# Towards a Portable Model for All-scale Predictions (PMAP) : FVM

Loïc Maurin, Fabrice Voitus

CNRM/GMAP/ALGO, Météo-France

April 18th, 2024

# Primary Goals

**Plan B** : Elaborate an efficient dynamical core for very-high resolution NWP applications as an alternative to the present AROME Dyncore

- ⛰ Stable and reliable NWP forecasts over very steep orography
- 💻 Scalability over heterogeneous and highly parallel HPC-clusters

# Why FVM ?

## Close collaboration with the ECMWF :

- Development a NWP Global Model prototype based upon Finite Volume Module (FVM) approach as an alternative to the current IFS Spectral Transform SISL Model [Kuhnlein et al. (2018)]

## Code adaptation strategy : Domain Specific Language (DSL) :

- As a proof of concept, a 3D Limited-area version of FVM Dyncore code has been recently developed by C. Kuhnlein in collaboration with ETH Zürich and MeteoSwiss based on GridTools for python (GT4py) adaptative programming language.

# AROME vs FVM
# Fully-Compressible Dynamical Cores

| | AROME | FVM |
|---|---|---|
| Terrain-following Vertical coord. | Mass-based | Height-based |
| Horizontal Discretization | Spectral Transform (ST) | Finite Volumes (FV) |
| SI Linearization | Constant Coefficients (CC) | Non-constant Coef. (NC) |
| Implicit solver | Direct | Preconditionned Krylov methods |
| Transport scheme | Semi-Lagrangian (SL) | Eulerian (MPDATA) |

# Mass vs Height -based

## terrain-following vertical coordinate

### Mass-based

- + seamless access to some "Hydrostatic part" of the flow ⇒ easy transition from EE to HPE ⇒ No need to prescribe an ambient state
- + In theory, no need for a top absorbing layer ($z_{top} \to \infty$)
- − Time dependent metric terms [$\pi_s = \pi_s(x, y, t)$]
- − Need for vertical integral operators (as well as vertical derivatives operators in EE case)

### Height-based

- + Time-independent Metric terms
- + Only derivative operators are involved
- − Need for a top absorbing layer ($z_{top} = Cst$)
- − Need to prescribe an hydrostatically-balanced ambient state. [for stability and accuracy reasons]

# Global vs. Local horizontal discretization Methods

## Spectral transform

- \+ Discretization method with the highest order of accuracy ⇒ help to achieve mimetic properties for discrete horizontal operators ⇒ no spurious sources [e.g, of vorticity $\nabla \times \nabla \Phi = 0$].
- − Global stencil ⇒ Strongly impose the use of Constant-coefficient Linear implicit problem ⇒ Stability issue over steep slopes.
- − Global communications (transpositions) → potential scalability issue.

## Finite-Volume

- \+ Very popular method in other NWP community (FV3, ICON, GungHo), using small local stencil ⇒ good scalability skills
- \+ Combined with flux-form approach ⇒ Good conservative property for the prognostic variables
- − Better mimetic property for the discrete horizontal pressure gradient force requires high-order FV schemes ⇒ may introduce spurious computational modes.

# Constant vs Non-constant Coefficients Implicit Schemes

## AROME - Constant Coefficients ICI

- Direct Spectral solver
  - + Exact solution (no inner-iteration)
- CC Iterative solver (replacing ST by local FD method )
  - + vertical separability allow to control the convergence rate ⇒ the number of inner-iterations can be set in advance.

− Stability issue at very steep slopes ($> 70^\circ$)

## FVM - Non-constant Coefficients - Iterative solver

+ metric terms are incorporated in the implicit part → Stable on steep slopes (up to 85°)

− non-separable implicit problem ⇒ Convergence is harder to control.

# Zängl steep orography experiment with FVM

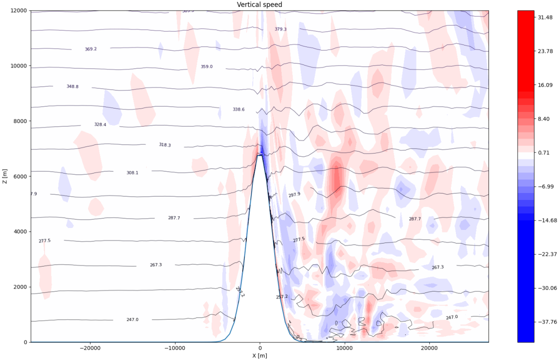


Figure 1: Zängl demanding experiment with orography : uniform horizontal wind $u = 20\ m.s^{-1}$ and isothermal initial conditions with a gaussian orography, maximum slope 75°. Results after 6 hours with $\Delta x = 30\,m$ and $\Delta t = 0.1\,s$.

# Solver : improving numerical efficiency

## Direct spectral solver

- − Poor weak-scalability due to global communications for the Spectral Transform

## Iterative Krylov GCR(k) solver

- + Near-constant weak scalability while increasing the resolution
- − Convergence rate depends on the prescribed ambient state
- − Global communication due to scalar-product (Gram–Schmidt orthogonalization process)

→ alternative : Multigrid methods

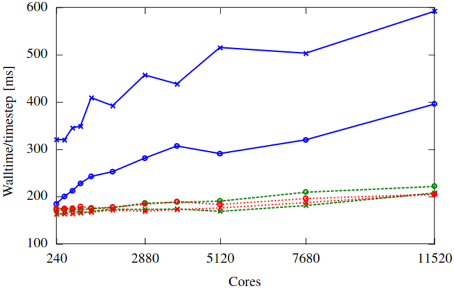


Figure 2: Degrauwe et al. (2020) : Weak-scalability experiments on spectral solver (solid blue), GCR(k) (dashed green), Richardson Multigrid (short-dashed red)

# Semi-Lagrangian vs. Eulerian MPDATA

## Pointwise Semi-Lagrangian scheme

- + Performs well with relatively long advective Courant number (CFL) between 4 and 10 : allowing long time-steps
- − Lipchitz stability condition becomes more stringent at cloud-resolving resolutions $\Rightarrow$ serious limitation on $\Delta t$ $\Rightarrow$ adversely affect the cost effectiveness of scheme.
- − Non conservative under severe flow deformation.

## MPDATA - *Multi Dimensional Positive Definite Advection Transport Algorithm*

- + Eulerian flux-form Conservative transport scheme
- − Conditional stability with CFL $< 0.5$ (implies very small time-steps due to more stringent vertical advective CFL, small $\Delta z/\Delta x$ aspect-ratio used in NWP)
- − Second-order accurate scheme at the best $\Rightarrow$ inherently diffusive scheme (more than high-order cubic SL).

→ Expecting MPDATA performances on GPU to compensate small timesteps

# FVM Code adaptation approach : Built on GT4Py + DaCe

GridTools 4python

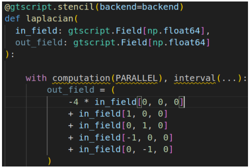
Python Domain Specific Language (DSL) for Weather and Climate HPC code generation

## GT4Py : GridTools for Python

- Portable across CPU and GPU (Nvidia, AMD) architectures
- Modularization of the code (dycore, physical packages) and OOP (Object Oriented Programming)
- Used by ICON (Exclaim), COSMO, and NOAA (FV3GFS)

## DaCe : Data Centric Parallel Programming

- Generating high-performance code for parts out of GT4Py
- DaCeML : Merging AI and Physics based models
  - Model inference using ONNX
  - Bindings with Pytorch



```
@gtscript.stencil(backend=backend)
def laplacian(
  in_field: gtscript.Field[np.float64],
  out_field: gtscript.Field[np.float64]
):

    with computation(PARALLEL), interval(...):
        out_field = (
            -4 * in_field[0, 0, 0]
            + in_field[1, 0, 0]
            + in_field[0, 1, 0]
            + in_field[-1, 0, 0]
            + in_field[0, -1, 0]
        )
```

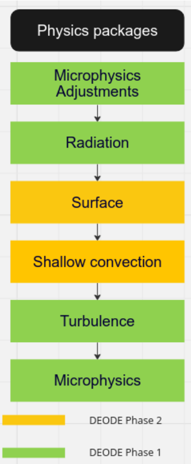
Figure 3: Laplacian operator in gt4py

# Physics for PMAP-FVM

## Porting manually physical processes from Fortran to GT4Py

- Finalizing and integrating GT4Py physics packages to PMAP-FVM :
  - ICE3 - Microphysics + Adjustments (MF)
  - ecRad (ECMWF)
  - Turbulence scheme from COSMO (ETHZ)

- (MF) Ongoing works (DEODE phase 2) :
  - CBR Turbulence TKE scheme used in AROME/Méso-NH
  - SURFEX : focus on options used operationnally in AROME
  - Shallow Convection



Physics packages

Microphysics Adjustments

Radiation

Surface

Shallow convection

Turbulence

Microphysics

🟨 DEODE Phase 2

🟩 DEODE Phase 1

Figure 4: Physics parametrizations

# On going works and perspectives

## Porting Physics packages to GT4Py

- Integration and Test of GT4py-ICE3 microphysics and GT4py-ecRad radiation scheme in PMAP-FVM.
- Development of GT4Py packages for SURFEX (Slim version), Shallow convection scheme, and CBR turbulence scheme.

## Running FVM on a realistic case over AROME domain

- Testing FVM over the Alps starting from AROME initial conditions and lateral boundaries.
- Perform like-to-like comparisons between AROME and FVM
- Further investigation on the convergence of the non-constant coefficient Iterative Solver and preconditionning.

# Translation of APLMPHYS into GT4Py (aside from FVM)

*Courtesy of Daan Degrauwe(RMI), Denis Haumont(RMI) and Santeri Karppinen (FMI)*

## Translation process : Fortran to GT4Py

- Using Loki to transform the Fortran code
  - Inlined called subroutines
  - Removed horizontal loops
  - Changed array dimensions and indices to stencil notations : e.g. (JLON, JLEV - 1 ) → (0, 0, -1)
- Using Loki python backend to generate python code
- Manual translation from python to GT4Py

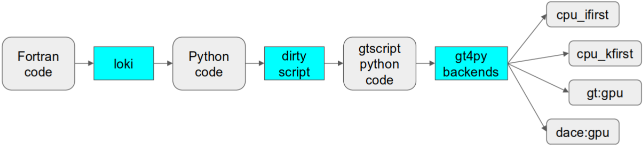


Figure 5: Translation toolchain : from Fortran to Python to GT4Py

# Is promise of performance portability reached ?

## Translation process : Fortran to GT4Py

- Portability is OK : APLMPHYS' gt4py version works out-of-the-box on AMD CPUs and GPUs
- Performance on LUMI-G (56 CPU threads, 1 GPU) :
    - First time using gt4py : need to build best practices on code optimization
    - APLMPHYS larger than examples from FVM

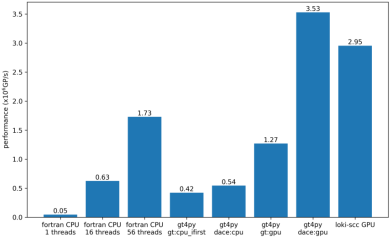


Figure 6: Performances of APLMPHYS GT4Py vs Fortran on LUMI

# References

- Kühnlein, C., Deconinck, W., Klein, R., Malardel, S., Piotrowski, Z. P., Smolarkiewicz, P. K., Wedi, N. P. (2019). FVM 1.0: A nonhydrostatic finite-volume dynamical core for the IFS. Geoscientific Model Development, 12(2), 651-676. doi:https://doi.org/10.5194/gmd-12-651-2019
- Degrauwe D, Voitus F, Termonia Piet. A non-spectral Helmholtz solver for numerical weather prediction models with a mass-based vertical coordinate. QJR Meteorol. Soc. 2020; 1-15.
- Burgot, T., Auger, L., and Bénard, P. (2021). Krylov solvers in a vertical-slice version of the semi-implicit semi-Lagrangian AROME model. Quarterly Journal of the Royal Meteorological Society, 147(736), 1497-1515.