Exploring a high-level performance-portable programming model for the NWP domain

Stefano Ubbiali[1], Christian Kühnlein[2], Christoph Schär[1], Linda Schlemmer[3], Thomas C. Schulthess[4,5], Michael Staneker[2], and Heini Wernli[1]

[1]Institute for Atmospheric and Climate Science (IAC), ETH Zürich, Switzerland
[2]European Centre for Medium Range Weather Forecasts (ECMWF), Bonn, Germany
[3]Deutscher Wetterdienst (DWD), Offenbach, Germany
[4]Institute for Theoretical Physics (ITP), ETH Zürich, Switzerland
[5]Swiss National Supercomputing Center (CSCS), Lugano, Switzerland

Correspondence: Stefano Ubbiali (subbiali@phys.ethz.ch)

Abstract. We explore the domain-specific Python library GT4Py (GridTools for Python) for implementing a representative physical parametrization scheme and the related tangent-linear & adjoint algorithms from the Integrated Forecasting System (IFS) of ECMWF. GT4Py encodes stencil operators in an abstract and hardware-agnostic fashion, thus enabling more concise, readable and maintainable scientific applications. The library achieves high performance by translating the application into targeted low-level coding implementations. Here, the main goal is to study the correctness and performance-portability of the Python rewrites with GT4Py against the reference Fortran codes and a number of other (automatic and manual) porting approaches at ECMWF. The present work is part of a larger cross-institutional effort to port weather and climate models to Python with GT4Py. The focus of the current work is the IFS prognostic cloud microphysics scheme, a core physical parametrization represented by a comprehensive code that takes a significant share of the total forecast model execution time. In order to verify GT4Py for NWP systems, we put additional emphasis on the implementation and validation of the tangent-linear & adjoint model versions which are employed in data assimilation. We benchmark all prototype codes on three European supercomputers characterized by diverse GPU and CPU hardware, node designs, software stacks and compiler suites. Once the application is ported to Python with GT4Py, we find robust execution and portability, paired with competitive performance.

1 Introduction

Soon after its first public release in 1957, Fortran has become the language of choice for weather and climate models (Méndez et al., 2014). On the one hand, its functional programming style and built-in support for multi-dimensional arrays has granted Fortran large popularity in the whole scientific computing community. On the other, its low-level nature guarantees fast execution of intensive mathematical operations on vector machines and conventional Central Processing Units (CPUs). In the last decades, these characteristics have permitted to run weather forecasts several times per day under tight operational schedules on high-performance computing (HPC) systems (Neumann et al., 2019).

In recent years, in response to the simultaneous end of Moore's law and Dennard scaling, and under the pressure of the ongoing energy crisis, the computer hardware landscape has been undergoing a rapid specialization to prevent unsustainable growth of the power envelope (Müller et al., 2019). Emerging supercomputers are dominated by a heterogeneous node design, where energy-efficient accelerators such as Graphics Processing Units (GPUs) co-exist. Because Fortran has been conceived with CPU-centric machines in mind, efficient programming of hybrid HPC platforms using the core Fortran language can be challenging (Méndez et al., 2014; Lawrence et al., 2018). Indeed, the sustained performance of legacy weather and climate model codes written in Fortran has decreased over the decades (Schulthess et al., 2018), revealing the urgency for algorithmic and software adaptations to remain competitive in the medium and long term (Bauer et al., 2021).

Compiler directives are an attractive solution to spread a workload across multiple CPU threads, and to offload data and computations to GPU. The most famous incarnations of this programming paradigm are OpenMP (Dagum and Menon, 1998) and OpenACC (Chandrasekaran and Juckeland, 2017). Because compiler directives accommodate incremental porting and enable non-disruptive software development workflows, they are being adopted by many weather and climate modeling groups, who are facing the grand challenge of accelerating large code-bases with thousands of source files and millions of lines of code, which stem from decades of scientific discoveries and software developments (Lapillonne et al., 2017, 2020; Randall et al., 2022). The annotation of Fortran codes with compiler directives can be automated using tools such as the CLAW compiler (Clement et al., 2019) or the source-to-source translation tool Loki[1]. Nevertheless, in real-case scenarios compiler directives might provide a sub-optimal solution performance-wise: to run efficiently on the target hardware, the code may require additional invasive transformations,

[1]github.com/ecmwf-ifs/loki

which could finally lead to code duplication and thus worsen maintainability (Dahm et al., 2023). Moreover, polluting the scientific code with low-level instructions threatens the overall readability of the code.

On the contrary, domain-specific languages (DSLs) separate the code describing the science from the code actually executing on the target hardware, thus enabling *performance-portability*, namely application codes that achieve near-optimal performance on a variety of computer architectures (Deakin et al., 2019). Many modeling systems are being completely re-factored using multiple and diverse DSLs, not necessarily embedded in Fortran. For instance, the dynamical core of the weather prediction model from the COnsortium for Small-scale MOdeling (COSMO; Baldauf et al., 2011) has been re-written in C++ using the GridTools library (Afanasyev et al., 2021) to port stencil-based operators to GPUs (Fuhrer et al., 2014, 2018). Similarly, HOMMEXX-NH (Bertagna et al., 2020) is an architecture-portable C++ implementation of the non-hydrostatic dynamical core of the Energy Exascale Earth System model (E3SM; Taylor et al., 2020) harnessing the Kokkos library to express on-node parallelism (Edwards et al., 2014). The GungHo project blends the LFRic infrastructure with the PSyclone code generator (Adams et al., 2019) to develop a new dynamical core for the U.K. Met Office Unified Model (MetUM; Brown et al., 2012), which is amenable to massively parallel supercomputers with a hybrid node design. Pace (Ben-Nun et al., 2022; Dahm et al., 2023) is a Python re-write of the Finite-Volume Cubed-Sphere Dynamical Core (FV3; Harris and Lin, 2013) using the GridTools for Python framework (GT4Py) to accomplish performance-portability. GT4Py is being explored also in other separate projects shaping the next iteration of the non-hydrostatic dynamical core FVM (Kühnlein et al., 2023) and the dynamical kernel of the ICOsahedral Non-hydrostatic modeling framework (ICON; Zängl et al., 2015, https://exclaim.ethz.ch/).

The common trait of all the portability efforts mentioned previously is their focus on the model *dynamics*, that is the part of the model solving the fluid-dynamics equations governing the evolution of atmospheric flows. The rationale is that the dynamics houses the majority of the parallel communications and takes the largest fraction of the model runtime (Bertagna et al., 2020). Notwithstanding, the performance of the physical parameterizations, addressing the bulk effect of diverse subgrid-scale processes on the resolved flow and forming the so-called *physics* of the model, should not be overlooked. Although there is large evidence that the vertical transport (i.e., convection) of moisture, momentum and heat can be partially resolved when approaching horizontal grid resolutions in the order of 1 km, thus avoiding the parameterization of deep convection (e.g. Miyamoto et al., 2013; Neumann et al., 2019; Schär et al., 2019; Stevens et al., 2019), it is equally clear that micro-scale dynamical processes (e.g., turbulent motions) and non-dynamical processes like radiation and cloud microphysics will be parameterized well beyond the kilometer-resolution. Parameterizations are being commonly ported to accelerators using OpenACC (e.g. Fuhrer et al., 2014; Yang et al., 2019; Kim et al., 2021). Wrappers around low-level legacy physics codes might then be designed to facilitate adoption within higher-level workflows (Monteiro et al., 2018; McGibbon et al., 2021). Lately, first efforts to fully re-factor physical parameterizations in a portable-fashion have been documented in the literature. For instance, Watkins et al. (2023) presented a re-write of the MPAS-Albany Land Ice (MALI) ice-sheet model using Kokkos. Here, we present a novel Python implementation for the cloud microphysics schemes CLOUDSC and CLOUDSC2, which are part of the physics suite of the Integrated Forecasting System (IFS), the flagship of the European Centre for Medium Range Weather Forecasts (ECMWF). Details on the formulation and validation of the schemes are discussed in Section 2. The proposed Python implementations build upon the GT4Py toolchain, and in the remainder of the paper we use the terms CLOUDSC-GT4Py and CLOUDSC2-GT4Py to refer to the GT4Py re-write of CLOUDSC and CLOUDSC2 (in all its three variants). The working principles of the GT4Py framework are illustrated in Section 3, where we also advocate the advantages offered by domain-specific approaches in scientific software development. Section 4 sheds some light on the infrastructure code, and how it can enable composable and re-usable model components. In Section 5, we compare the performance of CLOUDSC-GT4Py and CLOUDSC2-GT4Py, as measured on three leadership-class GPU-equipped supercomputers, to established implementations in Fortran and CUDA C. We conclude the paper with final remarks and future development paths.

2 Overview of the IFS Physics Suite

Several physical and chemical mechanisms occurring in the atmosphere are active on spatial scales that are significantly smaller than the highest affordable model resolution. It follows that these mechanisms cannot be properly captured by the resolved model dynamics, but need to be *parameterized*. Parameterizations express the bulk effect of subgrid-scale phenomena on the resolved flow in terms of the grid-scale variables. The equations underneath physical parameterizations are based on theoretical and semi-empirical arguments and their numerical treatment commonly adheres to the *single-column* abstraction, so that adjustments can only happen within individual columns, with no data dependencies between columns. The atmospheric module of the IFS includes parameterizations dealing with the radiative heat transfer (ecRad; Hogan and Bozzo, 2018), deep and shallow convection (Tiedtke, 1989; Bechtold et al., 2008, 2014), clouds and stratiform precipitation (Forbes and Tompkins, 2011; Forbes et al., 2011), surface exchange (Balsamo et al., 2009), turbulent mixing in the planetary boundary layer (Köhler et al., 2011), subgrid-scale orographic drag (Lott and Miller, 1997; Beljaars et al., 2004), non-orographic gravity wave drag (Orr et al., 2010), and methane oxidation (Monge-Sanz et al., 2013). The focus of this paper is on the cloud microphysics modules of the ECMWF: the CLOUDSC – used in operational forecasting – and the CLOUDSC2 – employed in the data assimilation. The motivation is two-fold: both schemes are representative of the computational patterns ubiquitous in physical parameterizations, and they are among the most computationally expensive parameterizations, with the CLOUDSC accounting for up to 10% of the total execution time of the high-resolution operational forecasts at ECMWF.

2.1 CLOUDSC: Cloud Microphysics Forecast Model

The CLOUDSC is a single-moment cloud microphysics scheme that parameterizes stratiform clouds and their contribution to surface precipitation (ECMWF, 2023). It was implemented in the IFS Cycle 36r4 and has been operational at ECMWF since November 2010. Compared to the pre-existing scheme, it accounts for five prognostic variables (cloud fraction, cloud liquid water, cloud ice, rain and show) and brings substantial enhancements in different aspects, including treatment of mixed-phase clouds, advection of precipitating hydrometeors (rain and snow), physical realism, and numerical stability (Nogherotto et al., 2016). For a comprehensive description of the scheme, we refer the reader to Forbes et al. (2011) and the references therein. Although the original CLOUDSC integrates the governing equations for the five variables at consideration using an implicit scheme, all the codes considered in this paper, including the novel Python re-write, do not encompass the time integration, but are limited to the computation of the tendencies for the prognostic variables and the retrieval of additional diagnostics. For each coding version using either single or double precision computations, the calculations are validated by direct comparison of the output against serialized language-agnostic reference data.

2.2 CLOUDSC2: Cloud Microphysics in the Context of Data Assimilation

The CLOUDSC2 scheme represents a streamlined version of CLOUDSC, devised for use in the four-dimensional variational assimilation (4D-Var) at ECMWF (Courtier et al., 1994). 4D-Var merges short-term model integrations with observations over a twelve-hour assimilation window to determine the best possible representation of the current state of the atmosphere. This then provides the initial conditions for longer-term forecasts (Janisková and Lopez, 2023). The optimal synthesis between model and observational data is found by minimizing a cost function, which is evaluated using the *tangent-linear* of the *non-linear* forecasting model, while the *adjoint* model is employed to compute the gradient of the cost function (Errico, 1997; Janisková et al., 1999). For the sake of computational economy, the tangent-linear and adjoint operators are derived from a simplified and regularized version of the full non-linear model. The CLOUDSC2 is one of the physical parameterizations included in the ECMWF's simplified model, together with radiation, vertical diffusion, orographic wave drag, moist convection, and non-orographic gravity wave activity (Janisková and Lopez, 2023). In the following, we provide a mathematical and algorithmic representation of the tangent-linear and adjoint versions of CLOUDSC2.

Let $F : \boldsymbol{x} \mapsto \boldsymbol{y}$ be the functional description of CLOUDSC2, connecting the input fields $\boldsymbol{x}$ with the output variables $\boldsymbol{y}$. The tangent-linear operator F' of F is derived from the Taylor series expansion

$$F\left(\boldsymbol{x} + \delta\boldsymbol{x}\right) = \boldsymbol{y} + \delta\boldsymbol{y} = F\left(\boldsymbol{x}\right) + F'\left[\boldsymbol{x}\right]\left(\delta\boldsymbol{x}\right) + \mathcal{O}\left(||\delta\boldsymbol{x}||^2\right), \tag{1}$$

where $\delta\boldsymbol{x}$ and $\delta\boldsymbol{y}$ are variations on $\boldsymbol{x}$ and $\boldsymbol{y}$, and $||\cdot||$ is a suitable norm. The formal correctness of the coding implementation of F' can be assessed through the Taylor test (also called the "V-shape" test), which ensures that the following condition is satisfied up to machine precision:

$$\lim_{\lambda \to 0} \frac{F\left(x + \lambda\delta\boldsymbol{x}\right) - F\left(\boldsymbol{x}\right)}{F'\left[\boldsymbol{x}\right]\left(\lambda\delta\boldsymbol{x}\right)} = 1 \qquad \forall \boldsymbol{x}, \delta\boldsymbol{x}\,. \tag{2}$$

The logical steps carried out in the actual implementation of the Taylor test are sketched in Algorithm 1.

The adjoint operator F^* of F' is defined such that for the inner product $<\cdot,\cdot>$:

$$<\delta\boldsymbol{x}, F^*\left[\boldsymbol{y}\right]\left(\delta\boldsymbol{y}\right)> = <\delta\boldsymbol{y}, F'\left[\boldsymbol{x}\right]\left(\delta\boldsymbol{x}\right)> \qquad \forall \boldsymbol{x}, \delta\boldsymbol{x}, \boldsymbol{y}, \delta\boldsymbol{y}\,. \tag{3}$$

In particular, (3) must hold for $\boldsymbol{y} = F\left(\boldsymbol{x}\right)$ and $\delta\boldsymbol{y} = F'\left[\boldsymbol{x}\right]\left(\delta\boldsymbol{x}\right)$:

$$<\boldsymbol{x}, F^*\left[F\left(\boldsymbol{x}\right)\right]\left(F'\left[\boldsymbol{x}\right]\left(\delta\boldsymbol{x}\right)\right)> = <F'\left[\boldsymbol{x}\right]\left(\delta\boldsymbol{x}\right), F'\left[\boldsymbol{x}\right]\left(\delta\boldsymbol{x}\right)> \qquad \forall \boldsymbol{x}, \delta\boldsymbol{x}\,. \tag{4}$$

The latter condition is at the hearth of the so-called symmetry test for F^* (see Algorithm 2).

2.3 Cloud Microphysics Dwarfs

The ECMWF has established in the community the concept of weather and climate "computational dwarfs", or simply "dwarfs". These are model components shaped into stand-alone software packages to serve as archetypes of relevant computational motifs (Müller et al., 2019). Dwarfs are primarily intended to provide a convenient platform for

Algorithm 1 The Taylor test assessing the formal correctness of the coding implementation of the tangent-linear formulation of CLOUDSC2, denoted as CLOUDSC2TL. The three-dimensional arrays $\mathbf{x}$ and $\mathbf{y}$ collect the grid point values for all *nin* input fields and *nout* output fields of CLOUDSC2, respectively. The corresponding variations are $\delta\mathbf{x}$ and $\delta\mathbf{y}$. The grid consists of *ncol* columns, each containing *nlev* vertical levels. Note that compared to its functional counterpart $F'[\boldsymbol{x}] : \delta\boldsymbol{x} \mapsto \delta\boldsymbol{y}$, CLOUDSC2TL$(\mathbf{x}, \delta\mathbf{x})$ returns both $\mathbf{y}$ and $\delta\mathbf{y}$. The coding implementation of the non-linear CLOUDSC2 is indicated as CLOUDSC2NL.

```
function TotalNorm($ncol, nlev, nout, \mathbf{y}, \mathbf{y}_j, \delta\mathbf{y}_j$)    ▷ $\mathbf{y}, \mathbf{y}_j, \delta\mathbf{y}_j \in \mathbb{R}^{ncol \times nlev \times nout}$
    $total\_norm \leftarrow 0$
    $total\_count \leftarrow 0$
    for $l \leftarrow 1$ to $nout$ do
        $\beta \leftarrow \left| \sum_{i=1}^{nlev} \sum_{k=1}^{ncol} \delta\mathbf{y}_j(i,k,l) \right|$
        if $\beta > 0$ then
            $total\_norm \leftarrow total\_norm + \left| \sum_{i=1}^{nlev} \sum_{k=1}^{ncol} (\mathbf{y}_j(i,k,l) - \mathbf{y}(i,k,l)) \right| / \beta$
            $total\_count \leftarrow total\_count + 1$
    if $total\_count > 0$ then
        return $total\_norm / total\_count$
    else
        return 0

procedure TaylorTest($ncol, nlev, nin, nout, \mathbf{x}$)    ▷ $\mathbf{x} \in \mathbb{R}^{ncol \times nlev \times nin}$
    $\delta\mathbf{x} \leftarrow 0.01 * \mathbf{x}$
    $(\mathbf{y}, \delta\mathbf{y}) \leftarrow$ Cloudsc2TL($\mathbf{x}, \delta\mathbf{x}$)    ▷ $\mathbf{y}, \delta\mathbf{y} \in \mathbb{R}^{ncol \times nlev \times nout}$
    $norms \leftarrow ()$
    $jstart \leftarrow 1$
    for $j \leftarrow 1$ to 10 do
        $\mathbf{y}_j \leftarrow$ Cloudsc2NL($\mathbf{x} + 10^{-j} * \delta\mathbf{x}$)
        $norms \leftarrow norms \cup \left(1 - \text{TotalNorm}(ncol, nlev, nout, \mathbf{y}, \mathbf{y}_j, 10^{-j} * \delta\mathbf{y})\right)$
        if $jstart = 1$ & $norms(j) < 0.5$ then
            $jstart \leftarrow j$
    $test \leftarrow -10$
    $negat \leftarrow$ True
    for $j \leftarrow jstart$ to 9 do
        if $negat$ & $norms(j+1) \geq norms(j)$ then
            $test \leftarrow test + 10$
        $negat \leftarrow norms(j+1) < norms(j)$
    if $test = -10$ then
        $test \leftarrow 11$
    if $\min_{jstart \leq j \leq 10}(norms(j)) > 10^{-5}$ then
        $test \leftarrow test + 7$
    if $\min_{jstart \leq j \leq 10}(norms(j)) > 10^{-6}$ then
        $test \leftarrow test + 5$
    if $test \leq 5$ then
        print "The Taylor test passed."
    else
        print "The Taylor test failed."
```

Algorithm 2 The symmetry test assessing the formal correctness of the coding implementation of the adjoint formulation of CLOUDSC2, denoted as CLOUDSC2AD. The machine epsilon is indicated as ε; all other symbols have the same meaning as in Algorithm 1. Note that compared to its functional counterpart $F^* [F(\boldsymbol{x})] : \delta \boldsymbol{y} \mapsto \delta \boldsymbol{x}^*$, CLOUDSC2AD($\mathbf{x}$, $\delta\mathbf{y}$) returns both $\mathbf{y}$ and $\delta\mathbf{x}^*$.

```
 1: function ColumnWiseInnerProduct($ncol$, $nlev$, $ndim$, $\mathbf{a}$, $\mathbf{b}$)        ▷ $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{ncol \times nlev \times ndim}$
 2:     $\boldsymbol{c} \leftarrow \mathbf{0} \in \mathbb{R}^{ncol}$
 3:     for $l \leftarrow 1$ to $ndim$ do
 4:         for $i \leftarrow 1$ to $ncol$ do
 5:             $\boldsymbol{c}(i) \leftarrow \boldsymbol{c}(i) + \sum_{k=1}^{ncol} \mathbf{a}(i,k,l) * \mathbf{b}(i,k,l)$
 6:     return $\boldsymbol{c}$

 7: procedure SymmetryTest($ncol$, $nlev$, $nin$, $nout$, $\mathbf{x}$, $\varepsilon$)        ▷ $\mathbf{x} \in \mathbb{R}^{ncol \times nlev \times nin}$
 8:     $\delta\mathbf{x} \leftarrow 0.01 * \mathbf{x}$
 9:     $(\mathbf{y}, \delta\mathbf{y}) \leftarrow$ Cloudsc2TL($\mathbf{x}$, $\delta\mathbf{x}$)        ▷ $\mathbf{y}, \delta\mathbf{y} \in \mathbb{R}^{ncol \times nlev \times nout}$
10:     $\boldsymbol{c}_{\mathbf{y}} \leftarrow$ ColumnWiseInnerProduct($ncol$, $nlev$, $nout$, $\delta\mathbf{y}$, $\delta\mathbf{y}$)
11:     $(\mathbf{y}, \delta\mathbf{x}^*) \leftarrow$ Cloudsc2AD($\mathbf{x}$, $\delta\mathbf{y}$)        ▷ $\mathbf{x}^*, \delta\mathbf{x}^* \in \mathbb{R}^{ncol \times nlev \times nin}$
12:     $\boldsymbol{c}_{\mathbf{x}} \leftarrow$ ColumnWiseInnerProduct($ncol$, $nlev$, $nin$, $\delta\mathbf{x}$, $\delta\mathbf{x}^*$)
13:     $success \leftarrow$ True
14:     for $i \leftarrow 1$ to $ncol$ do
15:         if $\boldsymbol{c}_{\mathbf{x}}(i) = 0$ then
16:             $c \leftarrow |\boldsymbol{c}_{\mathbf{y}}(i)| / \varepsilon$
17:         else
18:             $c \leftarrow |\boldsymbol{c}_{\mathbf{y}}(i) - \boldsymbol{c}_{\mathbf{x}}(i)| / |\varepsilon * \boldsymbol{c}_{\mathbf{x}}(i)|$
19:         $success \leftarrow success$ & $c < 10^3$
20:     if $success$ then
21:         print "The symmetry test passed."
22:     else
23:         print "The symmetry test failed."
```

performance optimizations and portability studies (Bauer et al.). In recent years, the ECMWF created the CLOUDSC and CLOUDSC2 dwarfs as prototypes of physical parameterizations used in NWP. The original Fortran codes for CLOUDSC and CLOUDSC2, corresponding respectively to the IFS Cycle 41r2 and 46r1, have been pulled out of the IFS codebase, slightly polished and finally made available in public code repositories[2]. Later, the repositories have been enriched with alternative coding implementations, using different languages and programming paradigms; the most relevant implementations will be discussed in Section 5. The source codes for CLOUDSC-GT4Py and CLOUDSC2-GT4Py live in separate forks of the CLOUDSC and CLOUDSC2 repositories, and will not be merged until the licensing model of GT4Py complies with the long-established licensing model of ECMWF software. We are optimistic that a solution for the licensing issue that satisfies both parties will be found soon.

[2]https://github.com/ecmwf-ifs/dwarf-p-cloudsc and https://github.com/ecmwf-ifs/dwarf-p-cloudsc2-tl-ad

3 A Domain-Specific Approach to Scientific Software Development

In scientific software development, it is common practice to conceive a first proof-of-concepts implementation of a numerical algorithm in a high-level programming environment like MATLAB/Octave (Lindfield and Penny, 2018), or Python. Because these languages do not require any compilation, they provide a breeding ground for fast prototyping. However, the direct adoption of interpreted languages in HPC has historically been hindered by their intrinsic slowness. To squeeze more performance out of the underlying silicon, the initial proof-of-concept is translated into either Fortran, C or C++. This leads to the so-called "two-language problem", where the programming language used for the germinal prototyping is abandoned in favor of a faster language that might be more complicated to use. The lower-level code can be parallelized for shared memory platforms using OpenMP directives, while distributed memory machines can be targeted using Message Passing Interface (MPI) libraries to handle data movements between remote memories. The resulting code can later be migrated to GPUs, offering outstanding compute throughput and memory bandwidth especially for Single Instruction Multiple Data (SIMD) applications. GPU porting is accomplished using either OpenACC or OpenMP directives, or via a CUDA re-writing. To efficiently run the model at scale on multiple GPUs, a CUDA-aware MPI build should be chosen, so to avoid costly memory transfers between host and device.

The schematic visualization on the left side of Fig. 1 highlights how the above workflow leads to multiple coding versions of the same scientific application. This unavoidably complicates software maintainability: ideally, any modification in the model should be encoded in all implementations, so to preserve the coherency across the hierarchy. The maintainability problem is exacerbated as the number of lines of code, the pool of platforms to support, and the user-base increase. This situation has been known as the "software productivity gap" (Lawrence et al., 2018), and we argue that it cannot be alleviated by relying on general-purpose programming paradigms and monolithic code designs. Instead, it calls for a more synergistic collaboration between domain scientists (which here include model developers, weather forecasters, and weather and climate scientists) and computer experts. A path forward is provided by DSLs through *separation of concerns* (right side of Fig. 1), so that domain scientists can express the science using syntactic constructs that are aligned with the semantics of the application domain and abstract away all the architecture-specific details. The resulting source code is thus hardware-agnostic, more concise and easier to read. A toolchain developed by software engineers then employs automatic code generation techniques to synthesize optimized parallel code for the target computer architecture in a transparent fashion.

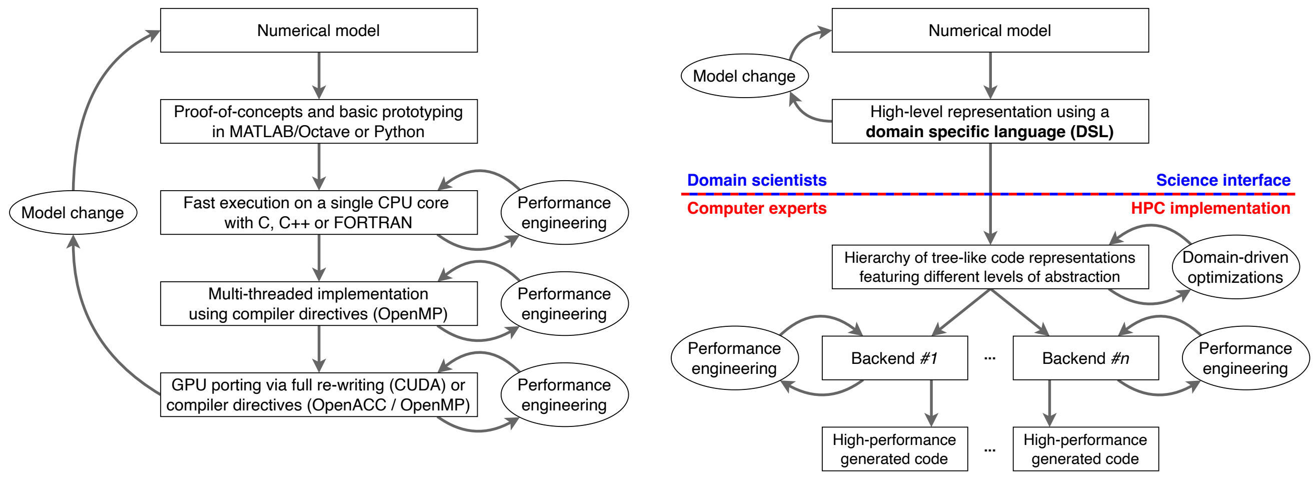
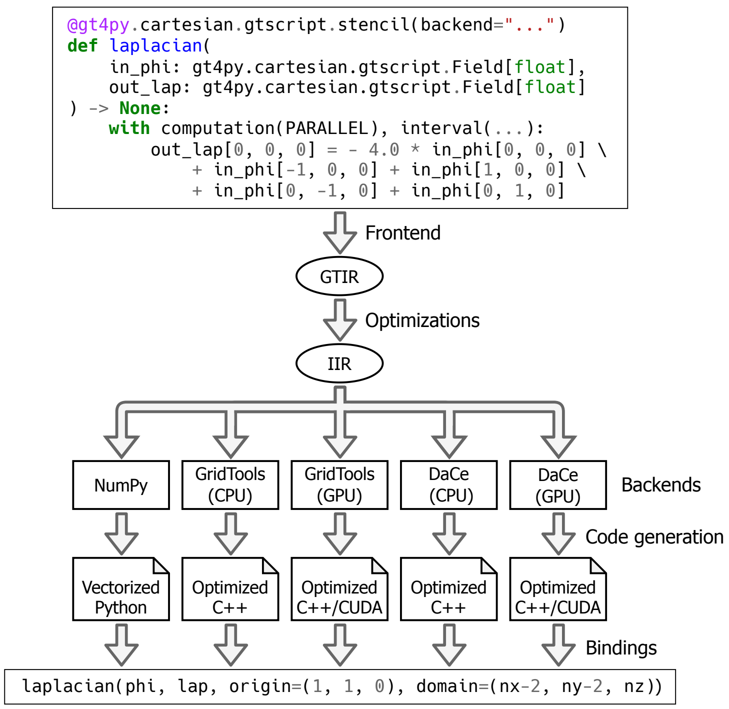

Numerical model
Proof-of-concepts and basic prototyping in MATLAB/Octave or Python
Fast execution on a single CPU core with C, C++ or FORTRAN
Multi-threaded implementation using compiler directives (OpenMP)
GPU porting via full re-writing (CUDA) or compiler directives (OpenACC / OpenMP)
Model change
Performance engineering
Performance engineering
Performance engineering
Numerical model
Model change
High-level representation using a domain specific language (DSL)
Domain scientists
Science interface
Computer experts
HPC implementation
Hierarchy of tree-like code representations featuring different levels of abstraction
Domain-driven optimizations
Performance engineering
Backend #1
...
Backend #n
Performance engineering
High-performance generated code
...
High-performance generated code

Figure 1. Diagrams comparing (a) a well-established workflow in scientific software development and (b) a DSL-based approach resembling the software engineering strategy advocated in this paper. The red-and-blue dashed line on the right mark the separation-of-concerns between the domain scientists and the computer experts.

3.1 The GT4Py framework

GT4Py[3] is a Python library to generate high-performance implementations of stencil[4] kernels as found in weather and climate applications. The library is being jointly developed by the Swiss National Supercomputing Center (CSCS), the Swiss Federal Office of Meteorology and Climatology (MeteoSwiss), and the Paul Allen Institute for Artificial Intelligence (AI^2). The choice of embedding the GT4Py framework in Python has been primarily dictated by the following factors: (i) Python is taught in many academic courses due its clean, intuitive and expressive syntax, so that a significant fraction of early-career domain scientists is exposed to the language; (ii) it admits a powerful ecosystem of open source packages for building end-to-end applications; (iii) it is possible to seamlessly interface Python with lower-level languages with minimal overhead and virtually no memory copies; (iv) under the thrust of the Artificial Intelligence and Machine Learning community (AI/ML), the popularity and adoption of Python across the whole scientific community is constantly growing, as opposed to Fortran (Shipman and Randles, 2023). The proposed Python implementation of CLOUDSC and CLOUDSC2 are based on the first public release of GT4Py, which only supports Cartesian grids. Latest advancements to support unstructured meshes (contained in the sub-package `gt4py.next`) are not discussed in the following.

[3]https://github.com/GridTools/gt4py
[4]A *stencil* is an operator which computes array elements by accessing a fixed pattern of neighbouring items.



```
@gt4py.cartesian.gtscript.stencil(backend="...")
def laplacian(
    in_phi: gt4py.cartesian.gtscript.Field[float],
    out_lap: gt4py.cartesian.gtscript.Field[float]
) -> None:
    with computation(PARALLEL), interval(...):
        out_lap[0, 0, 0] = - 4.0 * in_phi[0, 0, 0] \
            + in_phi[-1, 0, 0] + in_phi[1, 0, 0] \
            + in_phi[0, -1, 0] + in_phi[0, 1, 0]
```

```
laplacian(phi, lap, origin=(1, 1, 0), domain=(nx-2, ny-2, nz))
```

Figure 2. Simplified view on the internal stages carried out within the GT4Py toolchain to generate a high-performance CPU or GPU implementation of the horizontal Laplacian stencil starting from its GTScript definition. For the sake of visualization, only two intermediate representations (IRs) are included: the GridTools IR (GTIR) and the Implementation IR (IIR).

Figure 2 showcases the main steps undertaken by the GT4Py toolchain to translate the high-level definition of the horizontal Laplacian operator into optimized code which can be directly called from within Python. The stencil definition is given as a regular Python function using the GTScript DSL. GTScript abstracts for-loops away: computations are described for a single point of a three-dimensional Cartesian grid, and can be differentiated for the vertical boundaries using the `interval` context manager. Vertical loops are replaced by `computation` contexts, which define the iteration order along the vertical axis: either `PARALLEL` (meaning no vertical data dependencies between horizontal planes), `FORWARD` or `BACKWARD`. Each assignment statement within a computation block can be thought of as a loop over a horizontal plane; no horizontal data dependencies are allowed. Neighbouring points are accessed through relative offsets, with the first two offsets being the horizontal offsets, and the last offset being the vertical offset.

Any function marked with the `gt4py.cartesian.gtscript.stencil` decorator is translated by the GT4Py *frontend* into a hierarchy of tree-like Intermediate Representations (IRs), featuring different levels of abstractions to accommodate diverse optimizations and transformations (Gysi et al., 2021). The lowest-level IR (denoted as

Implementation IR, or IIR) is consumed by the *backends* to generate code which is either optimized for a given architecture, or suited to a specific purpose. The following backends are currently available:

- NumPy (Harris et al., 2020) is the *de facto* standard for array computing in Python, and can be used for debugging and fast-prototyping;
- GridTools (Afanasyev et al., 2021) is a set of libraries and utilities to write performance-portable applications in the area of weather and climate;
- DaCe (Ben-Nun et al., 2019) is a parallel programming framework which internally uses the Stateful DataFlow multiGraph (SDFG) data-centric intermediate representation to decouple domain science and performance engineering.

The generated code is compiled under the hood, and Python bindings for the resulting executable are automatically produced, so that the stencil can finally be executed by passing the input and output fields and by specifying the origin and size of the computation domain. GT4Py provides convenient utilities to allocate arrays with an optimal memory layout for any given backend, relying on NumPy for CPU storages and CuPy (Nishino and Loomis, 2017) for GPU storages. In the latter respect, we highlight that GT4Py supports both NVIDIA and AMD GPUs.

A more realistic and pertinent code sample is provided in Listing 1. It is an abridged GT4Py implementation of the procedure computing the saturation water vapor pressure as a function of air pressure and temperature. The code is extracted from the CLOUDSC2-GT4Py dwarf and highlights two additional features of GTScript: functions and external symbols. Functions can be thought of as macros, and can be used to improve composability, re-usability and readability. External symbols are used to encode those scalar parameters (e.g. physical constants) which are kept constant throughout a simulation, and might only change between different model setups. External values must be provided at stencil compilation time. The functionalities provided by the package `ifs_physics_common` will be discussed in the following section.

4 Infrastructure Code

Over the decades, the major factor driving the improvement of the forecasting skills of atmospheric models has been the shrinking of the horizontal grid spacing. However, the continual growth of the model resolution demands an increasing specialization to address the physical processes which emerge on smaller and smaller scales (Gross et al., 2016, 2018). This has resulted in a high compartmentalization of the model development, with dynamical cores and

Listing 1 GTScript (the Python-embedded DSL exposed by GT4Py) functions and stencil computing the saturation water vapor pressure given the air pressure and temperature. Abridged excerpt from the CLOUDSC2-GT4Py dwarf.

```python
@gt4py.cartesian.gtscript.function
def foealfcu(t):
    from __externals__ import RTICECU, RTWAT, RTWAT_RTICECU_R
    return min(1.0, ((max(RTICECU, min(RTWAT, t)) - RTICECU) * RTWAT_RTICECU_R) ** 2)

@gt4py.cartesian.gtscript.function
def foeewmcu(t):
    from __externals__ import R2ES, R3IES, R3LES, R4IES, R4LES, RTT
    return R2ES * (
        foealfcu(t) * exp(R3LES * (t - RTT) / (t - R4LES))
        + (1 - foealfcu(t)) * (exp(R3IES * (t - RTT) / (t - R4IES)))
    )

@ifs_physics_common.framework.stencil.stencil_collection("saturation")
def saturation(
    in_ap: gtscript.Field[float], in_t: gtscript.Field[float], out_qsat: gtscript.Field[float]
):
    from __externals__ import LPHYLIN, QMAX, R2ES, R3IES, R3LES, R4IES, R4LES, RETV, RTT
    with computation(PARALLEL), interval(...):
        if LPHYLIN:
            alfa = foealfa(in_t)
            foeewl = R2ES * exp(R3LES * (in_t - RTT) / (in_t - R4LES))
            foeewi = R2ES * exp(R3IES * (in_t - RTT) / (in_t - R4IES))
            foeew = alfa * foeewl + (1 - alfa) * foeewi
            qs = min(foeew / in_ap, QMAX)
        else:
            ew = foeewmcu(in_t)
            qs = min(ew / in_ap, QMAX)
        out_qsat[0, 0, 0] = qs / (1.0 - RETV * qs)
```

physics packages mostly developed in isolation. In turn, this has eased the proliferation of software components with incompatible structure. There are several examples of parameterizations that have been devised for a specific model, and that can be transferred to another model only upon a significant amount of work and complex interfaces (Randall, 1996). Such an approach entails a lot of duplicated code and hinders the transfer of knowledge and expertise between research groups and modeling systems. This is in direct contrast with the need of a comprehensive assessment of the impact of the time-stepping on weather forecasts and climate projections (Ubbiali et al., 2021).

The recognition of the need for standardizing Earth system models dates back to the 1980s (Pielke and Arritt, 1984). In Kalnay et al. (1989), the authors suggested a list of basic programming rules to design *plug-compatible* physics packages which enable a high degree of scientific code exchange. This led to the idea of a common software infrastructure which couples different components while enhancing inter-operability, usability, software reuse, and portability (Dickinson et al., 2002). Since then, several frameworks have appeared which comply with this view (Guilyardi et al., 2003; Hill et al., 2004; Balaji, 2012; Craig et al., 2012; Theurich et al., 2016), all identifying components with the major physical domains which constitute the Earth system: atmosphere, ocean, land surface, sea ice, ice sheet, and biogeochemistry. In the majority of legacy codes, each component is implemented as a monolithic piece of software, with the process sequence being hard-coded for efficiency reasons. It follows that changing the ocean component of a coupled model might be far more immediate than changing the turbulence scheme within the atmospheric component. Recently, the Common Community Physics Package initiative (CCPP; Heinzeller et al., 2019; Donahue et al., 2021) has lowered the concept of component to the level of individual physics processes. This design choice gives the ability to choose the order of parameterizations, to subcycle individual parameterizations, and to interleave dynamics with physics computations.

The same approach fostered by CCPP has been pursued by Sympl (Monteiro et al., 2018), a tool set of Python abstract base classes (ABCs) and utilities to write self-contained, self-documented and inter-changeable model components. Components interact through dictionaries whose keys are the names of the model variables (fields), and whose values are xarray's `DataArrays` (Hoyer and Hamman, 2017) collecting the grid point values, the labelled dimensions, the axis coordinates, and the units for those variables. The most relevant component exposed by Sympl is `TendencyComponent`, producing tendencies for prognostic variables and retrieving diagnostics. The class defines a minimal interface to declare the list of input and output fields, and initialize and run an instance of the class. This imposes minor constraints on model developers when writing a new physics package.

The bespoke infrastructure code for CLOUDSC-GT4Py and CLOUDSC2-GT4Py is bundled as an installable Python package called `ifs_physics_common`[5]. It builds upon Sympl and extends it with grid-aware and stencil-oriented functionalities. Both the CLOUDSC cloud microphysics and the non-linear, tangent-linear and adjoint formulations of CLOUDSC2 are encoded as stand-alone `TendencyComponent` classes settled over a `ComputationalGrid`. The latter is a collection of index spaces for different grid locations. For instance, `(I, J, K)` corresponds to cell centers, while `(I, J, K-1/2)` denotes vertically-staggered grid points. For any input and output field, its name, units and grid location are specified as class properties. When running the component via the *dunder* method `__call__`, Sympl transparently extracts the raw data from the input `DataArrays` according to the information provided in the class properties. This step may involve units conversion and axis transposition. The resulting storages are forwarded to the method `array_call`, which carries out the actual computations.

[5]https://github.com/stubbiali/ifs-physics-common

Listing 2 A Python class to compute the saturation water vapor pressure given the air pressure and temperature. Abridged excerpt from the CLOUDSC2-GT4Py dwarf.

```
import cupy as cp
from functools import cached_property
import numpy as np
from typing import Optional, Union
from ifs_physics_common.framework.components import DiagnosticComponent
from ifs_physics_common.framework.config import GT4PyConfig
from ifs_physics_common.framework.grid import ComputationalGrid, I, J, K

# type alias originally defined in ifs_physics_common.utils.typingx
StorageDict = dict[str, Union[cp.ndarray. np.ndarray]]

class Saturation(DiagnosticComponent):
    def __init__(
        self,
        computational_grid: ComputationalGrid,
        lphylin: bool,
        yoethf_parameters: Optional[dict[str, float]] = None,
        yomcst_parameters: Optional[dict[str, float]] = None,
        gt4py_config: GT4PyConfig,
    ) -> None:
        super().__init__(computational_grid, gt4py_config)
        externals = {"LPHYLIN": lphylin, "QMAX": 0.5}
        externals.update(yoethf_parameters or {})
        externals.update(yomcst_parameters or {})
        self.saturation = self.compile_stencil("saturation", externals)

    @cached_property
    def _input_properties(self):
        return {"ap": {"grid": (I, J, K), "units": "Pa"}, "t": {"grid": (I, J, K), "units": "K"}}

    @cached_property
    def _diagnostic_properties(self):
        return {"qsat": {"grid": (I, J, K), "units": "g g^-1"}}

    def array_call(self, state: StorageDict, out: StorageDict) -> None:
        self.saturation(
            in_ap=state["ap"],
            in_t=state["t"],
            out_qsat=out["qsat"],
            origin=(0, 0, 0),
            domain=self.computational_grid.grids[I, J, K].shape,
        )
```

Listing 2 brings a concrete example from CLOUDSC2-GT4Py: a model component leveraging the stencil defined in Listing 1 to compute the saturation water vapor pressure. The class inherits `DiagnosticComponent`, a stripped-down version of `TendencyComponent` which only retrieves diagnostic quantities. Within the instance initializer `__init__`, the stencil from Listing 1, registered using the decorator `ifs_physics_common.framework.stencil.stencil`, is compiled using the utility method `compile_stencil`. The options configuring the stencil compilation (e.g. the GT4Py backend) are fetched from the dataclass `GT4PyConfig`.

5 Performance Analysis

The performance of diverse coding versions of the CLOUDSC and CLOUDSC2 prototypes are sampled on three supercomputers:

(i) Piz Daint[6], an HPE Cray XC40/XC50 system installed at CSCS in Lugano, Switzerland;

(ii) MeluXina[7], an ATOS BullSequana XH2000 machine hosted by LuxConnect in Bissen, Luxembourg, and procured by the EuroHPC Joint Undertaking (JU) initiative;

(iii) the Cray HPE EX235a supercomputer LUMI[8], an EuroHPC pre-exascale machine at the Science Information Technology Center (CSC) in Kajaani, Finland.

On each machine, the codes are executed on a single hybrid node, housing one or multiple GPU accelerators alongside the host CPU. An overview of the node architecture for all three test bed supercomputers can be found in Table 1.

Beside the GT4Py re-write, four lower-level implementations of the cloud microphysics schemes are considered.

(a) The baseline Fortran version, enriched with OpenMP directives for multi-threading execution on CPU.

(b) An optimized GPU-enabled version based on OpenACC using the single-column coalesced (SCC) loop layout in combination with loop fusion and temporary local array demotion (or "k-caching"). While the SCC loop layout yields more efficient accesses to device memory, the k-caching technique reduces the memory footprint by restricting temporary fields on individual horizontal planes, rather than on the full three-dimensional domain.

(c) An optimized GPU-enabled version using the source-to-source translation tool Loki.

(d) An optimized GPU-enabled CUDA C version of CLOUDSC including loop fusion and temporary local array demotion.

Tables 2-4 report the compiler suite employed for each coding implementation on Piz Daint, MeluXina and LUMI, respectively. To accommodate a fair performance-wise comparison, compiler optimizations are enabled for all implementations, provided that the underlying code manipulations do not harm validation. CPU-only codes are executed on all the cores available on the node (possibly spanning multiple sockets)[9], while device code is launched on

[6]https://www.cscs.ch/computers/piz-daint
[7]https://docs.lxp.lu/
[8]https://docs.lumi-supercomputer.eu/
[9]The compute nodes of the GPU partition of LUMI have the low-noise mode activated. This mode reserves one core per Non-Uniform Memory Access (NUMA) node to the operating system, so that only 56 out of 64 cores are available to the jobs.

System	CPU	GPU	RAM
Piz Daint	1x Intel Xeon E5-2690v3 12c @ 2.60 GHz	1x NVIDIA Tesla P100 16GB	64 GB
MeluXina	2x AMD EPYC Rome 7452 32c @ 2.35 GHz	4x NVIDIA Tesla A100 40GB	512 GB
LUMI	1x AMD EPYC Trento 7A53 64c @ 2.00 GHz	4x AMD Instinct MI250x	512 GB

Table 1. Overview of the node architecture for the hybrid partition of Piz Daint, MeluXina and LUMI. Only the technical specifications which are most relevant for the purposes of this paper are reported.

a single GPU (i.e. no multi-GPU capabilities are exploited). The execution times for the CLOUDSC, the non-linear formulation of CLOUDSC2, and the symmetry test for the tangent-linear and adjoint formulations of CLOUDSC2 are visualized in Figs. 3-5 for Piz Daint, MeluXina and LUMI, respectively. In each figure, performance counters are provided for both double precision (FP64; top row) and single precision (FP32; bottom row) implementations. We remark, however, that for all the implementations at consideration, the symmetry test fails under single precision. This is indeed a well-known fact, given that the test is highly sensitive to round-off errors.

Within each panel of Figs. 3-5, bars refer to different coding versions, and any missing bar signifies that the corresponding implementation is either not available or not working properly. More in detail:

– the Fortran version of the adjoint formulation of CLOUDSC2 can only run on a single OpenMP thread on MeluXina (the issue is still under investigation);

– a 32-bit C CUDA version of CLOUDSC does not exist at the time of writing, and no C CUDA implementations are available for CLOUDSC2;

– CUDA is a parallel programming platform designed specifically for NVIDIA GPUs and cannot be run on AMD GPUs, as those available on LUMI;

– all Fortran-based implementations of the three formulations of CLOUDSC2 can only use double precision computations;

– the Loki version of CLOUDSC fails to compile on LUMI (the issue is still under investigation);

– a Loki version of the tangent-linear and adjoint formulations of CLOUDSC2 is not available at the time of writing.

On the contrary, the GT4Py re-write of both CLOUDSC and CLOUDSC2 runs on every CPU and GPU architecture included in the study, and can fully employ either double precision or single precision floating point arithmetic. With

Implementation	CLOUDSC	CLOUDSC2: Non-linear	CLOUDSC2: Symmetry test
Fortran: OpenMP (CPU)	Intel Fortran 2021.3.0	Intel Fortran 2021.3.0	Intel Fortran 2021.3.0
Fortran: OpenACC (GPU)	NVIDIA Fortran 21.3-0	-	-
Fortran: Loki (GPU)	NVIDIA Fortran 21.3-0	NVIDIA Fortran 21.3-0	-
C: CUDA (GPU)	NVIDIA CUDA 11.2.67	-	-
GT4Py: CPU k-first	g++ (GCC) 10.3.0	g++ (GCC) 10.3.0	g++ (GCC) 10.3.0
GT4Py: DaCe (GPU)	NVIDIA CUDA 11.2.67	NVIDIA CUDA 11.2.67	NVIDIA CUDA 11.2.67

Table 2. For each coding version of the CLOUDSC and CLOUDSC2 dwarfs considered in the performance analysis, the table reports the compiler suite used to compile the codes on Piz Daint. The codes are compiled with all major optimization options enabled. Those implementations which are either not available or not working are marked with a dash; more details, as well as a high-level description of each coding implementation, are provided in the text.

Implementation	CLOUDSC	CLOUDSC2: Non-linear	CLOUDSC2: Symmetry test
Fortran: OpenMP (CPU)	NVIDIA Fortran 22.7-0	NVIDIA Fortran 22.7-0	-
Fortran: OpenACC (GPU)	NVIDIA Fortran 22.7-0	-	-
Fortran: Loki (GPU)	NVIDIA Fortran 22.7-0	NVIDIA Fortran 22.7-0	-
C: CUDA (GPU)	NVIDIA CUDA 11.7.64	-	-
GT4Py: CPU k-first	g++ (GCC) 11.3.0	g++ (GCC) 11.3.0	g++ (GCC) 11.3.0
GT4Py: DaCe (GPU)	NVIDIA CUDA 11.7.64	NVIDIA CUDA 11.7.64	NVIDIA CUDA 11.7.64

Table 3. As Table 2 but for the MeluXina supercomputer.

Implementation	CLOUDSC	CLOUDSC2: Non-linear	CLOUDSC2: Symmetry test
Fortran: OpenMP (CPU)	Cray Fortran 14.0.2	Cray Fortran 14.0.2	-
Fortran: OpenACC (GPU)	Cray Fortran 14.0.2	-	-
Fortran: Loki (GPU)	Cray Fortran 14.0.2	Cray Fortran 14.0.2	-
C: CUDA (GPU)	-	-	-
GT4Py: CPU k-first	g++ (GCC) 11.2.0	g++ (GCC) 11.2.0	g++ (GCC) 11.2.0
GT4Py: DaCe (GPU)	AMD Clang 14.0.0	AMD Clang 14.0.0	AMD Clang 14.0.0

Table 4. As Table 2 but for the LUMI supercomputer.

GT4Py, changing the target architecture (namely, the backend), as well as the precision of computations is as easy as setting a namelist parameter. Particularly, the GT4Py porting of the tangent-linear and adjoint formulations of CLOUDSC2 represents the first ever coding version of such schemes offloading the computationally-intensive stencil kernels to GPU. Nonetheless, we observe that the performance delivered by GT4Py falls short of native implementations both on CPU and GPU. Across all systems and test cases, multi-threaded Fortran is up to three times faster than the GridTools CPU backend of GT4Py using the k-first (C-like) memory layout, while on CLOUDSC the DaCe backend of GT4Py is about 50% slower than CUDA C. However, we observe a significant sensitivity of the GPU performance with respect to the thread block size[10]: for values smaller than 128, performance are degraded across all implementations, with the gap between CUDA C and GT4Py/DaCe being smaller. On the one hand, this signals that even for the DSL approach, some (small) hand-tuning, driven by an in-depth knowledge of the underlying

[10]In the Fortran dialect, the thread block size corresponds to the NPROMA.

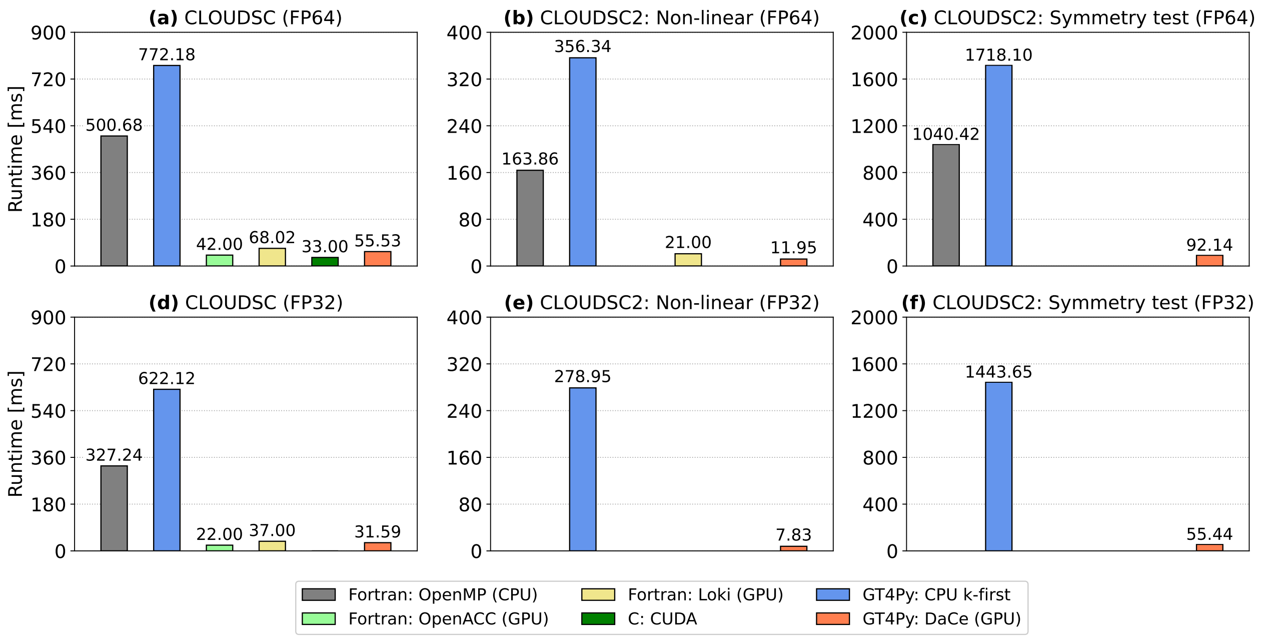


Figure 3. Execution time on a single hybrid node of the Piz Daint supercomputer for CLOUDSC (left), CLOUDSC2 (center) and the symmetry test for the tangent-linear and adjoint formulations of CLOUDSC2 (right) using either double precision (top row) or single precision (bottom row) floating point arithmetic. Displayed are the multi-threaded Fortran baseline using OpenMP (grey); two GPU-accelerated Fortran implementations, either using OpenACC directives (lime) or the source-to-source translation tool Loki (yellow); an optimized CUDA C version (green); and the GT4Py re-write, either using the GridTools C++ CPU backend with k-first data ordering (blue) or the DaCe GPU backend (orange). All numbers should be intended as an average over 50 realizations. The panels only show the code versions available and validating at the time of writing.

compute architecture, is still needed to achieve best performance. On the other, it indicates that there is still room for improvement in the DSL optimization toolchain. In this regard, future endeavors at ECMWF might pick up the code generated by GT4Py and manually iterate over it, in view of its integration into the IFS. We also remark that minimal effort has been invested into performance engineering the application side of both CLOUDSC-GT4Py and CLOUDSC2-GT4Py. This was a deliberate choice, not to affect the readability and modularity of the Python codes. Nonetheless, there exist more näive CUDA C versions of CLOUDSC performing more poorly than GT4Py/DaCe.

The performance of GT4Py compares more favourably with OpenACC. Although OpenACC is consistently faster on CLOUDSC both on Piz Daint and MeluXina, it is significantly slower on LUMI, especially when using double precision. This could be ascribed to the not-yet-refined support for OpenACC offered by the HPE Cray compiler. In this regard, it should be added that as of now, only the HPE Cray compiler implements GPU offloading capabilities for OpenACC directives on AMD GPUs, meaning that Fortran OpenACC codes require an HPE Cray platform

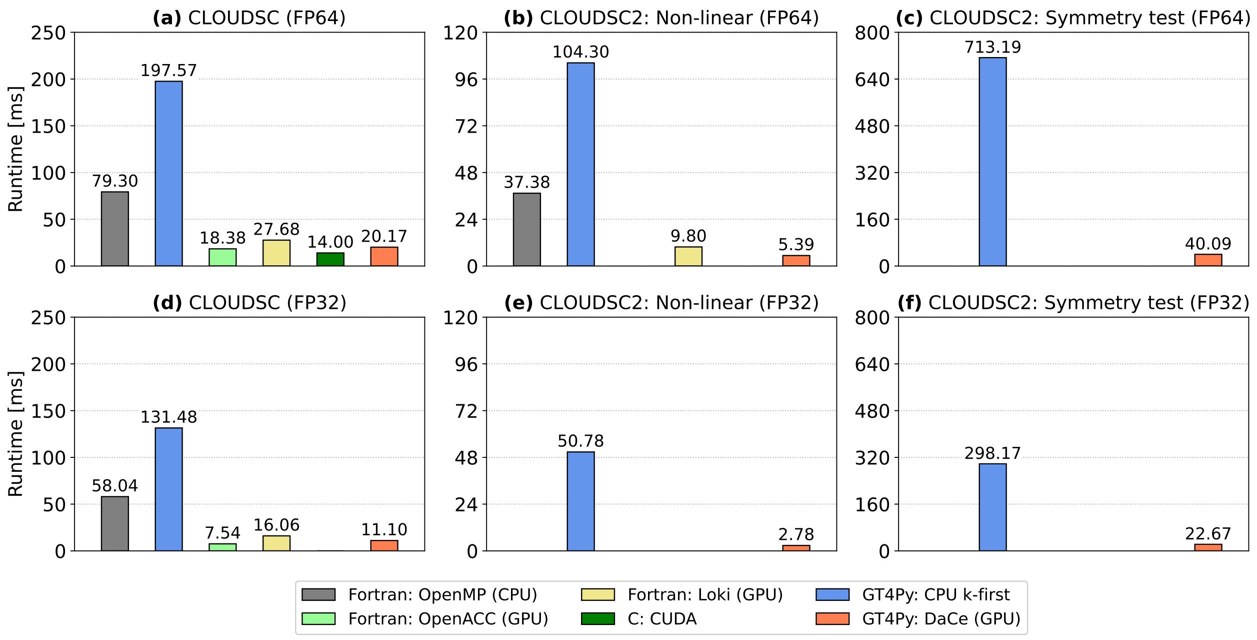


Figure 4. As Fig. 3 but for the MeluXina supercomputer.

to run on AMD GPUs. On the other hand, GT4Py relies on the HIPCC compiler driver developed by AMD to compile device code for AMD accelerators, and this should guarantee a proper functioning irrespective of the machine vendor. It is interesting to note that the DaCe backend of GT4Py executes roughly two-times faster on MeluXina's NVIDIA A100 GPUs, compared to LUMI's AMD Instinct MI250x GPUs. It should be mentioned, however, that from a software perspective, each physical GPU module on LUMI is considered as two GPUs, so that the code is actually executed on half of a physical GPU module. We can therefore speculate that when using both dies of an AMD Instinct MI250x GPU, performance are on par with the NVIDIA A100 GPU.

Finally, we highlight that CLOUDSC-GT4Py and CLOUDSC2-GT4Py are faster than Loki-based implementations on all three machines. This is particularly relevant, since Loki promises the same advantages in terms of portability and productivity as GT4Py.

6 Conclusions

The CLOUDSC and CLOUDSC2 cloud microphysics schemes from the IFS physics suite have served as demonstrators to showcase the benefits deriving from a high-level domain-specific approach to physical parameterizations for NWP.

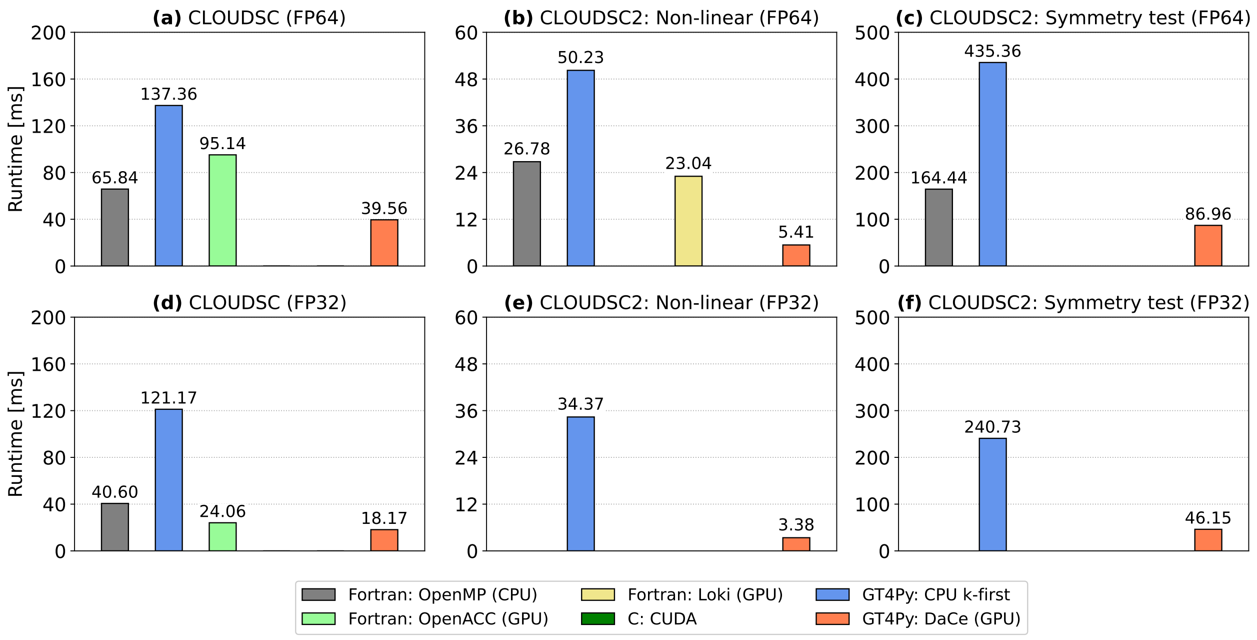


Figure 5. As Fig. 3 but for the LUMI supercomputer.

We presented two novel Python implementations based on the GT4Py framework, where the scientific code is insulated from hardware-specific specializations. The result is a hardware-agnostic user application with enhanced readability and maintainability, which can run on a wide spectrum of compute architectures. The latter point is of primary interest in the light of the current HPC technology landscape, where general-purpose CPUs are increasingly complemented by domain-specific architectures (DSAs) such as GPUs, Tensor Processor Units (TPUs), Field Programmable Gate Arrays (FPGAs), and Application-Specific Integrated Circuits (ASICS).

In both CLOUDSC-GT4Py and CLOUDSC2-GT4Py, the stencil kernels are encapsulated within model components sharing a minimal and clear interface. By avoiding any assumption on the host model, the interface is aimed to devise inter-operable and plug-and-play physical packages, which can be transferred more easily between different modeling systems with virtually no performance penalty.

We carried out a comprehensive study to assess the portability of the Python codes across three top-class supercomputers, differing for the manufacturer, the node architecture, the software stack and the compiler suites. We showed that the GPU performance of GT4Py codes are comparable to optimized Fortran OpenACC implementations and superior to the source-to-source translation tool Loki. However, CLOUDSC-GT4Py and CLOUDSC2-GT4Py cannot yet attain the same performance as native implementations, both on CPU (Fortran) and GPU (CUDA C).

On the one hand, we are not particularly concerned about the overall performance of the CPU backends of GT4Py. The results shown in this paper help create clear evidence that weather and climate model codes can execute significantly faster on GPUs (Fuhrer et al., 2018), and the number of world-wide HPC systems offering some sort of accelerators is steadily increasing[11]. Therefore, we envision that CPUs will be increasingly relegated to non time-critical tasks in the future.

On the other hand, we recognize that GPU performance are of utter importance in view of a future adoption of GT4Py in an operational context. It should be mentioned, however, that the CUDA C implementation of CLOUDSC has been written from scratch by CUDA experts, and the final version of the code is the result of a long iterative hand-optimization process which is hardly scalable to the full model. On the contrary, we believe that the GPU porting of a full-fledge model via a ground-up re-writing using a DSL is not only possible, but also more sustainable and cost-effective than a directive-based approach. Indeed, DSLs enable a separation of concerns between domain scientists and computer experts, so that both can better exploit the respective field of expertise and thus boost productivity. Notwithstanding, we stress that the development and maintenance of the DSL toolchain with respect to emerging computational patterns and heterogeneous compute platforms is complex and does not come for free. However, provided that the design of the toolchain is driven by a close synergy between domain scientists and computer experts, the advantages for all downstream applications are likely to offset the investment in the DSL.

Finally, we remind that GT4Py has been originally devised to express the computational motifs ubiquitous in dynamical cores. It is then not surprising that some patterns found in the CLOUDSC and CLOUDSC2 microphysics codes are not natively supported by the DSL. It is thus not clear yet whether GT4Py will be the most appropriate tool to port all physical packages to graphics accelerators. We plan to conduct more feasibility and portability studies targeting diverse physical parameterizations, possibly recycling the prototype infrastructure code presented in this manuscript.

Author contributions. TODO

Competing interests. The authors declare that they have no conflict of interest.

[11]In the 61st edition of the TOP500 list published in June 2023, 185 out of the 500 most powerful supercomputers in the world use graphics accelerator technology (https://www.top500.org/lists/top500/2023/06/highs/).

Acknowledgements. TODO

References

Adams, S. V., Ford, R. W., Hambley, M., Hobson, J., Kavčič, I., Maynard, C. M., Melvin, T., Müller, E. H., Mullerworth, S., Porter, A., et al.: LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models, Journal of Parallel and Distributed Computing, 132, 383–396, 2019.

Afanasyev, A., Bianco, M., Mosimann, L., Osuna, C., Thaler, F., Vogt, H., Fuhrer, O., VandeVondele, J., and Schulthess, T. C.: GridTools: A framework for portable weather and climate applications, SoftwareX, 15, 100 707, 2021.

Balaji, V.: The flexible modeling system, in: Earth System Modelling-Volume 3, pp. 33–41, Springer, 2012.

Baldauf, M., Seifert, A., Förstner, J., Majewski, D., Raschendorfer, M., and Reinhardt, T.: Operational convective-scale numerical weather prediction with the COSMO model: Description and sensitivities, Monthly Weather Review, 139, 3887–3905, 2011.

Balsamo, G., Beljaars, A., Scipal, K., Viterbo, P., van den Hurk, B., Hirschi, M., and Betts, A. K.: A revised hydrology for the ECMWF model: Verification from field site to terrestrial water storage and impact in the Integrated Forecast System, Journal of Hydrometeorology, 10, 623–643, 2009.

Bauer, P., Quintino, T., Wedi, N., Bonanni, A., Chrust, M., Deconinck, W., Diamantakis, M., Düben, P., English, S., Flemming, J., et al.: The ECMWF scalability programme: Progress and plans, ECMWF Technical Memo No. 857.

Bauer, P., Dueben, P. D., Hoefler, T., Quintino, T., Schulthess, T. C., and Wedi, N. P.: The digital revolution of Earth-system science, Nature Computational Science, 1, 104–113, 2021.

Bechtold, P., Köhler, M., Jung, T., Doblas-Reyes, F., Leutbecher, M., Rodwell, M. J., Vitart, F., and Balsamo, G.: Advances in simulating atmospheric variability with the ECMWF model: From synoptic to decadal time-scales, Quarterly Journal of the Royal Meteorological Society, 134, 1337–1351, 2008.

Bechtold, P., Semane, N., Lopez, P., Chaboureau, J.-P., Beljaars, A., and Bormann, N.: Representing equilibrium and nonequilibrium convection in large-scale models, Journal of the Atmospheric Sciences, 71, 734–753, 2014.

Beljaars, A. C., Brown, A. R., and Wood, N.: A new parametrization of turbulent orographic form drag, Quarterly Journal of the Royal Meteorological Society, 130, 1327–1347, 2004.

Ben-Nun, T., de Fine Licht, J., Ziogas, A. N., Schneider, T., and Hoefler, T.: Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, 2019.

Ben-Nun, T., Groner, L., Deconinck, F., Wicky, T., Davis, E., Dahm, J., Elbert, O. D., George, R., McGibbon, J., Trümper, L., et al.: Productive performance engineering for weather and climate modeling with python, in: SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–14, IEEE, 2022.

Bertagna, L., Guba, O., Taylor, M. A., Foucar, J. G., Larkin, J., Bradley, A. M., Rajamanickam, S., and Salinger, A. G.: A performance-portable nonhydrostatic atmospheric dycore for the Energy Exascale Earth System Model running at cloud-resolving resolutions, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–14, IEEE, 2020.

Brown, A., Milton, S., Cullen, M., Golding, B., Mitchell, J., and Shelly, A.: Unified modeling and prediction of weather and climate: A 25-year journey, Bulletin of the American Meteorological Society, 93, 1865–1877, 2012.

Chandrasekaran, S. and Juckeland, G.: OpenACC for Programmers: Concepts and Strategies, Addison-Wesley Professional, 2017.

Clement, V., Marti, P., Lapillonne, X., Fuhrer, O., and Sawyer, W.: Automatic Port to OpenACC/OpenMP for Physical Parameterization in Climate and Weather Code Using the CLAW Compiler, Supercomputing Frontiers and Innovations, 6, 51–63, 2019.

Courtier, P., Thépaut, J.-N., and Hollingsworth, A.: A strategy for operational implementation of 4D-Var, using an incremental approach, Quarterly Journal of the Royal Meteorological Society, 120, 1367–1387, 1994.

Craig, A. P., Vertenstein, M., and Jacob, R.: A new flexible coupler for earth system modeling developed for CCSM4 and CESM1, The International Journal of High Performance Computing Applications, 26, 31–42, 2012.

Dagum, L. and Menon, R.: OpenMP: an industry standard API for shared-memory programming, IEEE Computional Science and Engineering, 5, 46–55, 1998.

Dahm, J., Davis, E., Deconinck, F., Elbert, O., George, R., McGibbon, J., Wicky, T., Wu, E., Kung, C., Ben-Nun, T., et al.: Pace v0. 2: a Python-based performance-portable atmospheric model, Geoscientific Model Development, 16, 2719–2736, 2023.

Deakin, T., McIntosh-Smith, S., Price, J., Poenaru, A., Atkinson, P., Popa, C., and Salmon, J.: Performance portability across diverse computer architectures, in: 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), pp. 1–13, IEEE, 2019.

Dickinson, R. E., Zebiak, S. E., Anderson, J. L., Blackmon, M. L., De Luca, C., Hogan, T. F., Iredell, M., Ji, M., Rood, R. B., Suarez, M. J., et al.: How can we advance our weather and climate models as a community?, Bulletin of the American Meteorological Society, 83, 431–436, 2002.

Donahue, A., Bertagna, L., and Caldwell, P.: A Flexible and Modern Design for the Next Generation E3SM Atmosphere Model, in: AGU Fall Meeting Abstracts, vol. 2021, pp. A25N–1863, 2021.

ECMWF: IFS Documentation CY48R1 - Part IV: Physical Processes, ECMWF, 2023.

Edwards, H. C., Trott, C. R., and Sunderland, D.: Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, Journal of Parallel Distributed Computing, 74, 3202–3216, 2014.

Errico, R. M.: What is an adjoint model?, Bulletin of the American Meteorological Society, 78, 2577–2592, 1997.

Forbes, R. and Tompkins, A.: An improved representation of cloud and precipitation, ECMWF Newsletter No. 129, pp. 13–18, 2011.

Forbes, R. M., Tompkins, A. M., and Untch, A.: A new prognostic bulk microphysics scheme for the IFS, ECMWF Techical Memo. No. 649, 2011.

Fuhrer, O., Osuna, C., Lapillonne, X., Gysi, T., Cumming, B., Bianco, M., Arteaga, A., and Schulthess, T. C.: Towards a performance portable, architecture agnostic implementation strategy for weather and climate models, Supercomputing Frontiers and Innovations, 1, 45–62, 2014.

Fuhrer, O., Chadha, T., Hoefler, T., Kwasniewski, G., Lapillonne, X., Leutwyler, D., Lüthi, D., Osuna, C., Schär, C., Schulthess, T. C., et al.: Near-global climate simulation at 1 km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0, Geoscientific Model Development, 11, 1665–1681, 2018.

Gross, M., Malardel, S., Jablonowski, C., and Wood, N.: Bridging the (Knowledge) Gap between Physics and Dynamics, Bulletin of the American Meteorological Society, 97, 137–142, https://doi.org/10.1175/BAMS-D-15-00103.1, https://doi.org/10.1175/BAMS-D-15-00103.1, 2016.

Gross, M., Wan, H., Rasch, P. J., Caldwell, P. M., Williamson, D. L., Klocke, D., Jablonowski, C., Thatcher, D. R., Wood, N., Cullen, M., et al.: Physics–Dynamics Coupling in Weather, Climate, and Earth System Models: Challenges and Recent Progress, Monthly Weather Review, 146, 3505–3544, 2018.

Guilyardi, E., Budich, R., Komen, G., and Brasseur, G.: PRISM system specification handbook, version 1, PRISM report series, p. 230, 2003.

Gysi, T., Müller, C., Zinenko, O., Herhut, S., Davis, E., Wicky, T., Fuhrer, O., Hoefler, T., and Grosser, T.: Domain-specific multi-level IR rewriting for GPU: The Open Earth compiler for GPU-accelerated climate simulation, ACM Transactions on Architecture and Code Optimization (TACO), 18, 1–23, 2021.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al.: Array programming with NumPy, Nature, 585, 357–362, 2020.

Harris, L. M. and Lin, S.-J.: A two-way nested global-regional dynamical core on the cubed-sphere grid, Monthly Weather Review, 141, 283–306, 2013.

Heinzeller, D., Bernardet, L., Firl, G., Carson, L., Schramm, J., Zhang, M., Dudhia, J., Gill, D., Duda, M., Goldhaber, S., et al.: The Common Community Physics Package CCPP: unifying physics across NOAA and NCAR models using a common software framework, in: Geophysical Research Abstracts, vol. 21, 2019.

Hill, C., DeLuca, C., Balaji, V., Suarez, M., and Silva, A. d.: The architecture of the earth system modeling framework, IEE Computational Science and Engineering, 6, 18–28, 2004.

Hogan, R. J. and Bozzo, A.: A flexible and efficient radiation scheme for the ECMWF model, Journal of Advances in Modeling Earth Systems, 10, 1990–2008, 2018.

Hoyer, S. and Hamman, J.: xarray: ND labeled arrays and datasets in Python, Journal of Open Research Software, 5, 2017.

Janisková, M. and Lopez, P.: Linearised physics: the heart of ECMWF's 4D-Var, ECMWF Newsletter No. 175, pp. 20–26, 2023.

Janisková, M., Thépaut, J.-N., and Geleyn, J.-F.: Simplified and regular physical parameterizations for incremental four-dimensional variational assimilation, Monthly Weather Review, 127, 26–45, 1999.

Kalnay, E., Kanamitsu, M., Pfaendtner, J., Sela, J., Suarez, M., Stackpole, J., Tuccillo, J., Umscheid, L., and Williamson, D.: Rules for interchange of physical parameterizations, Bulletin of the American Meteorological Society, 70, 620–622, 1989.

Kim, J. Y., Kang, J.-S., and Joh, M.: GPU acceleration of MPAS microphysics WSM6 using OpenACC directives: Performance and verification, Computers & Geosciences, 146, 104 627, 2021.

Köhler, M., Ahlgrimm, M., and Beljaars, A.: Unified treatment of dry convective and stratocumulus-topped boundary layers in the ECMWF model, Quarterly Journal of the Royal Meteorological Society, 137, 43–57, 2011.

Kühnlein, C., Ehrengruber, T., Ubbiali, S., Krieger, N., Papritz, L., Calotoiu, A., and Wernli, H.: ECMWF collaborates with Swiss partners on GPU porting of FVM dynamical core, ECMWF Newsletter No. 175, 175, 11–12, 2023.

Lapillonne, X., Osterried, K., and Fuhrer, O.: Using OpenACC to port large legacy climate and weather modeling code to GPUs, in: Parallel Programming with OpenACC, pp. 267–290, Elsevier, 2017.

Lapillonne, X., Sawyer, W., Marti, P., Clement, V., Dietlicher, R., Kornblueh, L., Rast, S., Schnur, R., Esch, M., Giorgetta, M., et al.: Global climate simulations at 2.8 km on GPU with the ICON model, in: EGU General Assembly Conference Abstracts, p. 10306, 2020.

Lawrence, B. N., Rezny, M., Budich, R., Bauer, P., Behrens, J., Carter, M., Deconinck, W., Ford, R., Maynard, C., Mullerworth, S., et al.: Crossing the chasm: how to develop weather and climate models for next generation computers?, Geoscientific Model Development, 11, 1799–1821, 2018.

Lindfield, G. and Penny, J.: Numerical Methods: Using MATLAB, Academic Press, 2018.

Lott, F. and Miller, M. J.: A new subgrid-scale orographic drag parametrization: Its formulation and testing, Quarterly Journal of the Royal Meteorological Society, 123, 101–127, 1997.

McGibbon, J., Brenowitz, N. D., Cheeseman, M., Clark, S. K., Dahm, J. P., Davis, E. C., Elbert, O. D., George, R. C., Harris, L. M., Henn, B., et al.: fv3gfs-wrapper: a Python wrapper of the FV3GFS atmospheric model, Geoscientific Model Development, 14, 4401–4409, 2021.

Méndez, M., Tinetti, F. G., and Overbey, J. L.: Climate models: challenges for Fortran development tools, in: 2014 Second International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering, pp. 6–12, IEEE, 2014.

Miyamoto, Y., Kajikawa, Y., Yoshida, R., Yamaura, T., Yashiro, H., and Tomita, H.: Deep moist atmospheric convection in a subkilometer global simulation, Geophysical Research Letters, 40, 4922–4926, 2013.

Monge-Sanz, B., Chipperfield, M., Untch, A., Morcrette, J.-J., Rap, A., and Simmons, A.: On the uses of a new linear scheme for stratospheric methane in global models: water source, transport tracer and radiative forcing, Atmospheric Chemistry and Physics, 13, 9641–9660, 2013.

Monteiro, J. M., McGibbon, J., and Caballero, R.: sympl (v. 0.4. 0) and climt (v. 0.15. 3) – Towards a flexible framework for building model hierarchies in Python, Geoscientific Model Development, 11, 3781–3794, 2018.

Müller, A., Deconinck, W., Kühnlein, C., Mengaldo, G., Lange, M., Wedi, N., Bauer, P., Smolarkiewicz, P. K., Diamantakis, M., Lock, S.-J., et al.: The ESCAPE project: energy-efficient scalable algorithms for weather prediction at exascale, Geoscientific Model Development, 12, 4425–4441, 2019.

Neumann, P., Düben, P., Adamidis, P., Bauer, P., Brück, M., Kornblueh, L., Klocke, D., Stevens, B., Wedi, N., and Biercamp, J.: Assessing the scales in numerical weather and climate predictions: will exascale be the rescue?, Philosophical Transactions of the Royal Society A, 377, 20180 148, 2019.

Nishino, R. and Loomis, S. H. C.: CuPy: A NumPy-compatible library for NVIDIA GPU calculations, 31st conference on neural information processing systems, p. 151, 2017.

Nogherotto, R., Tompkins, A. M., Giuliani, G., Coppola, E., and Giorgi, F.: Numerical framework and performance of the new multiple-phase cloud microphysics scheme in RegCM4.5: precipitation, cloud microphysics, and cloud radiative effects, Geoscientific Model Development, 9, 2533–2547, 2016.

Orr, A., Bechtold, P., Scinocca, J., Ern, M., and Janiskova, M.: Improved middle atmosphere climate and forecasts in the ECMWF model through a nonorographic gravity wave drag parameterization, Journal of Climate, 23, 5905–5926, 2010.

Pielke, R. A. and Arritt, R. W.: A proposal to standardize models, Bulletin of the American Meteorological Society, 65, 1082–1082, 1984.

Randall, D. A.: A university perspective on global climate modeling, Bulletin of the American Meteorological Society, 77, 2685–2690, 1996.

Randall, D. A., Hurrell, J. W., Gettelman, A., Loft, R., Skamarock, W. C., Hauser, T., Dazlich, D. A., and Sun, L.: Simulations With EarthWorks, in: AGU Fall Meeting Abstracts, vol. 2022, pp. A33E–02, 2022.

Schär, C., Fuhrer, O., Arteaga, A., Ban, N., Charpilloz, C., Di Girolamo, S., Hentgen, L., Hoefler, T., Lapillonne, X., Leutwyler, D., et al.: Kilometer-scale climate models: Prospects and challenges, Bulletin of the American Meteorological Society, 2019.

Schulthess, T. C., Bauer, P., Wedi, N., Fuhrer, O., Hoefler, T., and Schär, C.: Reflecting on the goal and baseline for exascale computing: A roadmap based on weather and climate simulations, Computing in Science & Engineering, 21, 30–41, 2018.

Shipman, G. M. and Randles, T. C.: An evaluation of risks associated with relying on Fortran for mission critical codes for the next 15 years, https://doi.org/10.2172/1970284, https://www.osti.gov/biblio/1970284, 2023.

Stevens, B., Satoh, M., Auger, L., Biercamp, J., Bretherton, C. S., Chen, X., Düben, P., Judt, F., Khairoutdinov, M., Klocke, D., et al.: DYAMOND: the DYnamics of the Atmospheric general circulation Modeled On Non-hydrostatic Domains, Progress in Earth and Planetary Science, 6, 1–17, 2019.

Taylor, M. A., Guba, O., Steyer, A., Ullrich, P. A., Hall, D. M., and Eldred, C.: An energy consistent discretization of the nonhydrostatic equations in primitive variables, Journal of Advances in Modeling Earth Systems, 12, e2019MS001 783, 2020.

Theurich, G., DeLuca, C., Campbell, T., Liu, F., Saint, K., Vertenstein, M., Chen, J., Oehmke, R., Doyle, J., Whitcomb, T., et al.: The earth system prediction suite: toward a coordinated US modeling capability, Bulletin of the American Meteorological Society, 97, 1229–1247, 2016.

Tiedtke, M.: A comprehensive mass flux scheme for cumulus parameterization in large-scale models, Monthly Weather Review, 117, 1779–1800, 1989.

Ubbiali, S., Schär, C., Schlemmer, L., and Schulthess, T. C.: A numerical analysis of six physics-dynamics coupling schemes for atmospheric models, Journal of Advances in Modeling Earth Systems, 13, e2020MS002 377, 2021.

Watkins, J., Carlson, M., Shan, K., Tezaur, I., Perego, M., Bertagna, L., Kao, C., Hoffman, M. J., and Price, S. F.: Performance portable ice-sheet modeling with MALI, The International Journal of High Performance Computing Applications, 37, 600–625, 2023.

Yang, Z., Halem, M., Loft, R., and Suresh, S.: Accelerating MPAS-A model radiation schemes on GPUs using OpenACC, in: AGU Fall Meeting Abstracts, vol. 2019, pp. A11A–06, 2019.

Zängl, G., Reinert, D., Rípodas, P., and Baldauf, M.: The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core, Quarterly Journal of the Royal Meteorological Society, 141, 563–579, 2015.