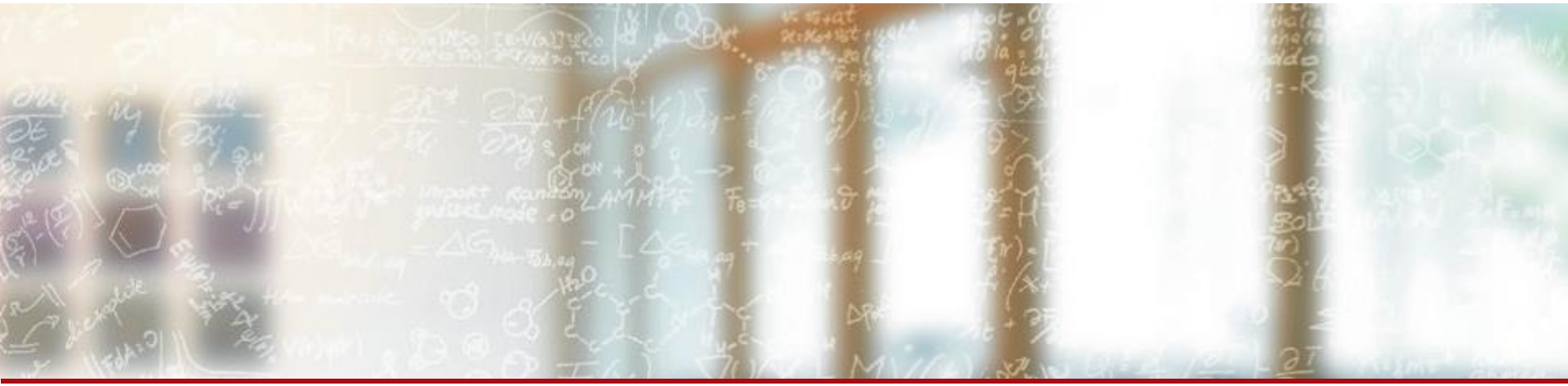
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Domain-specific Libraries for Weather and Climate

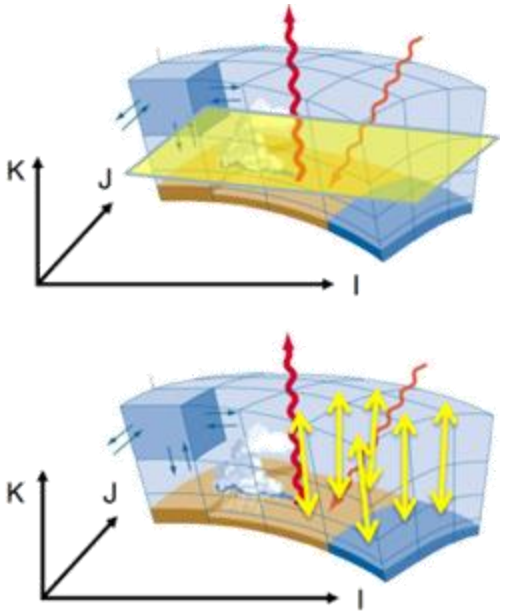
CSCS User Lab Day

Lukas Mosimann, CSCS

September 09, 2019

Weather and Climate Applications

- Application properties
  - Numerical methods
    - Explicit methods in the horizontal
    - Implicit methods in the vertical
  - Usually many inputs for one output
  - Mostly memory-bound
  - Distributed memory



$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

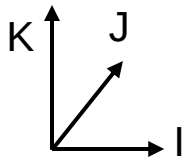
explicitly discretized

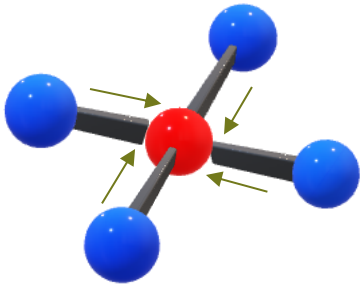
$$u_i^{n+1} = u_i^n + \alpha \frac{\Delta t}{(\Delta x)^2} \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n \right)$$

time step *stencil kernel* *field* *spatial step*

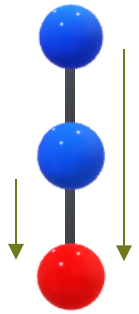
Algorithmic Patterns

- Domain is a cartesian grid
  - **2D / 3D *stencil-like*** computations
  - Embarrassingly parallel on the horizontal plane (*I-J*)
  - Dependencies on the vertical (*K*)
    - Forward, backward or parallel
    - Partial specializations for intervals
  - Halo-updates / boundary conditions





```
for (int i = 0; i < ni; ++i)
  for (int j = 0; j < nj; ++j)
    for (int k = 0; k < nk; ++k)
      out[i][j][k] = -4.0 * in[i][j][k]
        + in[i-1][j][k] + in[i+1][j][k]
        + in[i][j-1][k] + in[i][j+1][k];
```



Horizontal stencil

```
for (int i, j = ...)
  c[i][j][0] /= b[i][j][0]
  for (int k = 1; k < nk; ++k)
    c[i][j][k] /= b[i][j][k]
                  - a[i][j][k] * c[i][j][k-1]
```

Sequential vertical solver

Challenges

- Traditional weather and climate codes:
  - Large monolithic codebases hand-tuned for a specific architecture
  - Compute motifs hidden behind these optimizations
  - Hardware-specific acceleration usually added using pragmas, e.g. OpenMP or OpenACC
- Current hardware trends
  - Different architectures require very specific optimizations

- **Problem:** Lack of *domain* specific libraries to write models in a performance portable way for multiple architectures
- **Solution:** GridTools C++ Libraries

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

GridTools: Libraries for Applications on Grids

GridTools: Libraries for Applications on Grids

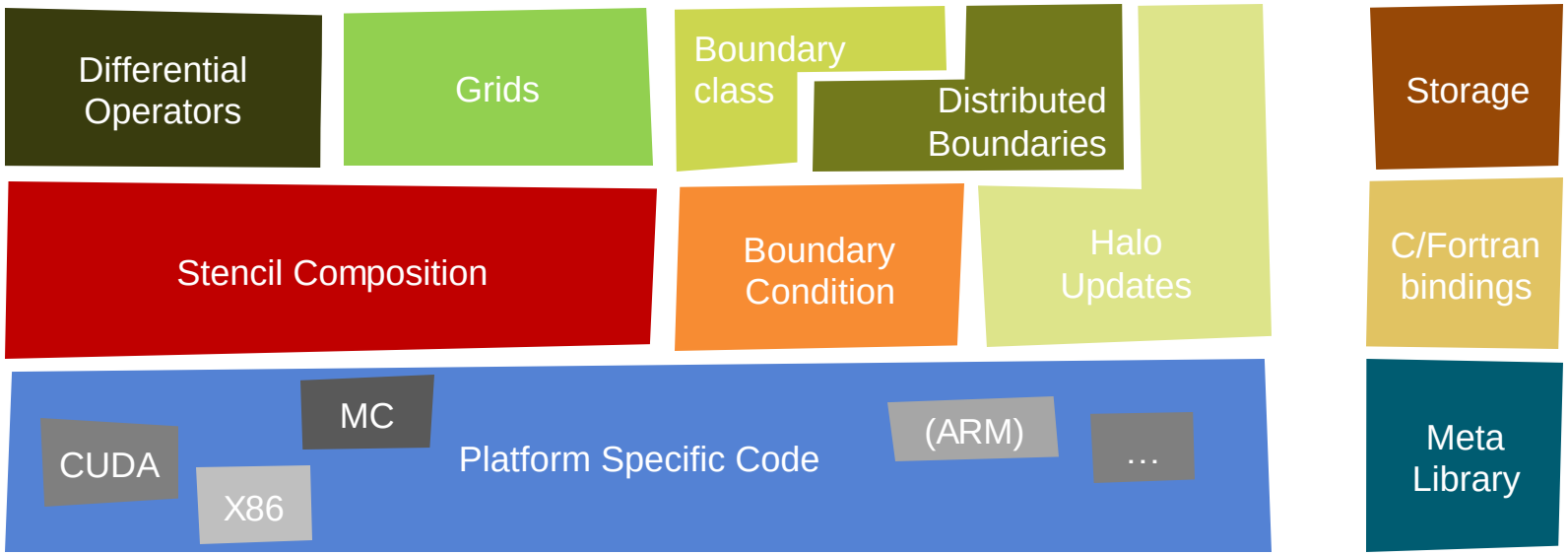
- Open Source C++ project
- Goals
  - Productivity, portability, performance
  - Integration with existing codebases, e.g. COSMO (**Fortran**)

Platform for Advanced Scientific Computing

C2SM Center for Climate Systems Modeling

MeteoSwiss

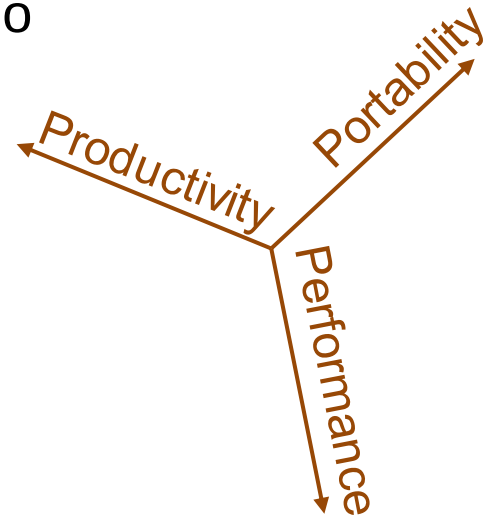
CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre



https://www.github.com/GridTools/gridtools

Declarative Programming in GridTools

- Separation of concerns between interface and implementation
  - Abstract the underlying execution architecture / strategy
  - Provide enough information about **what** to do
  - Let the implementation choose **how** to do it

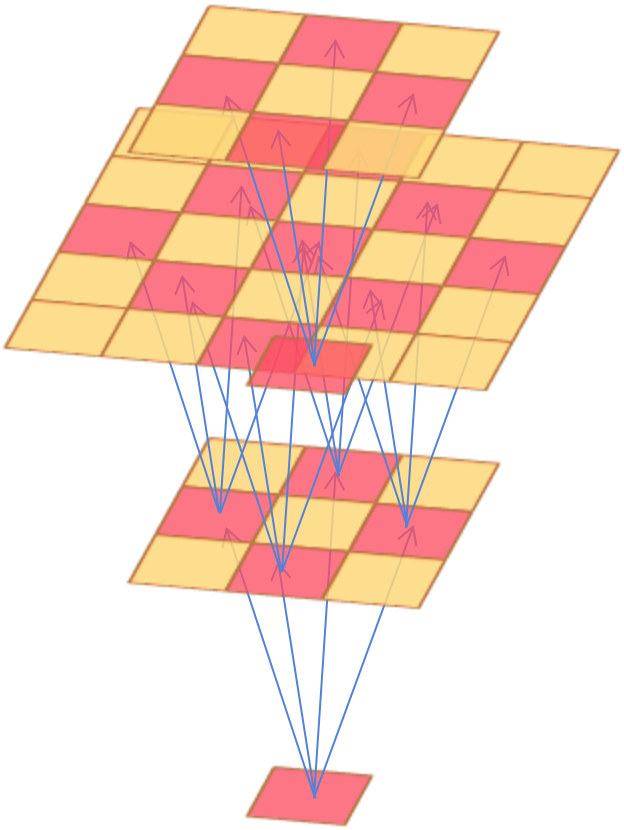


- Storages
  - Layout optimized for access patterns

Stencil Composition in GridTools

- What is a stencil from a programming point of view?
  - Loop control structures
  - Loop body

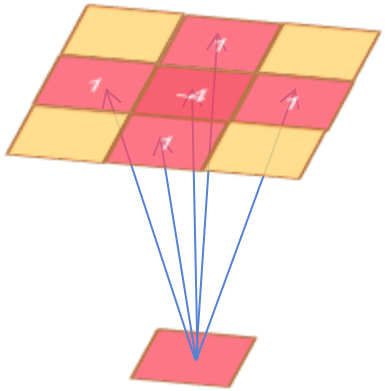
```
for (int i = 0; i < ni; ++i)
  for (int j = 0; j < nj; ++j)
    for (int k = 0; k < nk; ++k)
      out[i][j][k] = -4.0 * in[i][j][k]
        + in[i-1][j][k] + in[i+1][j][k]
        + in[i][j-1][k] + in[i][j+1][k];
```



- How do we achieve good performance for the GridTools execution model?
  - Naïve implementations are not efficient
  - Hardware dependent iterations
  - Loop fusion
    - Difficult to maintain manually
    - Hardware dependent

Stencil Composition Example

- Stencil operator: Laplacian



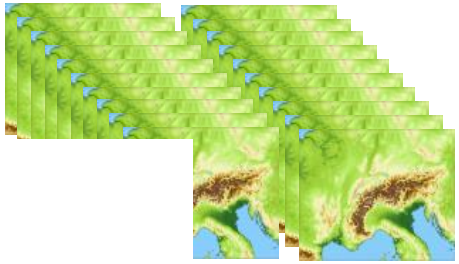

```
struct laplacian {
  using in = in_accessor<0, extent<-1, 1, -1, 1>>;
  using lap = inout_accessor<1>;
  using param_list = make_param_list<in, lap>;

  template <typename Evaluation>
  GT_FUNCTION static void apply(Evaluation const &eval) {
    eval(lap(i, j, k)) = -4. * eval(in(i, j, k))
                         + eval(in(i + 1, j, k))
                         + eval(in(i, j + 1, k))
                         + eval(in(i - 1, j, k))
                         + eval(in(i, j - 1, k));
  }
};
```

- Composing stencils

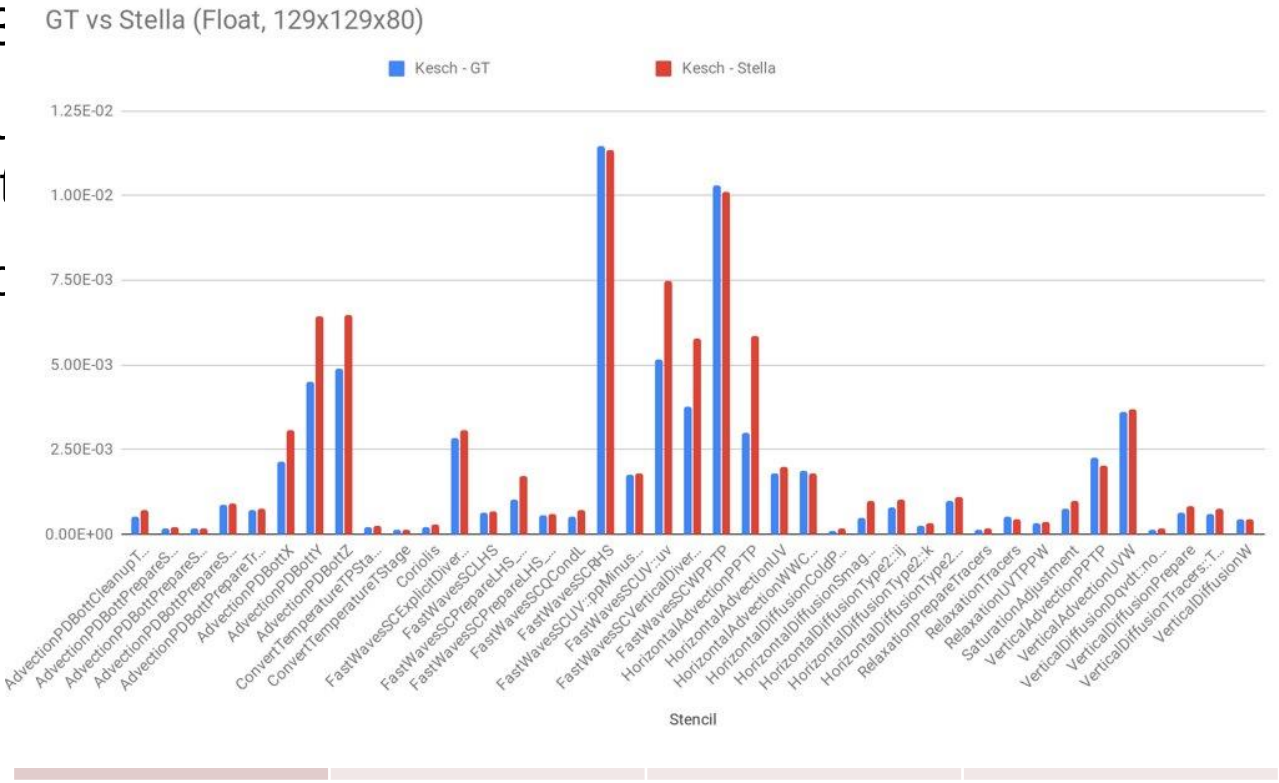
```
using arg_in = arg<0, data_store_t>;
using arg_lap = tmp_arg<1, data_store_t>;
using arg_out = arg<2, data_store_t>;
auto horizontal_diffusion = make_computation<backend_t>(
    my_grid,
    make_multistage(
        execute::parallel(),
        make_stage<laplacian>(arg_in(), arg_lap()),
        make_stage<laplacian>(arg_lap(), arg_out())
        ));
horizontal_diffusion.run(arg_in{} = in, arg_out{} = out);
```

COSMO Performance GridTools vs. Stella

- Testcase with COSMO-E on Piz Kesch using 4 K80s (8 GK210)
- 582 x 3
- Total ru simulat
- Compc



19/05/2016)
.2km grid size, 21 members



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

GridTools4Py: Python Interface to GridTools

GT4Py Goals

- C++ is less known in the community (Fortran, Python)
  - Too low-level for domain scientists
- **Solution:** Use Python as a higher level embedded DSL
  - Hide C++ templates boilerplate from the user
- Use GridTools benefits
  - Employ flexible execution model with multiple backends
  - Provide same performance / low overhead
- Raise level of abstraction
  - Automatic computation of
    - Halo extents and data dependencies
    - Stages and multi-stages / computation-on-the-fly
    - Temporary fields vs scalar variables
- First release in the coming weeks

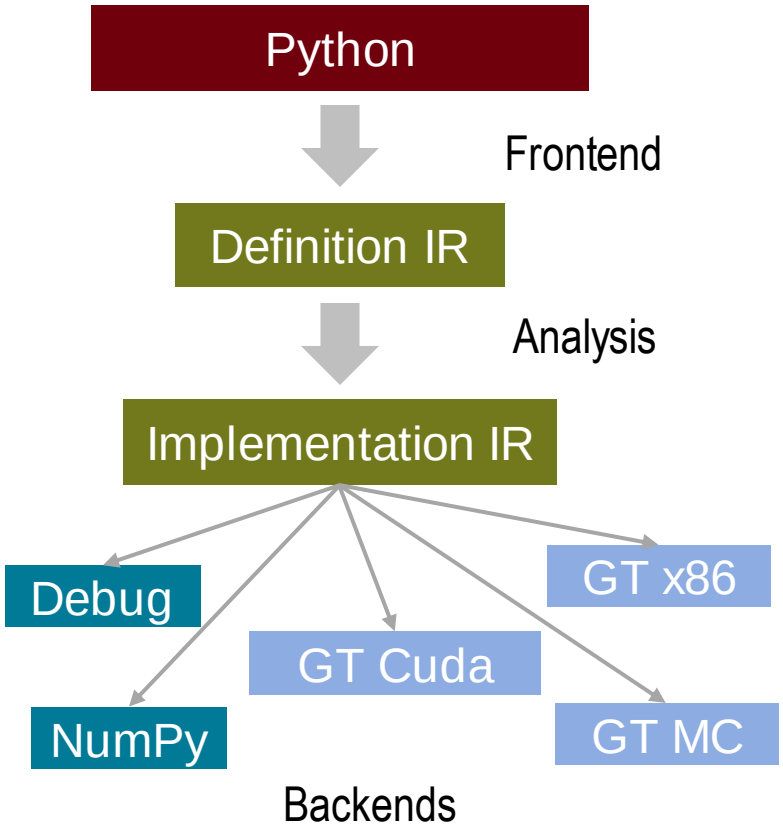
Horizontal Stencil Example

```python
import gridtools as gt

backend = "gtx86"

# Storage definition
float_st = gt.storage.StorageDescriptor(dtype=np.float32, grid_group="main")
```

```python
# Stencil definition
@gt.stencil(backend=backend)
def horizontal_diffusion(u: float_st, diffusion: float_st, coeff: float):
    laplacian = 4.0 * u[0, 0, 0] - (u[1, 0, 0] + u[-1, 0, 0] + u[0, 1, 0] + u[0, -1, 0])
    flux_i = laplacian[1, 0, 0] - laplacian[0, 0, 0]
    flux_i = 0.0 if flux_i * (u[1, 0, 0] - u[0, 0, 0]) > 0 else flux_i
    flux_j = laplacian[0, 1, 0] - laplacian[0, 0, 0]
    flux_j = 0.0 if flux_j * (u[0, 0, 0] - u[0, 1, 0]) > 0 else flux_j
    diffusion = u[0, 0, 0] - coeff * (
        flux_i[0, 0, 0] - flux_i[-1, 0, 0] + flux_j[0, 0, 0] - flux_j[0, -1, 0]
    )
```

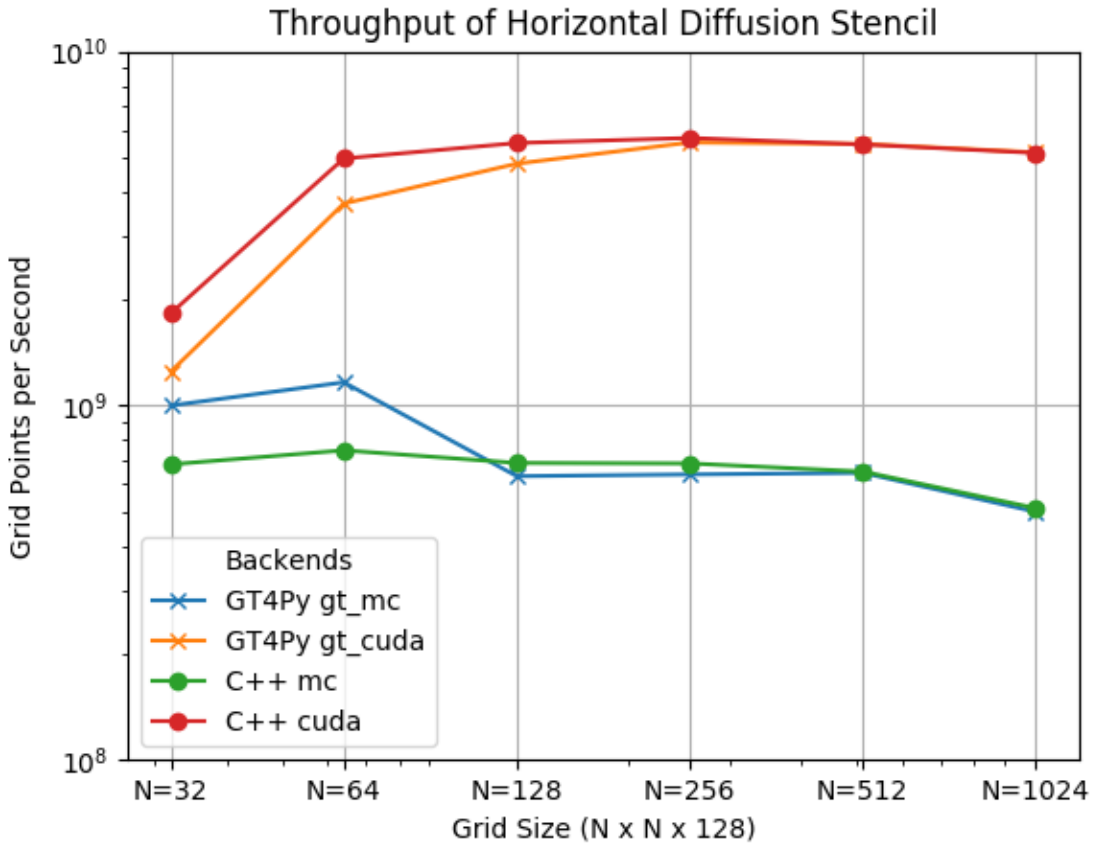
```python
# Storage initialization
# u_data_array = np.load(...)
u_storage = gt.storage.from_array(u_data_array,
                                  descriptor=float_st,
                                  backend=backend,
                                  halo=(2, 2, 0))

out_storage = gt.storage.empty(u_data_array.shape,
                               descriptor=float_st,
                               backend=backend,
                               halo=(2, 2, 0))

# Stencil execution
horizontal_diffusion(u_storage, out_storage, coeff=4.0)

print("Result: ", out_storage.data)
```

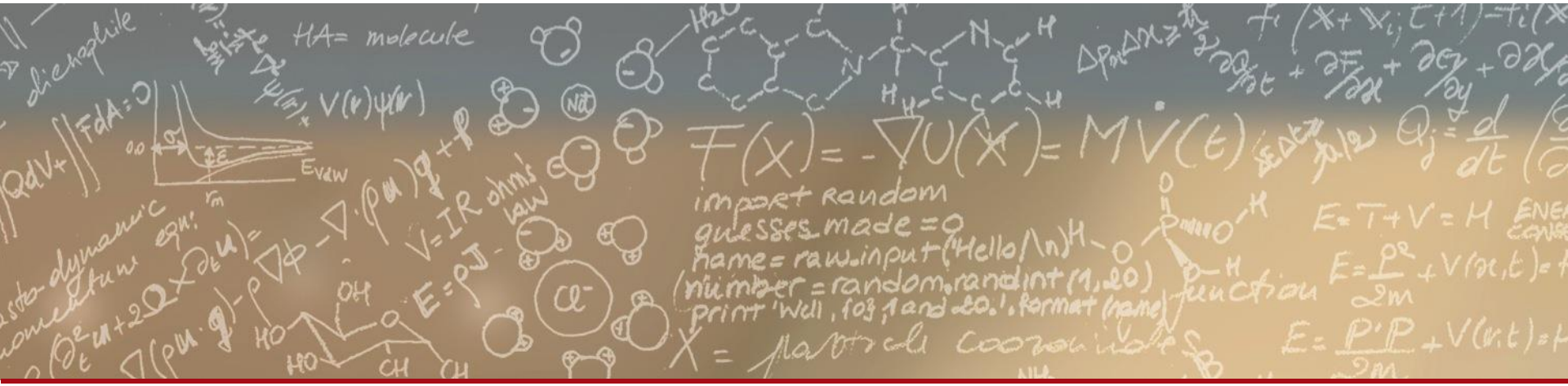
Pipeline


Python
Frontend
Definition IR
Analysis
Implementation IR
Debug
NumPy
GT Cuda
GT x86
GT MC
Backends

Preliminary Results

- Horizontal diffusion
  - GTMC and GTCuda backends (disabled caches)
  - Python overhead becomes neglectable for medium-sized domains





Thank you for your attention.