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

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).
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(inout):: 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
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

- **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

Usage: python -m pace.driver.run [OPTIONS] CONFIG_PATH
Run the driver.
CONFIG_PATH is the path to a DriverConfig yaml file.
Options: ...

https://github.com/ai2cm/pace

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

J. Dahm et al., “Pace v0.1: A Python-based Performance-Portable Implementation of the FV3 Dynamical Core”. EGUSphere’22
Declarative Abstraction (GT4Py)

- Acoustics
- Tracer Advection

Dynamical Core
- Remapping

Unit Tests
- Halo Exchange

Horizontal Stencil
Vertical Solver

Backend
Calibacks

Local Optimization
Transfer Tuning
Full-Program Optimization

Orchestration (DaCe)
Scientific Computing is Moving to Python

- IPython
- jupyter
- GridTools
- SciPy
- TensorFlow
- seaborn
- matplotlib
- SymPy
- NetworkX
- PyTorch
```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)
        # ...
```

de12cubed.py

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
https://github.com/ai2cm/pace/blob/main/examples/notebooks/stencil_definition.ipynb
Declarative Abstraction (GT4Py)

- Acoustics
- Tracer Advection
- Dynamical Core
- Unit Tests
- Halo Exchange

- Horizontal Stencil
- Vertical Solver
- Orchestration (DaCe)
- Callbacks

- Local Optimization
- Transfer Tuning
- Full-Program Optimization
- Backend
GridTools for Python (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

```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
```

https://github.com/GridTools/gt4py
Declarative Abstraction (GT4Py)

Acoustics
Tracer Advection
Dynamical Core

Unit Tests
Halo Exchange

Horizontal Stencil
Vertical Solver

Declarative Abstraction (GT4Py)

Local Optimization
Transfer Tuning
Full-Program Optimization

Orchestration (DaCe)

Callbacks

Backend
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)
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)

High-performance optimization = data movement reduction

Kwasniewski et al., “Red-blue pebbling revisited: near optimal parallel matrix-matrix multiplication” SC’19
**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**

\[
L \
R
\]

**System**

**Hardware Information**

**Compiler**

**CPU Binary**

**GPU Binary**

**FPGA Modules**

**Performance Results**

**Transformed Dataflow**

---

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

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
GT4Py Backend

Stencil Implementations

Orchestration and Global Optimization
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)
            # ...

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
plot_on_map(state.x_wind)
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)
        # ...

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

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
validate(state)
plot_on_map(state.x_wind)
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)
        # ...

@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)

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)

deade

def dycore_loop(state, dycore, time_steps):
    for _ in range(time_steps):
        dycore.step_dynamics(state)
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”
Initial Heuristics

Aligned addresses

Pre-padding
$(o)$

$$o = a - h$$

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

Start offset: $o = a - h$

Strides:

$$s_i = 1$$

$$s_j = a \left\lfloor \frac{I + 2h}{a} \right\rfloor$$

$$s_k = s_j \cdot (J + 2h)$$
Initial Heuristics
Module-Based Autotuning
Transfer-Tune to Full Application

Exhaustive tuning on graph cutouts

[ {copy_corners_y_nord: 5}, ...
{compute_y_flux: 2, final_fluxes: 1} ]

Store top-k patterns

Without transfer tuning:
≥30,302,185 configurations

With transfer tuning:
603

2:42 hours on Piz Daint
8:24 hours
Initial Heuristics → Module-Based Autotuning → Transfer-Tune to Full Application → Benchmark, Generate Perf. Model

Horizontal stencil
Vertical solver

% of Peak Memory Bandwidth
Initial Heuristics

Module-Based Autotuning

Transfer-Tune to Full Application

Benchmark, Generate Perf. Model

Suboptimal Kernel Inspection

---

**with** computation(PARALLEL), interval(\ldots):

\[
\text{vort} = \text{dt} \times (\text{delpc} \times 2.0 + \text{vort} \times 2.0) \times 0.5
\]
Initial Heuristics → Module-Based Autotuning → Transfer-Tune to Full Application → Benchmark, Generate Perf. Model

Fine Tuning → Suboptimal Kernel Inspection

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

Fine Tuning -> Suboptimal Kernel Inspection

% of Peak Memory Bandwidth

- Blue: Horizontal stencil
- Orange: Vertical solver
- Green: After fine-tuning

<table>
<thead>
<tr>
<th>% of Peak Memory Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>90</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>70</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Legend:
- lagrangian_contributions
- smagorinsky_diffusion
- approx
- corner_fill
- byrd_defn
- divergence
- corner_x_nord
- copy
- corner_x_stencil
- copy
- corner_y_nord
- copy
- corner_y_defn
- fill
- corner_bnd
- defn
- pp
- edge
- precompute
- compute_x_flux
- compute
- update
- mean
- x
- compute
- x

After fine-tuning
Evaluated Systems

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

<table>
<thead>
<tr>
<th>Domain Size (relative size)</th>
<th>FORTRAN Time [ms]</th>
<th>FORTRAN Scaling</th>
<th>GT4Py+DaCe Time [ms]</th>
<th>GT4Py+DaCe Scaling</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>128×128×80 (1x)</td>
<td>12.27</td>
<td>—</td>
<td>1.85</td>
<td>—</td>
<td>6.63×</td>
</tr>
<tr>
<td>192×192×80 (2.25x)</td>
<td>27.94</td>
<td>2.28</td>
<td>3.86</td>
<td>2.08</td>
<td>7.25×</td>
</tr>
<tr>
<td>256×256×80 (4x)</td>
<td>52.40</td>
<td>4.27</td>
<td>6.96</td>
<td>3.76</td>
<td>7.53×</td>
</tr>
<tr>
<td>384×384×80 (9x)</td>
<td>121.80</td>
<td>9.92</td>
<td>15.31</td>
<td>8.26</td>
<td>7.96×</td>
</tr>
</tbody>
</table>

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

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

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
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?

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

youtube.com/@spcl
twitter.com/spcl_eth
spcl.inf.ethz.ch
github.com/spcl