Workshop on

Domain-Specific Languages for Performance-Portable Weather and Climate Models

Content: Advanced Concepts II
(Decorations, regional computation, and the future of GT4Py)

Presenter: Johann Dahm

Learning goals for this session

- Understand the power of the syntax `@gtscript.stencil`
- How to restrict/specialize computation on portions of the IJ axes
- Future design of the toolchain: Storages, horizontal reductions, etc.

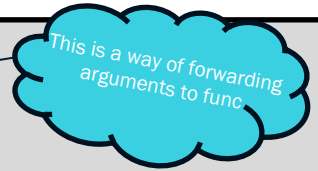
What does @gtscript.stencil actually do?

Python Decorators

A decorator is a function that accepts and returns a function.

```
import time
def timeit(func):
    def wrapper(*args, **kwargs):
        start = time.perf_counter()
        func(*args, **kwargs)
        print("Time: " + str(time.perf_counter() - start))
    return wrapper

@timeit
def say_whee():
    time.sleep(0.5)
    print("Whee!")
```




```
>>> say_whee()
```

What is output when say_whee is executed?

Python Decorators

A decorator is a function that accepts and returns a function.

```
import time
def timeit(func):
    def wrapper(*args, **kwargs):
        start = time.perf_counter()
        func(*args, **kwargs)
        print("Time: " + str(time.perf_counter() - start))
    return wrapper

@timeit
def say_whee():
    time.sleep(0.5)
    print("Whee!")
```

```
>>> say_whee()
Whee!
Time: 0.5012413430013112
```

This is the normal syntax for decorators

Python Decorators

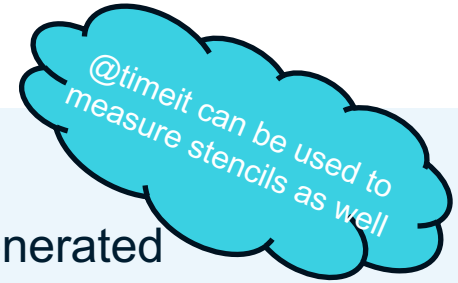
A decorator is a function that accepts and returns a function.

```
import time
def timeit(func):
    def wrapper(*args, **kwargs):
        start = time.perf_counter()
        func(*args, **kwargs)
        print("Time: " + str(time.perf_counter() - start))
    return wrapper

def say_whee():
    time.sleep(0.5)
    print("Whee!")
say_whee = timeit(say_whee)
```

Since decorators are just functions, they can be called in this way as well with the same effect

How does this relate to GT4Py?



`@gtscript.stencil` returns an entirely new function of generated code instead of a wrapper as `@timeit` does. It:

1. Analyzes the source code from the decorated function
2. Lowers through a series of intermediate representations
3. Generates optimized source code in some language (and if need be, compiles it)
4. Wraps the optimized version in a python function and returns it

Regional Computation

Porting Conditionals?

```
389
390   do j=js-1,jep1
391   do i=is-1,iep1
392   if ( ua(i,j) > 0. ) then
393       if ( i==1 ) then
394           ke(1,j) = uc(1,  j)*sin_sg(1,j,1)+v(1,j)*cos_sg(1,j,1)
395       elseif ( i==npx  ) then
396           ke(i,j) = uc(npx,j)*sin_sg(npx,j,1)+v(npx,j)*cos_sg(npx,j,1)
397       else
398           ke(i,j) = uc(i,j)
399       endif
400   else
401       if ( i==0   ) then
402           ke(0,j) = uc(1,  j)*sin_sg(0,j,3)+v(1,j)*cos_sg(0,j,3)
403       elseif ( i==(npx-1)   ) then
404           ke(i,j) = uc(npx,j)*sin_sg(npx-1,j,3)+v(npx,j)*cos_sg(npx-1,j,3)
405       else
406           ke(i,j) = uc(i+1,j)
407       endif
408   endif
409   enddo
410   enddo
```

Conditionals on the iteration position cannot be expressed based on gtscript introduced so far…

Review of the GridTools Parallel Model

Stencils iterate sequentially over ***computations*** in the order they appear in the code

A computation is composed of ***vertical intervals*** that are executed sequentially in the order defined by the ***iteration policy*** of the computation

A vertical interval is executed as a sequential for-loop over the K-range, following the order defined in the iteration policy

Vertical intervals are composed of ***statements***, with ***horizontal extents*** determined automatically

- Parallel model (left) iterates a single statement over the entire horizontal plane
- There are cases where special computation needs to happen on points of the domain
  - BCs
  - Corner cases on multiblock meshes
  - …

Review of the GridTools Parallel Model

Stencils iterate sequentially over ***computations*** in the order they appear in the code

A computation is composed of ***vertical intervals*** that are executed sequentially in the order defined by the ***iteration policy*** of the computation

A vertical interval is executed as a sequential for-loop over the K-range, following the order defined in the iteration policy

Vertical intervals are composed of ***statements***, with ***horizontal extents*** determined automatically

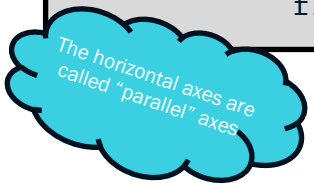
- Parallel model (left) iterates a single statement over the entire horizontal plane
- There are cases where special computation needs to happen on points of the domain
  - BCs
  - Corner cases on multiblock meshes
  - …

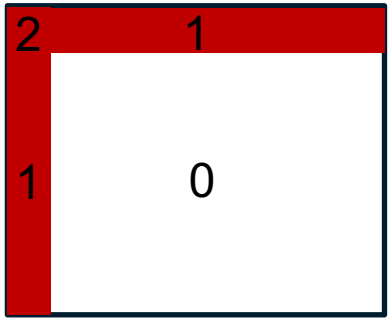
New Feature: Statements can now be restricted to a portion of the ***extended*** IJ plane

Frontend Feature

Simple stencil to set an inout field to the following values

```
@gtscript.stencil(…)
def set_field(field: Field[float]):
    with computation(PARALLEL), interval(…):
        field = 0.
        with parallel(region[I[0]:I[2], :],
                      region[:, I[-2]:]):
            field = 1.
        with parallel(region[:I[2], I[-2]:]):
            field = 2.
```





Wrap statements in `with parallel(…)` to restrict to portion(s) of the IJ axes

Frontend Feature

Simple stencil to set an inout field to the following values

```
@gtscript.stencil(…)
def set_field(field: Field[float]):
    with computation(PARALLEL), interval(…):
        field = 0.
        with parallel(region[I[0]:I[2], :],
                      region[:, I[-2]:]):
            field = 1.
        with parallel(region[:I[2], I[-2]:]):
            field = 2.
```

The indices used in these slices are relative to the start or end of the ***stencil compute domain***

Each argument to `parallel(…)` is a ***region***, which is like a 2D array using numpy slice notation

Frontend Feature

Simple stencil to set an inout field to the following values

```
@gtscript.stencil(…)
def set_field(field: Field[float]):
    with computation(PARALLEL), interval(…):
        field = 0.
        with parallel(region[I[0]:I[2], :],
                      region[:, I[-2]:]):
            field = 1.
        with parallel(region[:I[2], I[-2]:]):
            field = 2.
```

Statements are still executed in order, so the order matters to get the the corner value correct

Axis Offsets

New ***Axis Offset*** syntax subscripts the axes I, J, K inside the compute domain with wrap-around:

- `region[I[0], :]`: first I-axis point in compute domain (west edge)
- `region[I[-1], :]`: last I-axis point in the compute domain (east edge)

To access points in the ***extended compute domain*** for a statement, offset these:

- `region[I[0]-1, :]`: point in extended compute domain (if needed)
- `region[I[-1]+1, :]`: point in extended compute domain (if needed)

The region will only exist in generated code if

1. It sets a field value in the compute domain
2. It sets a temporary value that is consumed in the stencil domain (via offset)

Back to the Example…

Scenario: we want to fill a halo value on the east side, then consume it by using that field with an offset…

```
@gtscript.stencil(…)
def set_field(in_field: Field[float], out_field: Field[float]):
    with computation(PARALLEL), interval(…):
        with parallel(region[I[-1] + 1, :]):
            in_field = 0.5 * (in_field[-1, 0, 0] + in_field[-2, 0, 0])
        out_field = in_field[1, 0, 0]
```

The value is consumed here, so gt4py generates code for the region

Domain Decomposition & Distributed Memory

Regional computations can also be used to enable domain decomposition in GT4Py

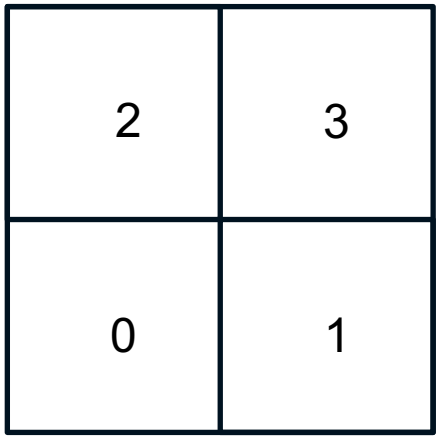
Axis Offsets like `I[0]` can be named externals, such as `istart`

Ranks may have a different value for these externals – leading to different code!

If `istart is None`, then regions using this external are automatically removed from the generated code

- `istart = I[0] if rank % 2 else None`
- `jstart = J[0] if rank < 2 else None`
- …

Domain distributed over 4 ranks



Domain Decomposition & Distributed Memory

Regional computations can also be used to enable domain decomposition in GT4Py

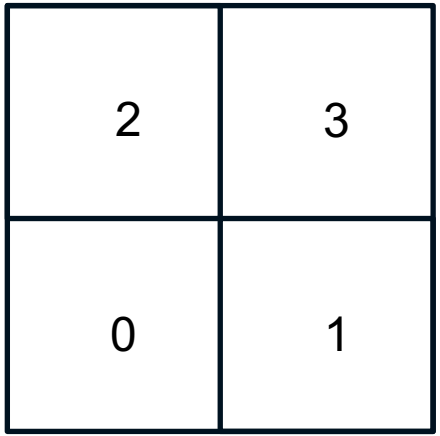
Axis Offsets like `I[0]` can be named externals, such as `istart`

Ranks may have a different value for these externals – leading to different code!

If `istart is None`, then regions using this external are automatically removed from the generated code

- `istart = I[0] if rank % 2 else None`
- `jstart = J[0] if rank < 2 else None`
- …

Domain distributed over 4 ranks




```
def stencil(…):
    from __externals__ import istart
    with computation(PARALLEL), interval(…):
        with parallel(region[istart, :]):
            field = 0.
```

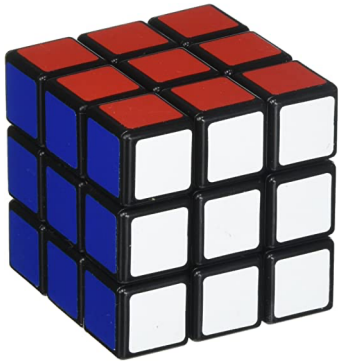
Applied to the Example...

```
iend = I[-1] if proc % 2 == 1 else None
@gtscript.stencil(…)
def set_field(in_field: Field[float], out_field: Field[float]):
    from __externals__ import iend
    with computation(PARALLEL), interval(…):
        with parallel(region[iend + 1, :]):
            in_field = 0.5 * (in_field[-1, 0, 0] + in_field[-2, 0, 0])
        out_field = in_field[1, 0, 0]
```

Can offset the Axis Offset in the region slices

FV3 Application: Corner Computation

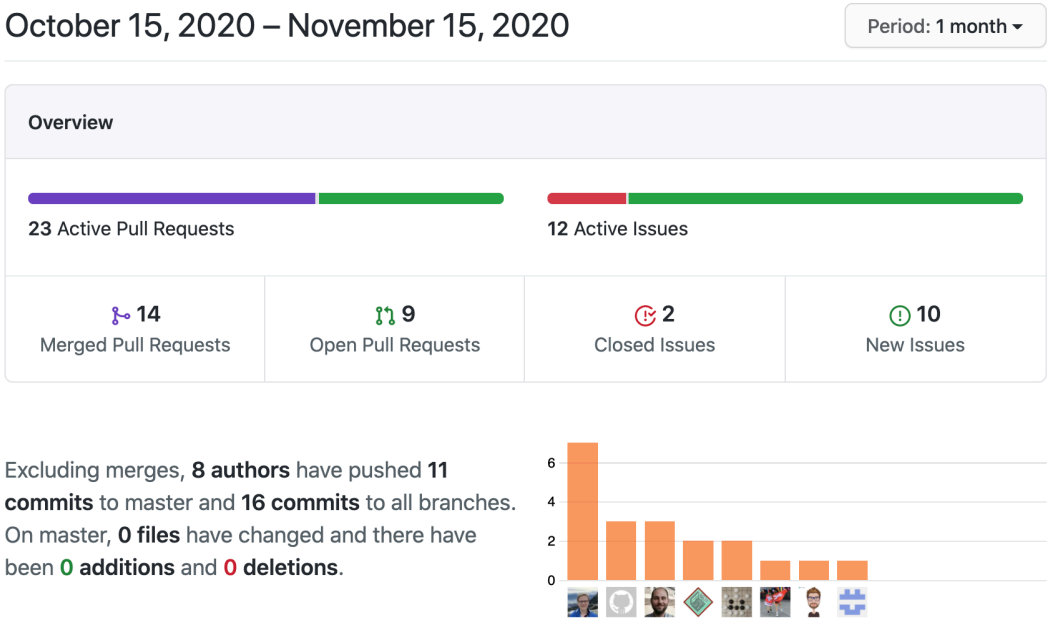
- Finite difference operators must be specialized at the corners of the cubed sphere tiles, since the values do not really exist
- The hands-on tutorial will walk you though an example that applies the bi-Laplacian operator in a cubed sphere context, where special computation happens at the corners



Which concepts / motifs that are missing in GT4Py?

GT4Py: Stable, yet Active Project

- Formal GDP-based process to introduce new features
- Everyday focus on improving testing and error messages



GT4Py Upcoming Features

Recent workshop identified the following items:

- Storages are being replaced by a framework independent of numpy, allowing for cupy, xarray, etc. to be tightly integrated
- Frontend feature to to explicitly obtain the I, J, K positions
- Enable compact access syntax compatible with lower-dimensional storages
  - `field[0, 0, 1] = field[K+1]`
  - `field[1, 1, 0] = field[I+1,J+1]`
- Further reduce the overhead of calling a python stencil
- Simple optimization passes to compute redundant code to remove memory reads

October 2020 GTScript Workshop

GT4Py Future Extensions and Features

We know there are other motifs that may not map directly to stencils that also need to be fast

- Horizontal reductions on fields
- …

We need your help to identify these. GT4Py can only be great if we all work together to bring ideas from the science domain to the DSL!

Questions?

Hands-on Session

Now it's your turn!

`Session-2B.ipynb` to work on

- Writing an efficient transient diffusion code (§1)
- Exploring corner computation (§2)
- Experimenting with decorators (§3)

See you on Slack! **Next huddle at 4:30pm EST**