Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

CONSORTIUM FOR SMALL SCALE MODELING
COSMO

ICON

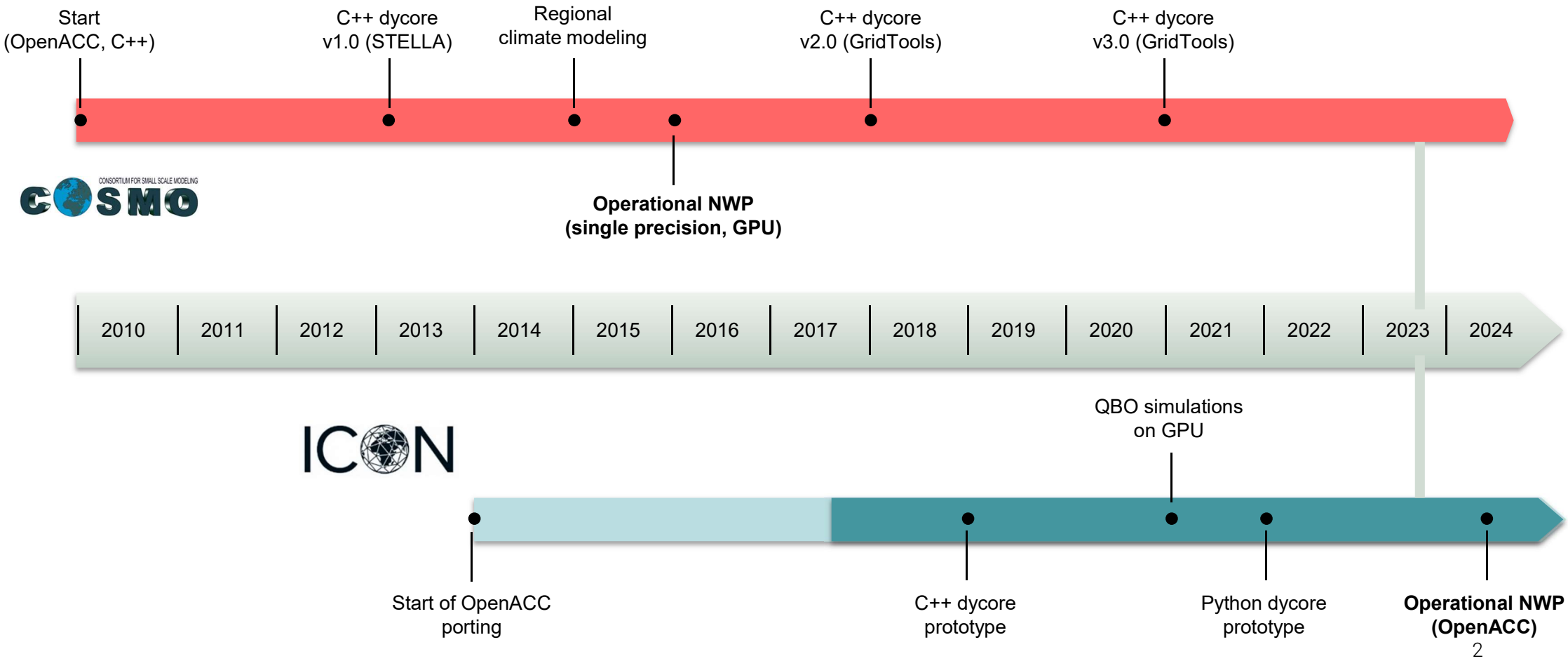
Enabling performance, productivity and portability by implementing weather and climate models in Python

Selection of on-going activities in the COSMO Consortium and ICON community

Oliver Fuhrer

EWGLAM/SRNWP Meeting 2023

The journey


Start
(OpenACC, C++)
C++ dycore
v1.0 (STELLA)
Regional
climate modeling
C++ dycore
v2.0 (GridTools)
C++ dycore
v3.0 (GridTools)
COSMO
CONSORTIUM FOR SMALL SCALE MODELING
Operational NWP
(single precision, GPU)
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
ICON
QBO simulations
on GPU
Start of OpenACC
porting
C++ dycore
prototype
Python dycore
prototype
Operational NWP
(OpenACC)

Priority Project IMPACT

Goal: Port ICON to GPUs

- Almost all components for operational ICON configuations ported to GPUs with OpenACC directives
- Running pre-operationally at MeteoSwiss starting Oct 2, 2023
- Project will be completed in December 2023

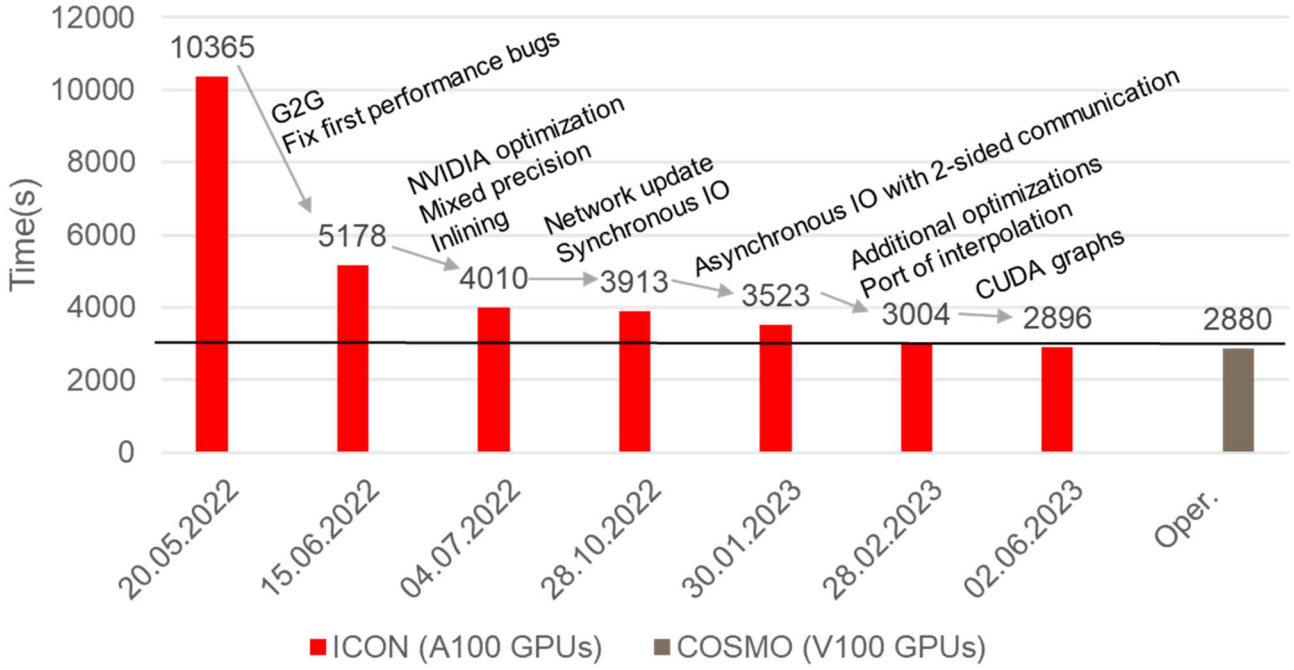
	ported	merged
nh_solve		
nh_hdiff		
transport		
2 way nesting		
convection		
microphysics (graupel)		
radiation		
radheat		
surface (terra)		
cover		
turbulence		
sea-ice		
SSO		
non-or. wave drag		
2 mom. microphysics		
NWP diagnostic		
Hailcast		
DA: LHN		
DA: conv. operatior		
DA: IAU (incr. anal. update)		
SPPT		
ICON-ART		
pollen		
EMVORADO		
ICON-HAMM		

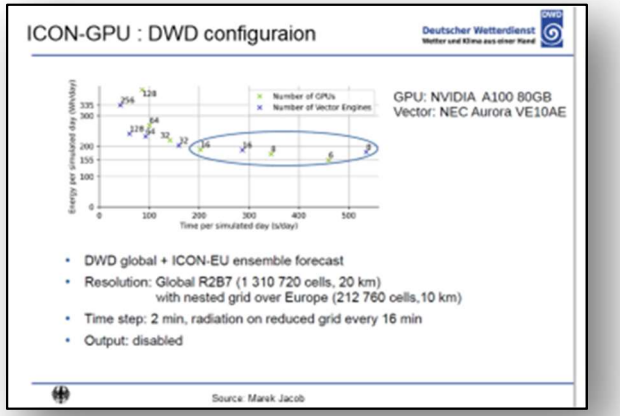
	completed
	in progress
	on CPU + data copy and interface ported
	not started

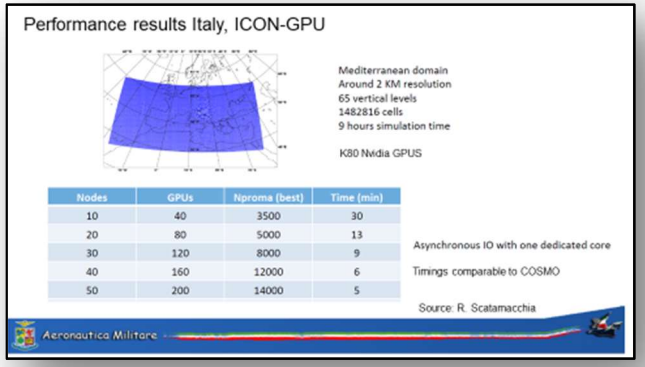
Main contributors
N. Burgdorfer[1], V. Cherkas[1], R. Dietlicher[1], E. Germann[1], F. Gessler[1], D. Hupp[1], X. Lapillonne[1], C. Müller[1], M. Röthlin[1], M. Stellio[1], G. Van Parys[1], G. Vollenweider[1], C. Osuna[1], A. Walser[1], M. Bettiol[3], R. Meli[3], A. Gopal[3], M. Jacob[4], A. Jocksch[3], J. Jucker[2], R. Meli[3], W. Sawyer[3], U. Schättler[4], D. Alexeev[5] , R. Scatamacchia
[1]MeteoSwiss, [2]C2SM, [3]CSCS, [4]DWD, [5]Nvidia, [6]ITAF Met

COSMO CONSORTIUM FOR SMALL SCALE MODELING ICON

Performance of ICON on GPU







Lessons learned

- Prioritize investments into software refactoring
- Plan on rewriting / refactoring your software multiple times
- Increase level of abstraction (no separation of concerns in Fortran + X)
- Choose a programming model that developers like
- Provide solutions for integrating with existing legacy code and incremental adoption

ICON-C - Consolidation Project

- ICON community infrastructure project to consolidate the ICON code and make it more **modular**, i.e. encapsulate the different components with clear/defined interfaces
- Allow for more **scalable development workflow**, in particular through an improved testing of individual components

Main contributors

Claudia Frauen; Joerg Behrens; Ralf Mueller; P. Samanta; Daniel Reinert; Florian Prill; Thomas Deppisch; Roland Wirth; Sergey Kosukhin; Luis Kornblueh; Reiner Schnur; Claudius Holeksa; Yen-Chen Chen ; Terry Cojean; William Sawyer ; Andreas Jocksch; Jonas Jucker; Christopher Bignamini; Xavier Lapillonne, Roland Potthast, Daniel Klocke, Hendryk Bockelmann, Gholamali Hoshyaripour, Oliver Fuhrer

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Deutscher Wetterdienst
Wetter und Klima aus einer Hand

MeteoSchweiz

Max-Planck-Institut für Meteorologie

KIT
Karlsruher Institut für Technologie

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

C2SM
Center for Climate Systems Modeling

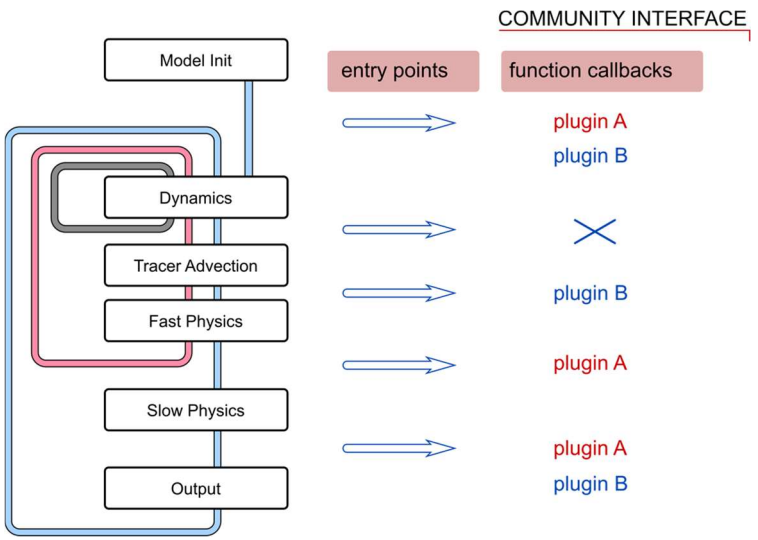
DKRZ

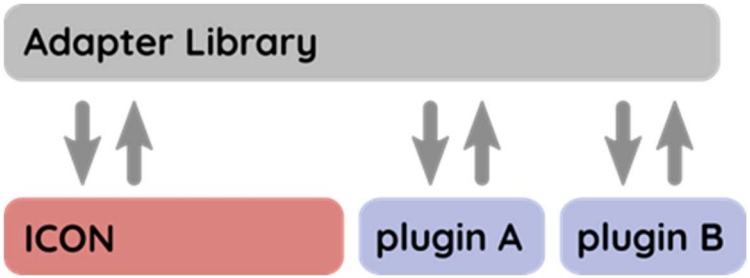
COSMO CONSORTIUM FOR SMALL SCALE MODELING

ICON

What are the aims of ComIn?

- Providing a standardized and stable **public interface** for third party codes (‘**plugins**’) coupled to ICON
- Significantly **reduced maintenance** for ICON as well as for third party code developers
- Plugins **easier to migrate** to new ICON releases
- Enables **multi-language support** (Fortran, C/C++, Python)



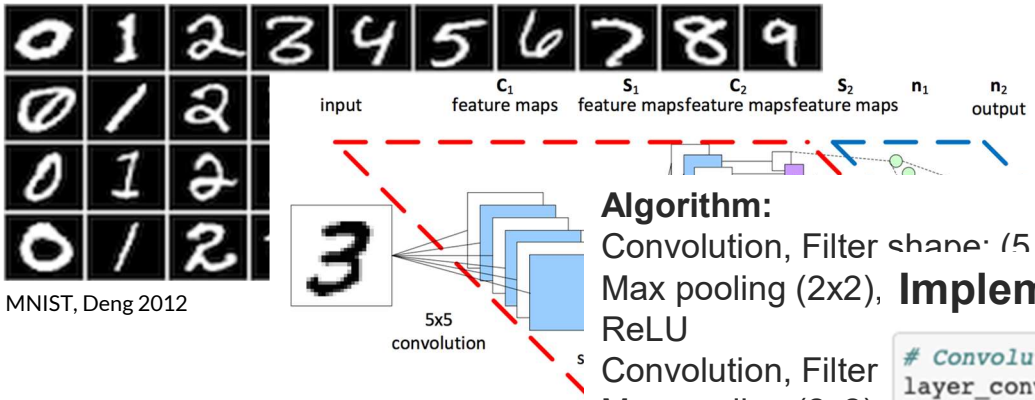


How does ComIn work in a nutshell?

- ComIn organizes the **data exchange** and **simulation events** between the ICON model and multiple plugins.
- **ComIn Callback Register**: Subroutines of the plugins are called at pre-defined events during the ICON simulation.
- The **ComIn Adapter Library** is included by ICON and the plugins. It contains descriptive data structures and regulates the access and the creation of model variables.

So what about Python?

Machine Learning Scientist



MNIST, Deng 2012

Algorithm:

Convolution, Filter shape: (5,5,6), Stride=1, Padding='SAME'
Max pooling (2x2),
ReLU
Convolution, Filter
Max pooling (2x2),
ReLU
Fully Connected L
ReLU
Fully Connected L
Softmax

Implementation (Python + Tensorflow):

```
# Convolutional Layer 1
layer_conv1, weights_conv1 = new_conv_layer(input=x_image, num_input_channels=1, fi
m_filters=6, name ="conv1")

# Pooling Layer 1
layer_pool1 = new_pool_layer(layer_conv1, name="pool1")

# RelU layer 1
layer_relu1 = new_relu_layer(layer_pool1, name="relu1")

# Convolutional Layer 2
layer_conv2, weights_conv2 = new_conv_layer(input=layer_relu1, num_input_channels=6
, num_filters=16, name= "conv2")

# Pooling Layer 2
layer_pool2 = new_pool_layer(layer_conv2, name="pool2")

# RelU layer 2
layer_relu2 = new_relu_layer(layer_pool2, name="relu2")

# Flatten Layer
num_features = layer_relu2.get_shape()[1:4].num_elements()
```

High-level of abstraction
Domain-specific language
Separation of concerns

COSMO CONSORTIUM FOR SMALL SCALE MODELING ICON

Atmospheric Scientist

$$\vec{u}(\vec{x}) = \sum_{i=1}^{k} \lambda_i \phi_i(\vec{x}) \vec{n}_i$$

Implementation (Fortran + OpenACC):

```
!$ACC PARALLEL &
!$ACC PRESENT( iqidx_d, ..., ptr_vn_d, e_flx_avg_d, vn_d, vt_d, rbf_vec_coeff_e_d )
!$ACC LOOP GANG PRIVATE( i_startidx, i_endidx, jb )
    DO jb = i_startblk, i_endblk
      IF ( i_startblk == jb ) THEN; i_startidx = e_startidx; ELSE; i_startidx = 1; ENDIF
      IF ( i_endblk == jb ) THEN; i_endidx = e_endidx; ELSE; i_endidx = nproma; ENDIF
!$ACC LOOP VECTOR COLLAPSE(2)
     DO je = i_startidx, i_endidx
        DO jk = 1, nlev
          iqidx_1 = iqidx_d(je,jb,1)
          ! Average normal wind components
          ptr_vn_d(je,jk,jb) = e_flx_avg_d(je,1,jb)*vn_now_d(je,jk,jb)&
            + e_flx_avg_d(je,2,jb)*vn_now_d(iqidx_1,jk,iqblk_1) &
            :
          ! RBF reconstruction of tangential wind component
          vt_now_d(je,jk,jb) = rbf_vec_coeff_e_d(1,je,jb)  &
            * vn_now_d(iqidx_1,jk,iqblk_1) &
            :
        ENDDO
      ENDDO
    ENDDO
!$ACC END PARALLEL
```

Low level of abstraction
General purpose language
No separation of concerns

CONSORTIUM FOR SMALL SCALE MODELING COSMO ICON

GT4Py

- Open-source, co-developed by CSCS, MeteoSwiss, and AI2
- https://github.com/gridtools/gt4py
- Build a “wrapper” around C++ library used to refactor COSMO
- Expose a productive, high-level language
- Inter-operate with Python ecosystem (e.g. NumPy)
- Often referred to as a **domain-specific language (DSL)**
- Activities at ECMWF (FVM), NOAA (X-SHiELD), NASA (GEOS)

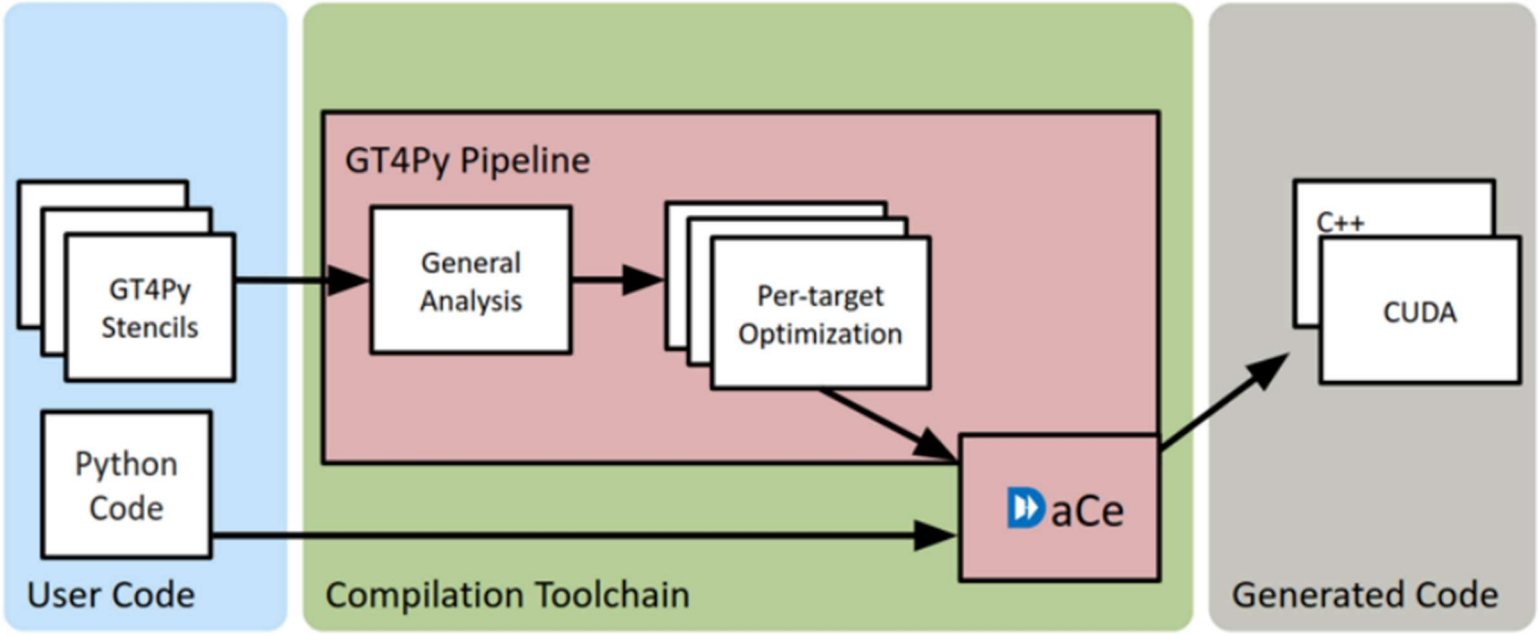
```python
import gt4py.gtscript as gtscript

@gtscript.function
def advection_x(dx, u, abs_u, phi):
    adv_phi_x = u[0, 0, 0] / (60. * dx) * (
        + 45. * (phi[1, 0, 0] - phi[-1, 0, 0])
        -  9. * (phi[2, 0, 0] - phi[-2, 0, 0])
        +       (phi[3, 0, 0] - phi[-3, 0, 0])
    ) - abs_u[0, 0, 0] / (60. * dx) * (
        +       (phi[3, 0, 0] + phi[-3, 0, 0])
        -  6. * (phi[2, 0, 0] + phi[-2, 0, 0])
        + 15. * (phi[1, 0, 0] + phi[-1, 0, 0])
        - 20. * phi[0, 0, 0]    )
    return adv_phi_x

@gtscript.stencil(backend="numpy")
def advection(
    in_u: gtscript.Field[np.float64],
    out_adv: gtscript.Field[np.float64],
    *,
    dx: float
):
    with computation(PARALLEL), interval(...):
        abs_u = absolute_value(phi=in_u)
        out_adv = advection_x(dx=dx, u=in_u,
                              abs_u=abs_u, phi=u)
```

Code generation

GT4Py code needs to be translated into NumPy or fast code in order to be executed



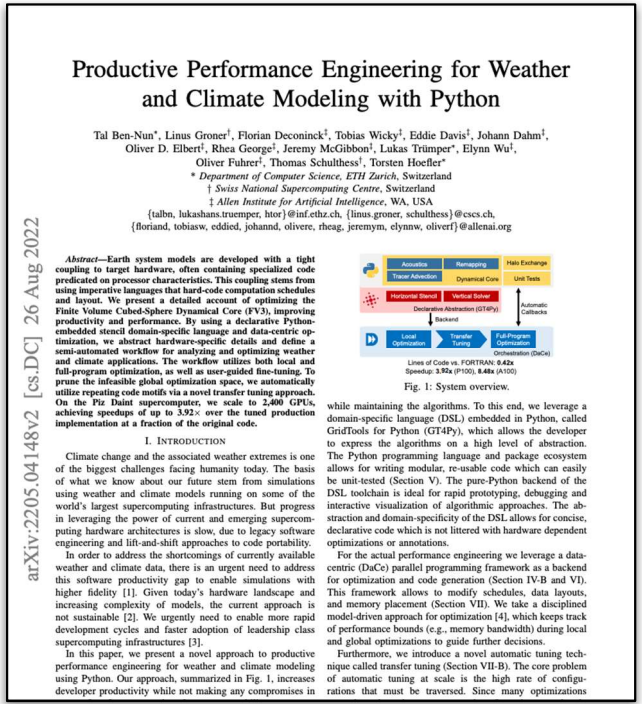
Similar to ML-frameworks or packages such as Numba, Cython, …

Demonstration: X-Shield Model

Rewrite X-SHiELD model using GT4Py

- dynamical core
- physical parameterizations
- Infrastructure

Project duration of 2 years



arXiv:2205.04148v2 [cs.DC] 26 Aug 2022

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*
* *Department of Computer Science, ETH Zurich*, Switzerland
† *Swiss National Supercomputing Centre*, Switzerland
‡ *Allen Institute for Artificial Intelligence*, WA, USA
{talbn, lukashans.truemper, htor}@inf.ethz.ch, {linus.groner, schulthess}@cscs.ch,
{floriand, tobiasw, eddied, johannd, olivere, rheag, jeremym, elynnw, oliverf}@allenai.org

***Abstract*—Earth system models are developed with a tight coupling to target hardware, often containing specialized code predicated on processor characteristics. This coupling stems from using imperative languages that hard-code computation schedules and layout. We present a detailed account of optimizing the Finite Volume Cubed-Sphere Dynamical Core (FV3), improving productivity and performance. By using a declarative Python-embedded stencil domain-specific language and data-centric optimization, we abstract hardware-specific details and define a semi-automated workflow for analyzing and optimizing weather and climate applications. The workflow utilizes both local and full-program optimization, as well as user-guided fine-tuning. To prune the infeasible global optimization space, we automatically utilize repeating code motifs via a novel transfer tuning approach. On the Piz Daint supercomputer, we scale to 2,400 GPUs, achieving speedups of up to 3.92× over the tuned production implementation at a fraction of the original code.**

Fig. 1: System overview.

I. Introduction

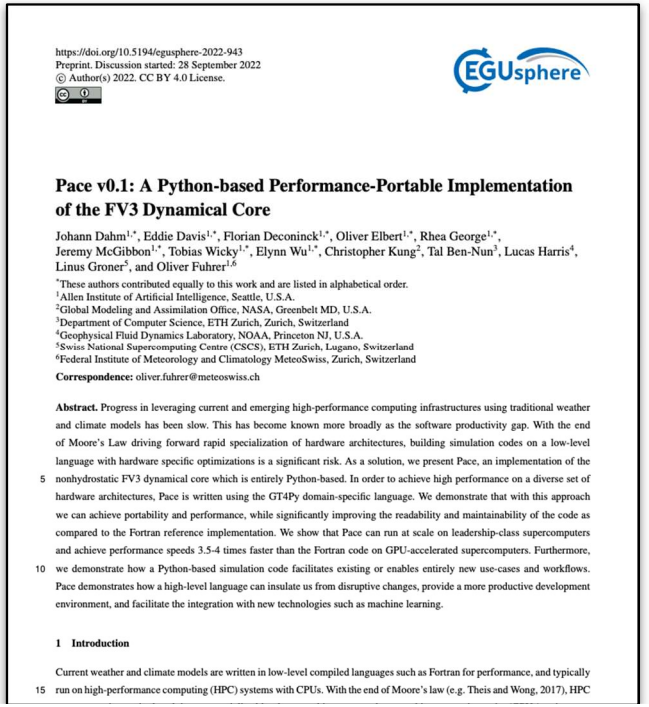
Climate change and the associated weather extremes is one of the biggest challenges facing humanity today. The basis of what we know about our future stem from simulations using weather and climate models running on some of the world's largest supercomputing infrastructures. But progress in leveraging the power of current and emerging supercomputing hardware architectures is slow, due to legacy software engineering and lift-and-shift approaches to code portability.

In order to address the shortcomings of currently available weather and climate data, there is an urgent need to address this software productivity gap to enable simulations with higher fidelity [1]. Given today's hardware landscape and increasing complexity of models, the current approach is not sustainable [2]. We urgently need to enable more rapid development cycles and faster adoption of leadership class supercomputing infrastructures [3].

In this paper, we present a novel approach to productive performance engineering for weather and climate modeling using Python. Our approach, summarized in Fig. 1, increases developer productivity while not making any compromises in while maintaining the algorithms. To this end, we leverage a domain-specific language (DSL) embedded in Python, called GridTools for Python (GT4Py), which allows the developer to express the algorithms on a high level of abstraction. The Python programming language and package ecosystem allows for writing modular, re-usable code which can easily be unit-tested (Section V). The pure-Python backend of the DSL toolchain is ideal for rapid prototyping, debugging and interactive visualization of algorithmic approaches. The abstraction and domain-specificity of the DSL allows for concise, declarative code which is not littered with hardware dependent optimizations or annotations.

For the actual performance engineering we leverage a data-centric (DaCe) parallel programming framework as a backend for optimization and code generation (Section IV-B and VI). This framework allows to modify schedules, data layouts, and memory placement (Section VII). We take a disciplined model-driven approach for optimization [4], which keeps track of performance bounds (e.g., memory bandwidth) during local and global optimizations to guide further decisions.

Furthermore, we introduce a novel automatic tuning technique called transfer tuning (Section VII-B). The core problem of automatic tuning at scale is the high rate of configurations that must be traversed. Since many optimizations



https://doi.org/10.5194/egusphere-2022-943
Preprint. Discussion started: 28 September 2022

EGUsphere

Pace v0.1: A Python-based Performance-Portable Implementation of the FV3 Dynamical Core

Johann Dahm[1,*], Eddie Davis[1,*], Florian Deconinck[1,*], Oliver Elbert[1,*], Rhea George[1,*], Jeremy McGibbon[1,*], Tobias Wicky[1,*], Elynn Wu[1,*], Christopher Kung[2], Tal Ben-Nun[3], Lucas Harris[4], Linus Groner[5], and Oliver Fuhrer[1,6]

[*]These authors contributed equally to this work and are listed in alphabetical order.
[1]Allen Institute of Artificial Intelligence, Seattle, U.S.A.
[2]Global Modeling and Assimilation Office, NASA, Greenbelt MD, U.S.A.
[3]Department of Computer Science, ETH Zurich, Zurich, Switzerland
[4]Geophysical Fluid Dynamics Laboratory, NOAA, Princeton NJ, U.S.A.
[5]Swiss National Supercomputing Centre (CSCS), ETH Zurich, Lugano, Switzerland
[6]Federal Institute of Meteorology and Climatology MeteoSwiss, Zurich, Switzerland

Correspondence: oliver.fuhrer@meteoswiss.ch

Abstract. Progress in leveraging current and emerging high-performance computing infrastructures using traditional weather and climate models has been slow. This has become known more broadly as the software productivity gap. With the end of Moore's Law driving forward rapid specialization of hardware architectures, building simulation codes on a low-level language with hardware specific optimizations is a significant risk. As a solution, we present Pace, an implementation of the nonhydrostatic FV3 dynamical core which is entirely Python-based. In order to achieve high performance on a diverse set of hardware architectures, Pace is written using the GT4Py domain-specific language. We demonstrate that with this approach we can achieve portability and performance, while significantly improving the readability and maintainability of the code as compared to the Fortran reference implementation. We show that Pace can run at scale on leadership-class supercomputers and achieve performance speeds 3.5-4 times faster than the Fortran code on GPU-accelerated supercomputers. Furthermore, we demonstrate how a Python-based simulation code facilitates existing or enables entirely new use-cases and workflows. Pace demonstrates how a high-level language can insulate us from disruptive changes, provide a more productive development environment, and facilitate the integration with new technologies such as machine learning.

1 Introduction

Current weather and climate models are written in low-level compiled languages such as Fortran for performance, and typically run on high-performance computing (HPC) systems with CPUs. With the end of Moore's law (e.g. Theis and Wong, 2017), HPC

“Comparison” on Piz Daint

X-SHiELD @ GT4Py (Dahm et al. 2023) runs at **0.073 SYPD** at 1.85 km, 80 levels, 4056 nodes, only microphysics

COSMO (Fuhrer et al. 2018) reported 0.23 SYPD at 1.9 km, 60 levels, 4888 nodes, only microphysics → *would scale to **~0.12 SYPD** for comparable simulation*

ICON (Giorgetta et al. 2022) reported 0.13 SYPD at 5 km, 191 levels, 1024 nodes, full physics → *would scale to **~0.068 SYPD** for comparable simulation*

IFS (Schulthess et al. 2018) reported 0.088 SYPD at 1.25 km, 62 levels, 4888 nodes, full physics → *would scale to **~0.20 SYPD** for comparable simulation*

Warning:  **vs.**  **comparison!!!**

EXCLAIM Project (ICON-X)

ETH zürich

EXCLAIM

EXascale **C**omputing platform for c**L**oud-resolving we**A**ther and c**l**imate **M**odels

C2SM Center for Climate Systems Modeling

CSCS

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

MeteoSwiss

SDSC

Empa
Materials Science and Technology

CONSORTIUM FOR SMALL SCALE MODELING COSMO

ICON

https://exclaim.ethz.ch/

ICON in GT4Py / Python

- Dynamical core, tracer advection and microphysics have been ported.
- Infrastructure for unit-testing and integration back into Fortran code-base
- Next step: Performance improvements (currently on-par with Fortran + OpenACC)

ICON-DSL dycore

COSMO CONSORTIUM FOR SMALL SCALE MODELING ICON

Extension to unstructured grids

Fortran + OpenMP + OpenACC

```
!$OMP PARALLEL PRIVATE(i_startblk,i_endblk)

      i_startblk = p_patch%edges%start_block(rl_start)
      i_endblk   = p_patch%edges%end_block(rl_end)

!$OMP DO PRIVATE(jb,i_startidx,i_endidx,jk,je,vn_vert1,vn_vert2,vn_vert3,vn_vert4, &
!$OMP            dvt_norm,dvt_tang), ICON_OMP_RUNTIME_SCHEDULE
      DO jb = i_startblk,i_endblk

        CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                           i_startidx, i_endidx, rl_start, rl_end)

        ! Computation of wind field deformation

!$ACC PARALLEL LOOP DEFAULT(NONE) GANG VECTOR COLLAPSE(2) ASYNC(1) IF( i_am_accel_node .AND. acc_on )
#ifdef __LOOP_EXCHANGE
        DO je = i_startidx, i_endidx
!DIR$ IVDEP
          DO jk = 1, nlev
#else
!$NEC outerloop_unroll(4)
        DO jk = 1, nlev
          DO je = i_startidx, i_endidx
#endif

            vn_vert1 =  u_vert(ividx(je,jb,1),jk,ivblk(je,jb,1)) * &
                        p_patch%edges%primal_normal_vert(je,jb,1)%v1 + &
                        v_vert(ividx(je,jb,1),jk,ivblk(je,jb,1)) * &
                        p_patch%edges%primal_normal_vert(je,jb,1)%v2

            vn_vert2 =  u_vert(ividx(je,jb,2),jk,ivblk(je,jb,2)) * &
                        p_patch%edges%primal_normal_vert(je,jb,2)%v1 + &
                        v_vert(ividx(je,jb,2),jk,ivblk(je,jb,2)) * &
                        p_patch%edges%primal_normal_vert(je,jb,2)%v2

            dvt_tang =  p_patch%edges%tangent_orientation(je,jb)* (   &
                        u_vert(ividx(je,jb,2),jk,ivblk(je,jb,2)) * &
                        p_patch%edges%dual_normal_vert(je,jb,2)%v1 + &
                        v_vert(ividx(je,jb,2),jk,ivblk(je,jb,2)) * &
                        p_patch%edges%dual_normal_vert(je,jb,2)%v2 - &
                        (u_vert(ividx(je,jb,1),jk,ivblk(je,jb,1)) * &
```

GT4Py Declarative

```
dvt_tang = (
    -(
        u_vert(E2C2V[0]) * dual_normal_vert_x(E2ECV[0])
        + v_vert(E2C2V[0]) * dual_normal_vert_y(E2ECV[0])
    )
) + (
    u_vert(E2C2V[1]) * dual_normal_vert_x(E2ECV[1])
    + v_vert(E2C2V[1]) * dual_normal_vert_y(E2ECV[1])
)
```

CONSORTIUM FOR SMALL SCALE MODELING COSMO

Summary

- ICON is available for production on GPU for NWP and climate applications (Fortran + OpenACC)
- On-going effort to improve software engineering, modularization and code maintainability (ICON-C)
- Rewrite of ICON in Python / GT4Py on-going
- Productivity is generally underappreciated

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

MeteoSchweiz
Operation Center 1
CH-8058 Zürich-Flughafen
T +41 58 460 91 11
www.meteoschweiz.ch

MeteoSvizzera
Via ai Monti 146
CH-6605 Locarno-Monti
T +41 58 460 92 22
www.meteosvizzera.ch

MétéoSuisse
7bis, av. de la Paix
CH-1211 Genève 2
T +41 58 460 98 88
www.meteosuisse.ch

MétéoSuisse
Chemin de l'Aérologie
CH-1530 Payerne
T +41 58 460 94 44
www.meteosuisse.ch