CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

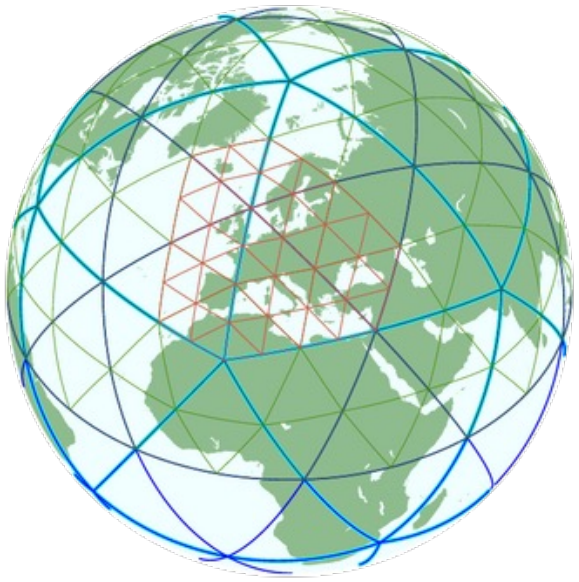
Performance portability in weather and climate HPC applications

SOS26 Workshop
Edoardo Paone, CSCS
March 12, 2024

Challenges in W&C application design

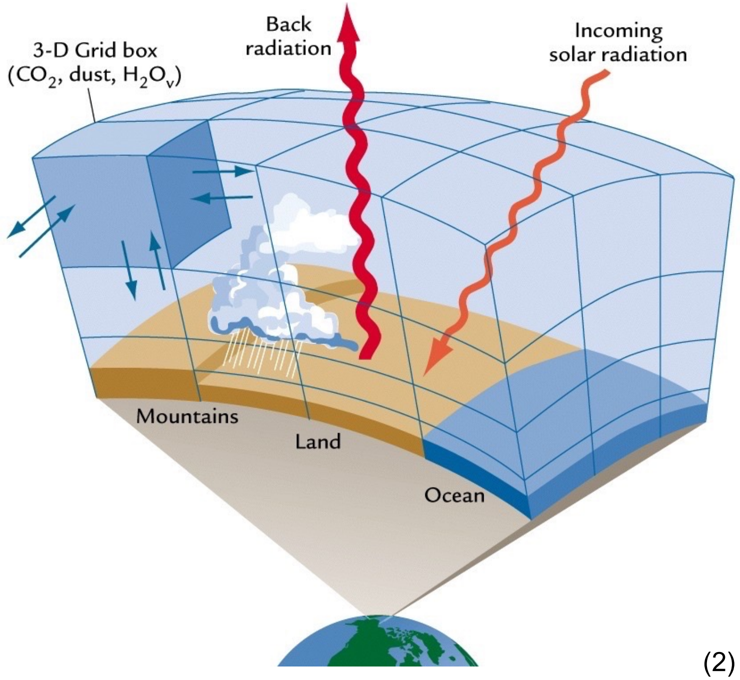
A multi-domain software stack

The data structure



(1)

The physics computation



(2)

SW optimization & deployment

(1) Jungclaus, J. & Lorenz, Stephan & Schmidt, H. & Brovkin, V. & Brüggemann, N. & Chegini, Fatemeh & Crueger, Traute & De-Vrese, P. & Gayler, V. & Giorgetta, Marco & Gutjahr, Oliver & Haak, Helmuth & Hagemann, Stefan & Hanke, M. & Ilyina, Tatiana & Korn, Peter & Kröger, Jürgen & Linardakis, L. & Mehlmann, C. & Claussen, M.. (2022). The ICON Earth System Model Version 1.0. Journal of Advances in Modeling Earth Systems. 14. 10.1029/2021MS002813.

(2) Wild, M. Introduction into parameterizations and parameterization of the planetary boundary layer. Institute for Atmospheric and Climate Science, ETH Zurich.

The ICON model

- ICON is a modelling framework for numerical weather and climate prediction.
  - Released as open-source code in January 2024.

```
!$OMP PARALLEL PRIVATE (rl_start,rl_end,i_startblk,i_endblk,jk_start)

!$OMP DO PRIVATE(jb,i_startidx,i_endidx,jk,jc,z_w_expl,z_contr_w_fl_l,z_contr_wq_fl_l,z_rho_expl,z_exner_expl, &
!$OMP   z_a,z_b,z_c,z_g,z_q,z_alpha,z_beta,z_beta_q,z_gamma,ic,z_flxdiv_mass,z_flxdiv_theta,z_flxdiv_vol  ) ICON_OMP_DEFAULT_SCHEDULE
    DO jb = i_startblk, i_endblk
        !$ACC PARALLEL IF(i_am_accel_node) DEFAULT(PRESENT) ASYNC(1)
        !$ACC LOOP GANG VECTOR COLLAPSE(2)
#ifdef __LOOP_EXCHANGE
        DO jc = i_startidx, i_endidx
!DIR$ IVDEP, PREFERVECTOR
          DO jk = 1, nlev
#else
!$NEC outerloop_unroll(8)
        DO jk = 1, nlev
          DO jc = i_startidx, i_endidx
#endif
            z_flxdiv_mass(jc,jk) =  p_nh%metrics%deepatmo_divh_mc(jk) * (                   &
              p_nh%diag%mass_fl_e(ieidx(jc,jb,1),jk,ieblk(jc,jb,1)) * p_int%geofac_div(jc,1,jb) + &
              p_nh%diag%mass_fl_e(ieidx(jc,jb,2),jk,ieblk(jc,jb,2)) * p_int%geofac_div(jc,2,jb) + &
              p_nh%diag%mass_fl_e(ieidx(jc,jb,3),jk,ieblk(jc,jb,3)) * p_int%geofac_div(jc,3,jb) )

            z_flxdiv_theta(jc,jk) = p_nh%metrics%deepatmo_divh_mc(jk) * (              &
              z_theta_v_fl_e(ieidx(jc,jb,1),jk,ieblk(jc,jb,1)) * p_int%geofac_div(jc,1,jb) + &
              z_theta_v_fl_e(ieidx(jc,jb,2),jk,ieblk(jc,jb,2)) * p_int%geofac_div(jc,2,jb) + &
              z_theta_v_fl_e(ieidx(jc,jb,3),jk,ieblk(jc,jb,3)) * p_int%geofac_div(jc,3,jb) )
          END DO
        END DO
        !$ACC END PARALLEL
    ENDDO
!$OMP END DO

!$OMP END PARALLEL
```

application code and optimization code inter-mixed in one place

Challenges for code maintainability and performance portability

- Imperative programming languages describe *how* computation is done, rather than *what* it does.
- Compiler directives and macros reduce code readability.
- Optimization code grows with the number of different target architectures.

The XCLAIM project

- Aims at developing a computing platform based on ICON model that is capable of running kilometer-scale climate simulations.
- Rewrites the ICON model using GT4Py, a declarative domain specific language (DSL) supporting the computational patterns of W&C applications.

```fortran
outerloop_unroll(8)
DO jk = 1, nlev
  DO jc = i_startidx, i_endidx

   z_flxdiv_mass(jc,jk) =  p_nh%metrics%deepatmo_divh_mc(jk) * (                 &
     p_nh%diag%mass_fl_e(ieidx(jc,jb,1),jk,ieblk(jc,jb,1)) * p_int%geofac_div(jc,1,jb) + &
     p_nh%diag%mass_fl_e(ieidx(jc,jb,2),jk,ieblk(jc,jb,2)) * p_int%geofac_div(jc,2,jb) + &
     p_nh%diag%mass_fl_e(ieidx(jc,jb,3),jk,ieblk(jc,jb,3)) * p_int%geofac_div(jc,3,jb) )

   z_flxdiv_theta(jc,jk) = p_nh%metrics%deepatmo_divh_mc(jk) * (               &
     z_theta_v_fl_e(ieidx(jc,jb,1),jk,ieblk(jc,jb,1)) * p_int%geofac_div(jc,1,jb) + &
     z_theta_v_fl_e(ieidx(jc,jb,2),jk,ieblk(jc,jb,2)) * p_int%geofac_div(jc,2,jb) + &
     z_theta_v_fl_e(ieidx(jc,jb,3),jk,ieblk(jc,jb,3)) * p_int%geofac_div(jc,3,jb) )
  END DO
END DO
!$ACC END PARALLEL
```

← ICON Fortran

GT4Py DSL →

```python
@field_operator
def _compute_divergence_of_fluxes_of_rho_and_theta(
    geofac_div: Field[[CEDim], wpfloat],
    mass_fl_e: Field[[EdgeDim, KDim], wpfloat],
    z_theta_v_fl_e: Field[[EdgeDim, KDim], wpfloat],
) -> tuple[Field[[CellDim, KDim], vpfloat], Field[[CellDim, KDim], vpfloat]]:

    z_flxdiv_mass_wp = neighbor_sum(geofac_div(C2CE) * mass_fl_e(C2E), axis=C2EDim)
    z_flxdiv_theta_wp = neighbor_sum(geofac_div(C2CE) * z_theta_v_fl_e(C2E), axis=C2EDim)
    return astype((z_flxdiv_mass_wp, z_flxdiv_theta_wp), vpfloat)

@program(backend=dace_gpu, grid_type=GridType.UNSTRUCTURED)
def compute_divergence_of_fluxes_of_rho_and_theta():
    _compute_divergence_of_fluxes_of_rho_and_theta(
        geofac_div,
        mass_fl_e,
        z_theta_v_fl_e,
        out=(z_flxdiv_mass, z_flxdiv_theta),
        domain={
            CellDim: (i_startidx, i_endidx),
            KDim: (1, nlev),
        },
    )
```

performance portable, HW-agnostic user code

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

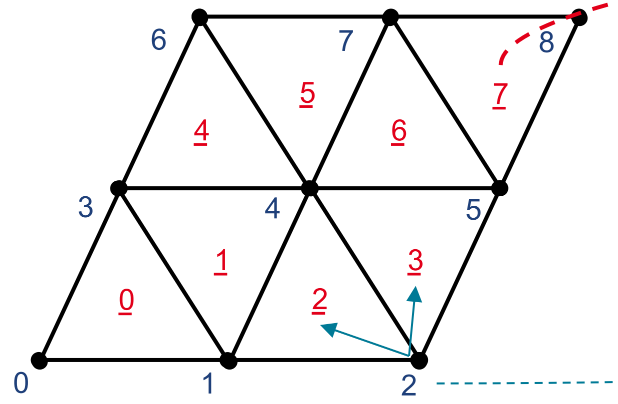
ETH zürich

GT4Py

GridTools 4python

GT4Py programming model

- The GT4Py DSL provides a data structure (called `Field`) and a set of operators to express stencil-based computations.
  - A field maps a position in the form of tuple of indices to a value.




```
c: Field[[Cell], float] = [c(0), c(1), ..., c(7)]
v: Field[[Vertex], float] = [v(0), v(1), ..., v(8)]

V2C: Connectivity[origin=Vertex, neighbor=Cell] =
        [[0], [0, 1, 2], [2, 3], ..., [7]]

fv: Field[[Vertex], float] =
        neighbor_sum(c(V2C), axis=V2CDim)
```

GT4Py programming model

- The GT4Py DSL provides a data structure (called `Field`) and a set of operators to express stencil-based computations.
  - A field maps a position in the form of tuple of indices to a value.
  - Field operators cover most patterns of explicit finite-difference and finite-volume discretization and are composable.

```
@field_operator
def lap(u: Field[[I, J], float]) -> Field[[I, J], float]:
    return (-4.0 * u + u(I[1]) + u(J[1]) + u(I[-1]) + u(J[-1]))

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

GT4Py programming model

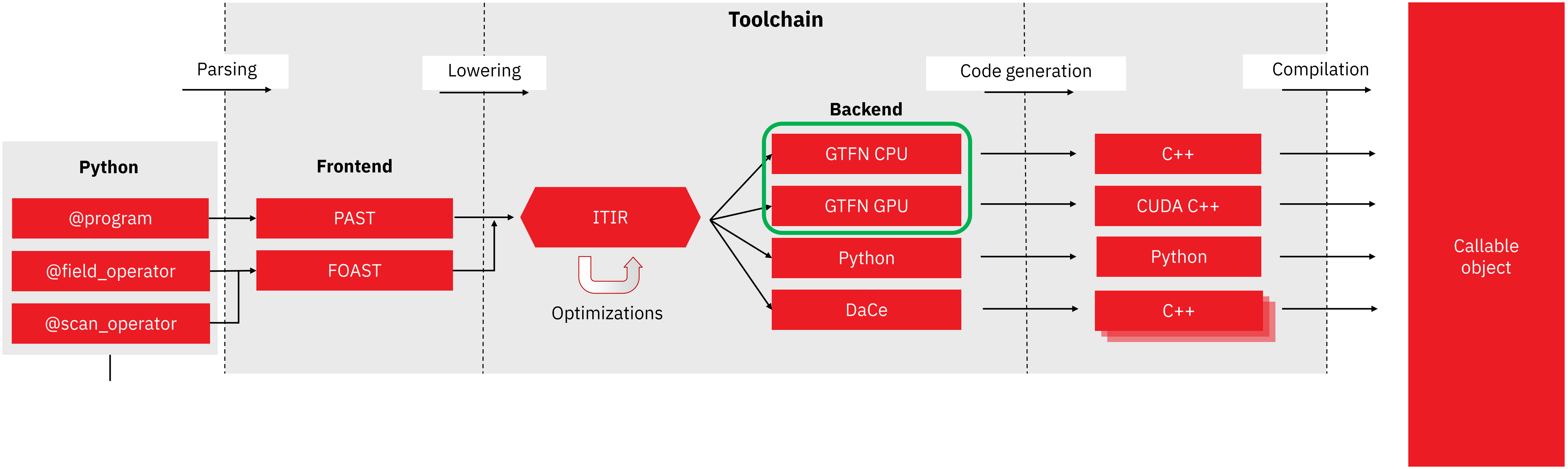
- The GT4Py DSL provides a data structure (called `Field`) and a set of operators to express stencil-based computations.
  - A field maps a position in the form of tuple of indices to a value.
  - Field operators cover most patterns of explicit finite-difference and finite-volume discretization and are composable.
  - A program is a sequence of (stateful) operator calls transforming the input arguments and writing the result to the specified output field.

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

GT4Py programming model

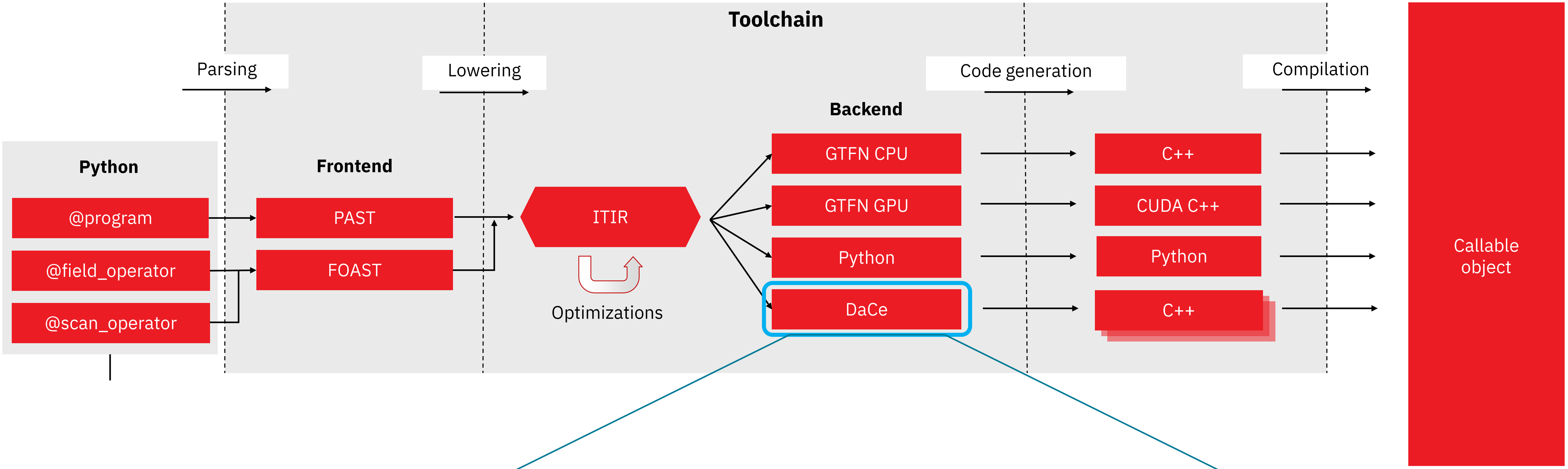
- A declarative language, that describes *what* the computation does, not *how* to execute it.
- Makes the highly-productive Python ecosystem available to domain scientists.
- By selecting a different backend users can switch to a different hardware architecture (e.g GPUs) with the change of a single line.

GT4Py toolchain

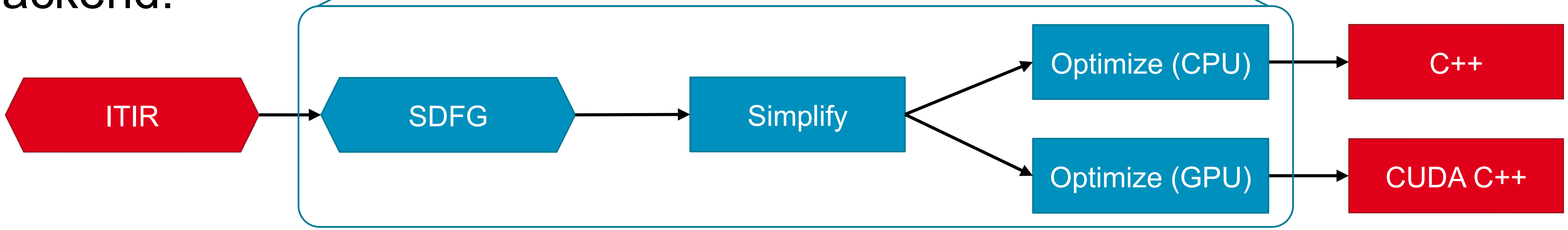


- GTFN backends rely on GridTools:
  - Optimizations on the iterator intermediate representation (ITIR) exploit the semantics of field operators to improve the schedule of operations in the stencil program.

GT4Py+DaCe toolchain



- DaCe backend:



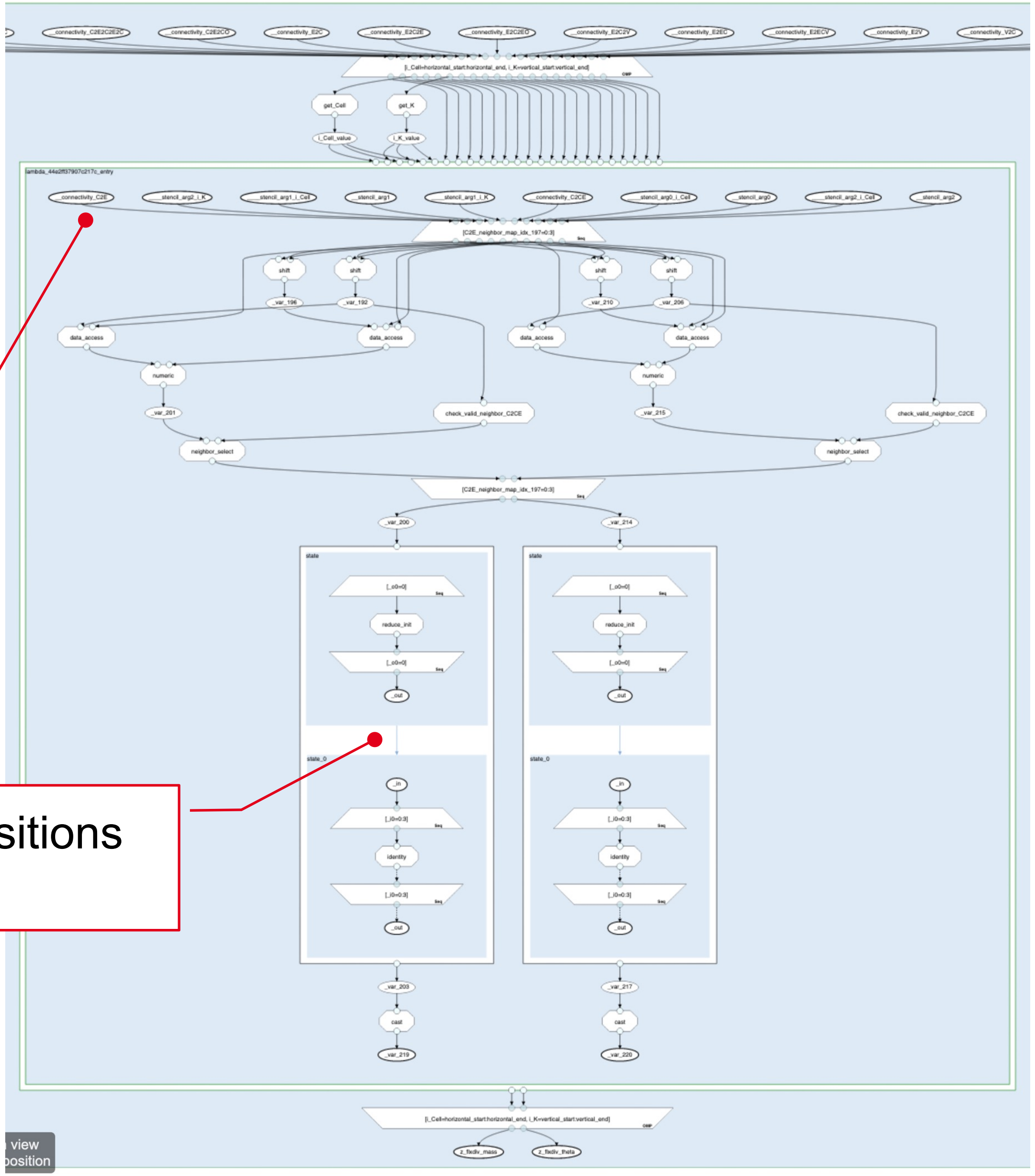
DaCe

Stateful Dataflow multiGraphs (SDFG)

- An SDFG is a hierarchical state machine of acyclic dataflow multigraphs.

explicit data movements

state transitions

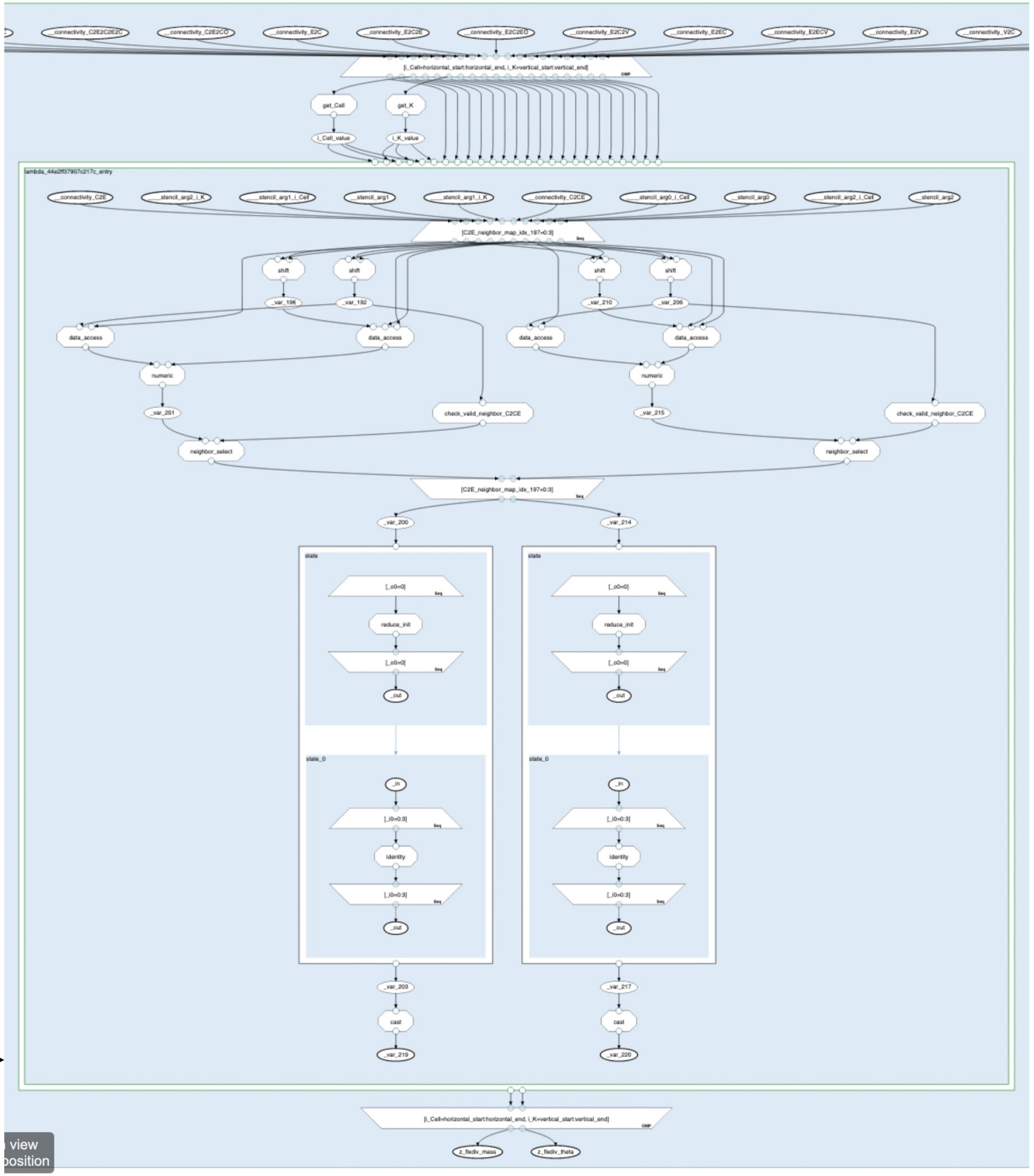


Stateful Dataflow multiGraphs (SDFG)

- An SDFG is a hierarchical state machine of acyclic dataflow multigraphs.
- Create the SDFG programmatically from ITIR using the SDFG builder API.

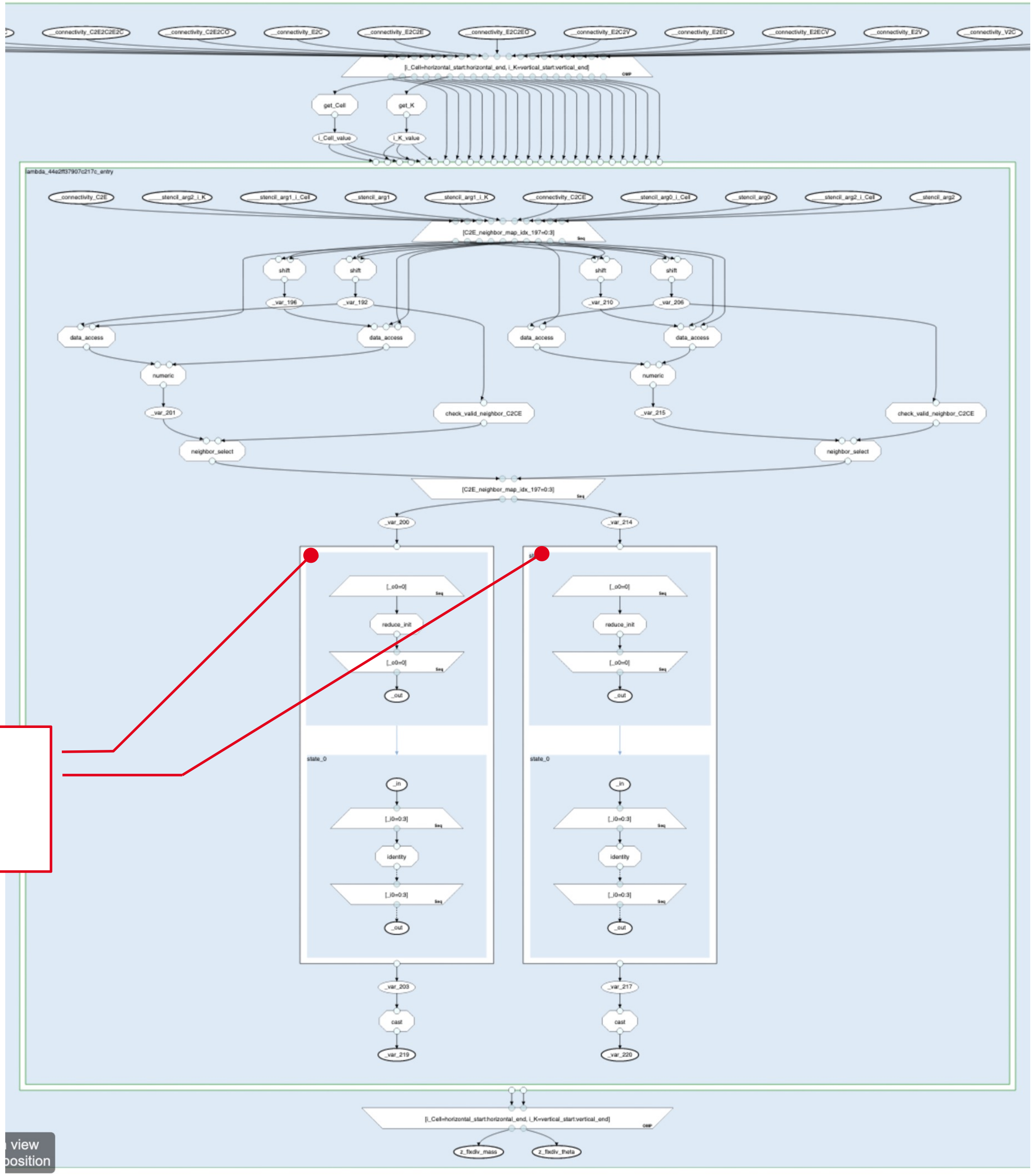
```
@field_operator
def _compute_divergence_of_fluxes_of_rho_and_theta(
    geofac_div: Field[[CEDim], wpfloat],
    mass_fl_e: Field[[EdgeDim, KDim], wpfloat],
    z_theta_v_fl_e: Field[[EdgeDim, KDim], wpfloat],
) -> tuple[Field[[CellDim, KDim], vpfloat], Field[[CellDim, KDim], vpfloat]]:

    z_flxdiv_mass_wp = neighbor_sum(geofac_div(C2CE) * mass_fl_e(C2E), axis=C2EDim)
    z_flxdiv_theta_wp = neighbor_sum(geofac_div(C2CE) * z_theta_v_fl_e(C2E), axis=C2EDim)
    return astype((z_flxdiv_mass_wp, z_flxdiv_theta_wp), vpfloat)
```



Stateful Dataflow multiGraphs (SDFG)

- An SDFG is a hierarchical state machine of acyclic dataflow multigraphs.
- Create the SDFG programmatically from ITIR using the SDFG builder API.
- The DaCe frontend allows to parse and include external stencil-SDFGs.

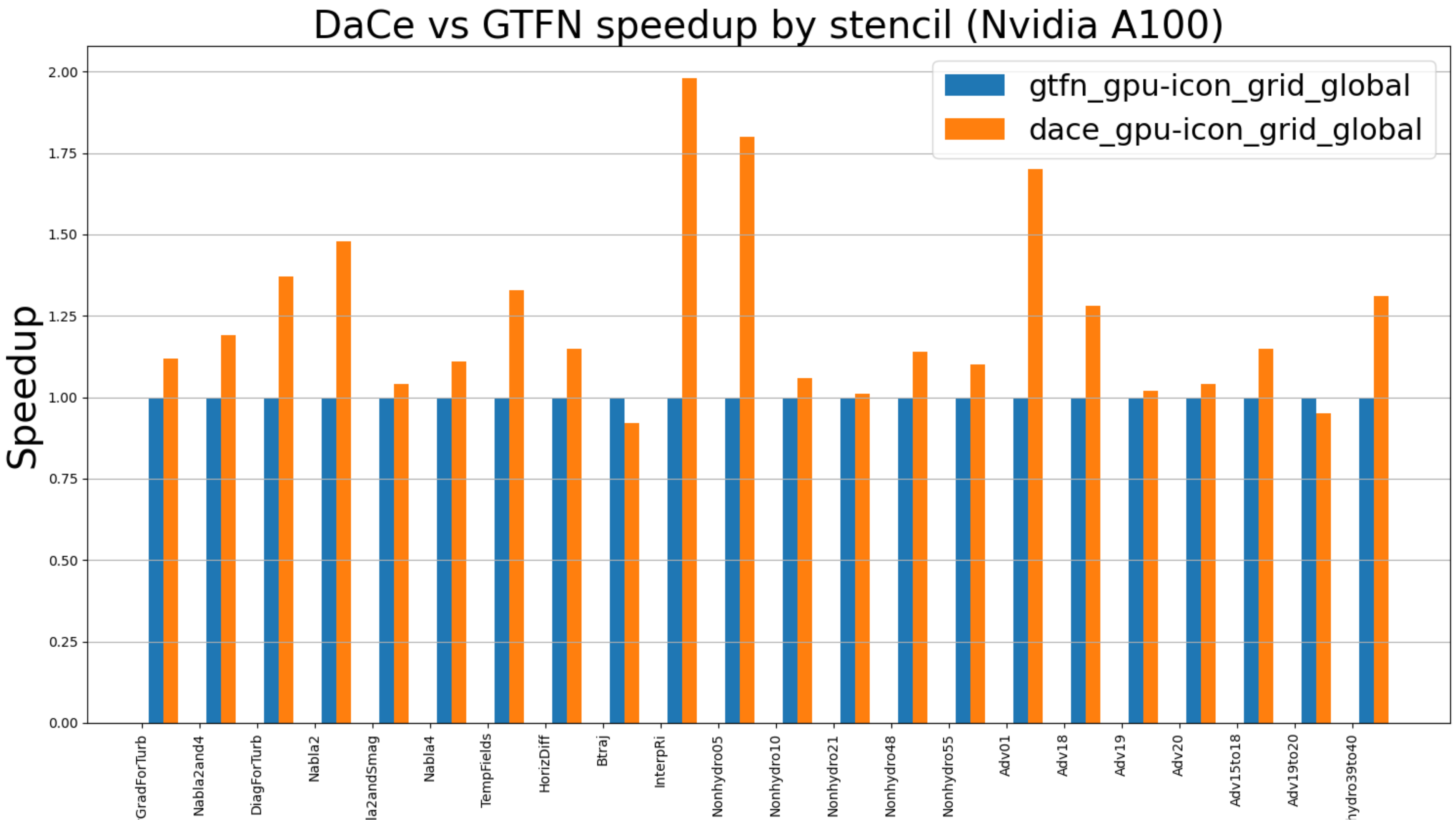


nested SDFGs

DaCe optimization

- Different means for SDFG optimization:
  - Graph rewriting transformations that preserve the SDFG semantics.
  - Local and global auto-tuning (e.g. data layout, map permutation, map tiling, map fusion).

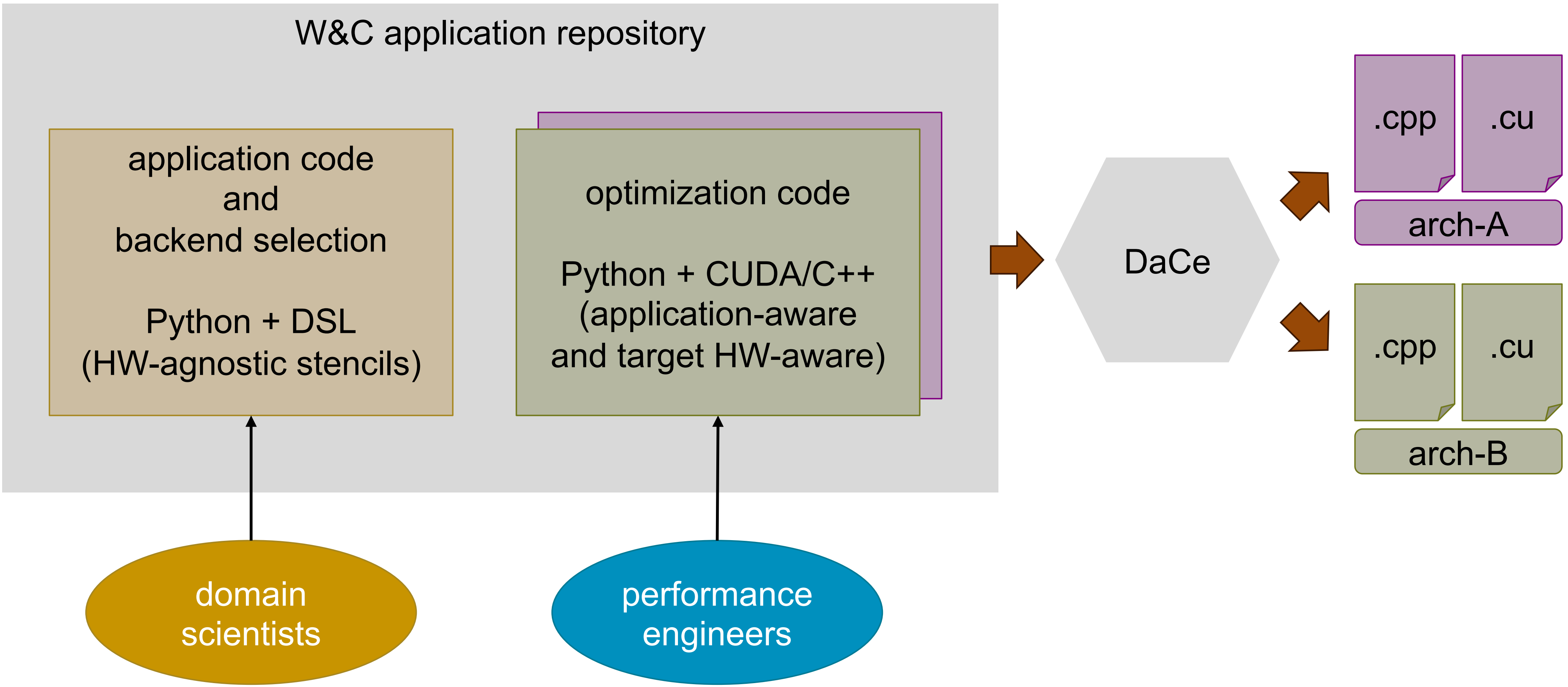
CUDA code generation



CSCS

ETH zürich

Packaging application code and optimization code



Conclusions

Performance portability in W&C HPC applications

GT4Py ➕ DaCe

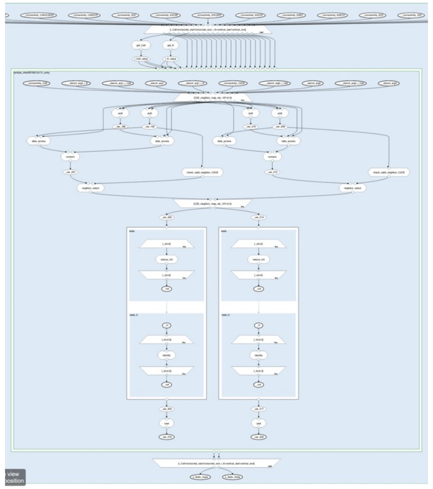
- Improve productivity of domain scientists by providing a DSL + the Python SW ecosystem.
- Improve code portability and maintainability by separating application code from optimization code.

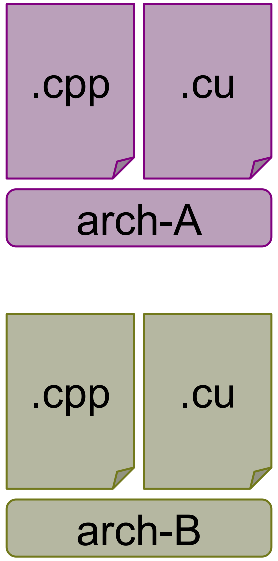
- Provide additional optimization opportunities by means of data-centric analysis.
- Produce C++/CUDA source code optimized for the target architecture.

```
@field_operator
def _compute_divergence_of_fluxes_of_rho_and_theta(
    geofac_div: Field[[CEDim], wpfloat],
    mass_fl_e: Field[[EdgeDim, KDim], wpfloat],
    z_theta_v_fl_e: Field[[EdgeDim, KDim], wpfloat],
) -> tuple[Field[[CellDim, KDim], vpfloat], Field[[CellDim, KDim], vpfloat]]:

    z_flxdiv_mass_wp = neighbor_sum(geofac_div(C2CE) * mass_fl_e(C2E), axis=C2EDim)
    z_flxdiv_theta_wp = neighbor_sum(geofac_div(C2CE) * z_theta_v_fl_e(C2E), axis=C2EDim)
    return astype((z_flxdiv_mass_wp, z_flxdiv_theta_wp), vpfloat)


@program(backend=dace_gpu, grid_type=GridType.UNSTRUCTURED)
def compute_divergence_of_fluxes_of_rho_and_theta():
    _compute_divergence_of_fluxes_of_rho_and_theta(
        geofac_div,
        mass_fl_e,
        z_theta_v_fl_e,
        out=(z_flxdiv_mass, z_flxdiv_theta),
        domain={
            CellDim: (i_startidx, i_endidx),
            KDim: (1, nlev),
        },
    )
```





Thank you for your attention.

EXCLAIM

GridTools 4python

SPCL