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

M. Bianco1, T. Ehrengruber2, N. Farabullini2, A. Gopal2, L. Groner1, R. Häuselmann1, P. Kardos2, S. Kellerhals2, M. Luz2, C. Müller2, E. G. Paredes1, M. Roethlin3, F. Thaler3, H. Vogt1, B. Weber3, T. Schultess1,4

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 parameterizations. An optimizing toolchain then transforms this high-level representation into a fine-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, out1):
    operator1(inp1, out1)
    operator2(inp1, 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 operators 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.

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 parameterizations. The output from the previous level (i.e., k-1 or k-2, 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=EDim, forward=True, init=0.0)
def simple_scan_operator(src:Field, carry:float, current_value:float) -> float:
    return current_value + src

Toolchain

Example - Upwind advection scheme

\[
\frac{\partial \phi}{\partial t} + \nabla \cdot (\rho \mathbf{v} \phi) = 0 \quad \text{on } \Omega \\
\]

(Advection equation)

@field_operator

def advection_scheme_upwind:
    rho_field([Vertex], float),
    dx_field: Field([Edge], float),
    u: Field([Vertex], float),
    v: Field([Vertex], float),
    vol: Field([Vertex], float),
    dual_face_orientation: Field([Vertex, V2EDim], float),
    dual_face_length: Field([Edge], float),
    normal_velocities: tuple([Field([Edge], float), Field([Vertex, V2EDim], float)])
    -> Field([Vertex], float):

        flux = upwind_flux(rho_field, vel, dual_face_normal, dual_face_length)
    return rho_field - flux / vol

Projects using GT4Py

- ECMWF develops the hydrostatic FVM dynamics core using GT4Py. A new high-performance distributed model on Cartesian grids and an LES configuration are already implemented and run on the Swiss supercomputer resources (CSCS) as well as own infrastructure (UBC, Norway) and the agnostic. The global model operating on the政法form (ECMWF) ecosystem grid is currently under development with the declarative GT4Py.

- The EXLASM project in developing an exascale computing and data platform for weather and climate modeling based on the ICESim data abstraction/Model (ICOM) system. The second version of GT4Py is used to replace the current Fortran-based model components (test poster: Rüegg et al.).

References