GT4Py: A Python Framework for the Development of High-Performance Weather and Climate Applications

M. Bianco[1], T. Ehrengruber[1], N. Farabullini[2], A. Gopal[2], L. Groner[1], R. Häuselmann[1], P. Kardos[2], S. Kellerhals[2], M. Luz[2], C. Müller[3], E. G. Paredes[1] , M. Roethlin[3] , F. Thaler[1], H. Vogt[1], B. Weber[3], T. Schulthess[1,4]

[1] Swiss National Supercomputing Center, CSCS [2] Institute for Atmospheric and Climate Science, ETH Zurich
[3] Federal Office of Meteorology and Climatology, MeteoSwiss [4] Institute for Theoretical Physics, ETH Zurich

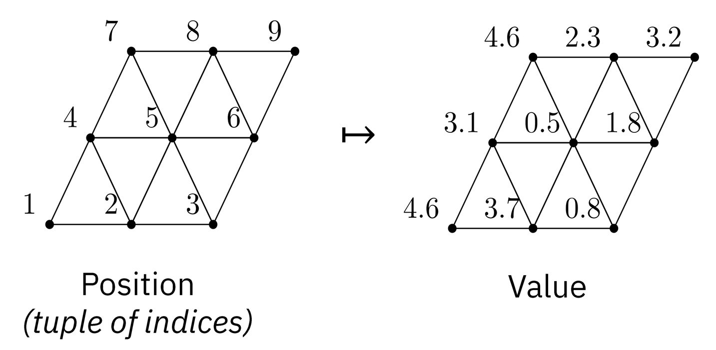
Introduction

GT4Py is a Python framework for weather and climate applications simplifying the development and maintenance of high-performance codes in prototyping and production environments. GT4Py separates model development from hardware architecture dependent optimizations, instead of intermixing both together in source code, as regularly done in lower-level languages like Fortran, C, or C++.

Domain scientists focus solely on numerical modeling using a declarative embedded domain specific language (DSL) supporting common computational patterns of dynamical cores and physical parametrizations. An optimizing toolchain then transforms this high-level representation into a finely-tuned implementation for the target hardware architecture. This separation of concerns allows performance engineers to implement new optimizations or support new hardware architectures without requiring changes to the application, increasing productivity for domain scientists and performance engineers alike.

Fields

Inspired by the concept of a field in physics the central datastructure used in GT4Py is a `Field`. A field maps a position in the form of a tuple of indices to a value or composite, e.g. tuple, thereof.

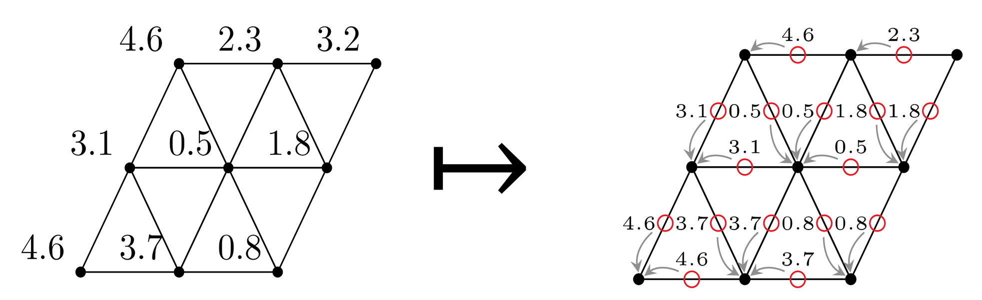


`Field[[Vertex], float]`

Remap operation

Aside from regular arithmetic and trigonometric operations, fields can be remapped in order to obtain a new field defined on a different domain of neighboring positions (e.g. from Vertices to Edges). GT4Py is mesh agnostic allowing domain scientists to use existing infrastructure and libraries (e.g. ATLAS, ICON) to generate meshes.

`vertex_field(E2V[0])`



Neighbor reductions

In case of a variable number of neighboring positions, a set of neighbor reductions (e.g. sum, maximum, minimum) can be used.

`neighbor_sum(flux(V2E), axis=V2EDim)`

Programs & Operators

Program (`@program`)

A program is a sequence of (stateful) operator calls transforming the input arguments and writing back the return value to a specified output field.

```
@program(backend=...)
def program1(inp1: AnyField, out1: AnyField, out2: AnyField):
  operator1(inp1, out=out1)
  operator2(inp1, out=out2)
  ...
```

By selecting a different backend users can switch to a different hardware architecture (e.g. GPUs) with the change of a single line.

Field operator (`@field_operator`)

Covering most patterns of explicit finite-difference and finite-volume discretizations multiple field operations can be grouped together into a field operator.

```
@field_operator
def edge_average(vertex_field: Field[[Vertex], float])
    -> Field[[Edge], float]:
  return 0.5*(vertex_field(E2V[0])+vertex_field(E2V[1]))
```

Field operators are composable, allowing the description of high-level operators from basic building blocks.

```
@field_operator
def laplap(u: Field[[I, J], float]) -> Field[[I, J], float]:
  return lap(lap(u))
```

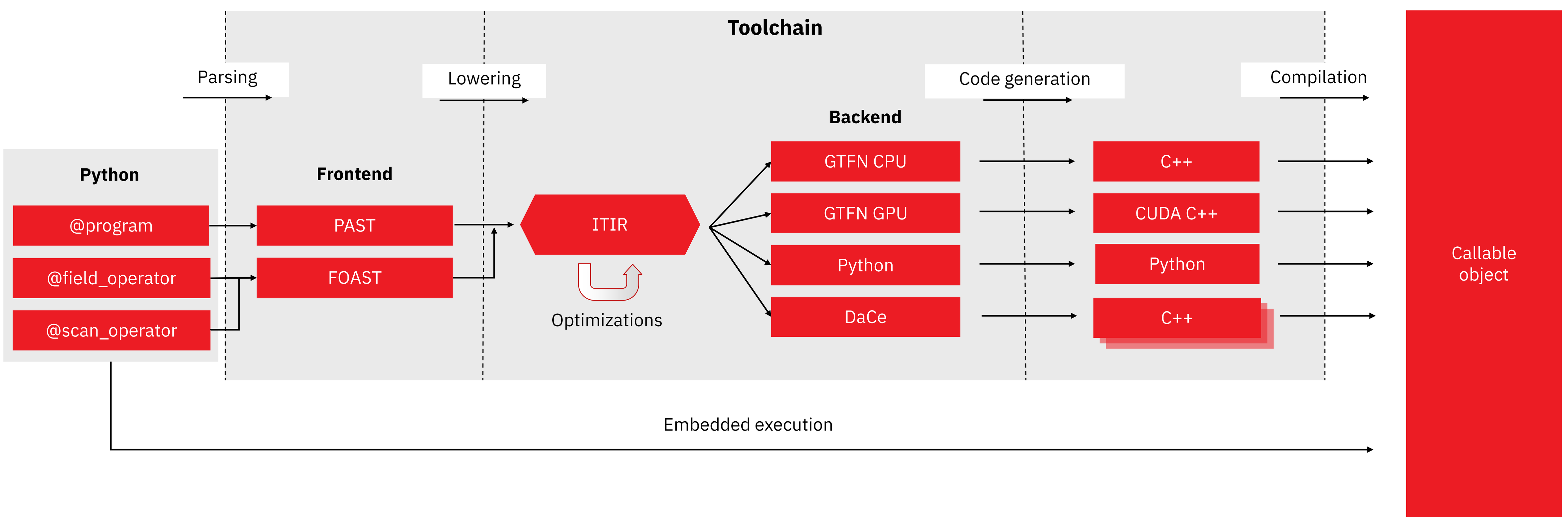
Scan operator (`@scan_operator`)

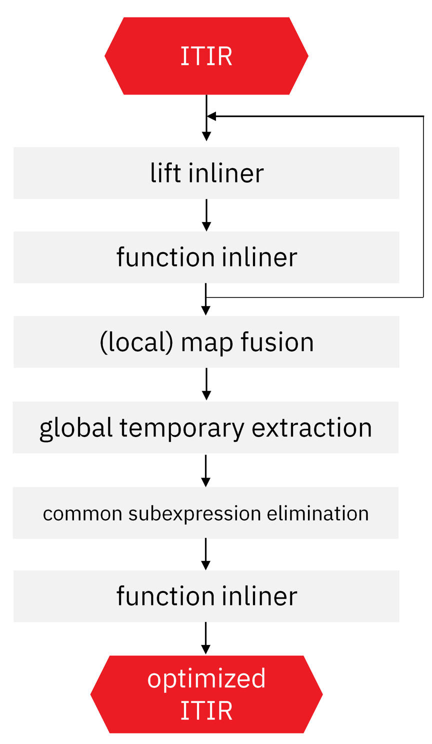
Scan operators are useful for expressing computations with dependencies across an entire dimension, which commonly occur in implicit solvers and physical parametrizations. The output from the previous level (i.e., `k+1` or `k-1`, depending on the direction) is used by a scalar function to derive a new value for the current grid point, iteratively building up a complete field.

```
@scan_operator(axis=KDim, forward=True, init=0.0)
def simple_scan_operator(
    carry: float, current_value: float
) -> float:
    return carry + current_value

simple_scan_operator(inp_field, out=out)
```

Toolchain





Example - Upwind advection scheme

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \text{ on } \Omega \qquad \text{(Advection equation)}$$

```
@field_operator
def upstream_flux(
    rho: Field[[Vertex], float],
    vel: tuple[Field[[Edge], float], Field[[Edge], float]],
    dual_face_normal: tuple[Field[[Edge], float], Field[[Edge], float]],
    dual_face_length: Field[[Edge], float]
) -> Field[[Edge], float]:
    normal_velocity = vel[0] * dual_face_normal[0] \
                    + vel[1] * dual_face_normal[1]
    return where(normal_velocity > 0.0, rho(E2V[0]), rho(E2V[1])) \
           * normal_velocity * dual_face_length
```

```
@field_operator
def advection_scheme_upwind(
    rho: Field[[Vertex], float],
    dt: float,
    vel: tuple[Field[[Vertex], float], Field[[Vertex], float]],
    vol: Field[[Vertex], float],
    dual_face_orientation: Field[[Vertex, V2EDim], float],
    dual_face_normal: tuple[Field[[Edge], float], Field[[Edge], float]],
    dual_face_length: Field[[Edge], float]
) -> Field[[Vertex], float]:
    flux = upwind_flux(rho, vel, dual_face_normal, dual_face_length)
    return rho - (dt / vol) * neighbor_sum(
        flux(V2E) * dual_face_orientation, axis=V2EDim)
```

Projects using GT4Py

- **ECMWF** develops the non-hydrostatic **FVM** dynamical core using GT4Py. A new high-performance distributed model on Cartesian grids and an LES configuration are already implemented and available for research in the PASC project KILOS (cf. poster Ubbiali et al. and Krieger et al.). The global model operating on the quasi-uniform ECMWF octahedral grid is currently under development with the declarative GT4Py.
- The **EXCLAIM** project is developing an exascale computing and data platform for weather and climate modelling based on the ICOsahedral Nonhydrostatic Model (**ICON**) system. The second version of GT4Py is used to replace the currently Fortran-based model components (cf. poster Müller et al.).

References

- Afanasyev et al. (2021). *GridTools: A framework for portable weather and climate applications*. SoftwareX, 15.
- Ben-Nun, T., et al. (2022). *Productive performance engineering for weather and climate modeling with Python*. SC '22: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.
- GT4Py - GridTools for Python: https://github.com/GridTools/gt4py
- DaCe - Data-Centric Parallel Programming: https://github.com/spcl/dace