ETH zürich

spcl.inf.ethz.ch
@spcl_eth

D INFK

Productive Performance Engineering for Weather and Climate Modeling with Python

Tal Ben-Nun[*], Linus Groner[†], Florian Deconinck[‡], Tobias Wicky[‡], Eddie Davis[‡], Johann Dahm[‡], Oliver D. Elbert[‡], Rhea George[‡], Jeremy McGibbon[‡], Lukas Trümper[*], Elynn Wu[‡], Oliver Fuhrer[‡], Thomas Schulthess[†], Torsten Hoefler[*]

Supercomputing '22, Dallas, TX, Nov. 2022

SPCL [*]

CSCS[†]
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Ai2 [‡]

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 program (grant agreements PSAP, No. 101002047; DEEP-SEA, No. 955606; and MAELSTROM, No. 955513).

erc


Photo by Dan Meyers on Unsplash


Photo by John Middelkoop on Unsplash


DALL-E

```
subroutine q_j_stencil(is,ie,js,je,npz,x_area_flux,area_with_x_flux,q,area,fx1,fx2,q_j)
  integer, intent(in):: is, ie, js, je, npz
  real, intent(in):: x_area_flux(is:ie+1, js:je, npz)
  real, intent(in):: area_with_x_flux(is:ie, js:je, npz)
  real, intent(in):: q(isd:ied, js:je, npz)
  real, intent(in):: area(is:ie, js:je)
  real, intent(inout):: fx1(is:ie+1, js:je, npz)
  real, intent(in):: fx2(is:ie+1, js:je, npz)
  real, intent(out):: q_j(is:ie, js:je, npz)
  integer:: i, j, k
  do k = 1, npz
    do j = js, je
      do i = is, ie+1
        fx1(i,j,k) = x_area_flux(i,j,k)*fx2(i,j,k)
      enddo
      do i = is, ie
        area_with_x_flux(i,j,k) = area(i, j)+x_area_flux(i,j,k)-x_area_flux(i+1,j,k)
      enddo
      do i = is, ie
        q_j(i,j,k) = (q(i,j,k)*area(i,j)+fx1(i,j,k)-fx1(i+1,j,k))/area_with_x_flux(i,j,k)
      enddo
    enddo
  enddo
end subroutine q_j_stencil
```

Domain size embedded to computation

Loop order is fixed

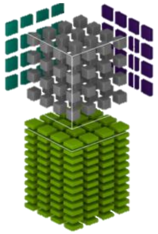
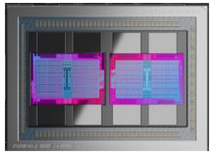
Schedule (fusion, recomputation) is fixed

Memory layout is fixed

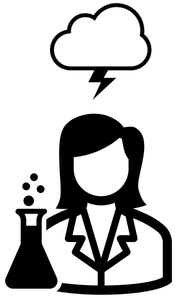
Hardcoded tiling strategies, rank distribution

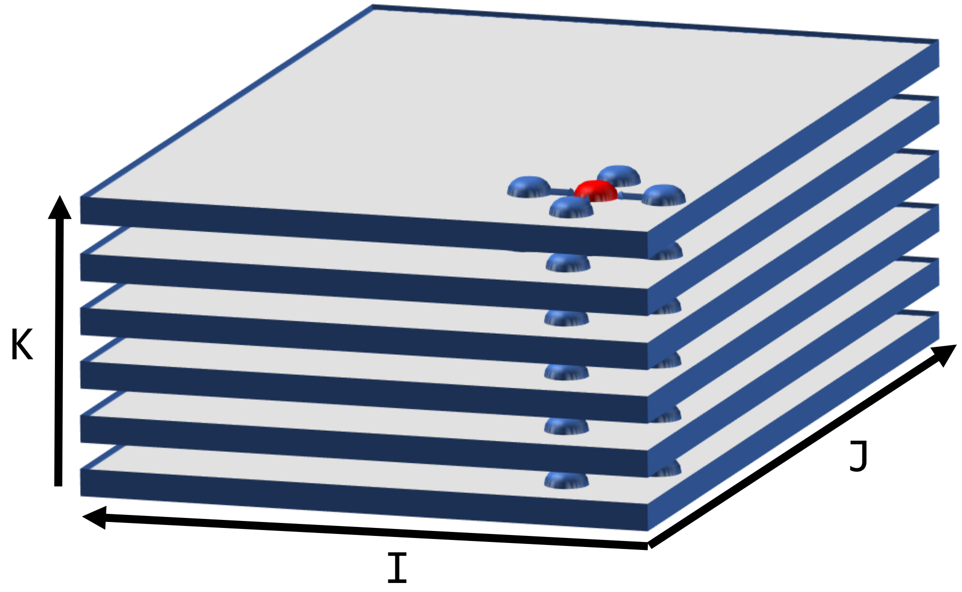
…

Hardware details fixed

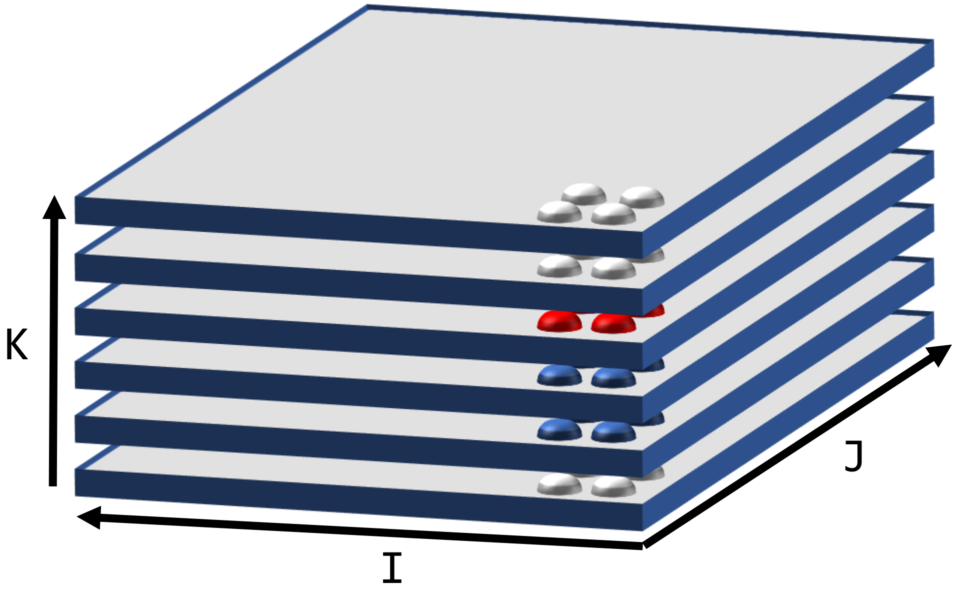









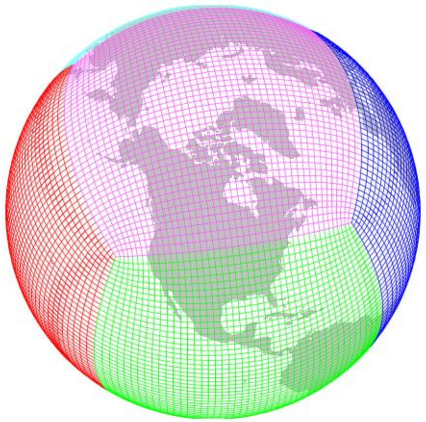
Horizontal Stencil



Vertical Solver

The FV3GFS Model

- **Finite-Volume Cubed-Sphere global climate model**
- **Dynamical core of models used by NOAA GFDL (e.g., X-SHiELD), NASA (GEOS, MCM), and other systems worldwide**
- **Distributed across at least 6 nodes (faces of the cubed sphere)**
  - Cubed-sphere grid balances uniform resolution, performance and simple code
- **Horizontal finite volume dynamics**
- **Vertical Lagrangian dynamics with remapping**
- **Baseline: highly-optimized FORTRAN for x86 CPU architectures**



https://www.gfdl.noaa.gov/fv3/

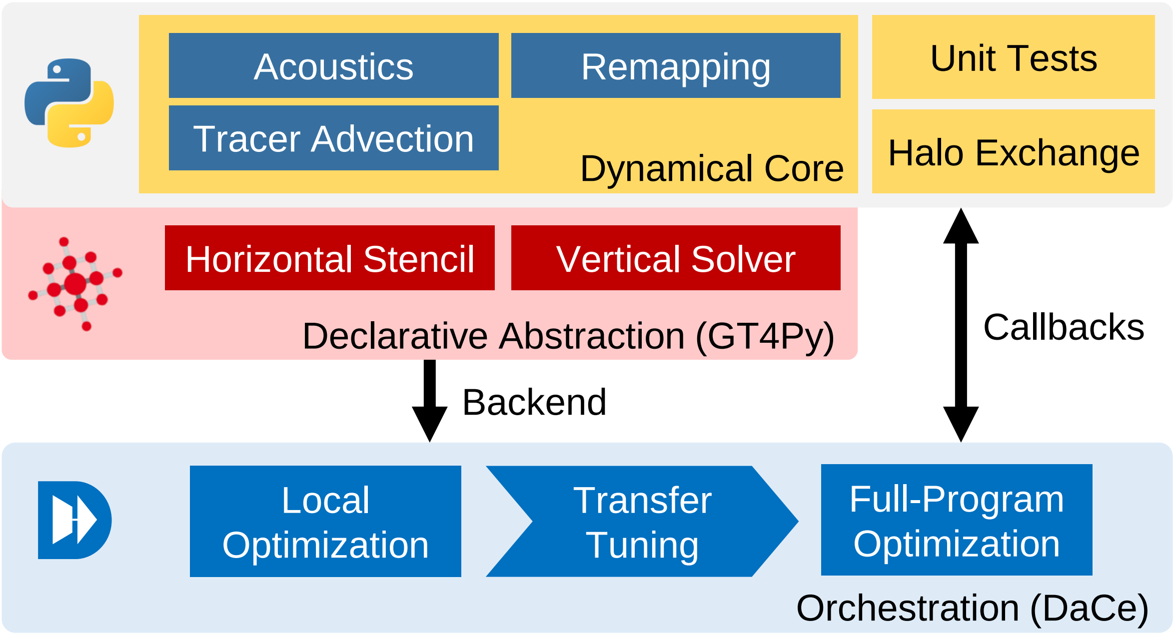
The Pace Project

https://github.com/ai2cm/pace

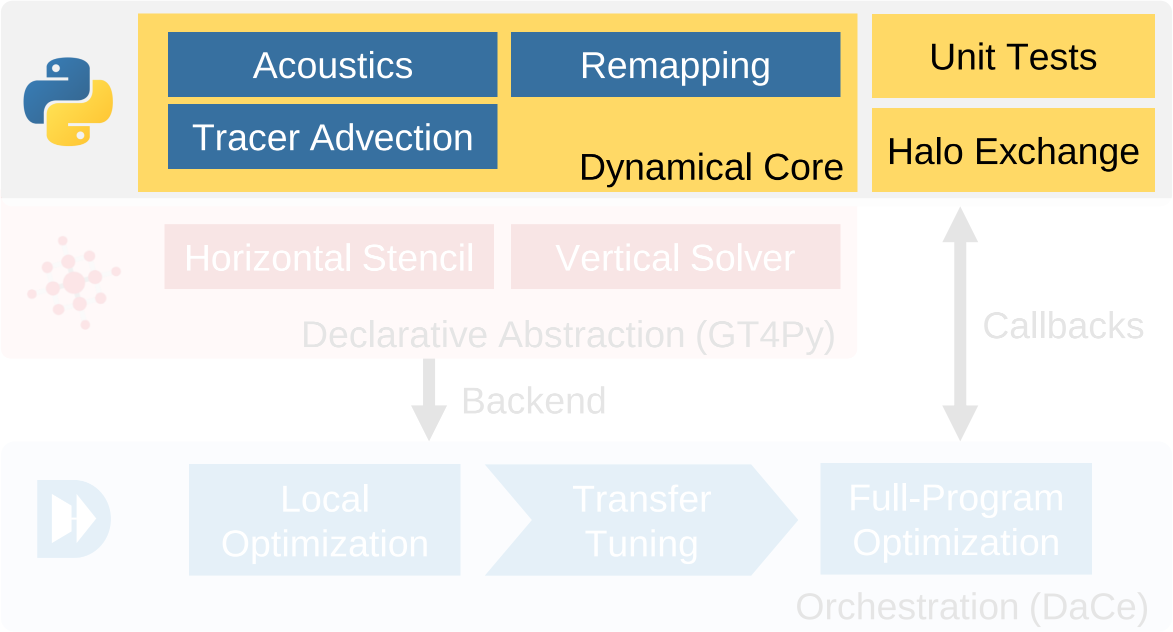
```
Usage: python -m pace.driver.run [OPTIONS] CONFIG_PATH

  Run the driver.

  CONFIG_PATH is the path to a DriverConfig yaml file.

Options:
  ...
```

- **FV3 reimagined in Python**
  - Goal: Atmospheric model that can run at scale on modern supercomputers
  - No FORTRAN involved
- **Full dynamical core: 12,450 Python LoC across 36 modules**

  vs. 29,458 in the baseline implementation



J. Dahm et al., "Pace v0.1: A Python-based Performance-Portable Implementation of the FV3 Dynamical Core". EGUSphere'22


Acoustics
Remapping
Tracer Advection
Dynamical Core
Unit Tests
Halo Exchange
Horizontal Stencil
Vertical Solver
Declarative Abstraction (GT4Py)
Callbacks
Backend
Local Optimization
Transfer Tuning
Full-Program Optimization
Orchestration (DaCe)

Scientific Computing is Moving to Python

IPython

jupyter

MD ANALYSIS

GridTools

python Package Index

SciPy

TensorFlow

seaborn

pandas

PyTorch

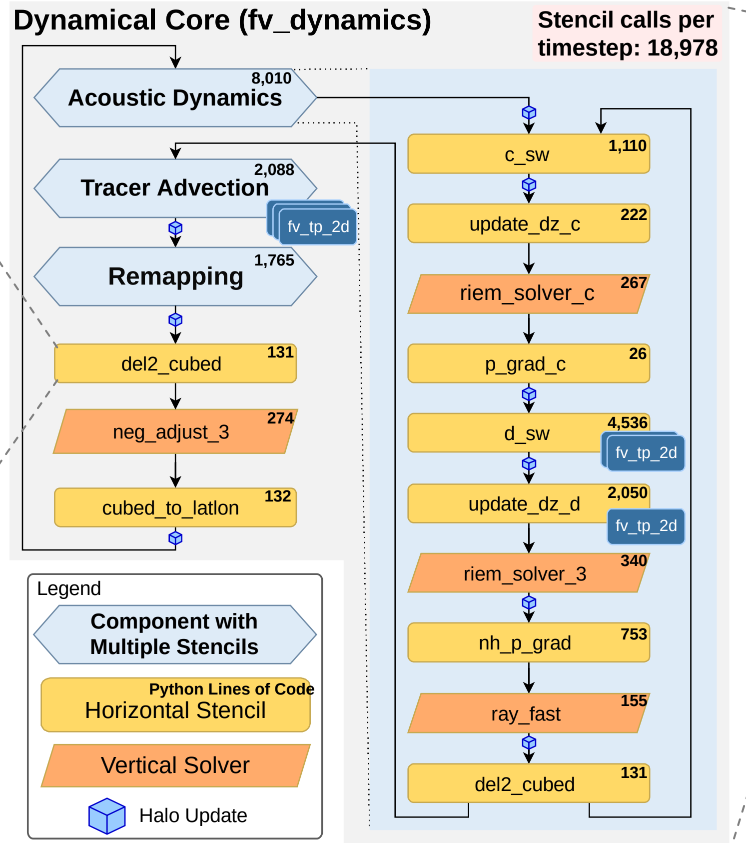
matplotlib

SymPy

NetworkX Network Analysis in Python

```python
class HyperdiffusionDamping:
  # ...
  def __call__(self, qdel: FloatField, cd: float):
    # ...
    for n in range(self._ntimes):
      nt = self._ntimes - (n + 1)
      self._corner_fill(qdel, self._q)

      if nt > 0:
        self._copy_corners_x(self._q)

      self._compute_zonal_flux[n](
          self._fx, self._q, self._del6_v)
      # ...
```

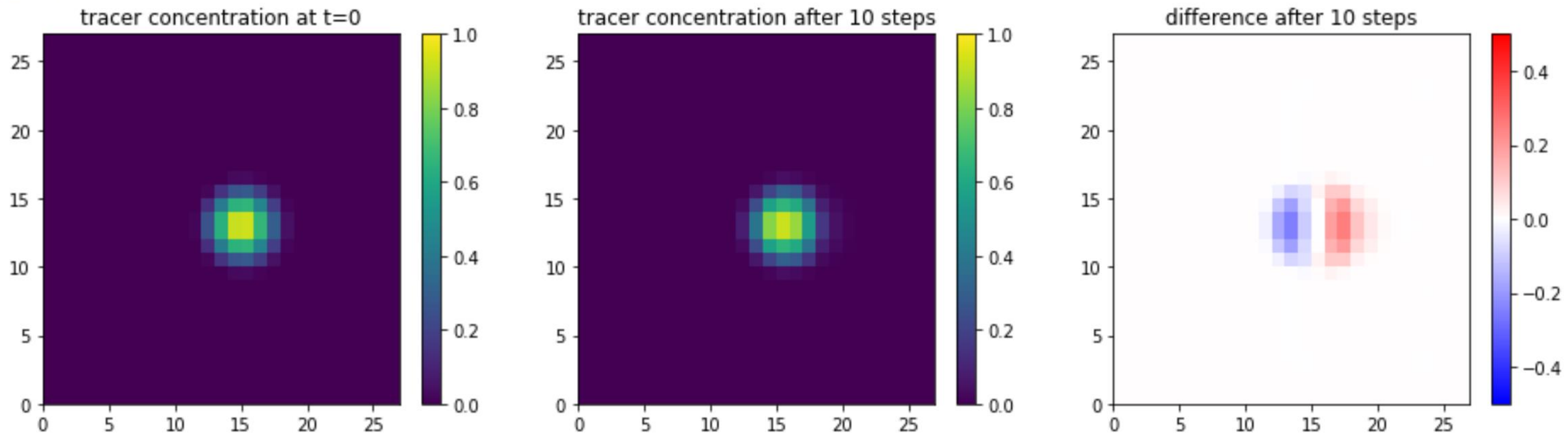
del2cubed.py




```python
def dycore_loop(state, dycore, time_steps):

  for _ in range(time_steps):
    dycore.step_dynamics(state)



# ...




state = initialize_state(...)  # Data loading
dycore = fv_dynamics.DynamicalCore(...)

# Invoke function
dycore_loop(state, dycore, T)

validate(state)

plot_on_map(state.x_wind)
```

dynamics.py

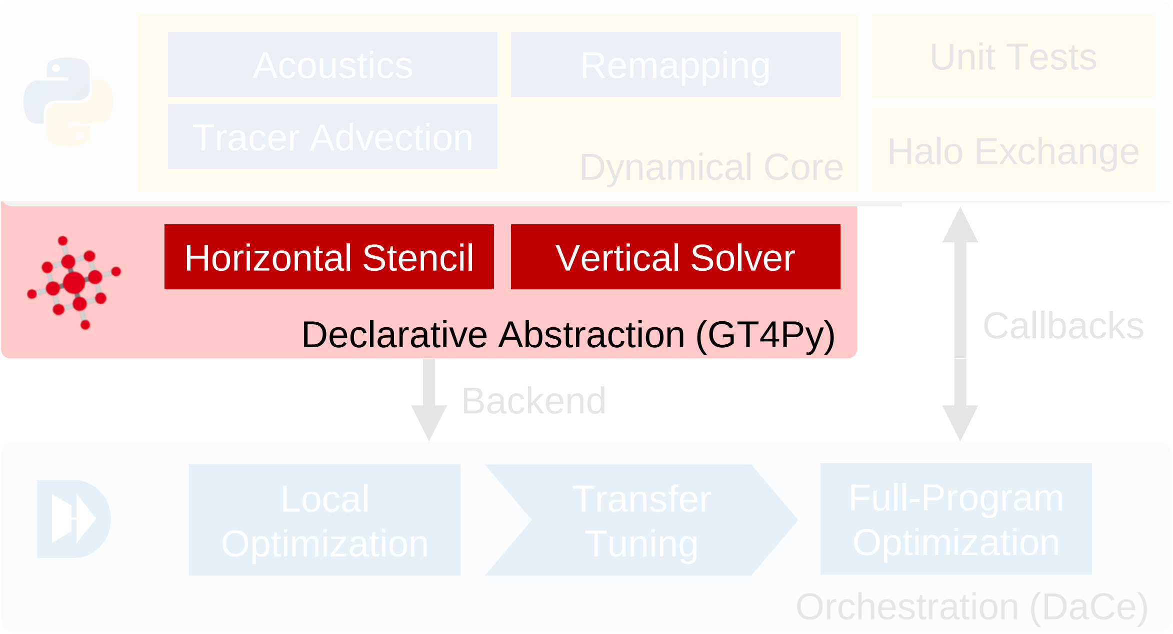
[output:0]




```
plot_on_map(state.x_wind)
```

dynamics.py

https://github.com/ai2cm/pace/blob/main/examples/notebooks/stencil_definition.ipynb

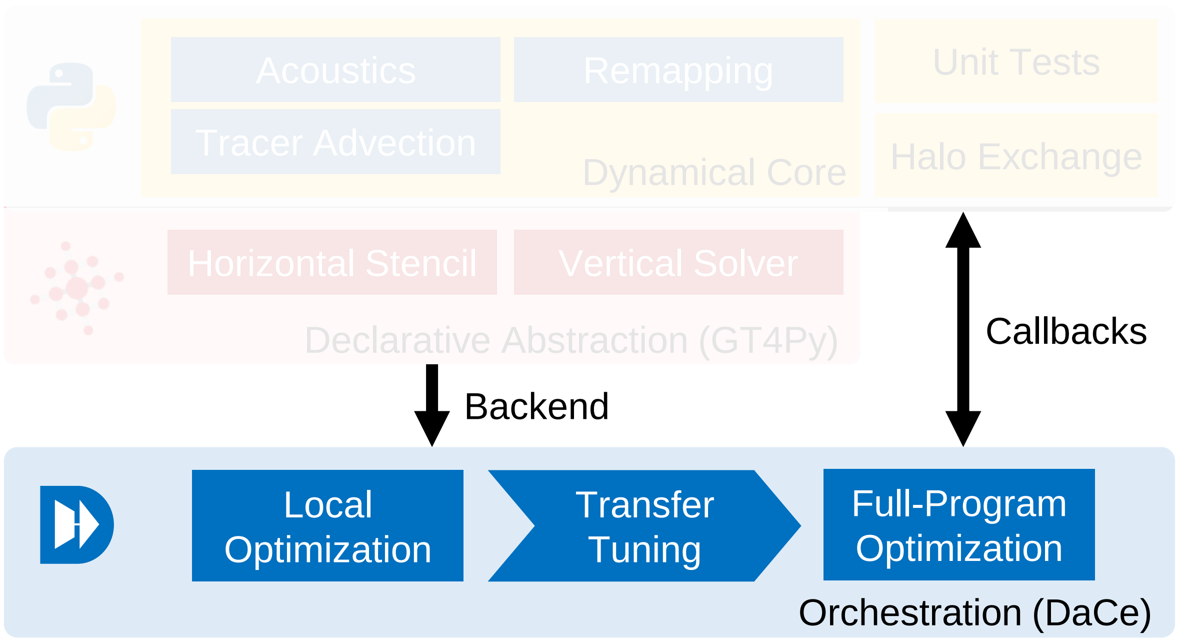

Acoustics
Remapping
Tracer Advection
Dynamical Core
Unit Tests
Halo Exchange
Horizontal Stencil
Vertical Solver
Declarative Abstraction (GT4Py)
Callbacks
Backend
Local Optimization
Transfer Tuning
Full-Program Optimization
Orchestration (DaCe)

GridTools for Python (GT4Py)

https://github.com/GridTools/gt4py

- **Domain Specific Language (DSL) for Weather and Climate**
- **A declarative approach to define stencils ("what", not "how")**
  - 3D stencils and vertical solvers
- **Computation domain is abstracted**
  - Relative indexing
  - Automatic iteration ranges and halo regions
- **Implementation concerns are delegated to backends**
  - Execution schedules
  - Memory allocation
  - Target language

```
@gtscript.stencil(backend='dace:gpu')
def q_j_stencil(q: FloatField, area: FloatFieldIJ,
                x_area_flux: FloatField, fx2: FloatField,
                q_j: FloatField):

    with computation(PARALLEL), interval(...):

      fx1 = x_area_flux * fx2

      area_with_x_flux = area + x_area_flux - x_area_flux[1, 0, 0]

      q_j = (q * area + fx1 - fx1[1, 0, 0]) / area_with_x_flux
```


Acoustics
Remapping
Unit Tests
Tracer Advection
Dynamical Core
Halo Exchange
Horizontal Stencil
Vertical Solver
Declarative Abstraction (GT4Py)
Callbacks
Backend
Local Optimization
Transfer Tuning
Full-Program Optimization
Orchestration (DaCe)

Domain Scientist

```
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += A[i, k] * B[k, j]
```

(or `C += A @ B`)

333 LoC

System

```
__syncthreads();

// Compute a grid of C matrix tiles in each warp.
#pragma unroll
for (int k_step = 0; k_step < CHUNK_K; k_step++) {
    wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::row_major> a[WARP_COL_TILES];
    wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> b[WARP_ROW_TILES];

    #pragma unroll
    for (int i = 0; i < WARP_COL_TILES; i++) {
        size_t shmem_idx_a = (warpId/2) * M * 2 + (i * M);
        const half *tile_ptr = &shmem[shmem_idx_a][k_step * K];

        wmma::load_matrix_sync(a[i], tile_ptr, K * CHUNK_K + SKEW_HALF);

        #pragma unroll
        for (int j = 0; j < WARP_ROW_TILES; j++) {
            if (i == 0) {
                // Load the B matrix fragment once, because it is going to be reused
                // against the other A matrix fragments.
                size_t shmem_idx_b = shmem_idx_b_off + (WARP_ROW_TILES * N) * (warpId%2)
                                     + (j * N);
                const half *tile_ptr = &shmem[shmem_idx_b][k_step * K];

                wmma::load_matrix_sync(b[j], tile_ptr, K * CHUNK_K + SKEW_HALF);
            }

            wmma::mma_sync(c[i][j], a[i], b[j], c[i][j]);
        }
    }
}

__syncthreads();
```

Tensor Core NVIDIA Code Sample

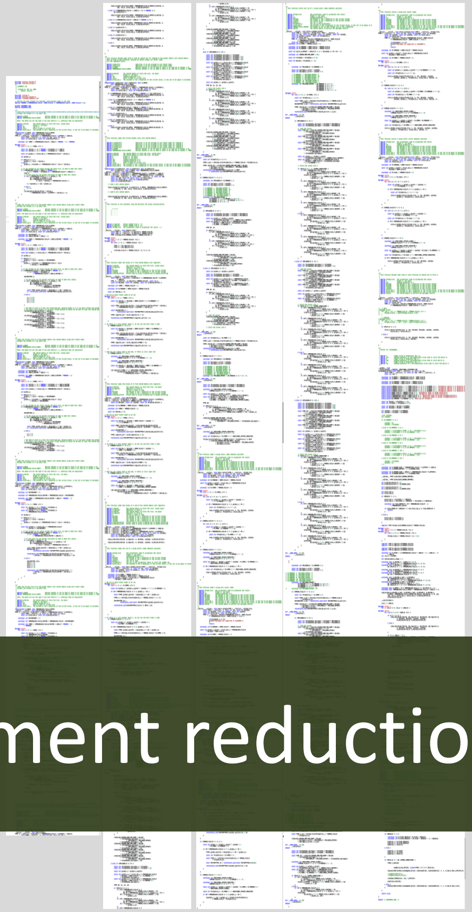
Domain Scientist

```python
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += A[i, k] * B[k, j]
```

(or `C += A @ B`)

System

1,717 LoC



COSMA on CUDA/HIP

High-performance optimization = data movement reduction

Kwasniewski et al., "Red-blue pebbling revisited: near optimal parallel matrix-matrix multiplication" SC'19 https://github.com/spcl/dace

DaCe Overview

Domain Scientist

Problem Formulation

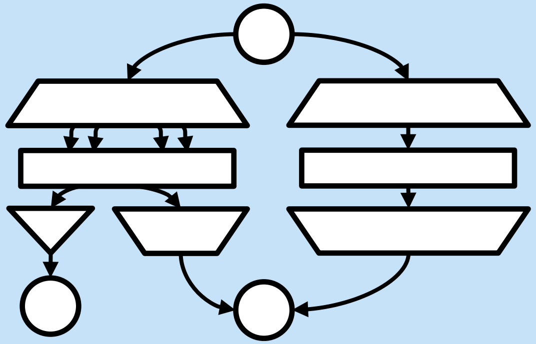
$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

Python	GT4Py
PyTorch	C

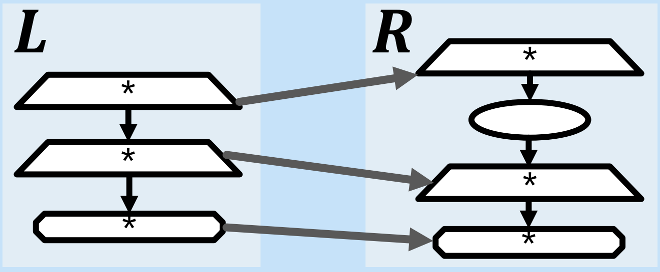
...

Data-Centric Program

Performance Engineer



Data-Centric Intermediate Representation (SDFG)



Graph Transformations

Transformed Dataflow

Performance Results

System

Hardware Information

Compiler

Runtime

CPU Binary

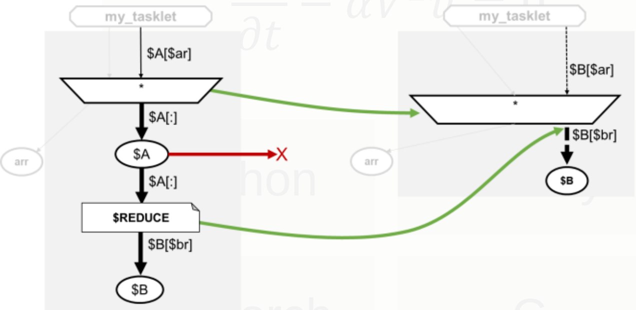
GPU Binary

FPGA Modules

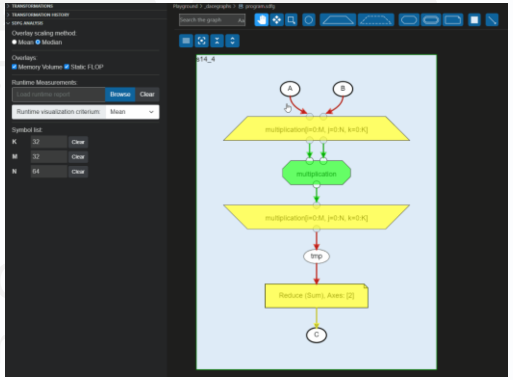
Ben-Nun et al., Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures, SC'19.

DaCe Overview

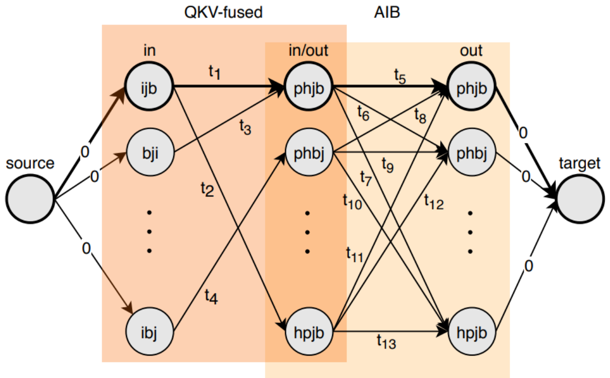
https://github.com/spcl/dace



Graph Rewriting Transformations



Interactive Transformation and Instrumentation



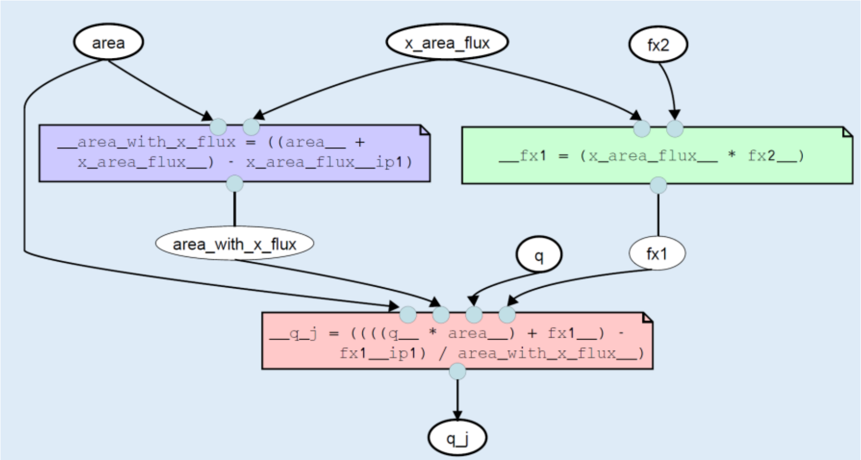
Local and Global Tuning Interface

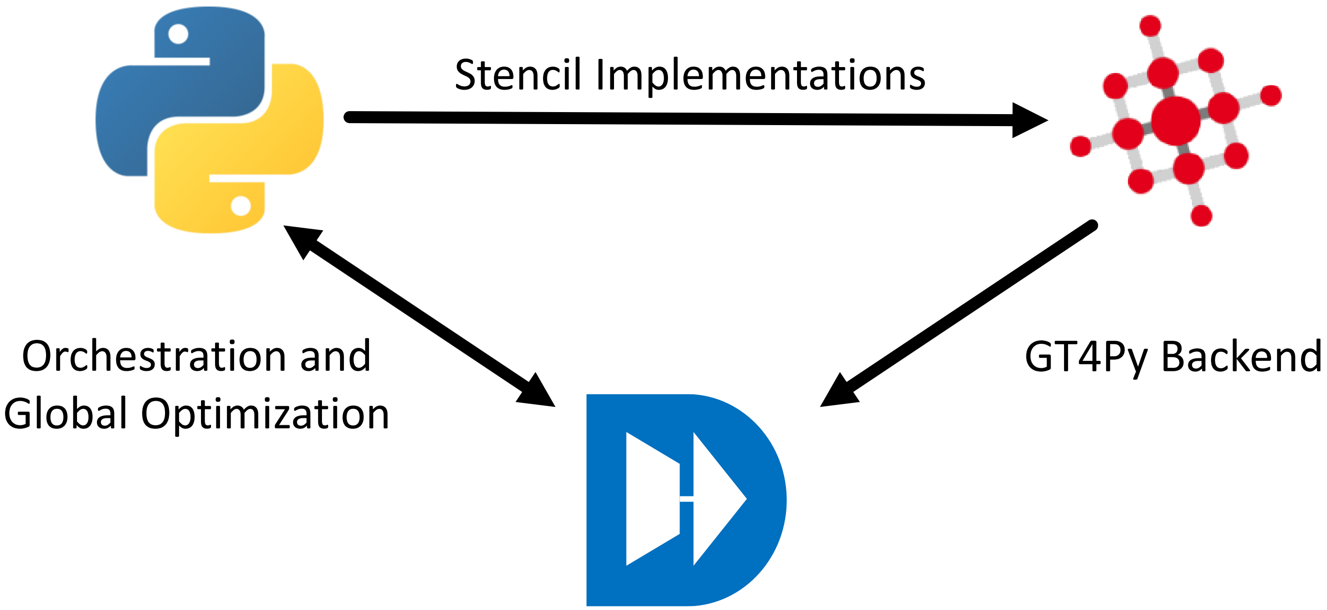
Ben-Nun et al., Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures, SC'19.

```python
@gtscript.stencil(backend='dace:gpu')
def q_j_stencil(q: FloatField, area: FloatFieldIJ,
                x_area_flux: FloatField, fx2: FloatField,
                q_j: FloatField):
    with computation(PARALLEL), interval(...):
      fx1 = x_area_flux * fx2
      area_with_x_flux = area + x_area_flux - x_area_flux[1, 0, 0]
      q_j = (q * area + fx1 - fx1[1, 0, 0]) / area_with_x_flux
```

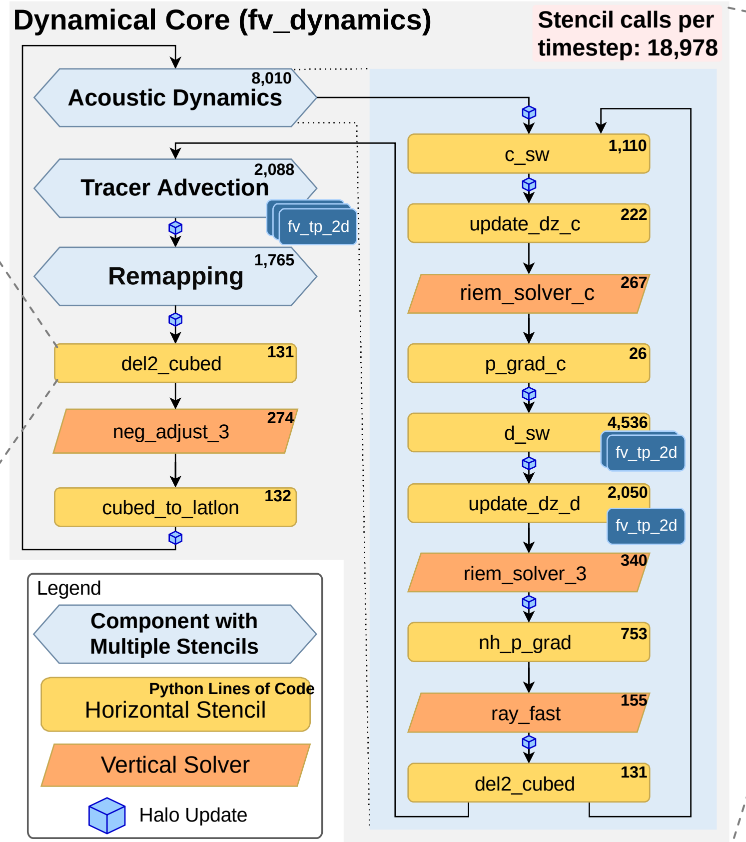
Stencil Implementations

GT4Py Backend




Stencil Implementations
Orchestration and
Global Optimization
GT4Py Backend

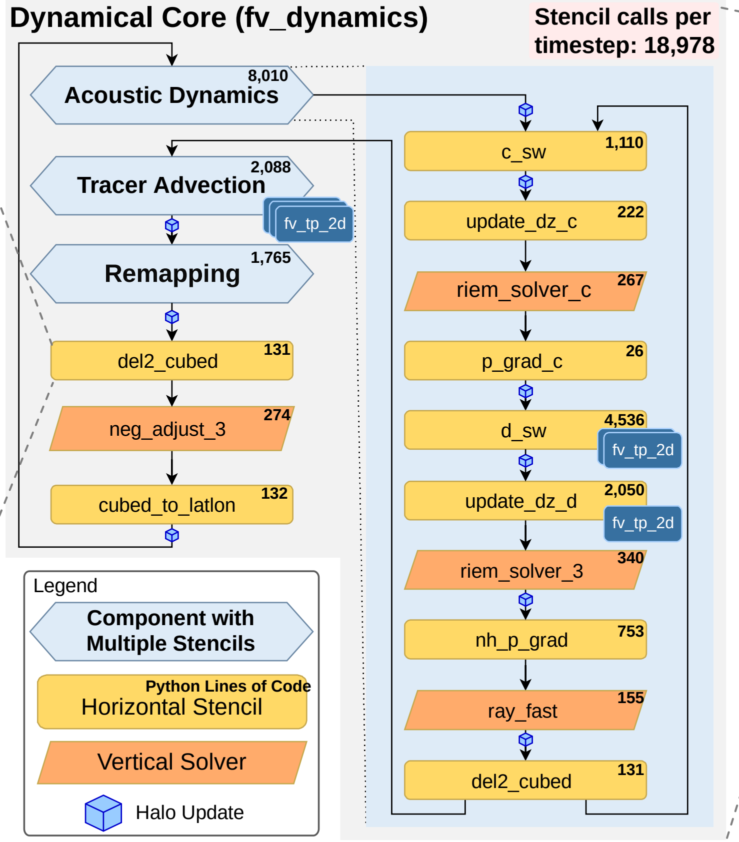
```python
class HyperdiffusionDamping:
  # ...
  def __call__(self, qdel: FloatField, cd: float):
    # ...
    for n in range(self._ntimes):
      nt = self._ntimes - (n + 1)
      self._corner_fill(qdel, self._q)

      if nt > 0:
        self._copy_corners_x(self._q)

      self._compute_zonal_flux[n](
          self._fx, self._q, self._del6_v)
      # ...
```




```python
def dycore_loop(state, dycore, time_steps):

  for _ in range(time_steps):
    dycore.step_dynamics(state)



# ...



state = initialize_state(...)  # Data loading
dycore = fv_dynamics.DynamicalCore(...)

# Invoke function
dycore_loop(state, dycore, T)

validate(state)

plot_on_map(state.x_wind)
```

```python
class HyperdiffusionDamping:
  # ...
  def __call__(self, qdel: FloatField, cd: float):
    # ...
    for n in range(self._ntimes):
      nt = self._ntimes - (n + 1)
      self._corner_fill(qdel, self._q)

      if nt > 0:
        self._copy_corners_x(self._q)

      self._compute_zonal_flux[n](
          self._fx, self._q, self._del6_v)
      # ...
```

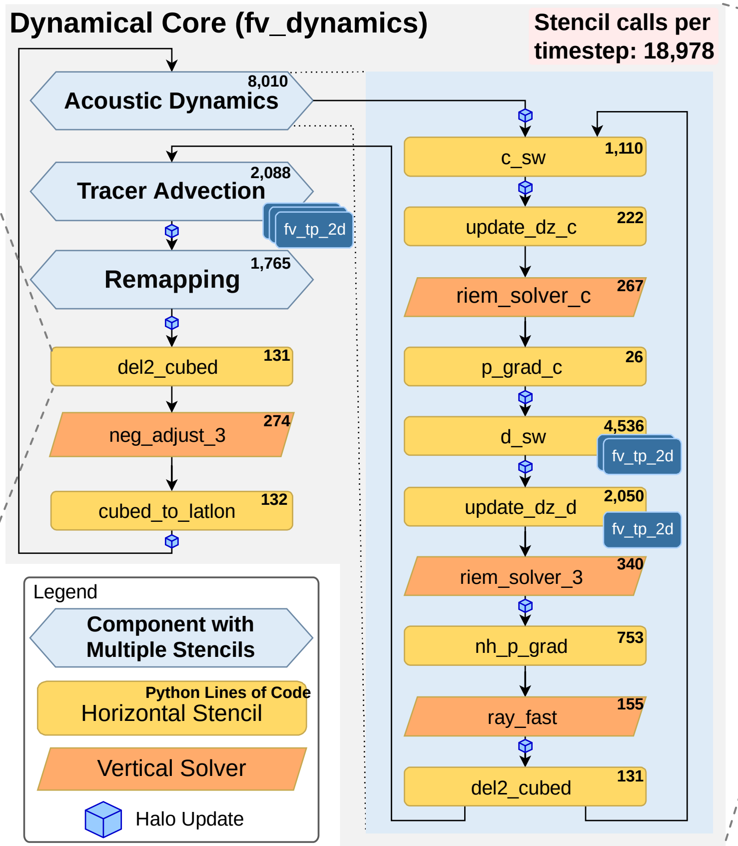
```python
@gtscript.stencil
def compute_zonal_flux(flux: FloatField,
                       a_in: FloatField,
                       del_term: FloatFieldIJ):
    with computation(PARALLEL), interval(...):
      flux = del_term * (a_in[-1, 0, 0] - a_in)
```




```python
def dycore_loop(state, dycore, time_steps):

  for _ in range(time_steps):
    dycore.step_dynamics(state)



# ...



state = initialize_state(...)  # Data loading
dycore = fv_dynamics.DynamicalCore(...)

# Invoke function
dycore_loop(state, dycore, T)

validate(state)

plot_on_map(state.x_wind)
```

```python
class HyperdiffusionDamping:
  # ...
  def __call__(self, qdel: FloatField, cd: float):
    # ...
    for n in range(self._ntimes):
      nt = self._ntimes - (n + 1)
      self._corner_fill(qdel, self._q)

      if nt > 0:
        self._copy_corners_x(self._q)

      self._compute_zonal_flux[n](
          self._fx, self._q, self._del6_v)
      # ...
```

```python
@gtscript.stencil
def compute_zonal_flux(flux: FloatField,
                       a_in: FloatField,
                       del_term: FloatFieldIJ):
    with computation(PARALLEL), interval(...):
      flux = del_term * (a_in[-1, 0, 0] - a_in)
```




```python
@dace
def dycore_loop(state, dycore, time_steps):

  for _ in range(time_steps):
    dycore.step_dynamics(state)
```

Compile

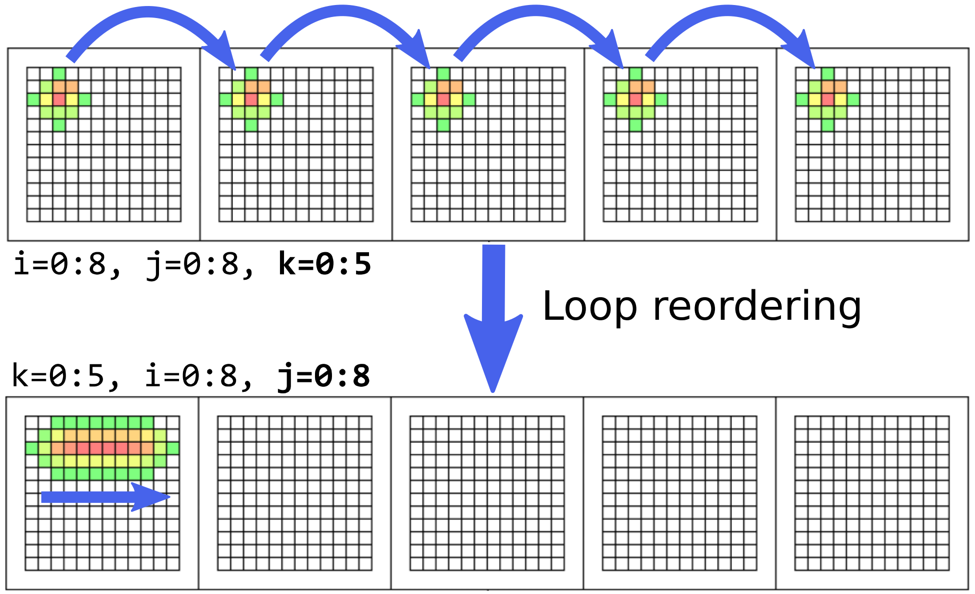
`libdycore_loop.so`

```python
state = initialize_state(...)  # Data loading
dycore = fv_dynamics.DynamicalCore(...)

# Invoke compiled function
dycore_loop(state, dycore, T)

validate(state)

plot_on_map(state.x_wind)
```

Characterizing the optimization space

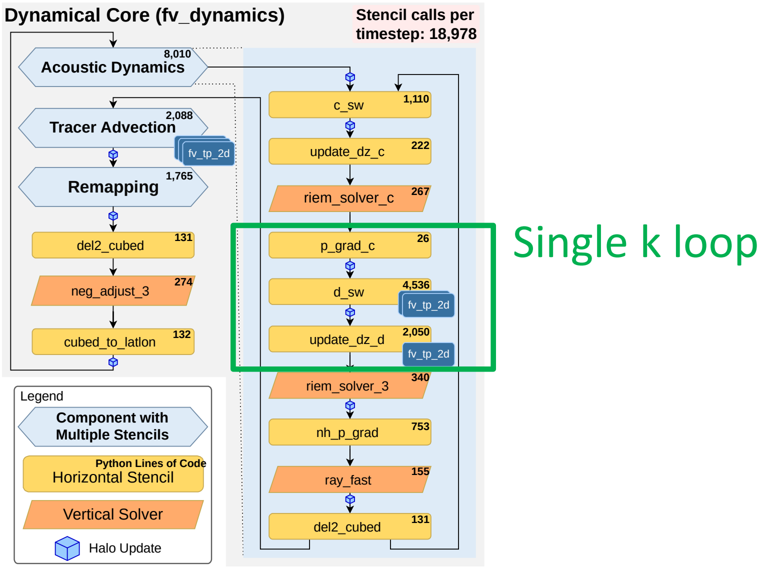
Within each stencil

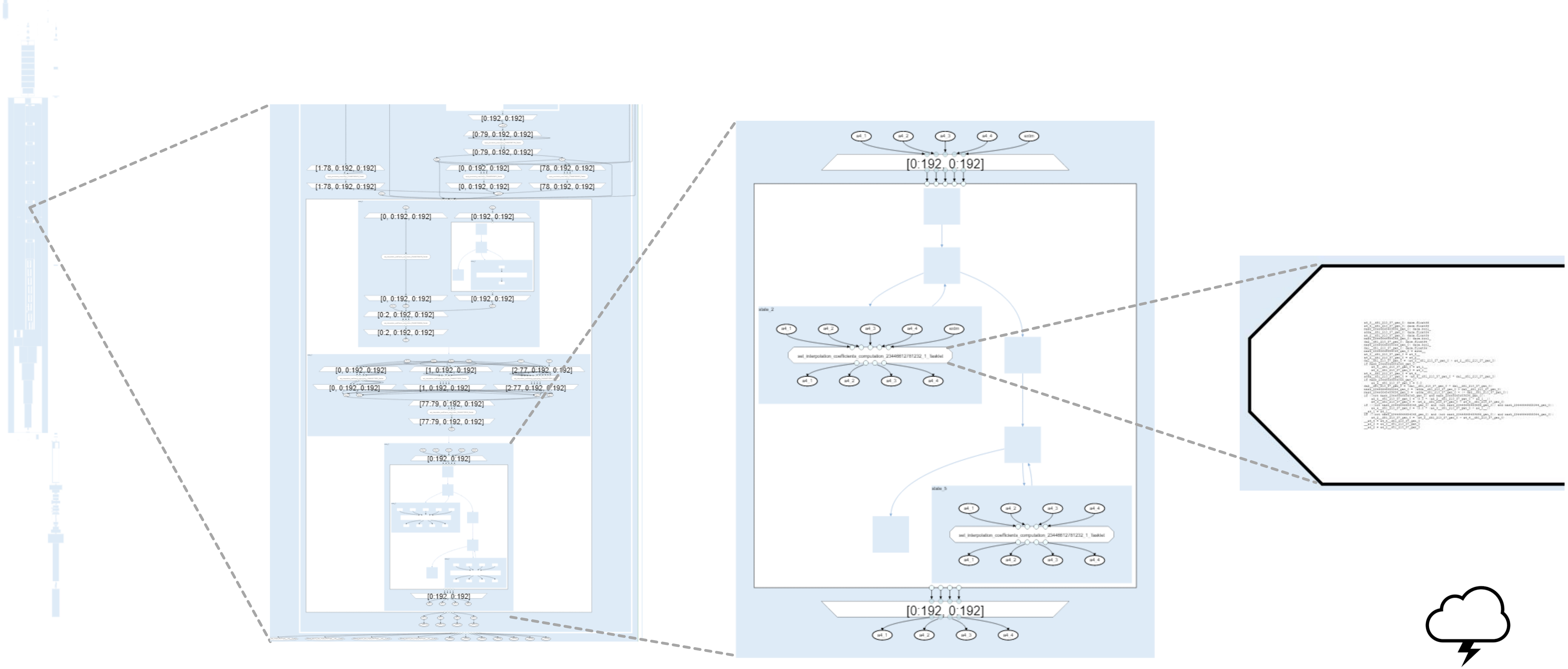
- Computational layout
- Data layout
- Other rescheduling passes in GT4Py (e.g., branch → predication)

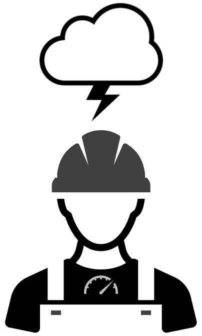


Between stencils

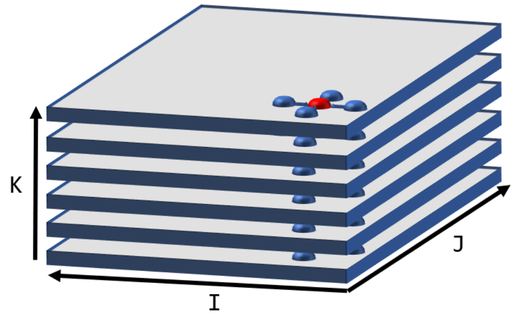
- Fusion
- Macro scheduling
- Pre-allocation (memory pool, static)
- Data layout "path"

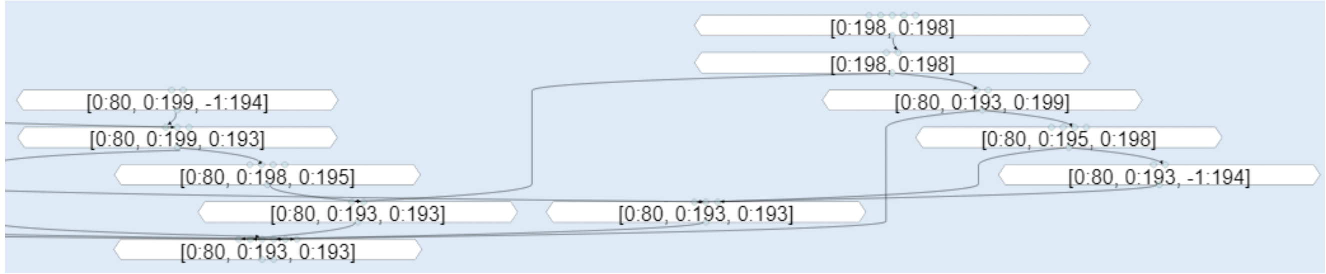



[0:192, 0:192]
[0:79, 0:192, 0:192]
[1:78, 0:192, 0:192]
[0, 0:192, 0:192]
[78, 0:192, 0:192]
[0:2, 0:192, 0:192]
[1, 0:192, 0:192]
[2:77, 0:192, 0:192]
[77:79, 0:192, 0:192]

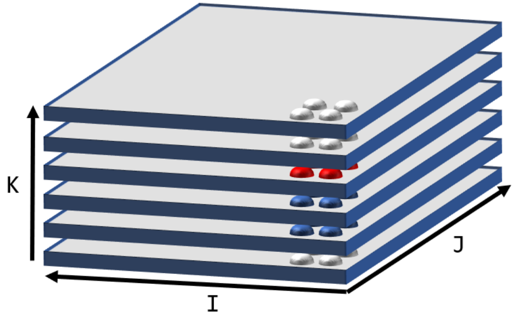


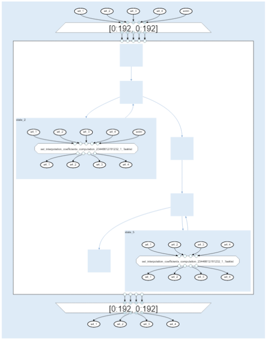
Initial Heuristics





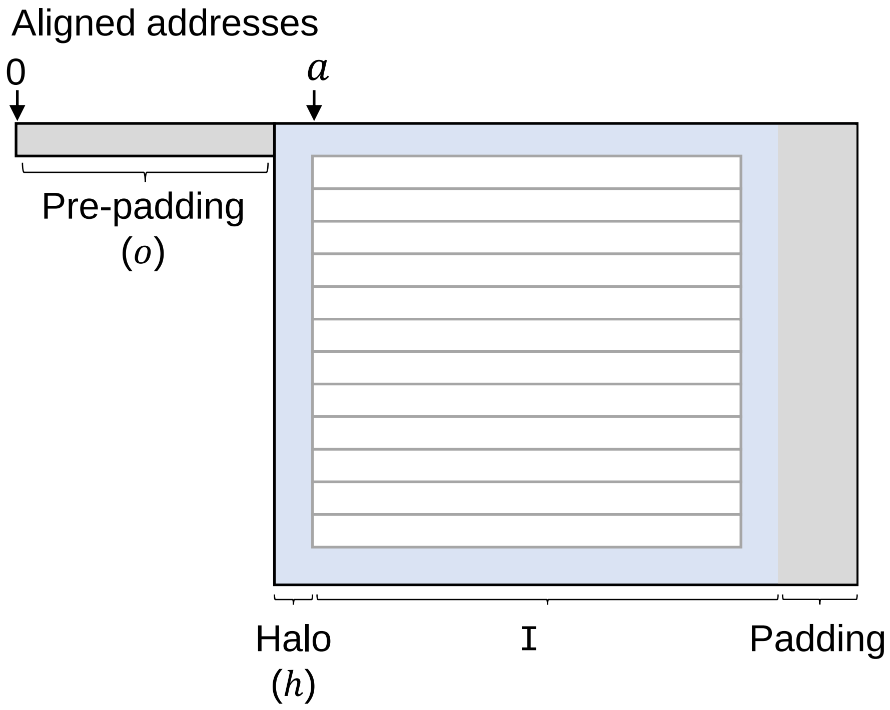
Interval, Operation, K, J, I





J, I, Interval, Operation, K

Initial Heuristics



Shape: $(I + 2h,\ J + 2h,\ K)$

Start offset: $o = a - h$

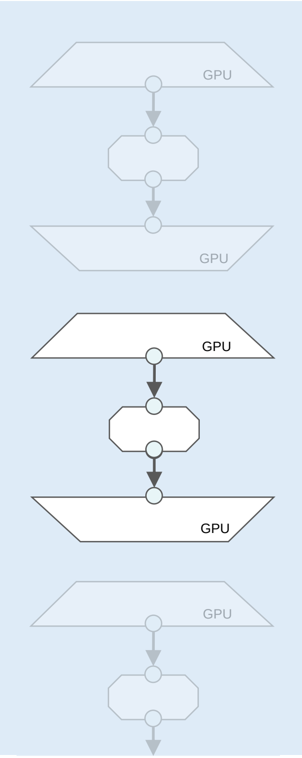
Strides:

$$s_i = 1$$

$$s_j = a \left\lceil \frac{I + 2h}{a} \right\rceil$$

$$s_k = s_j \cdot (J + 2h)$$

Initial Heuristics → Module-Based Autotuning → Transfer-Tune to Full Application

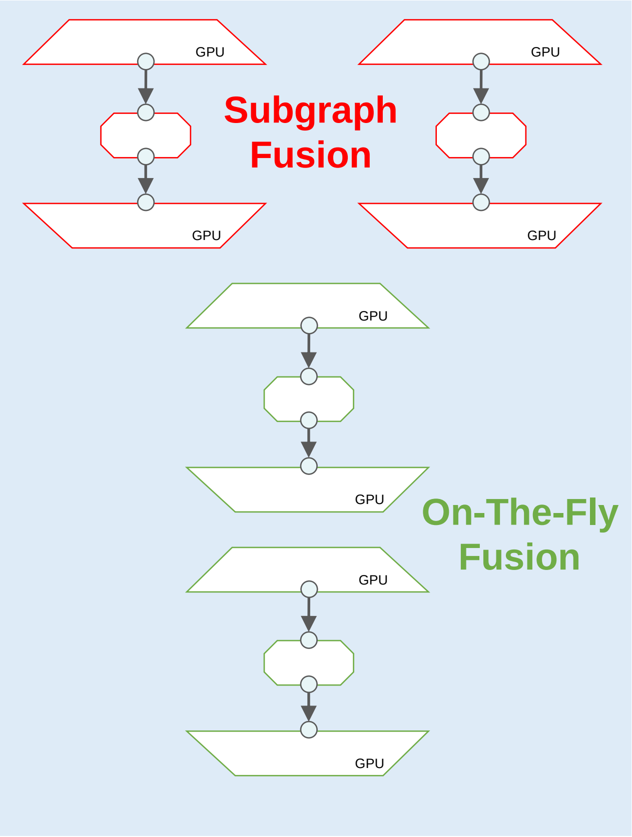


Exhaustive tuning on graph cutouts

2:42 hours on Piz Daint

```
[
 {copy_corners_y_nord: 5},

 ...

 {compute_y_flux: 2,
  final_fluxes: 1}
]
```

Store top-k patterns



Test and apply on full program

8:24 hours

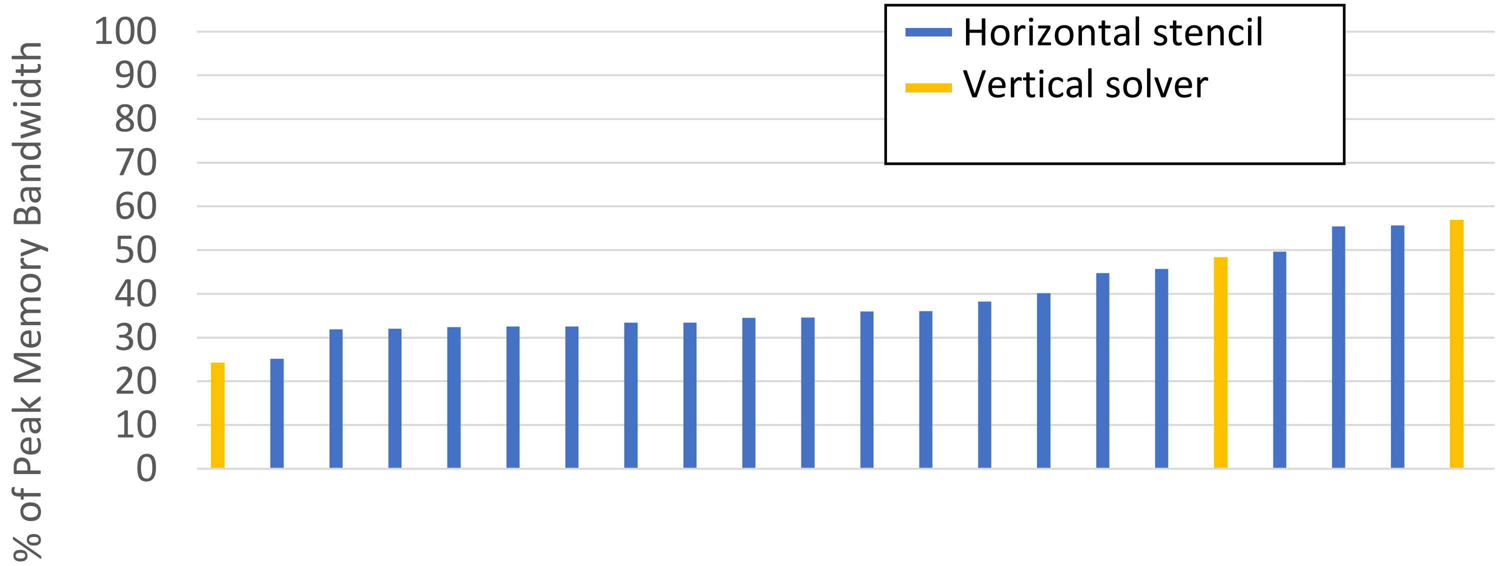
Without transfer tuning:

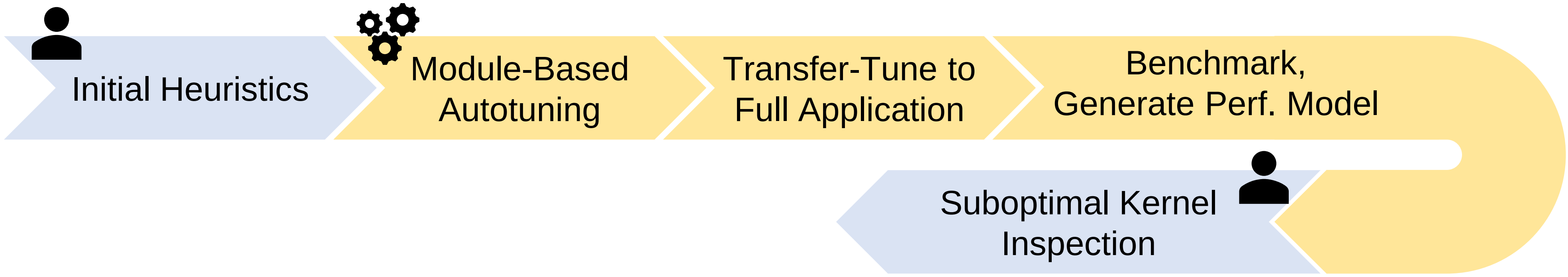
≥30,302,185 configurations

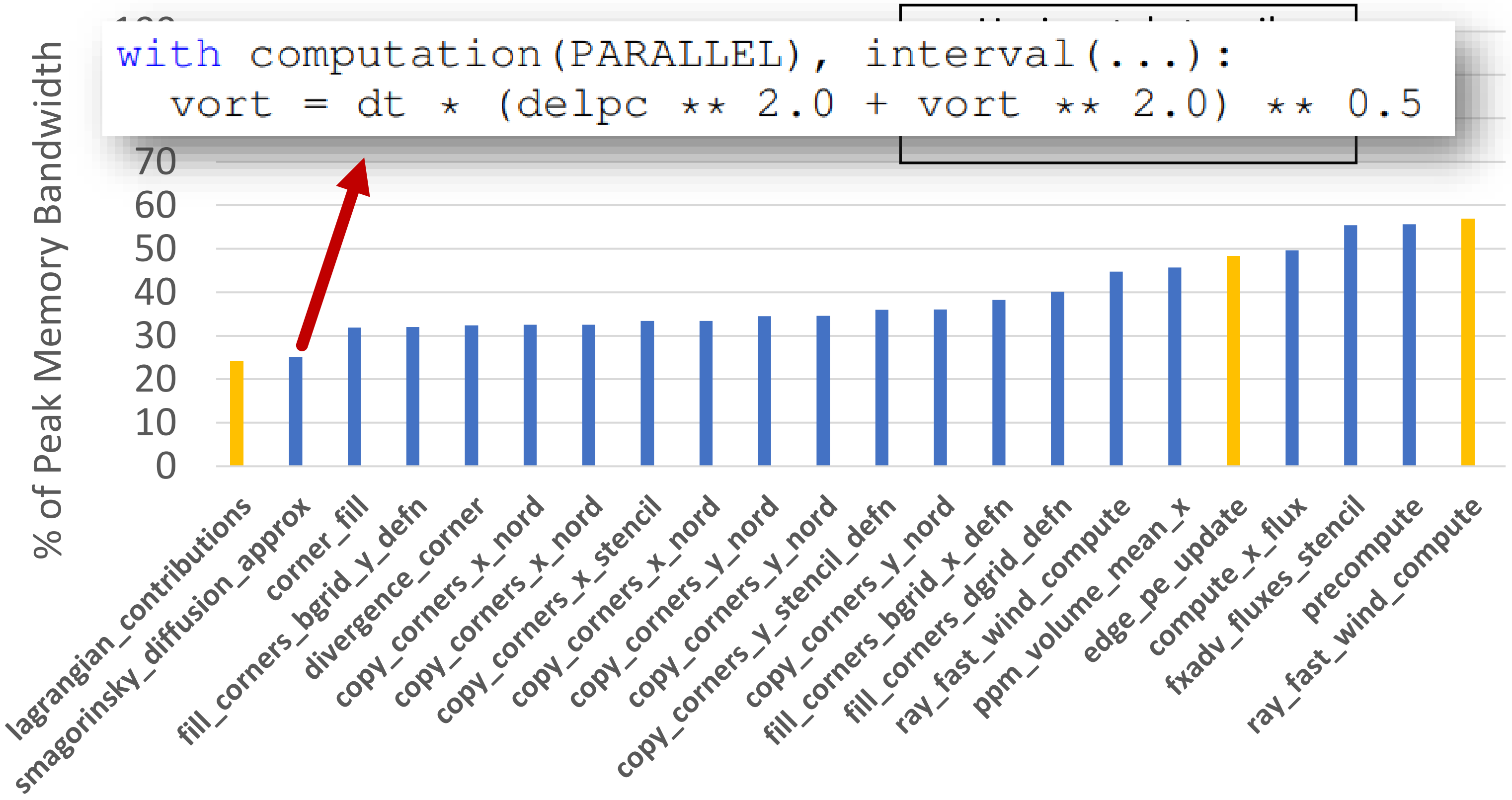
With transfer tuning:

603

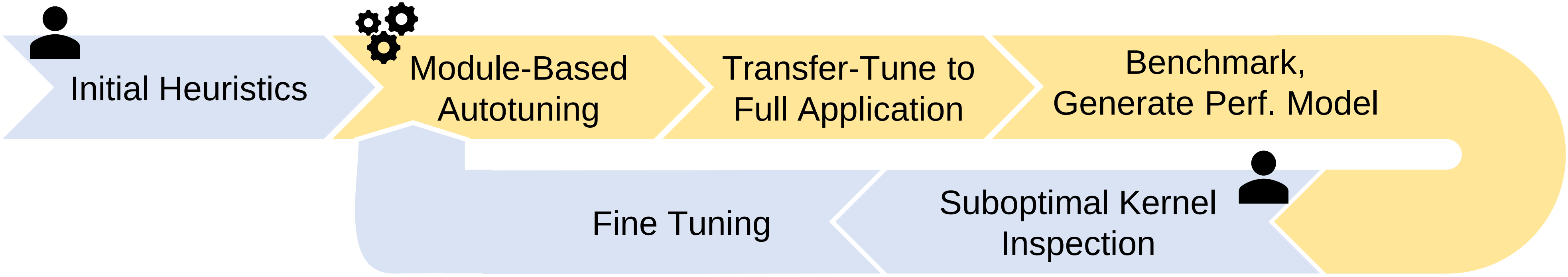
Initial Heuristics → Module-Based Autotuning → Transfer-Tune to Full Application → Benchmark, Generate Perf. Model



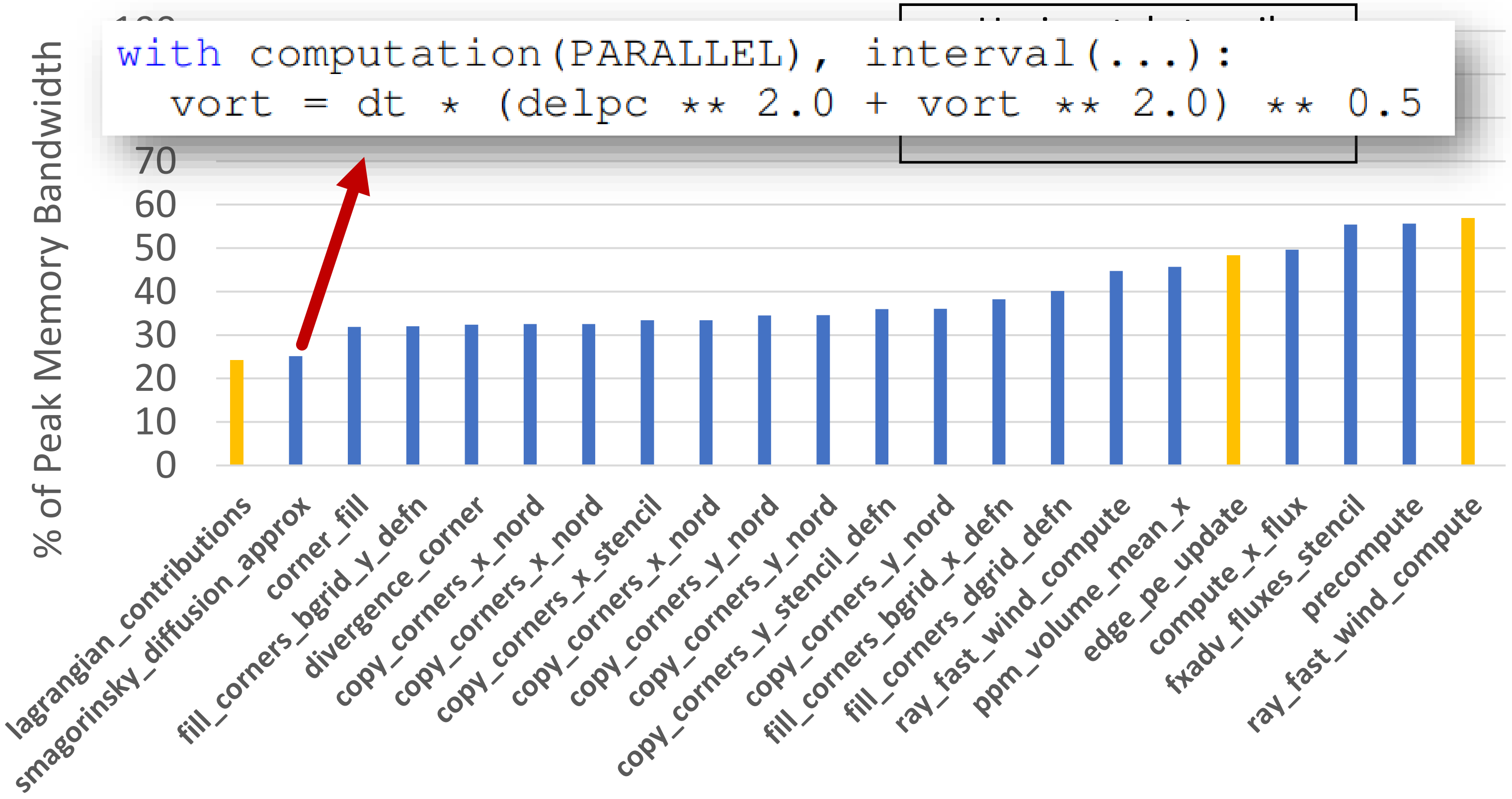


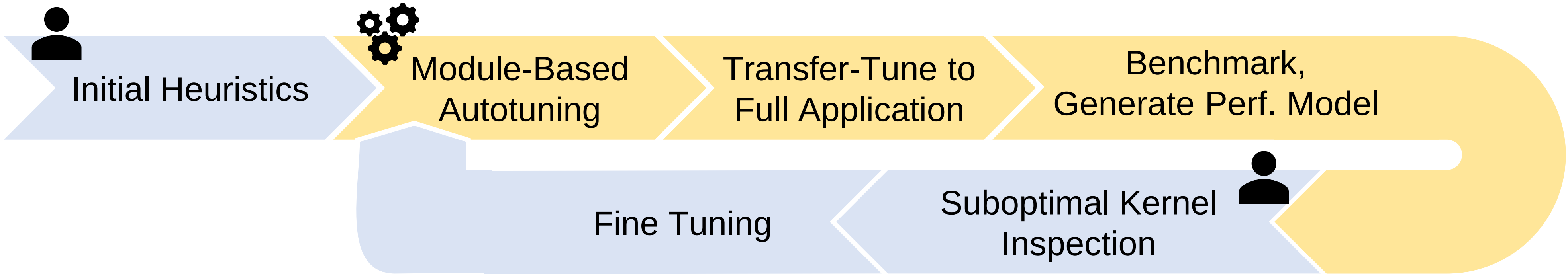


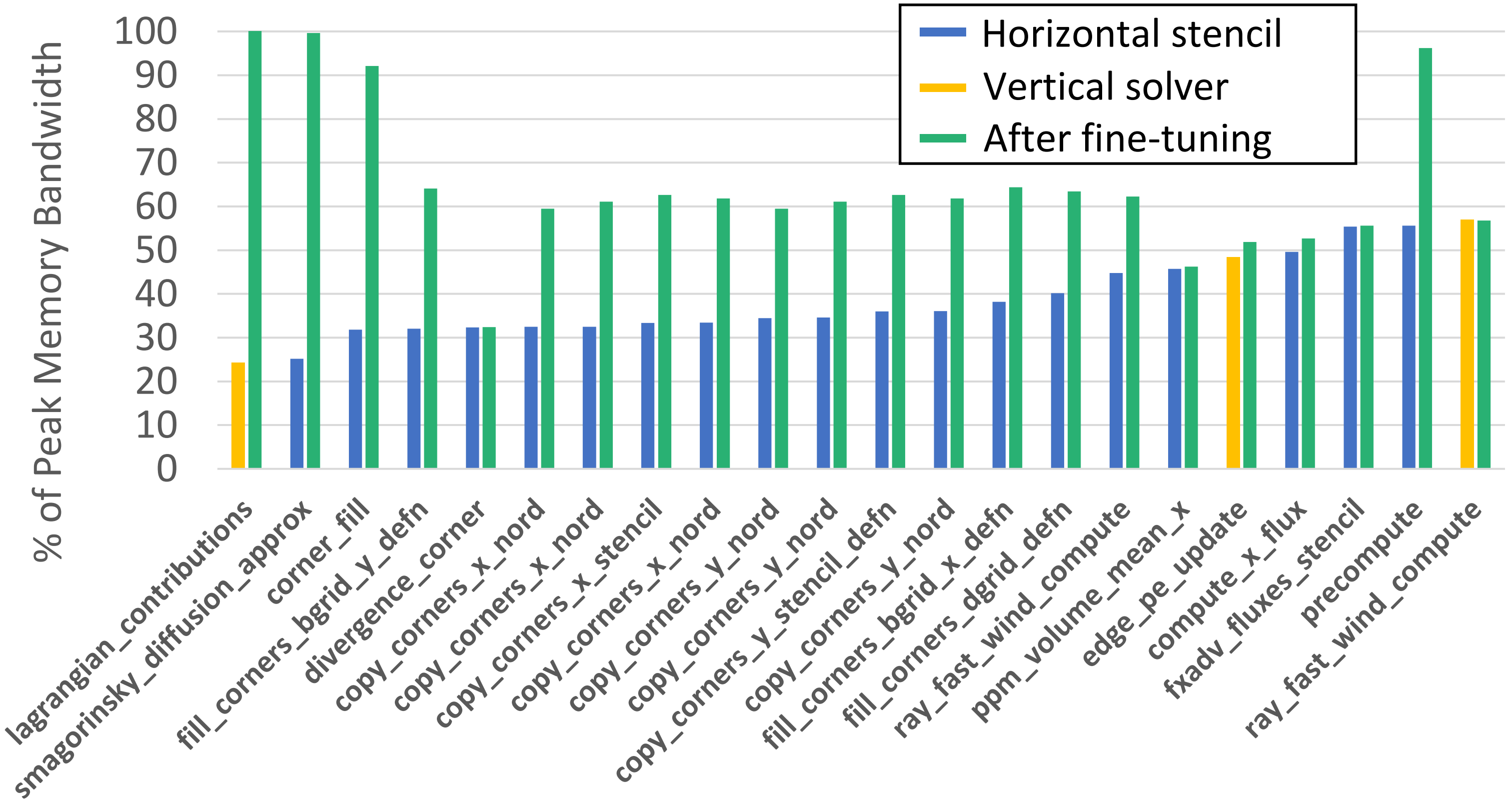

```
with computation(PARALLEL), interval(...):
  vort = dt * (delpc ** 2.0 + vort ** 2.0) ** 0.5
```




```
with computation(PARALLEL), interval(...):
  vort = dt * (delpc ** 2.0 + vort ** 2.0) ** 0.5
```




Initial Heuristics
Module-Based Autotuning
Transfer-Tune to Full Application
Benchmark, Generate Perf. Model
Suboptimal Kernel Inspection
Fine Tuning


% of Peak Memory Bandwidth
100
90
80
70
60
50
40
30
20
10
0
Horizontal stencil
Vertical solver
After fine-tuning
lagrangian_contributions
smagorinsky_diffusion_approx
corner_fill
fill_corners_bgrid_y_defn
divergence_corner
copy_corners_x_nord
copy_corners_x_nord
copy_corners_x_stencil
copy_corners_x_nord
copy_corners_y_nord
copy_corners_y_nord
copy_corners_y_stencil_defn
copy_corners_y_nord
fill_corners_bgrid_x_defn
fill_corners_dgrid_defn
ray_fast_wind_compute
ppm_volume_mean_x
edge_pe_update
compute_x_flux
fxadv_fluxes_stencil
precompute
ray_fast_wind_compute

Evaluated Systems

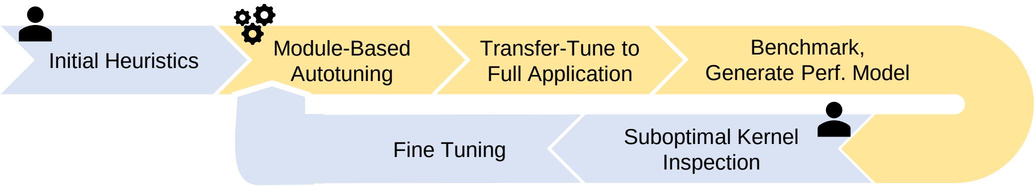


Photo courtesy of the Swiss National Supercomputing Centre

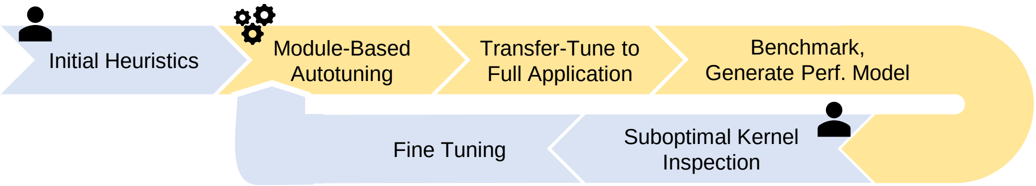
Photo courtesy of Forschungszentrum Jülich

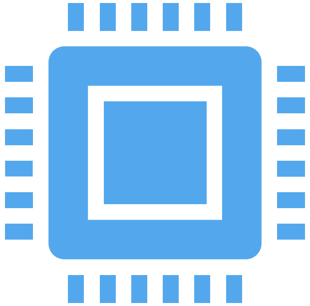
Piz Daint:
- GPU: 1 x NVIDIA Tesla P100 / Node
- CPU: Intel Xeon E5-2690 v3 (12 cores)

JUWELS Booster:
- GPU: 4 x NVIDIA Tesla A100 / Node
- CPU: AMD EPYC 7402 (2 sockets, 24 cores)

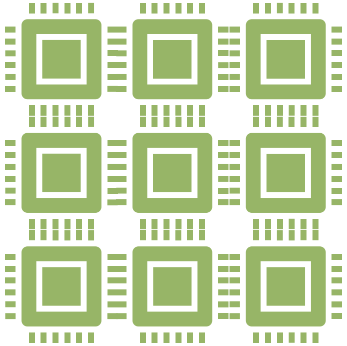
Domain size: 192x192x80

Memory Bounds





43.77 GB/s

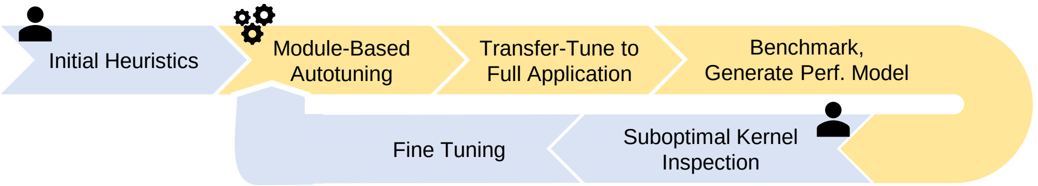


501.1 GB/s

Potential Speedup ≤ **11.45x**

Representative Vertical Solver

Riemann Solver (`riem_solver_c`)



Semi-implicit solver for nonhydrostatic terms of vertical velocity and pressure perturbation

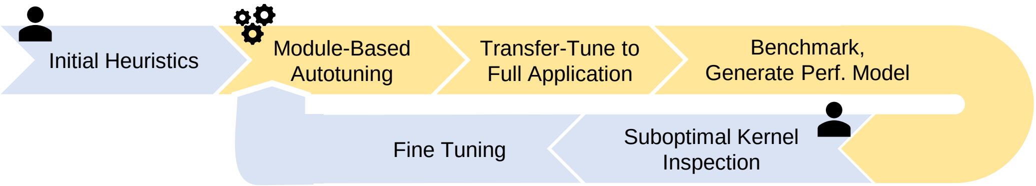
	FORTRAN		GT4Py+DaCe		
Domain Size (relative size)	Time [ms]	Scaling	Time [ms]	Scaling	Speedup
$128 \times 128 \times 80$ (1x)	12.27	—	1.85	—	6.63×
$192 \times 192 \times 80$ (2.25x)	27.94	2.28	3.86	2.08	7.25×
$256 \times 256 \times 80$ (4x)	52.40	4.27	6.96	3.76	7.53×
$384 \times 384 \times 80$ (9x)	121.80	9.92	15.31	8.26	7.96×

CPU cache runs out, data layout not ideal

Not enough parallelism

Representative Horizontal Stencil

Finite Volume Transport (`fv_tp_2d`)



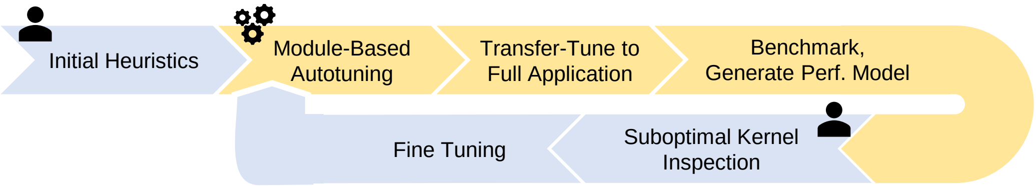
FORTRAN runs on a **single slice**, GT4Py/DaCe runs on entire 3D domain

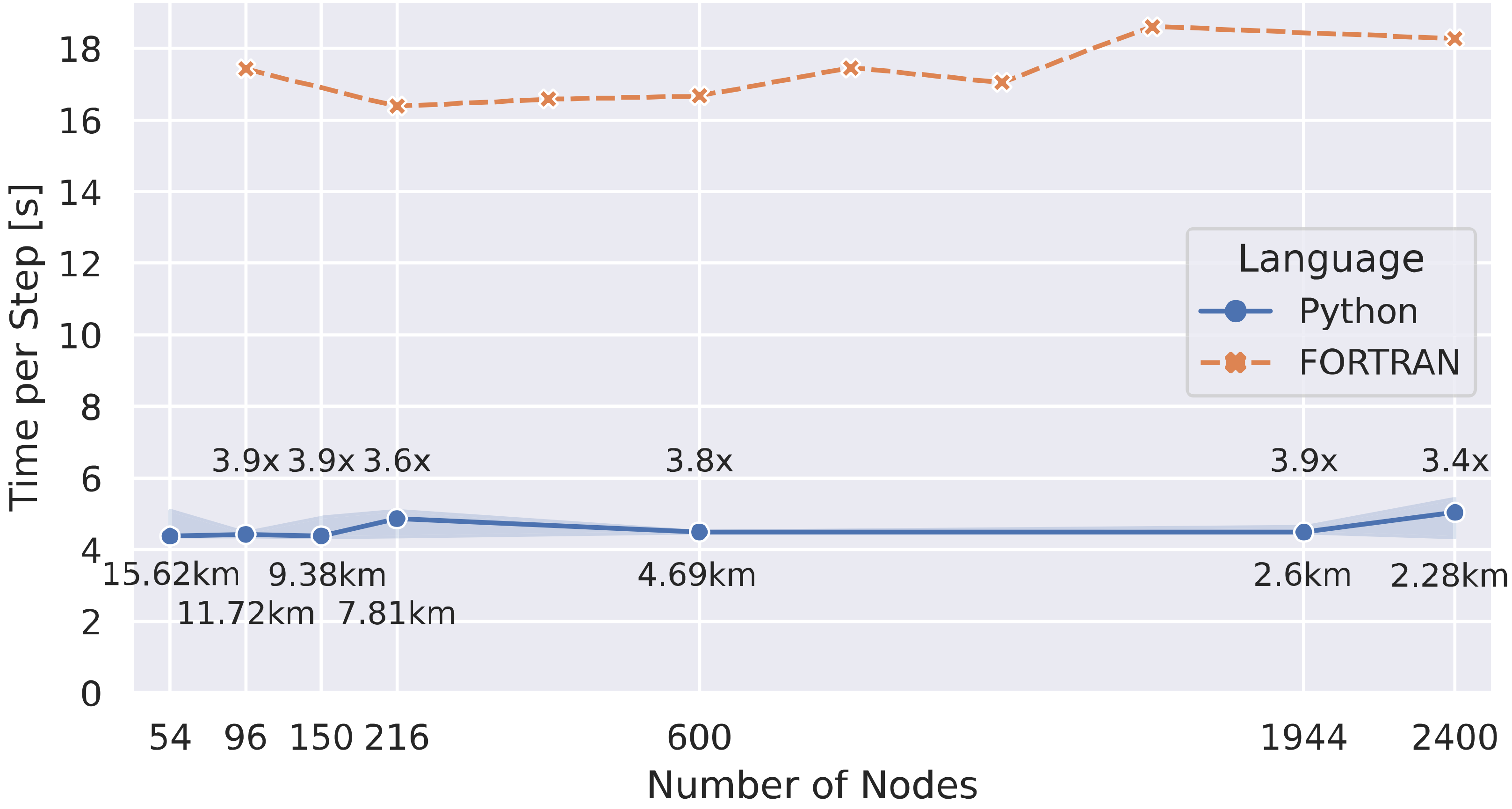
	FORTRAN		GT4Py+DaCe		
Domain Size (relative size)	Time [ms]	Scaling	Time [ms]	Scaling	Speedup
128×128×80 (1x)	3.41	—	1.81	—	1.88×
192×192×80 (2.25x)	12.31	3.61	3.41	1.88	3.61×
256×256×80 (4x)	35.79	10.49	5.67	3.13	6.31×
384×384×80 (9x)	106.66	31.27	13.10	7.23	8.14×

0.13% of load/stores are L3 misses

Closing gap to ideal memory bandwidth factor

Weak Scaling





Simulation throughput of **0.12 SYPD** at 2.6 km grid spacing

https://github.com/ai2cm/pace
https://github.com/GridTools/gt4py
https://github.com/spcl/dace

6
weeks of
work

10
optimization
revisions

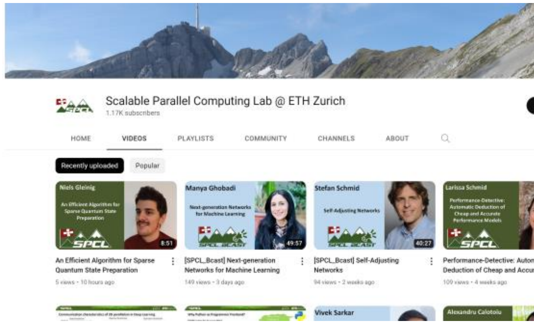
4
performance
engineers

3.92 – 8.48x
speedup vs.
production FORTRAN

0
model
changes

Want to know more?

youtube.com/@spcl



twitter.com/spcl_eth

spcl.inf.ethz.ch

github.com/spcl