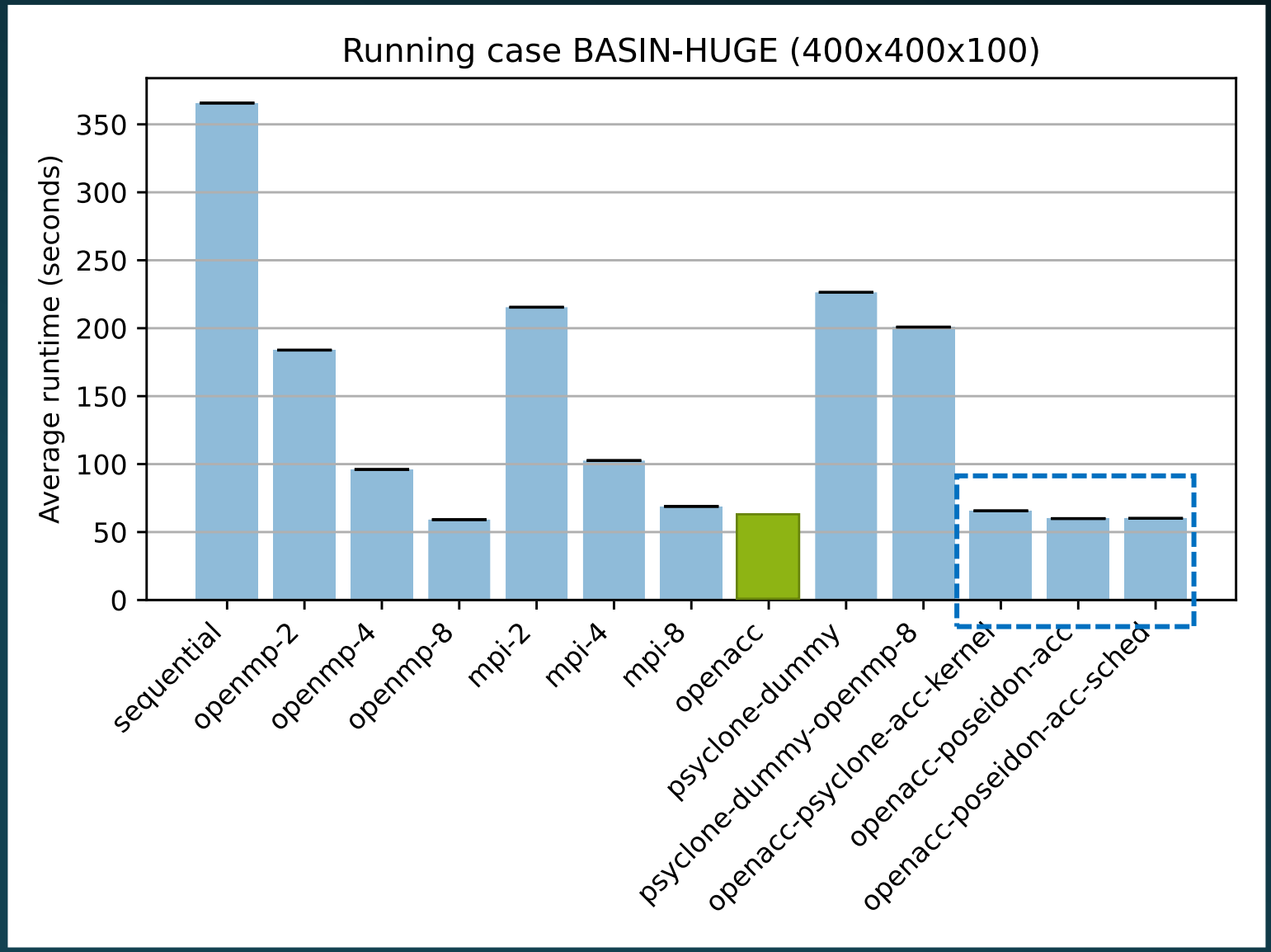
- The **transformation script** can be **unit tested** (python)

- It can **inject some safety checking code** (TODO).

Tested

Mode	Specificity
Sequential	• V1.3 • OMP_NUM_THREADS=1
Openmp-8	• V1.3 • **OMP_NUM_THREADS=8**
openacc	• **dev2022_GPU_merge**
psyclone-dummy	• V1.3 • OMP_NUM_THREADS=1 • **PSyClone rewrite (not transform)**
psyclone-acc	• V1.3 • OMP_NUM_THREADS=1 • **PSyClone Nemo ACC transform scripts**
openacc-psyclone-acc	• **dev2022_GPU_merge** • OMP_NUM_THREADS=1 • PSyClone Nemo ACC transform scripts
openacc-poseidon-acc	• **dev2022_GPU_merge** • OMP_NUM_THREADS=1 • PSyClone + my custom ACC transform scripts

Current status
step2d: OK - **step3d**: ongoing



On Bigfoot

Work in progress

- Case **BASIN** => **ONGOING**
- Handle loops => **ONGOING**
- GPU data transferts => **TODO**

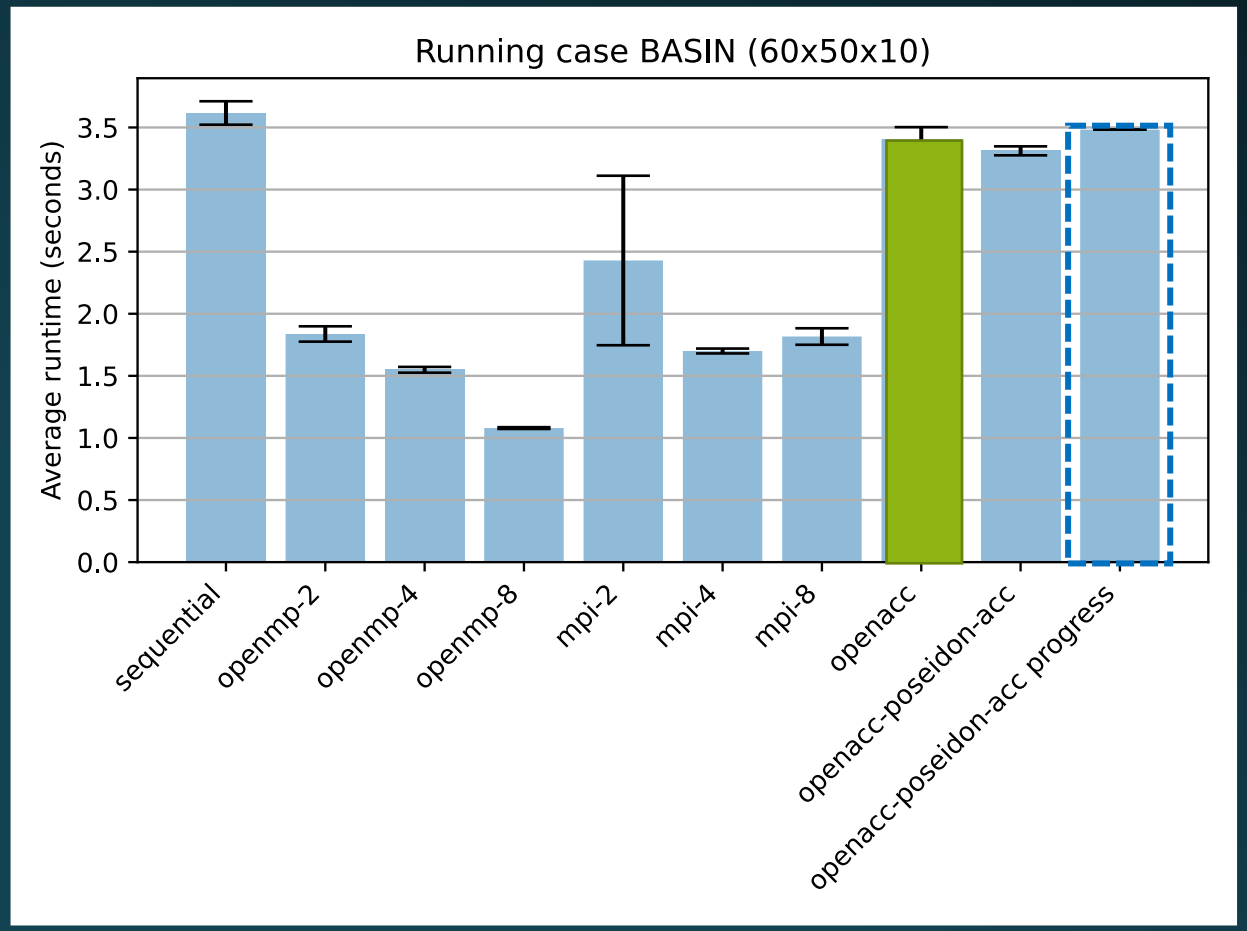
analytical_.f
diag_.f
get_vbc_.f
get_srflux_.f
get_stflux_.f
grid_stiffness_.f
omega_.f
pre_step3d_.f
prsgrd_.f
rho_eos_.f
rhs3d_.f
set_depth_.f
setup_grid1_.f
setup_grid2_.f
step2d_.f
step3d_t_.f
step3d_uv1_.f
step3d_uv2_.f

t3dbc_.f
t3dmix_.f
u2dbc_.f
u3dbc_.f
uv3dmix_.f
v2dbc_.f
v3dbc_.f
wvlcty_.f
zetabc_.f

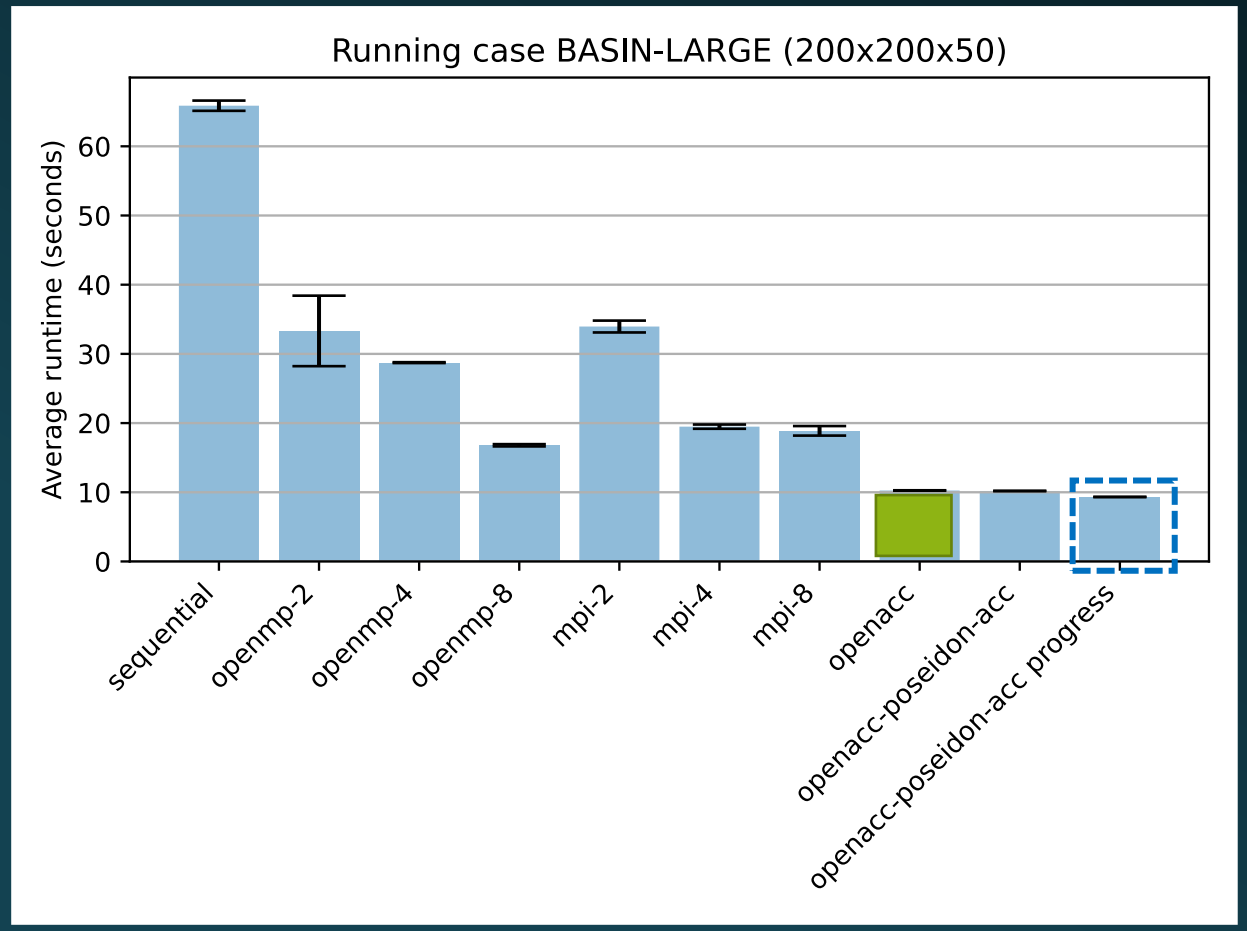
ana_grid_.f
ana_initial_.f

copy_to_device.f

On a small case



On a larger case



Good point

- Psyclone is **fully unit-tested**

- We can **easily write tests** to:
  - **validate** the **transformations**
  - **Outside** of the main code
  - Thanks to Python

- Can apply on the **various Psyclone**:
  - **Cases**
  - **Diverging branches**

Limitation

- **Caution** on that path…
- That's adding **complexity** on top of **complexity** !
- Should keep things **understandable** !

Thanks

What if …

- What if **a tool** can **do it for us**.

- **Not full perfect magic** but…

Issues of manual approch

- Need to **handle** all the **#if / #else / #include** cases…
- Need to handle **loop optimization**.
- How to **debug** / safety **checking** ?