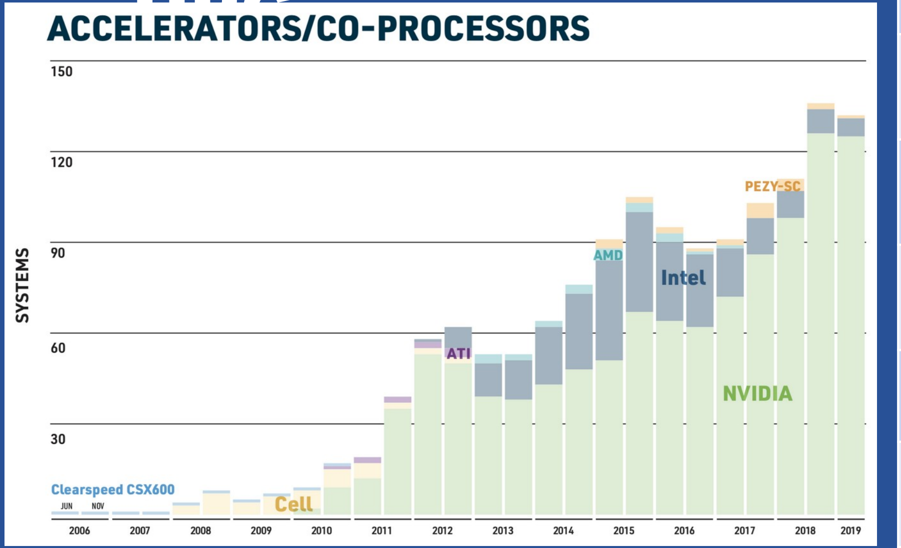
Pace: A GPU-enabled implementation of FV3GFS Using GT4Py

Oliver Elbert, NOAA GFDL

Johann Dahm, Eddie Davis, Florian Deconinck, Rhea George, Jeremy McGibbon, Tobias Wicky, Elynn Wu, Christopher Kung, Tal Ben-Nun, Lucas Harris, Linus Groner, and Oliver Fuhrer

UIFCW 2023
A UFS Collaboration Powered by EPIC

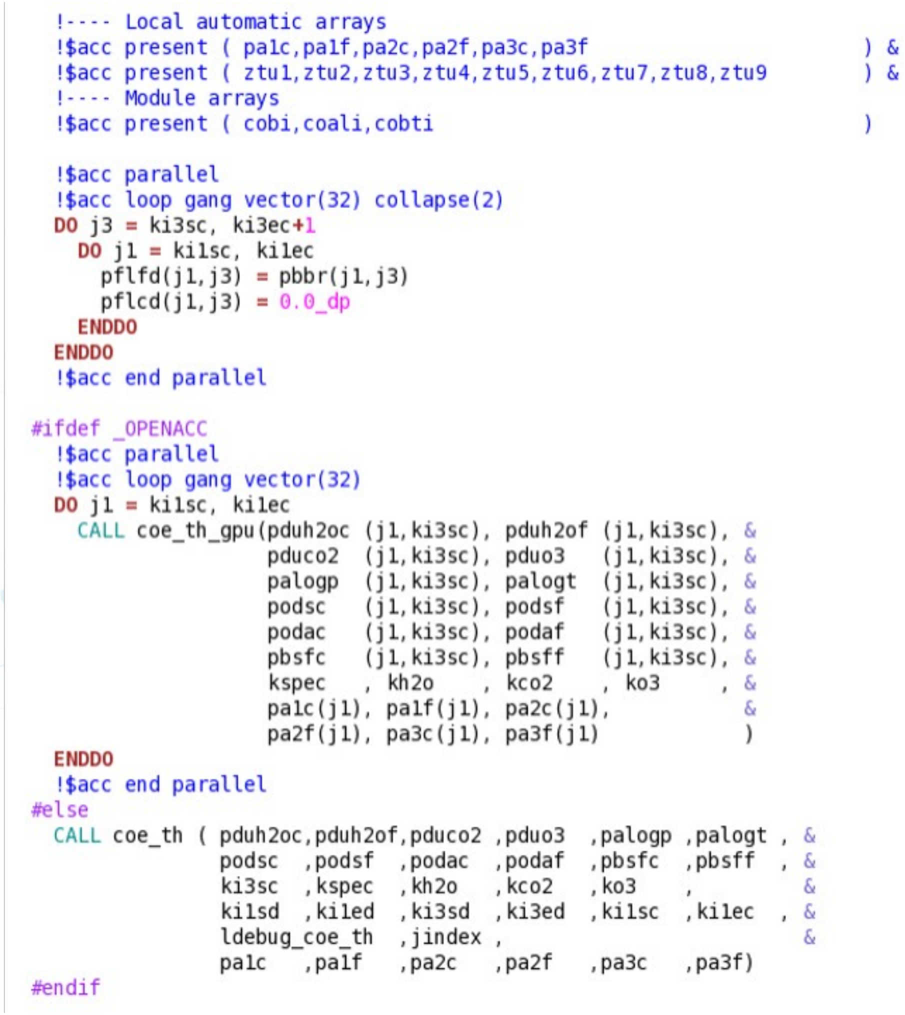
The Future of HPC



#	System	Nodes	Power [MW]	Rmax [PFlop/s]	Chip Technology
1	Frontier	9,472	21.1	1,102.0	1 x AMD, **4 x 250X**
2	Fugaku	158,976	29.9	442.0	1 x A64FX
3	Lumi	2,650*	6.02	309.1	1 x AMD, **4 x 250X***
4	Leonardo	3,456	7.4	238.7	1 x Xeon, **4 x A100**
5	Summit	4,608	10.1	148.6	2 x Power9, **6 x V100**
6	Sierra	4,474	7.4	94.6	2 x Power9, **4 x V100**
7	TaihuLight	40,960	15.4	93.0	**1 x SW26010**
8	Perlmutter	1,536	2.6	70.9	1 x AMD, **4 x A100**
9	Selene	560	2.6	63.5	2 x AMD, **8 x A100**

Approaches

- Rewrite model for hardware backend
- Adapt code with compiler directives
- Write model in a DSL



```
  !---- Local automatic arrays
  !$acc present ( pa1c,pa1f,pa2c,pa2f,pa3c,pa3f                 ) &
  !$acc present ( ztu1,ztu2,ztu3,ztu4,ztu5,ztu6,ztu7,ztu8,ztu9   ) &
  !---- Module arrays
  !$acc present ( cobi,coali,cobti                              )

  !$acc parallel
  !$acc loop gang vector(32) collapse(2)
  DO j3 = ki3sc, ki3ec+1
    DO j1 = ki1sc, ki1ec
      pflfd(j1,j3) = pbbr(j1,j3)
      pflcd(j1,j3) = 0.0_dp
    ENDDO
  ENDDO
  !$acc end parallel

#ifdef _OPENACC
  !$acc parallel
  !$acc loop gang vector(32)
  DO j1 = ki1sc, ki1ec
    CALL coe_th_gpu(pduh2oc (j1,ki3sc), pduh2of (j1,ki3sc), &
                    pduco2  (j1,ki3sc), pduo3   (j1,ki3sc), &
                    palogp  (j1,ki3sc), palogt  (j1,ki3sc), &
                    podsc   (j1,ki3sc), podsf   (j1,ki3sc), &
                    podac   (j1,ki3sc), podaf   (j1,ki3sc), &
                    pbsfc   (j1,ki3sc), pbsff   (j1,ki3sc), &
                    kspec   , kh2o    , kco2    , ko3     , &
                    pa1c(j1), pa1f(j1), pa2c(j1),           &
                    pa2f(j1), pa3c(j1), pa3f(j1)            )
  ENDDO
  !$acc end parallel
#else
  CALL coe_th ( pduh2oc,pduh2of,pduco2 ,pduo3  ,palogp ,palogt , &
                podsc  ,podsf  ,podac  ,podaf  ,pbsfc  ,pbsff  , &
                ki3sc  ,kspec  ,kh2o   ,kco2   ,ko3    ,         &
                ki1sd  ,ki1ed  ,ki3sd  ,ki3ed  ,ki1sc  ,ki1ec  , &
                ldebug_coe_th  ,jindex ,                         &
                pa1c   ,pa1f   ,pa2c   ,pa2f   ,pa3c   ,pa3f)
#endif
```

GridTools for Python

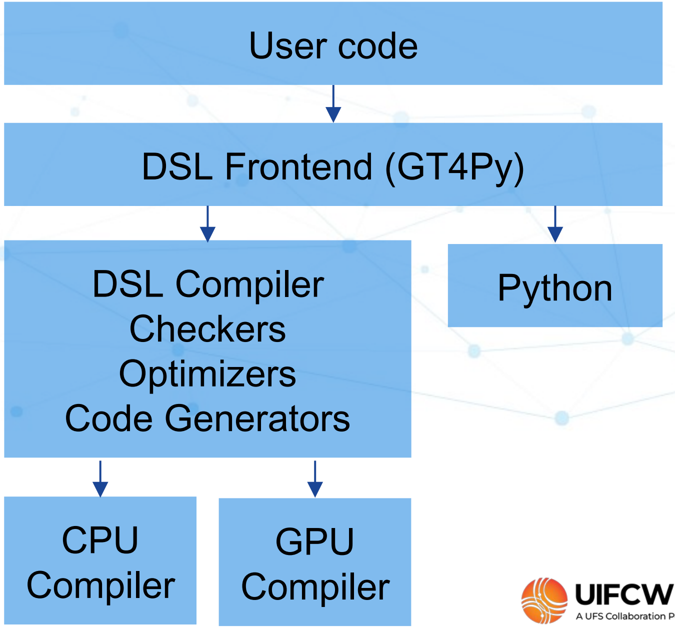
Write model code in Python using GT4Py library

Have access to Python environment for infrastructure, development, testing, etc…

DSL compiler translates code to C++/CUDA and optimizes code for hardware target

github.com/GridTools/gt4py

GridTools 4python



User code

DSL Frontend (GT4Py)

DSL Compiler
Checkers
Optimizers
Code Generators

Python

CPU Compiler

GPU Compiler

The Pace Model

GT4Py port of FV3 + v2 of GFDL cloud microphysics

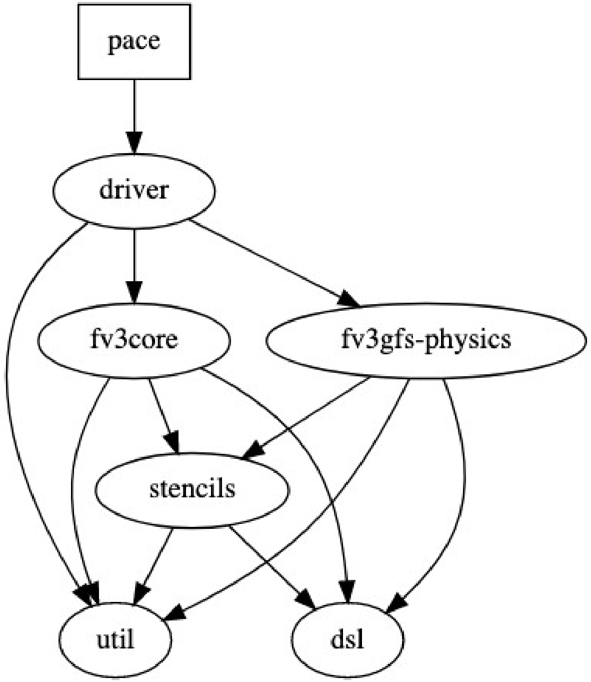
Rest of physics routines ported, not optimized or integrated yet

Contains infrastructure and utilities needed to run simulations, test code, profile performance…

https://github.com/NOAA-GFDL/pace

Dahm et al. 2023:

https://gmd.copernicus.org/articles/16/2719/2023/

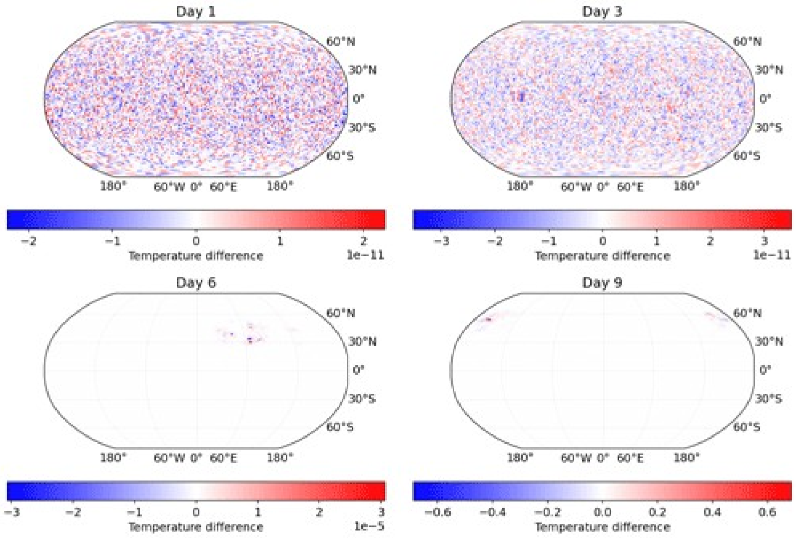


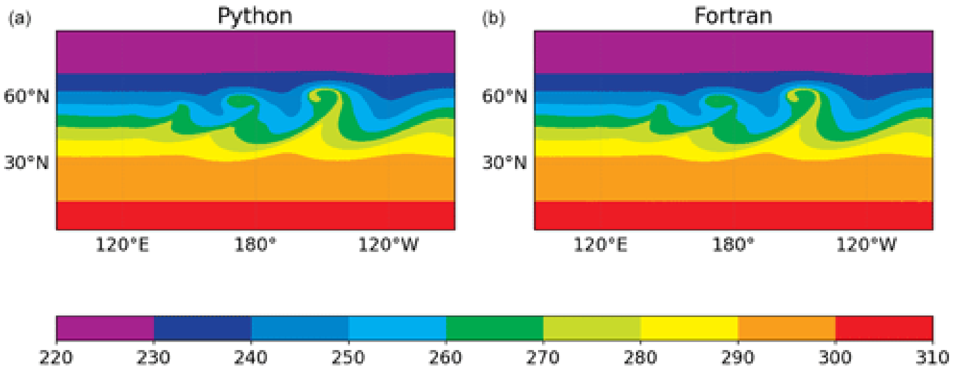
Comparing to Fortran

Moist baroclinic instability integrated for 9 days

Results match fairly well given arithmetic changes

Plotted: 850 mbar temperature





Performance

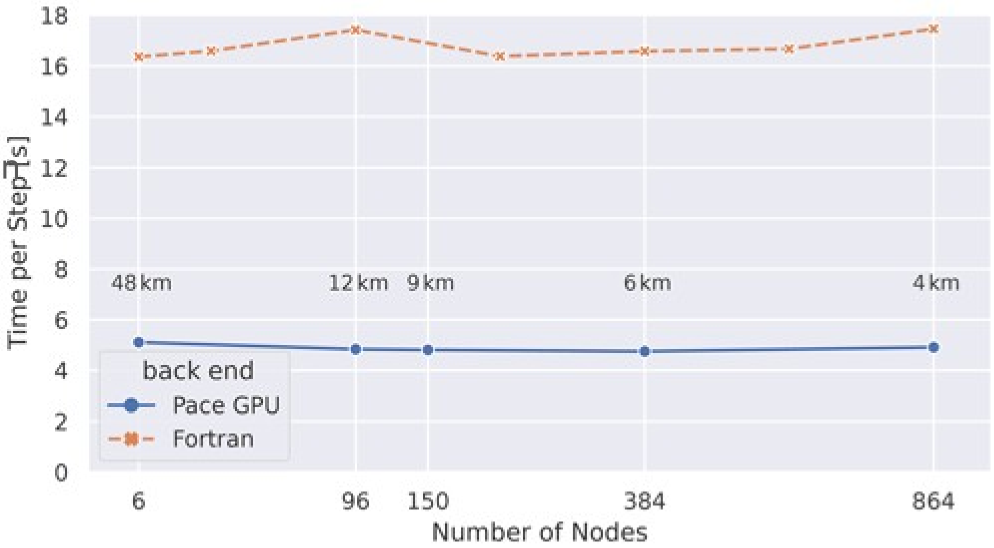
~3.6x speedup over Fortran on P100 GPUs, ~8.6x on A100 GPUs

CPU performance tuning coming soon, currently ~25% of Fortran speed

Ben-Nun et al. 2022:

https://dl.acm.org/doi/abs/10.5555/3571885.3571982

(https://arxiv.org/pdf/2205.04148.pdf)

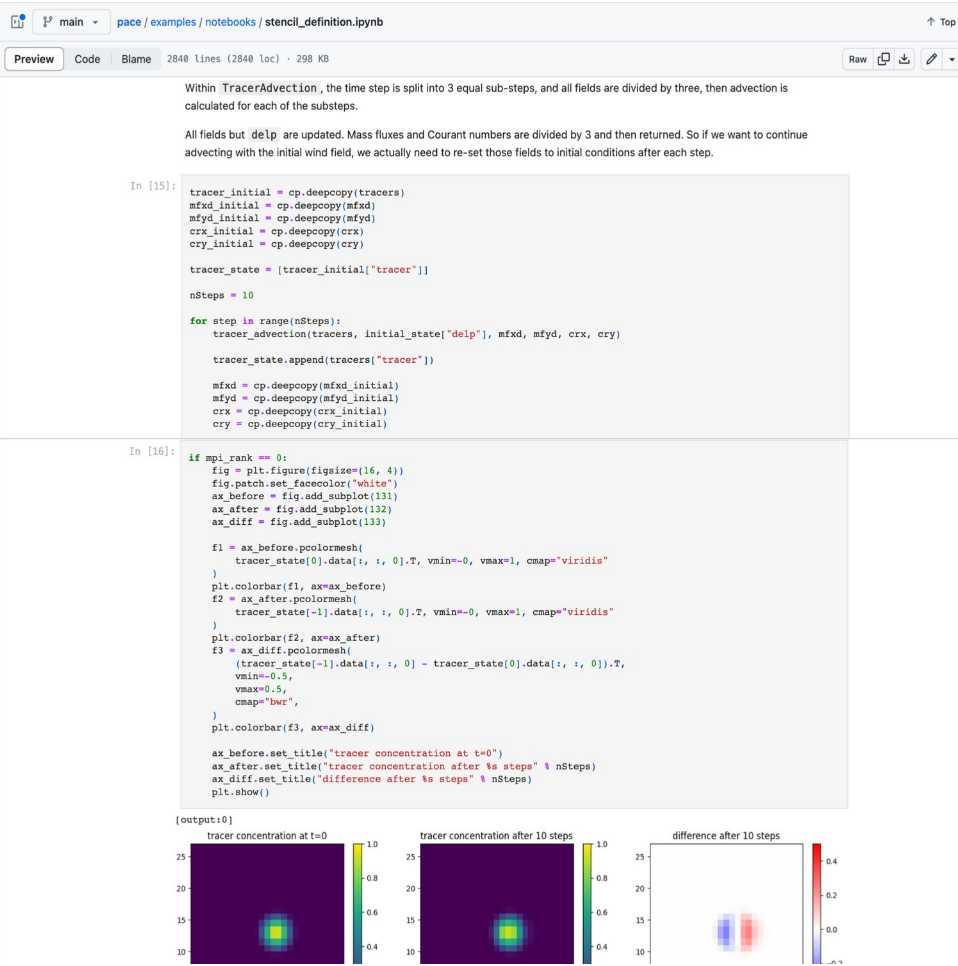


Take Advantage of Python

Python ecosystem available for model development

Can use Jupyter notebooks to run the model, or individual model components

Ex: run tracer advection for 10 steps and plot the results



Within `TracerAdvection`, the time step is split into 3 equal sub-steps, and all fields are divided by three, then advection is calculated for each of the substeps.

All fields but `delp` are updated. Mass fluxes and Courant numbers are divided by 3 and then returned. So if we want to continue advecting with the initial wind field, we actually need to re-set those fields to initial conditions after each step.

In [15]:

```
tracer_initial = cp.deepcopy(tracers)
mfxd_initial = cp.deepcopy(mfxd)
mfyd_initial = cp.deepcopy(mfyd)
crx_initial = cp.deepcopy(crx)
cry_initial = cp.deepcopy(cry)

tracer_state = [tracer_initial["tracer"]]

nSteps = 10

for step in range(nSteps):
    tracer_advection(tracers, initial_state["delp"], mfxd, mfyd, crx, cry)

    tracer_state.append(tracers["tracer"])

    mfxd = cp.deepcopy(mfxd_initial)
    mfyd = cp.deepcopy(mfyd_initial)
    crx = cp.deepcopy(crx_initial)
    cry = cp.deepcopy(cry_initial)
```

In [16]:

```
if mpi_rank == 0:
    fig = plt.figure(figsize=(16, 4))
    fig.patch.set_facecolor("white")
    ax_before = fig.add_subplot(131)
    ax_after = fig.add_subplot(132)
    ax_diff = fig.add_subplot(133)

    f1 = ax_before.pcolormesh(
        tracer_state[0].data[:, :, 0].T, vmin=-0, vmax=1, cmap="viridis"
    )
    plt.colorbar(f1, ax=ax_before)
    f2 = ax_after.pcolormesh(
        tracer_state[-1].data[:, :, 0].T, vmin=-0, vmax=1, cmap="viridis"
    )
    plt.colorbar(f2, ax=ax_after)
    f3 = ax_diff.pcolormesh(
        (tracer_state[-1].data[:, :, 0] - tracer_state[0].data[:, :, 0]).T,
        vmin=-0.5,
        vmax=0.5,
        cmap="bwr",
    )
    plt.colorbar(f3, ax=ax_diff)

    ax_before.set_title("tracer concentration at t=0")
    ax_after.set_title("tracer concentration after %s steps" % nSteps)
    ax_diff.set_title("difference after %s steps" % nSteps)
    plt.show()
```

```
[output:0]
```

Ongoing Development

Latest GFDL Cloud Microphysics

Integrate and optimize more physics parameterizations

CPU performance

Additional Capabilities

Put Pace to work!

Thank You!

Former AI2 DSL Team


Oliver Elbert

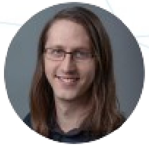

Oli Fuhrer


Johann Dahm


Tobias Wicky


Rhea George


Eddie Davis


Jeremy McGibbon


Elynn Wu


Florian Deconinck

Collaborators

Ai2

CSCS

NASA

W UNIVERSITY of WASHINGTON

ETH

MeteoSwiss

This slide intentionally left blank

UIFCW 2023

A UFS Collaboration Powered by EPIC

GPUs vs CPUs

CPU:

Complex instructions done in serial

"Add 3 to column 1, then divide the first row by 7, then multiply elements 1, 3, 7, 12, and 13 by 4…"

7	2	1	6
13	5	4	7
2	3	4.9e7	4
6	6	1	21

"Ok!"

GPU:

Simple instructions done in parallel

"Multiply all these by 5"

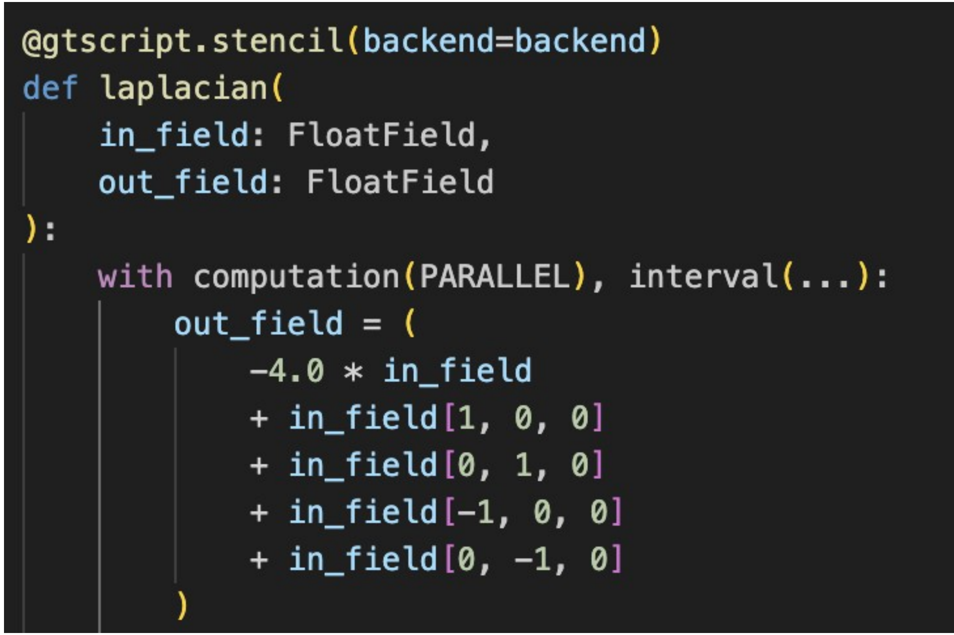
7	2	1	6
13	5	4	7
2	3	4.9e7	4
6	6	1	21

"Ok!"

GT4Py Stencils

Instead of looping through x, y, z define a function that gets moved around the grid and applied to all the points you want

Pass 3D fields as arguments



```
@gtscript.stencil(backend=backend)
def laplacian(
    in_field: FloatField,
    out_field: FloatField
):
    with computation(PARALLEL), interval(...):
        out_field = (
            -4.0 * in_field
            + in_field[1, 0, 0]
            + in_field[0, 1, 0]
            + in_field[-1, 0, 0]
            + in_field[0, -1, 0]
        )
```

More on Stencils

Stencils run on all horizontal indices in parallel

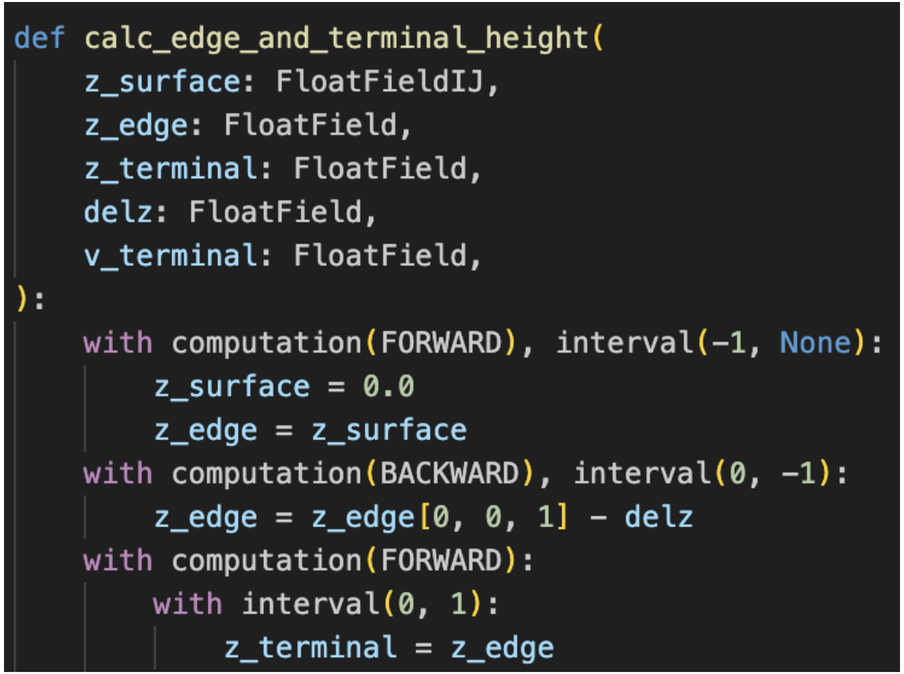
Can set vertical order and boundaries within stencil through `computation` and `interval`

`FORWARD` to go from k=0 to kend, `BACKWARD` to go in reverse

Interval slices like numpy

Use offsets instead of absolute indices i-1 instead of i=13

Can't write with offsets



```
def calc_edge_and_terminal_height(
    z_surface: FloatFieldIJ,
    z_edge: FloatField,
    z_terminal: FloatField,
    delz: FloatField,
    v_terminal: FloatField,
):
    with computation(FORWARD), interval(-1, None):
        z_surface = 0.0
        z_edge = z_surface
    with computation(BACKWARD), interval(0, -1):
        z_edge = z_edge[0, 0, 1] - delz
    with computation(FORWARD):
        with interval(0, 1):
            z_terminal = z_edge
```

Comparing Code

Fortran

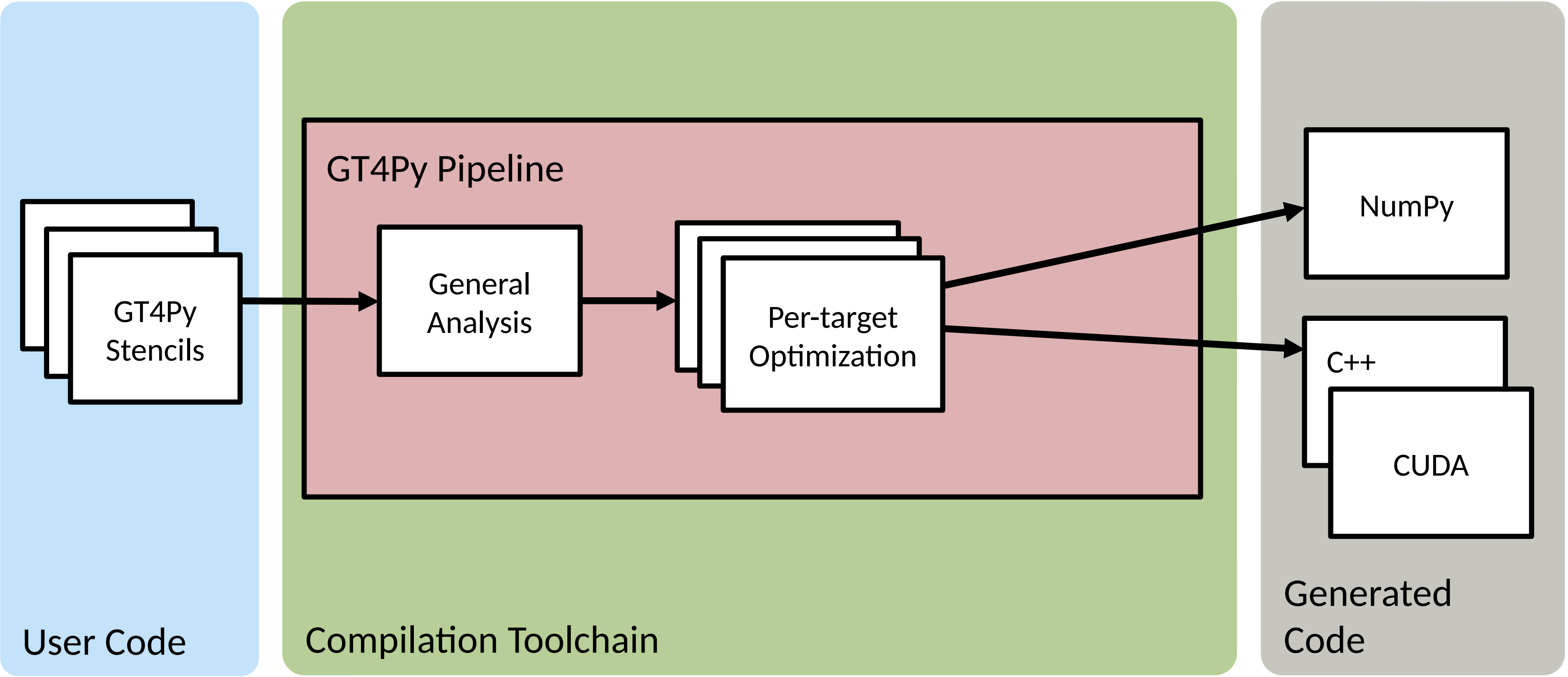
```
subroutine del2_cubed(q, cd, del6_v, del6_u, rarea, grid)

real :: fx(is:ie+1, js,je), fy(is:ie, js:je+1)
!$OMP parallel do default(none) shared(km, q,&
!$OMP is,ie,js,je, & cd) &
!$OMP private(fx, fy)
do k = 1, km
  do j = js, je
    do i = is, ie + 1
      fx(i,j) = del6_v(i,j) * ( q(i-1,j,k) - q(i,j,k) )
    enddo
  enddo

  do j = js, je + 1
    do i = is, ie
      fy(i,j) = del6_u(i,j) * ( q(i,j-1,k) - q(i,j,k) )
    enddo
  enddo

  do j = js, je
    do i = is, ie
      q(i,j,k) = q(i,j,k) + cd * rarea(i,j) * (
          fx(i,j) - fx(i+1,j) + fy(i,j) - fy(i,j+1) )
    enddo
  enddo
enddo
...

end subroutine del2_cubed

call del2_cubed(q, cd, del6_v, del6_u, rarea, grid)
```

GT4Py

```
@gtscript.function
def delx(q, weight):
    return weight * (q[-1, 0, 0] – q)

@gtscript.function
def dely(q, weight)
    return weight * (q[0, -1, 0] – q)

@gtscript.stencil(backend=‘numpy’)
def del2_cubed(q:field, rarea:field, del6_v:field, del6_u:field, cd:float):
    with computation(PARALLEL), interval(...):
        fx = delx(q, del6_v)
        fy = dely(q, del6_u)
        q = q + cd * rarea * (fx - fx[1, 0, 0] + fy - fy[0, 1, 0])

del2_cubed(q, del6_u, del6_v rarea, cd,
           origin=grid.compute_origin(), domain=grid.compute_domain())
```

- Horizontal **loops** removed, **OMP** removed
- Index offsets instead of absolute indices
- No explicit storage statements for **temporary variabl**
- Overhead-free, reusable **functions** -- inlining
- Less code
- No explicit parallelism or data storage layout
- Escaping into straight Python is possible

DaCe



GT4Py Stencils

GT4Py Pipeline

General Analysis

Per-target Optimization

NumPy

C++

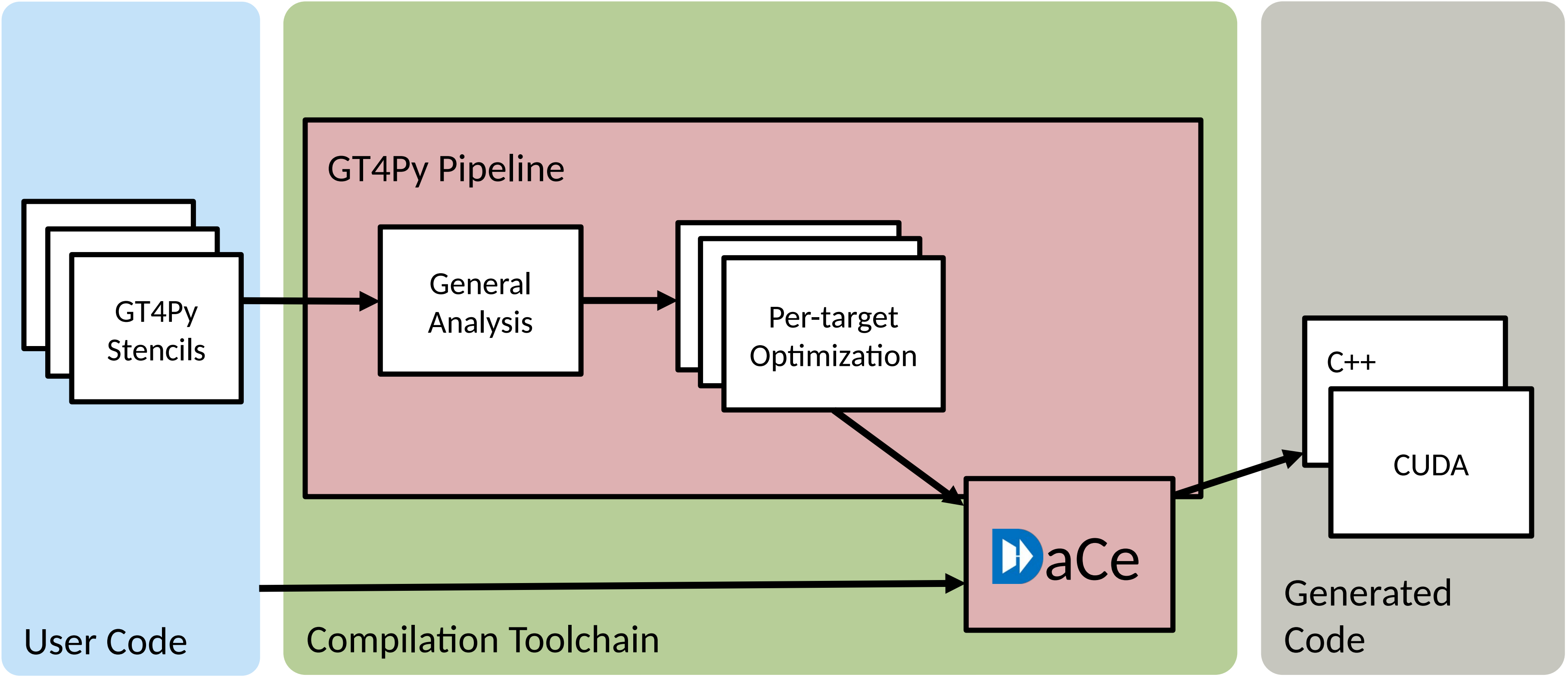
CUDA

User Code

Compilation Toolchain

Generated Code

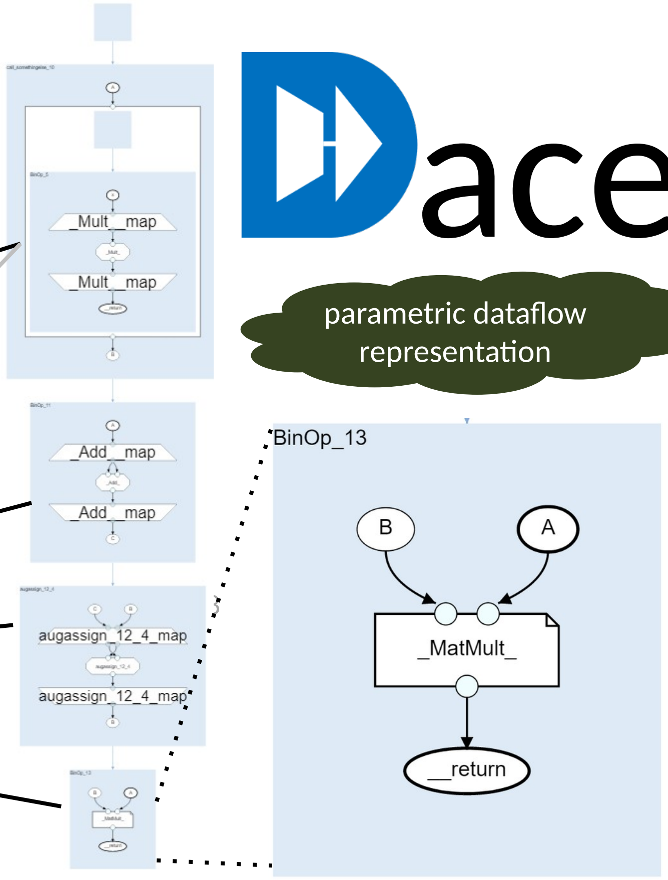
DaCe


GT4Py Pipeline
GT4Py Stencils
General Analysis
Per-target Optimization
DaCe
C++
CUDA
User Code
Compilation Toolchain
Generated Code

DaCe

```
import dace

@dace.program
def somethingelse(x):
    return x * 5

@dace.program
def example(A: dace.float6
4[20, 20]):
    B = somethingelse(A)
    C = A + A
    B += C
    return np.dot(B, A)
```

imperative code



parametric dataflow representation

Running the Dycore

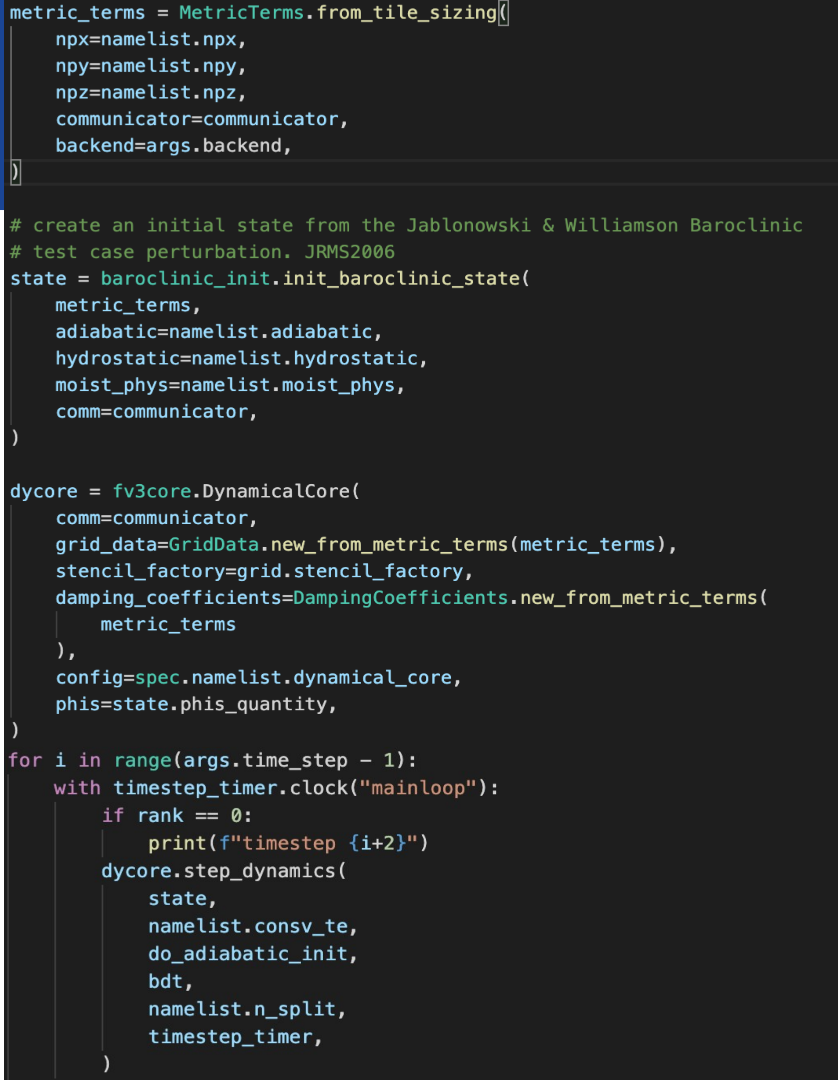
Python runfile.py

Import dynamical core, state setup modules

Load namelist from file, initialize grid, state, and dycore

Then simple loop over timesteps.

Full access to Python ecosystem for performance monitoring, postprocessing, etc.



```
metric_terms = MetricTerms.from_tile_sizing(
    npx=namelist.npx,
    npy=namelist.npy,
    npz=namelist.npz,
    communicator=communicator,
    backend=args.backend,
)

# create an initial state from the Jablonowski & Williamson Baroclinic
# test case perturbation. JRMS2006
state = baroclinic_init.init_baroclinic_state(
    metric_terms,
    adiabatic=namelist.adiabatic,
    hydrostatic=namelist.hydrostatic,
    moist_phys=namelist.moist_phys,
    comm=communicator,
)

dycore = fv3core.DynamicalCore(
    comm=communicator,
    grid_data=GridData.new_from_metric_terms(metric_terms),
    stencil_factory=grid.stencil_factory,
    damping_coefficients=DampingCoefficients.new_from_metric_terms(
        metric_terms
    ),
    config=spec.namelist.dynamical_core,
    phis=state.phis_quantity,
)
for i in range(args.time_step - 1):
    with timestep_timer.clock("mainloop"):
        if rank == 0:
            print(f"timestep {i+2}")
        dycore.step_dynamics(
            state,
            namelist.consv_te,
            do_adiabatic_init,
            bdt,
            namelist.n_split,
            timestep_timer,
        )
```

Object Orientation

Most stencils live inside classes

- Preserves temporary storages
- Stencils compile at init-time
- Simple organization

`__init__` creates an object of the class, handles stencil compilation, etc.

`__call__` means objects are called like functions

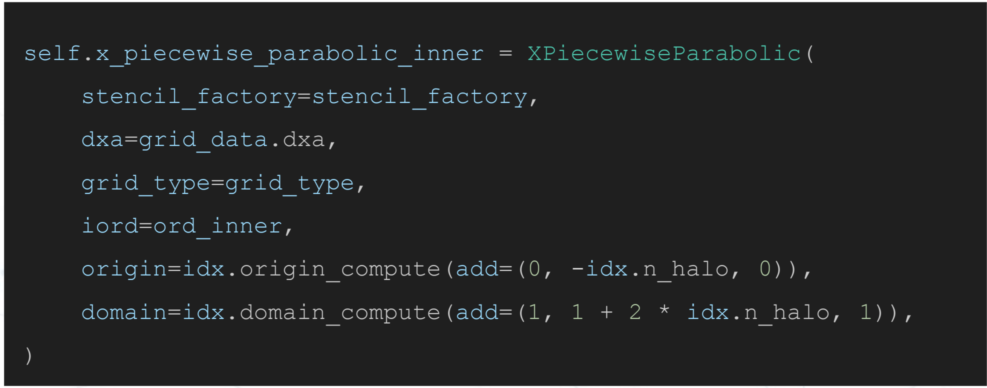
```
class XPiecewiseParabolic:
    """
    Fortran name is xppm
    """

    def __init__(
        self,
        stencil_factory: StencilFactory,
        dxa,
        grid_type: int,
        iord,
        origin: Index3D,
        domain: Index3D,
    ):
        assert grid_type < 3
        self._dxa = dxa
        ax_offsets = stencil_factory.grid_indexing.axis_offsets(origin, domain)
        self._compute_flux_stencil = stencil_factory.from_origin_domain(
            func=compute_x_flux,
            externals={
                "iord": iord,
                "mord": abs(iord),
                "xt_minmax": True,
                "i_start": ax_offsets["i_start"],
                "i_end": ax_offsets["i_end"],
            },
            origin=origin,
            domain=domain,
        )

    def __call__(
        self,
        q_in: FloatField,
        c: FloatField,
        q_mean_advected_through_x_interface: FloatField,
    ):
        """
        Args:
            q_in (in): scalar to be integrated
            c (in): Courant number (u*dt/dx) in x-direction defined on x-interfaces,
                indicates the fraction of the adjacent grid cell which will be
                advected through the interface in one timestep
            q_mean_advected_through_x_interface (out): defined on x-interfaces.
                mean value of scalar within the segment of gridcell to be advected
                through that interface in one timestep, in units of q_in
        """
        self._compute_flux_stencil(
            q_in, c, self._dxa, q_mean_advected_through_x_interface
        )
```

Object Orientation

Objects create objects inside their constructors

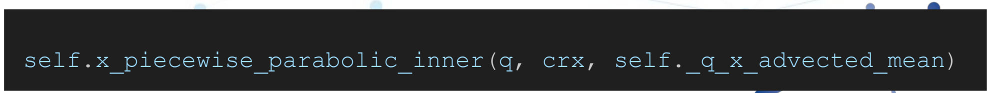
Ex: fvtp2d uses xppm

init:



```
self.x_piecewise_parabolic_inner = XPiecewiseParabolic(
    stencil_factory=stencil_factory,
    dxa=grid_data.dxa,
    grid_type=grid_type,
    iord=ord_inner,
    origin=idx.origin_compute(add=(0, -idx.n_halo, 0)),
    domain=idx.domain_compute(add=(1, 1 + 2 * idx.n_halo, 1)),
)
```

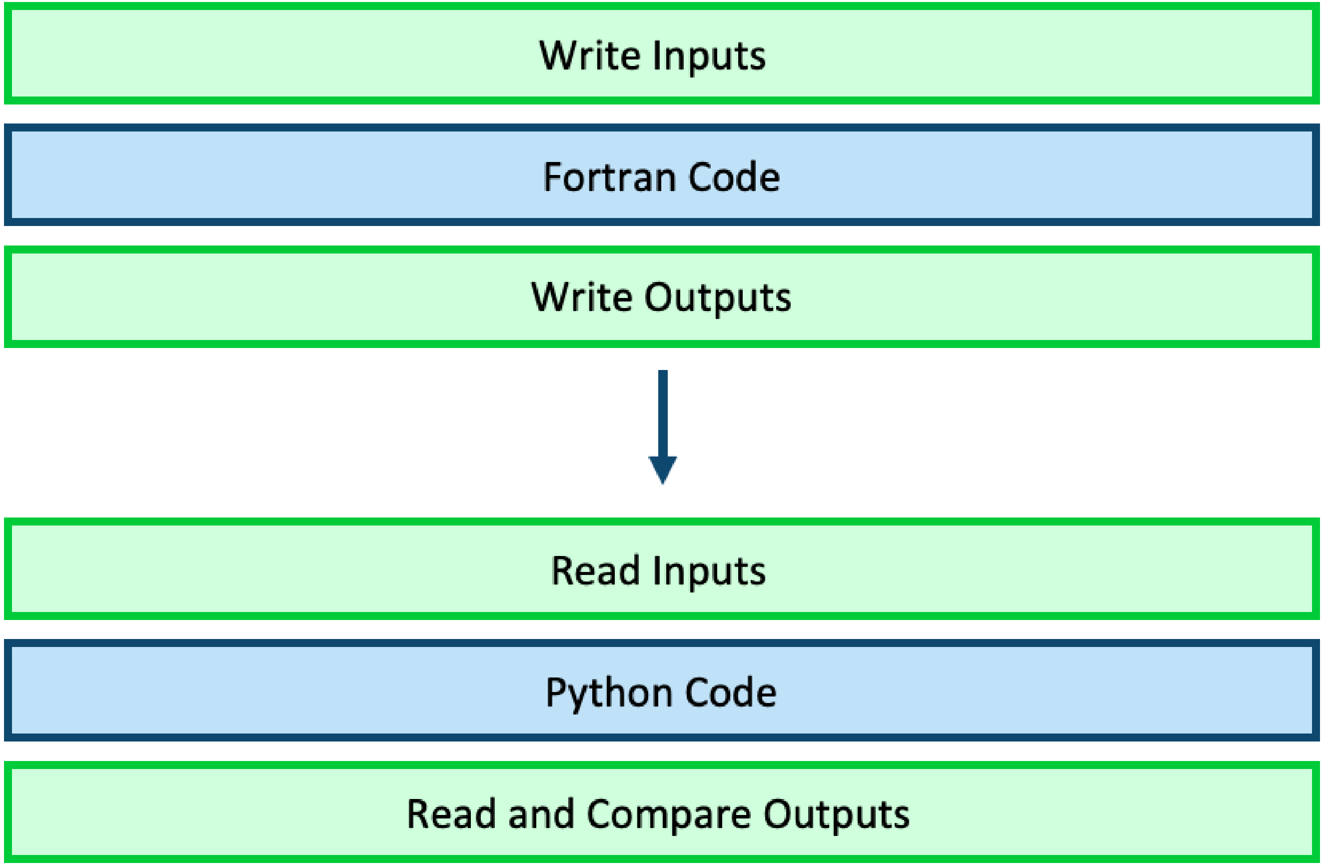
call:



```
self.x_piecewise_parabolic_inner(q, crx, self._q_x_advected_mean)
```

UIFCW 2023
A UFS Collaboration Powered by EPIC

Porting



Write Inputs

Fortran Code

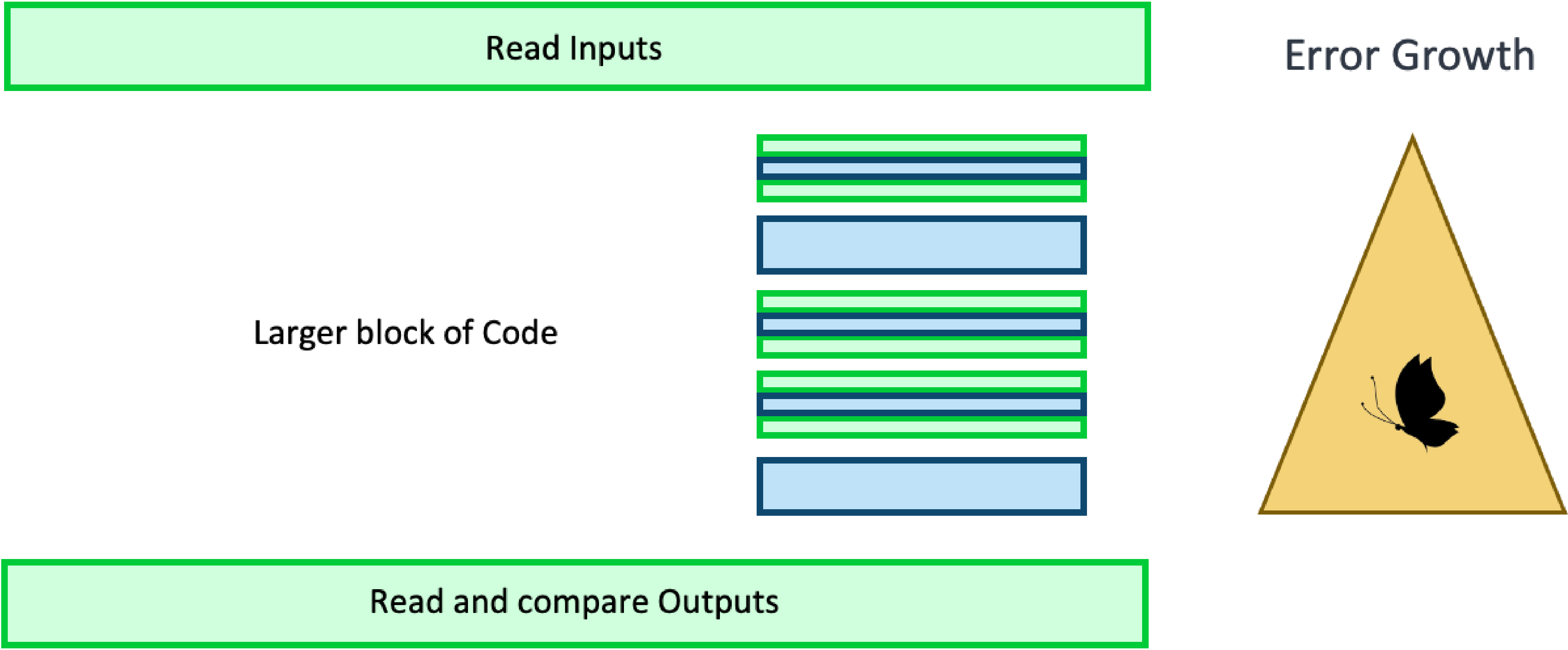
Write Outputs

Read Inputs

Python Code

Read and Compare Outputs

Porting



Read Inputs

Larger block of Code

Error Growth

Read and compare Outputs

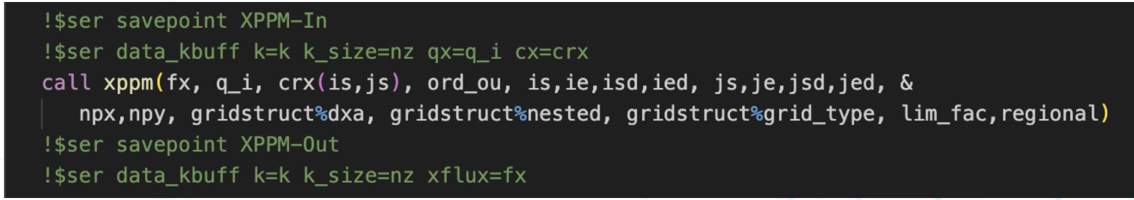
Suite of tests to make sure code matches Fortran as Pace builds up

Serialbox

Handy tool from GridTools team

Extracts data from C++, Python, Fortran codes

For Fortran: !$ser statements



```
!$ser savepoint XPPM-In
!$ser data_kbuff k=k k_size=nz qx=q_i cx=crx
call xppm(fx, q_i, crx(is,js), ord_ou, is,ie,isd,ied, js,je,jsd,jed, &
   npx,npy, gridstruct%dxa, gridstruct%nested, gridstruct%grid_type, lim_fac,regional)
!$ser savepoint XPPM-Out
!$ser data_kbuff k=k k_size=nz xflux=fx
```

Save data or execute code

Python preprocessor replaces !$ser with #ifdef SERIALIZE

```
!$ser verbatim bdt=dt_atmos/real(abs(p_split))
```

Compile Fortran with -DSERIALIZE and run