Implementation of a performance-portable global atmospheric model using a domain-specific language in Python

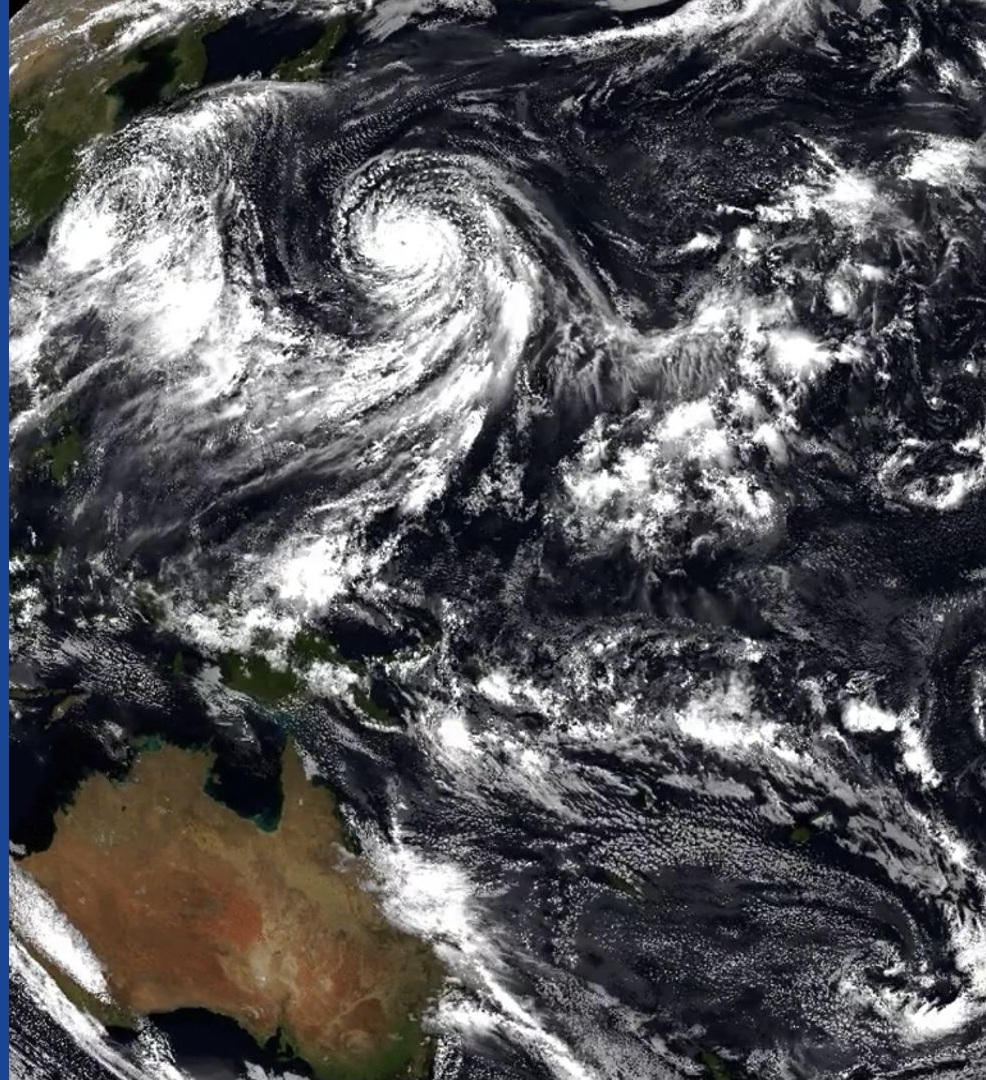
Oliver Fuhrer[1], Rusty Benson[2], Johann Dahm[1], Eddie Davis[1], Florian Deconinck[1], Oliver Elbert[1], Rhea George[1], Lucas Harris[2], Christopher Kung[3], Jeremy McGibbon[1], Tobias Wicky[1], Elynn Wu[1]

[1]Allen Institute for Artificial Intelligence, [2]Geophysical Fluid Dynamics Laboratory NOAA, [3]Global Modeling and Assimilation Office NASA

Goal(s)

Build a global-storm resolving model in Python using an embedded domain-specific language that can run at scale on modern supercomputers

Enable and support novel workflows, e.g. machine learning



The FV3GFS / SHiELD Model

- FV3 dynamical core integrated into UFS, GEOS, CESM, GFDL models
- xSHIELD = FV3 + GFS physics for global storm-resolving simulations (DYAMOND)
- Cubed-sphere grid balances uniform resolution and simple code
- Horizontal finite volume dynamics
- Vertical Lagrangian dynamics with remapping
- Highly optimized for x86 CPU architectures

Why Python?

→ Talk by Stefano Ubbiali is a great example!
→ See talk by Jeremy McGibbon on Friday!

AI2

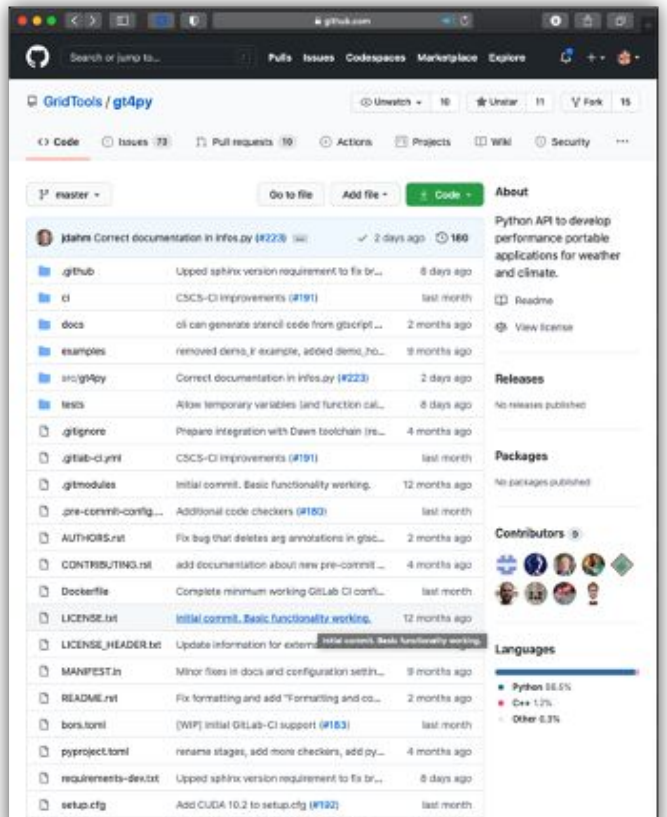
Why a DSL?

- Python is too slow
- No community standard for performance portability
- Incremental approaches (e.g. Fortran+OpenACC) do not always work
- Acceptance of C++ based approaches by community unclear
- Higher level of abstraction opens door to more optimizations
- Bridge the gap between existing performance portability libraries and domain scientists

GT4Py

Check it out on GitHub!
https://github.com/gridtools/gt4py

- Open-source, open-development project
- Joint development with ETH Zurich/CSCS and MeteoSwiss
- Part of the GridTools ecosystem of tools and libraries for weather and climate
- GT4Py is embedded in Python
- Emphasis on tight integration with scientific Python stack
- Multiple backends: Python (NumPy), CPU (C++/OpenMP), GPU (CUDA), …
- Demonstrations with ICON, IFS-FVM, FV3GFS in progress



Frontend: GTScript

$$y = \nabla^4 x = \Delta\left(\Delta \mathrm{x}\right)$$

```
import numpy as np
from gt4py.gtscript import function, Field, PARALLEL, computation, interval

@function
def laplacian(field):
    return - 4. * field[0, 0, 0] + field[-1, 0, 0] + field[1, 0, 0]
        + field[0, -1, 0] + field[0, 1, 0]

def laplap_stencil(in_field: Field[np.float64], out_field: Field[np.float64]):
    with computation(PARALLEL), interval(...):
        tmp = laplacian(in_field)
        out_field = laplacian(tmp)
```

Frontend: GTScript

$$y = \nabla^4 x = \Delta\left(\Delta \mathrm{x}\right)$$

```python
import numpy as np
from gt4py.gtscript import function, Field, PARALLEL, computation, interval

@function
def laplacian(field):
    return - 4. * field[0, 0, 0] + field[-1, 0, 0] + field[1, 0, 0]
        + field[0, -1, 0] + field[0, 1, 0]

def laplap_stencil(in_field: Field[np.float64], out_field: Field[np.float64]):
    with computation(PARALLEL), interval(...):
        tmp = laplacian(in_field)
        out_field = laplacian(tmp)
```

Stencils are valid Python functions with enhanced syntax

The DSL provides functions, etc. that GT4Py recognizes

Frontend: GTScript

$$y = \nabla^4 x = \Delta\left(\Delta \mathrm{x}\right)$$

```python
import numpy as np
from gt4py.gtscript import function, Field, PARALLEL, computation, interval

@function
def laplacian(field):
    return - 4. * field[0, 0, 0] + field[-1, 0, 0] + field[1, 0, 0]
        + field[0, -1, 0] + field[0, 1, 0]

def laplap_stencil(in_field: Field[np.float64], out_field: Field[np.float64]):
    with computation(PARALLEL), interval(...):
        tmp = laplacian(in_field)
        out_field = laplacian(tmp)
```

Functions called in the statements accept immutable inputs and return an output at each point

`computation` and `interval` provide the vertical context for the statements

AI2

Frontend: GTScript

$$y = \nabla^4 x = \Delta\left(\Delta \mathrm{x}\right)$$

```
import numpy as np
from gt4py.gtscript import function, Field, PARALLEL, computation, interval

@function
def laplacian(field):
    return - 4. * field[0, 0, 0] + field[-1, 0, 0] + field[1, 0, 0]
        + field[0, -1, 0] + field[0, 1, 0]

def laplap_stencil(in_field: Field[np.float64], out_field: Field[np.float64]):
    with computation(PARALLEL), interval(...):
        tmp = laplacian(in_field)
        out_field = laplacian(tmp)
```

Temporaries can be declared on-the-fly and are optimized by compiler.

Loop bounds in ij-direction are derived automatically (e.g. tmp)

Storages

```
import numpy as np
from gt4py import storage, gtscript
from gt4py.gtscript import Field, PARALLEL, computation, interval

# ... stencil definition ...

backend = "numpy"
shape = (12, 12, 7)
in_field = storage.empty(
    backend=backend, dtype=float, shape=shape, default_origin=(0, 0, 0)
)
in_field[:] = np.random.randn(*shape)
out_field = storage.empty(
    backend=backend, dtype=float, shape=shape, default_origin=(0, 0, 0)
)

laplap = gtscript.stencil(backend=backend, definition=laplap_stencil)

laplap(in_field, out_field)
```

Storages

Storages behave like NumPy/CuPy arrays

```
import numpy as np
from gt4py import storage, gtscript
from gt4py.gtscript import Field, PARALLEL, computation, interval

# ... stencil definition ...

backend = "numpy"
shape = (12, 12, 7)
in_field = storage.empty(
    backend=backend, dtype=float, shape=shape, default_origin=(0, 0, 0)
)
in_field[:] = np.random.randn(*shape)
out_field = storage.empty(
    backend=backend, dtype=float, shape=shape, default_origin=(0, 0, 0)
)

laplap = gtscript.stencil(backend=backend, definition=laplap_stencil)

laplap(in_field, out_field)
```

Storages

CPU and GPU backends available

```
import numpy as np
from gt4py import storage, gtscript
from gt4py.gtscript import Field, PARALLEL, computation, interval

# ... stencil definition ...

backend = "numpy"
shape = (12, 12, 7)
in_field = storage.empty(
    backend=backend, dtype=float, shape=shape, default_origin=(0, 0, 0)
)
in_field[:] = np.random.randn(*shape)
out_field = storage.empty(
    backend=backend, dtype=float, shape=shape, default_origin=(0, 0, 0)
)

laplap = gtscript.stencil(backend=backend, definition=laplap_stencil)

laplap(in_field, out_field)
```

Stencils are compiled for a backend

What it is… ✓

Architecture-agnostic expressive language focusing on algorithm
(abstracts parallelism and removes boilerplate, focus on stencils)

Single-node abstraction
(does not consider distributed memory parallelism)

Focus on patterns found in weather and climate models
(domain-knowledge used for optimization strategies)

What it isn't… ✗

Solution for everything
(currently does not do reductions, atomics, not meant for data assimilation, …)

Final product
(Under active development, frontend & backends are subject to change)

Drop-in replacement of NumPy
(expressive, domain-specific syntax)

AI2

Toolchain

```python
import numpy as np
from gt4py.gtscript import Field, PARALLEL,
computation, interval, function

@function
def laplacian(field):
    return - 4. * field[ 0, 0, 0] + field[-1, 0, 0]
        + field[1, 0, 0] + field[0, -1, 0]
        + field[0, 1, 0]

def laplap_stencil(in_field: Field[np.float64],
out_field: Field[np.float64]):
    with computation(PARALLEL), interval(...):
        tmp = laplacian(in_field)
        out_field = laplacian(tmp)
```

Application

Frontend

Toolchain

Backend

User code (DSL)

Python AST

GridTools IR

Analysis, Parallelization

Optimizable IR

General optimizations

Backend IR

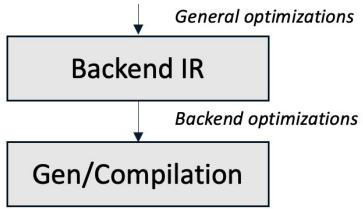
Backend optimizations

Gen/Compilation

Load and execute

AI2

Backends: CUDA, Numpy, GridTools C++, etc!



CUDA backend code

```
template <class Sid> struct loop_4780344224_f {
template <class Validator>
GT_FUNCTION_DEVICE void operator()(const int _i_block, const int
_j_block,
                                   Validator validator) const {
  // ...
  // VerticalLoopSection 4779410960
  if (_k_block >= 0 && _k_block < k_size + 0) {
    // HorizontalExecution 4779777280
    if (validator(extent<0, 0, 0, 0>())) {
      double tmp4 = -4.0 * in_field(0_c, -1_c, 0_c) + …;
      double tmp3 = -4.0 * in_field(0_c, 0_c, 0_c) + …;
      double tmp2 = -4.0 * in_field(0_c, 1_c, 0_c) + …;
      double tmp1 = -4.0 * in_field(-1_c, 0_c, 0_c) + …;
      double tmp0 = -4.0 * in_field(1_c, 0_c, 0_c) + …;
      out_field(0_c, 0_c, 0_c) = -4.0 * tmp3 + tmp1 + tmp0 + tmp2 + tmp4;
    }}}};
```

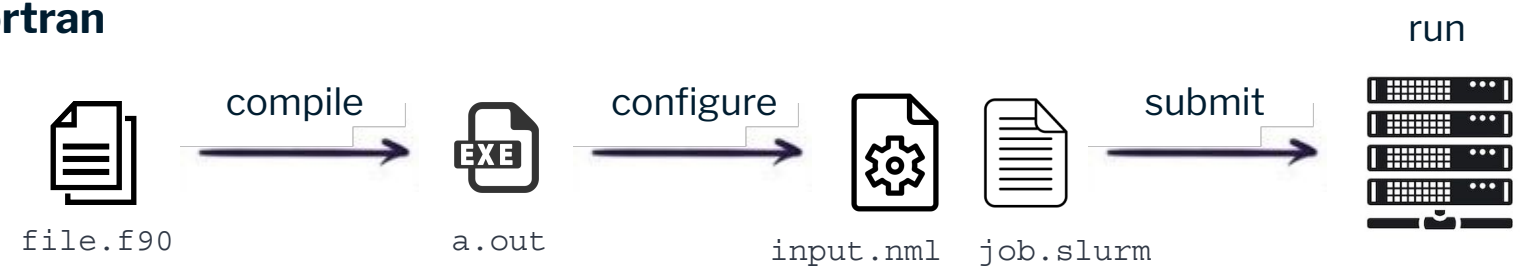
NumPy backend code

```
# VerticalLoopSection 4779410960
# HorizontalExecution 4779777280
in_field, out_field = \
    setup_accessors_4779777280(in_field_arr, out_field_arr)
tmp4 = -4.0 * in_field[i:I, j-1:J-1, k:K] + …
tmp3 = -4.0 * in_field[i:I, j:J, k:K] + …
tmp2 = -4.0 * in_field[i:I, j+1:J+1, k:K] + …
tmp1 = -4.0 * in_field[i-1:I-1, j:J, k:K] + …
tmp0 = -4.0 * in_field[i+1:I+1, j:J, k:K] + …
out_field[i:I, j:J k:K] = -4.0 * tmp3 + tmp1 + tmp0 + tmp2 + tmp4
```

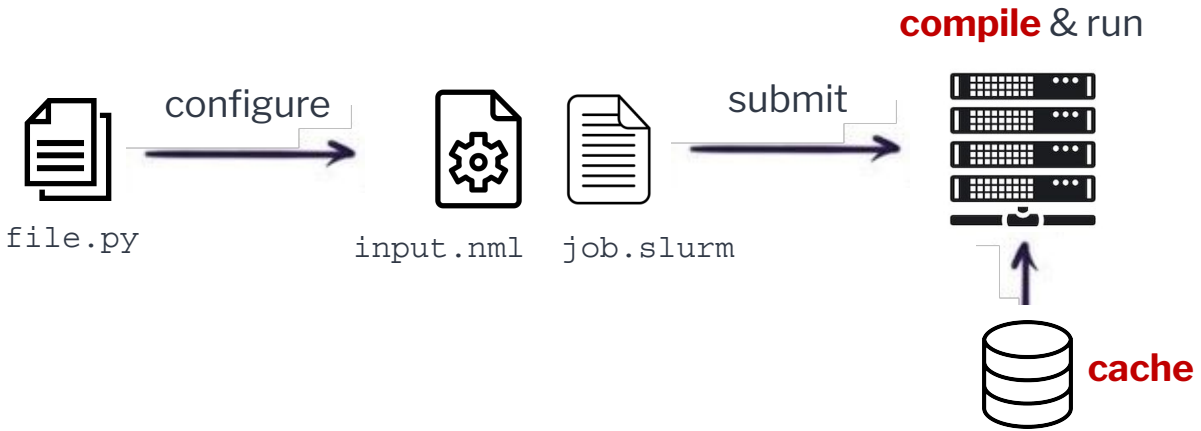
AI2

Just-in-time (JIT) compilation

Fortran



Python DSL



Just-in-time (JIT) compilation

User code

```
@gtscript.function
def compute_flux(in):
  b0 = 0.5 * (in[0, 0, 0] + in[1, 0, 0])
  bl = in[0, 0, 0]
  br = in[1, 0, 0]
  if __INLINED(mord == 5):
    q = bl * br < 0
  else:
    q = (3.0 * abs(b0)) < abs(bl - br)
  al = p1 * (q[-1, 0, 0] + q)
  with horizontal(region[i_start-1, :]):
    al = c1 * q[-2, 0, 0]
  return al[1, 0, 0] - al
```

mord=5

Not on edge!

Code passed to optimizer

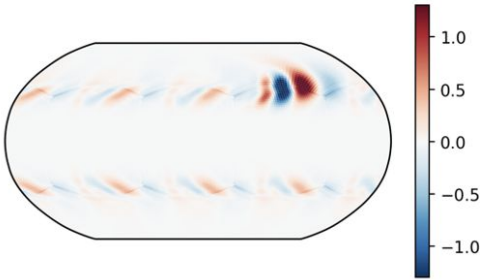
```
@gtscript.function
def compute_flux(in):
  b0 = 0.5 * (in[0, 0, 0] + in[1, 0, 0])
  bl = in[0, 0, 0]
  br = in[1, 0, 0]


  q = bl * br < 0



  al = p1 * (q[-1, 0, 0] + q)


  return al[1, 0, 0] - al
```

Dynamical Core (FV3)

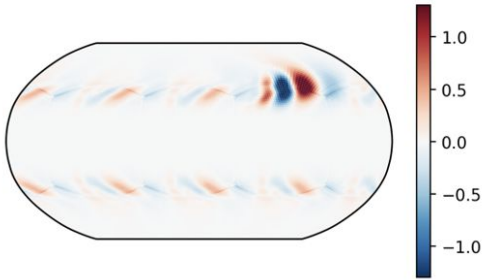
Check it out on GitHub!
https://github.com/ai2cm/fv3core

Baroclinic instability testcase (Jablonowski and Williamson 2006)
6 day surface temperature anomaly [K]

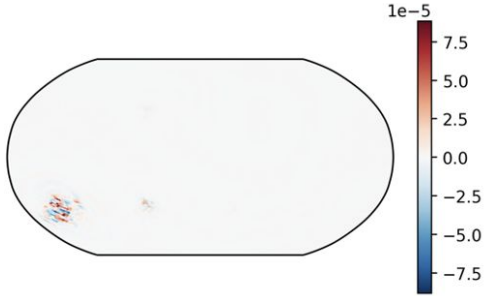
Reference
(Fortran, x86 CPU)



DSL Port
(Python, NVIDIA GPU)

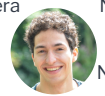


Difference



Physical Parameterizations

Check it out on GitHub!
https://github.com/ai2cm/physics_standalone

	Microphysics	PBL &Turbulence	Sea-Ice	Shallow Convection	LSM	Radiation
Authors	 Mikael	 Chris	 Vera  Nadja  Nicolai	 Mikael  Chenwei  Langwen	 Safira	 Andrew
Scheme	GFDL Cloud Microphysics Scheme	GFS scale-aware EDMF PBL and Free Atmospheric Turbulence Scheme	GFS Sea Ice Scheme	GFS SAS-based Mass-Flux Scheme for Shallow convection (sa-MF)	GFS Noah Land Surface Model	GFS RRTMG
Status	Ported	Ported	Ported	Ported	Ported	In progress

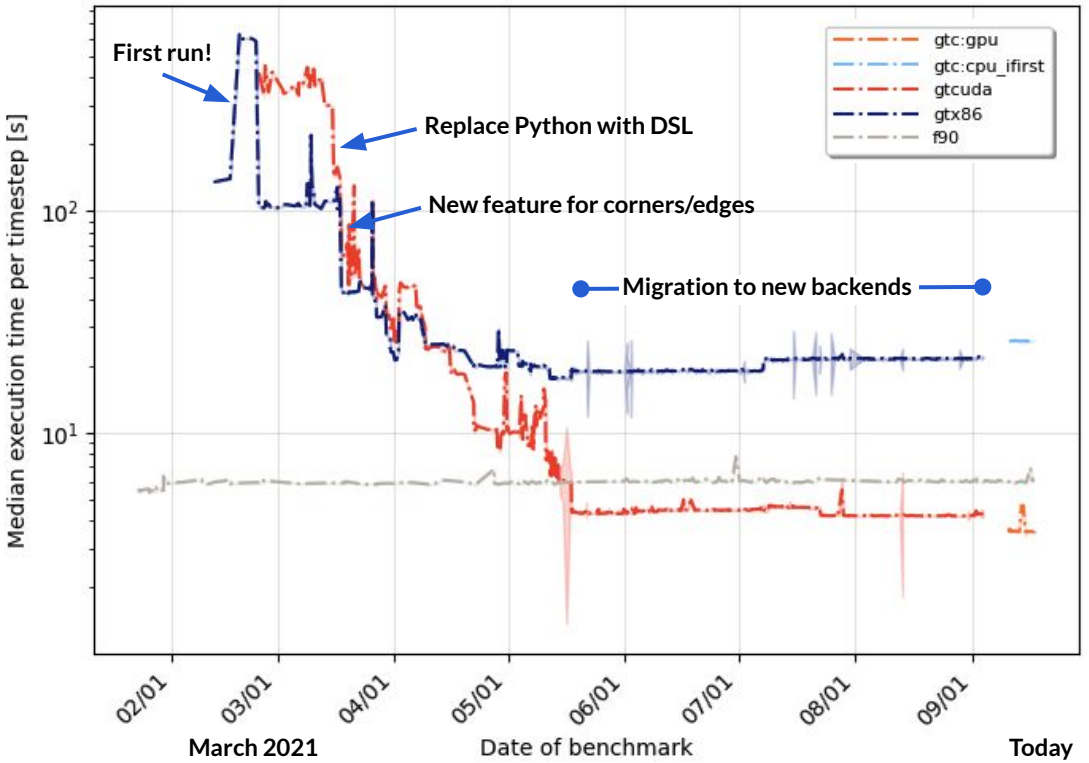
Performance: Dynamical Core

Validate, refactor, make it fast!

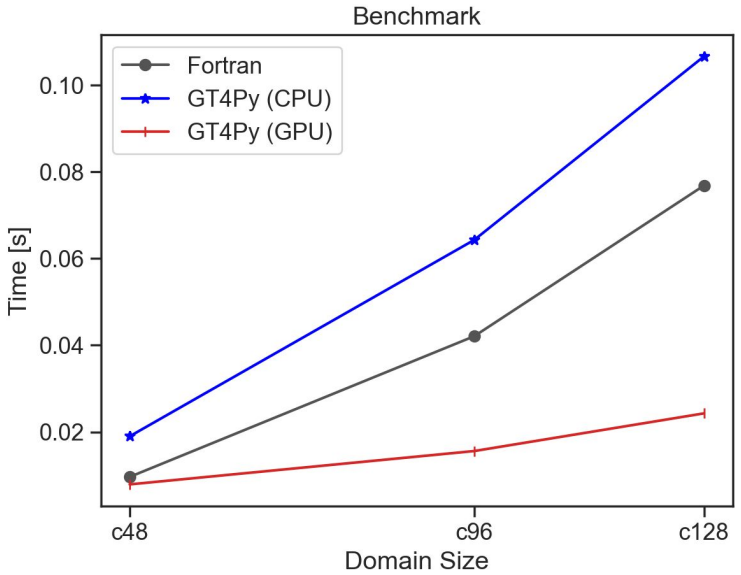
Main modifications

- Remove Python code
- Use new GT4py features to merge stencils
- Temporaries
- Asynchronous execution

Work in progress… Currently 2.8x slower on CPU and 1.6x faster on GPU as compared to Fortran reference



Performance: Microphysics



Domain size: 128 x 128 x 79

	Runtime [s]	Speedup
Fortran	0.077	REF
GT4Py (CPU)	0.107	0.72
GT4Py (GPU)	0.024	3.2

System: Piz Daint, CSCS
CPU: Intel Xeon E2690 v3 @ 2.6 GHz, 12 core
GPU: NVIDIA Tesla P100

Summary

Due to AI/ML, Python is rapidly growing in the HPC space

No turn-key solutions to achieve productivity, performance, and portability for weather and climate models

Domain-specific language in Python can…

- Bridge the semantic gap between domain scientists and existing portability frameworks (e.g. C++)
- Enable more aggressive, high-level optimizations otherwise not possible
- Enable efficient collaboration between domain science, computational science and computer science
- Enable new workflows inter

Thank you!
Questions?