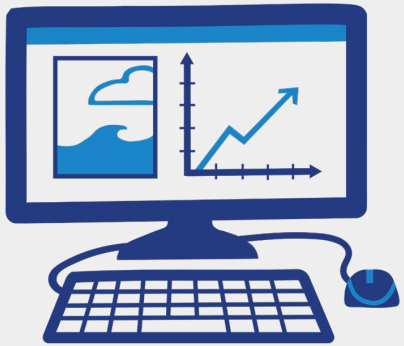


Pace and DSL Modeling
Oliver Elbert and Rusty Benson

Q1: Concerning GFDL's core strength of building and improving models of the weather, oceans, and climate for societal benefits, how can GFDL leverage advances in science and computational capabilities to improve its key models? What are the strengths, gaps, and new frontiers?

NOAA GEOPHYSICAL FLUID DYNAMICS LABORATORY

Performance Portability

GPU Supercomputing is dominant in exascale machines. To take advantage of these performance advances any application must be GPU-optimized.

Operational and research computers at NOAA[1] are primarily CPU-based; maintaining CPU capability and performance is a necessity for current modeling products and model development.

[1] *NOAA HPCC Supercomputing*

#	System	Nodes	Power [MW]	Rmax [PFlop/s]	Chip Technology
1	ElCapitan	11,136	29.6	1,742.0	1 x AMD, 4 x MI300X
2	Frontier	9,472	24.6	1,353.0	1 x AMD, 4 x MI250X
3	Aurora	10,624	38.7	1,012.0	2 x Xeon, 6 x IntelGPU
4	Eagle	3,600	–	561.2	1 x Xeon, 4 x NVidia H100
5	HPC6	3,330	8.46	477.9	1 x AMD, 4 x MI250X*
6	Fugaku	158,976	29.9	442.0	1 x A64FX
7	Alps	10,400	7.12	434.9	1 x G100, 1 x NVidia H100
8	Lumi	2,916	7.10	379.7	1 x AMD, 4 x MI250X*
9	Leonardo	3,456	7.49	241.2	1 x Xeon, 4 x NVidia A100
10	Tuolumne	4,608	3.39	208.1	1 x AMD, 1 x MI300X

The top 10 fastest supercomputers as of November 2024. GPU accelerators are highlighted in red. Credit: top500.org

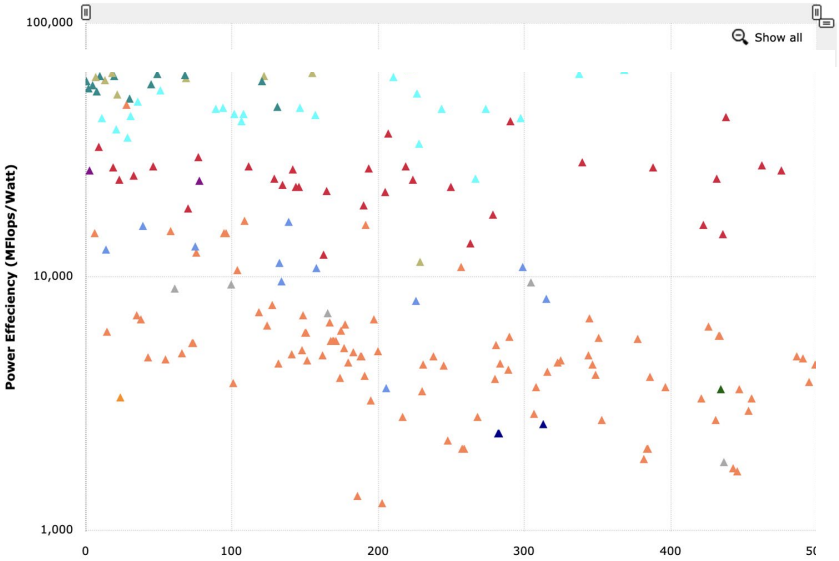
An Exascale Atmosphere

Weather and climate modeling can make excellent use of exascale technologies. Increasing performance will allow atmospheric models to **increase resolution**, improving representations of orography, convection, and turbulence, and will allow for **larger ensembles** to be run at current resolutions, increasing the statistical power of ensemble forecasts.

GFDL will use GPU supercomputing to advance our scientific modeling to advance our studies of tropical cyclones, extreme weather, and air-land-sea interactions.

GPUs can be an order of magnitude more energy-efficient than CPUs on a per-flop basis (~60,000 vs ~6,000 Mflops/Watt) but models need to effectively use the GPUs to realize those gains

CPU performance is still necessary for NOAA as our primary computational resources are still CPU-based[1]



Power efficiency of the top 500 supercomputers colored by hardware accelerator. Orange triangles show unaccelerated machines which are almost entirely CPU-based. The latest generation of GPU architectures are shown in teal, olive, and cyan, representing AMD MI300, NVIDIA GH100, and NVIDIA H100 systems. Credit: top500.org

[1] *NOAA HPCC Supercomputing*

Domain Specific Languages, GT4Py, and DaCe

A domain specific language (DSL) is a programming language or library that is designed for a particular set of tasks: e.g. TensorFlow for machine learning, LaTeX for typesetting.

GT4Py is a DSL for weather and climate modeling developed by a team of computer scientists at ETH Zurich.

Computation is done in stencils (stateless functions) that express algorithms in a 3D domain. The GT4Py toolchain automatically optimizes stencils for target hardware architectures or can run stencils directly using NumPy.

DaCe is a separate compilation framework for optimizing code based on data flow within an application. We use DaCe to optimize infrastructure code between GT4Py stencils and further optimize stencil code by analyzing data flow within and between numerical operations. DaCe is developed by ETH Zurich's Scalable Parallel Computing Laboratory.

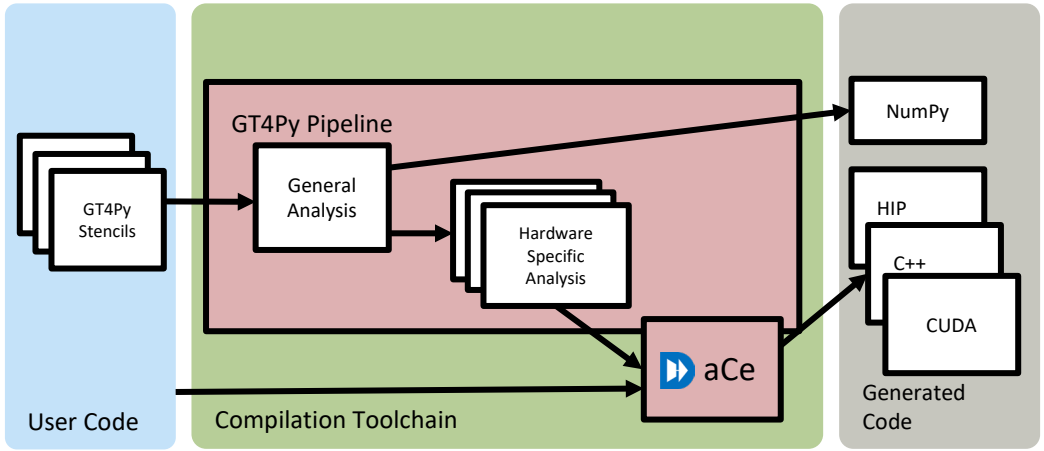


Illustration of the GT4Py and DaCe toolchain. User code is written in Python with numerics expressed as GT4Py stencils. The GT4Py compiler analyzes the stencils and either outputs NumPy code or optimizes them for the target hardware. DaCe similarly optimizes the non-stencil code and the dataflow between GT4Py stencils. The compiler then generates optimized code to run on the target HPC system.

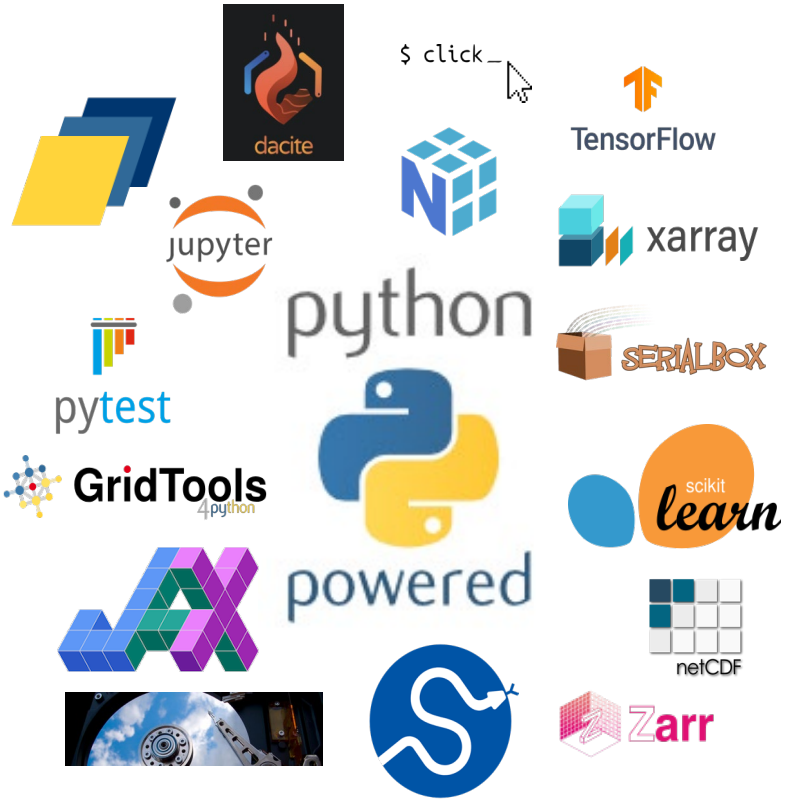
Python as a Modeling Language

Beyond GT4Py, Python is an attractive language for model development. It has been widely adopted for post-processing and data analysis within the scientific modeling community and is more accessible to non-specialists than classical programming languages.

The Python ecosystem provides access to a wide range of tools for model development. Pytest and pdb allow developers and engineers to easily test and debug code. Jupyter notebooks allow for interactive model development, where a scientist can immediately investigate and share the results of any numerical or algorithmic changes.

Python is the language of choice for AI/ML, and a model written in Python can easily be used to train an ML model or directly coupled to ML model components. Python also provides access to libraries like JAX which can calculate gradients of model code to introduce differentiability, opening new avenues for inline AI/ML model development and data assimilation.

Continuing with Fortran specialization limits the diversity of programmers we can hire and their ramp-up time to productivity.



The Pace model

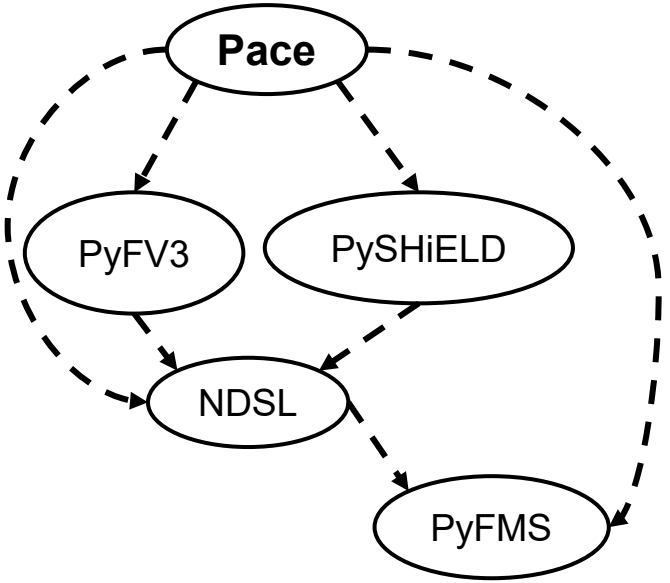
Pace is a Python port of the SHiELD weather model written in GT4Py.

Initially developed as a proof-of-concept at the Allen Institute for Artificial Intelligence in collaboration with GFDL, development and ownership transferred to GFDL in January 2023.

Pace is comprised of PyFV3 dynamical core and PySHiELD physics, and uses the NOAA/NASA DSL Middleware (NDSL) to interface with GT4Py and DaCe.

Pace integrates into the GFDL modeling ecosystem via a Python interface to FMS (PyFMS).

Pace and all components run out-of-the-box, within Docker containers, and contain example Jupyter notebooks to demonstrate how to use them.

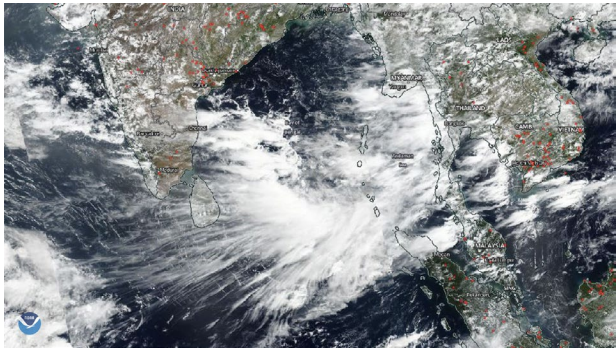


https://github.com/NOAA-GFDL/pace

Scientific Targets and Applications

Our initial targets for scientific applications of Pace are **large eddy simulations** at 50-100 m resolution for cloud studies, using the radiation, microphysics, and turbulence schemes in SHiELD. This will allow us to probe the mechanisms of features like anvil cloud formation and improve their representation in GCMs.

Once full physics are implemented in Python, we will extend our scientific efforts to **high-resolution global** studies of tropical cyclones, with the goal of improving hurricane forecasts, and further high resolution studies of weather and climate dynamics.



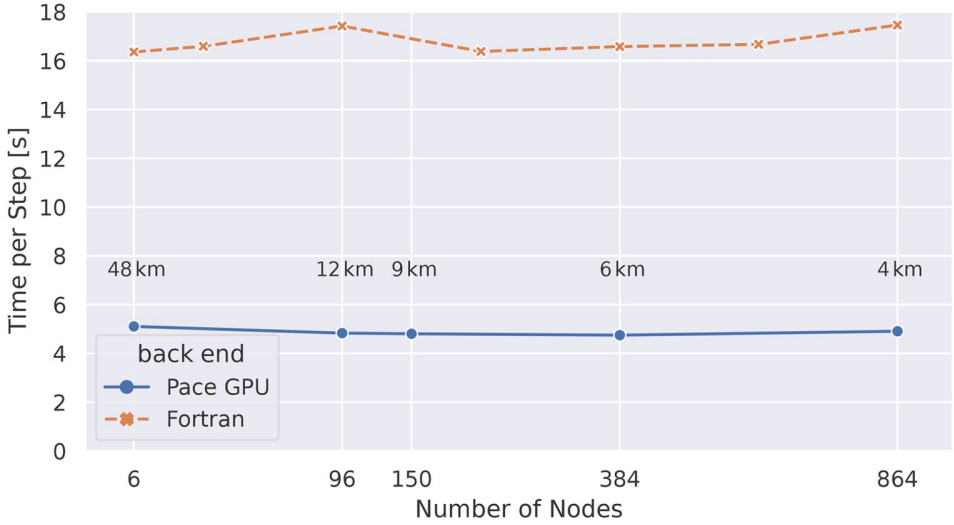
Anvil clouds over the Bay of Bengal. Image credit: NESDIS

PyFV3

AI2's Climate Modeling DSL team prototyped the initial port of the FV3 dynamical core and conducted preliminary GPU performance engineering studies. This port only supported high-resolution modeling on a uniform-resolution cubed sphere, but did demonstrate the viability of the Python DSL as a solution for performance-portable climate modeling.

We are extending PyFV3 to cover doubly-periodic domains, nested and stretched grids, and to encompass the full breadth of the numerical schemes used in FV3 applications at multiple resolutions.

This development is a collaboration with NASA GSFC's Advanced Science & Technology group, who are implementing it in the GEOS model.



Weak scaling performance comparison between the PyFV3 dynamical core and GFS microphysics and the original Fortran code, from Dahm et al. 2022. Numbers above the blue line show horizontal resolution in km. This analysis was performed on the Piz Daint supercomputer with one P100 GPU per node

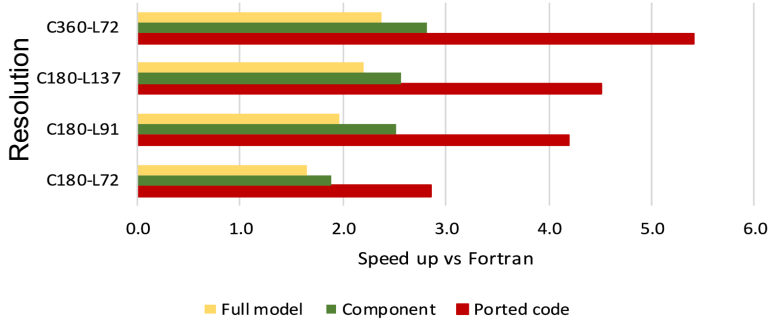
Applications of PyFV3

PyFV3 can be a drop-in dynamical core for other models both within GFDL (e.g. AM5) and NOAA more broadly, such as the UFS and HAFS.

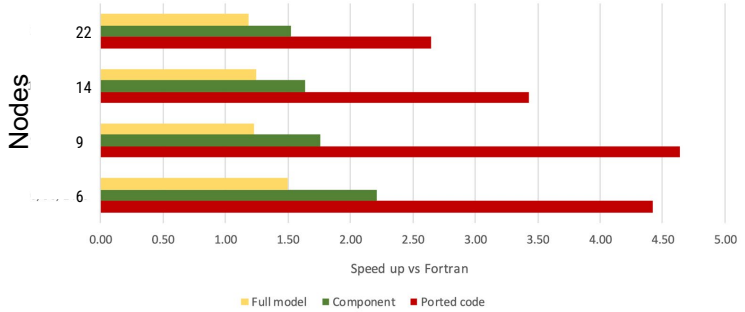
As part of our collaboration with NASA, PyFV3 is integrated into the GEOS model using a hybrid Python-Fortran paradigm yielding good initial performance gains. Just like GFDL, NASA is developing GT4Py ports of the GEOS physics to couple directly to PyFV3.

Until the full atmospheric system is implemented in GT4Py, data transfer between PyFV3 on the GPU and the physics running on the CPU limits the overall performance boost from PyFV3 to a factor of 2 speedup over existing Fortran.

This approach allows us to rewrite models for performance portability component-by-component



PyFV3-based GEOS speedup compared to Fortran FV3 at multiple resolutions on two ranks of Discover. Yellow shows speedup of the full model, green shows the boost to the dynamical core including the Fortran-Python interface, red shows the speedup to the ported sections of code



Strong scaling results from GEOS simulations at 12.5 km resolution, showing performance boosts from using PyFV3 with increasing resources. Colors are the same as in the above plot.

PySHiELD

PySHiELD is a GT4Py port of the atmospheric physical parameterizations from the SHiELD model family.

Preliminary ports of many physics schemes were written by the AI2 team using older versions of GT4Py, which lacked many features that are now available, and many of the parameterizations have been updated in the subsequent years.

The list of physical parameterizations from SHiELD to be included are:

- the sea-ice and slab ocean models have been ported and validated, as has the surface flux parameterization
- version 3 of GFDL's cloud microphysics (Zhou et al. 2022) has been ported, but is waiting on new features in GT4Py to be released prior to integration into Pace
- a port of SHiELD's TKE-EDMF PBL scheme is complete
- the shallow convection scheme port is finished
- the NOAH land-surface model has been rewritten in GT4Py and is currently in-test
- the orographic and convective gravity wave drag schemes have not yet been ported

Because our scientific targets will resolve deep convection explicitly, there is no need for a deep convection parameterization. Radiation will use a GPU-capable version of the RTE-RRTMGP (Pincus et al. 2019) radiation scheme via a Python interface under development.

PySHiELD is Advancing Science Globally

Our work on PySHiELD is pushing forward the state-of-the-art in DSL modeling and computer science.

Early versions of GT4Py only supported numerical operations of simple stencils. As the AI2 team developed PyFV3 more features were added to cover the algorithmic requirements of the FV3 dycore.

Our development of PySHiELD is similarly pushing GT4Py to support more computational motifs that are needed for physical parameterizations, such as the GFDL microphysics. Incorporating these features into the GT4Py compiler is compelling further developments in computer science to allow automatic optimization of more complex numerical algorithms.

By providing access to the computational power of GPUs, PySHiELD enables more accurate model numerics, e.g. directly calculating quantities like saturation vapor pressure in place of interpolated lookup tables.

PySHiELD is leading the way for other modeling centers that are adopting GT4Py and DaCe (ECMWF, NASA GSFC, MeteoSwiss, etc.), prototyping DSL features for the other centers to use in their model development.

NDSL: The NOAA/NASA DSL Middleware

Developed in collaboration with NASA GSFC, NDSL is our foundation for building DSL models. It contains everything needed in one interface.

NDSL gives model developers a simple interface to GT4Py functionality and smooths the incorporation of DaCe within the GT4Py toolchain. NDSL also has entrypoints for performance engineers to access the DSL layers to improve performance.

NDSL provides simple abstractions for common computational patterns such as GT4PY stencil compilation, array allocation, domain decomposition, mpi communication, and halo exchanges. Stencils for frequently-used numerical functions such as solving tridiagonal matrices, converting to lat-lon grids, and efficient array copies are also included in NDSL.

NDSL can easily be extended to support the tightly-coupled use of machine learning applications or differentiating model components via JAX, TensorFlow, etc.

NDSL Makes Modeling Easier

For example, the StencilFactory pattern abstracts the compilation of GT4Py stencils, requiring only the numerical function, the computational boundaries, and any compile-time variables. Hardware targets (e.g. CPU or GPU), optimization passes, and configuration specifics are handled inside NDSL.

Similarly, halo exchanges are handled by invoking NDSL's WrappedHaloUpdater object by simply specifying which variables are to be updated. The defined methods in the NDSL will ensure the proper data extents are communicated.

```python
from ndsl import StencilFactory
def compute_x_flux(
    q: FloatField, courant: FloatField, dxa: FloatFieldIJ, xflux: FloatField
):
    from __externals__ import mord

    with computation(PARALLEL), interval(...):
        if __INLINED(mord < 8):
            al = compute_al(q, dxa)
            xflux = get_flux(q, courant, al)
        else:
            bl, br = compute_blbr_ord8plus(q, dxa)
            xflux = get_flux_ord8plus(q, courant, bl, br)
```

An Example stencil function computing horizontal fluxes in PyFV3

```python
        self._compute_flux_stencil = stencil_factory.from_origin_domain(
            func=compute_x_flux,
            externals={
                "mord": abs(iord),
            },
            origin=origin,
            domain=domain,
        )
```

The model code that compiles the stencil using NDSL's StencilFactory pattern. The stencil factory takes the function and the computational boundaries for it, here specified as an origin and domain for computation, and any compile-time variables –"externals", and returns a compiled stencil. This serves as a simple frontend for GT4Py's more complex and detailed stencil compilation infrastructure and abstracts the interface to DaCe as well.

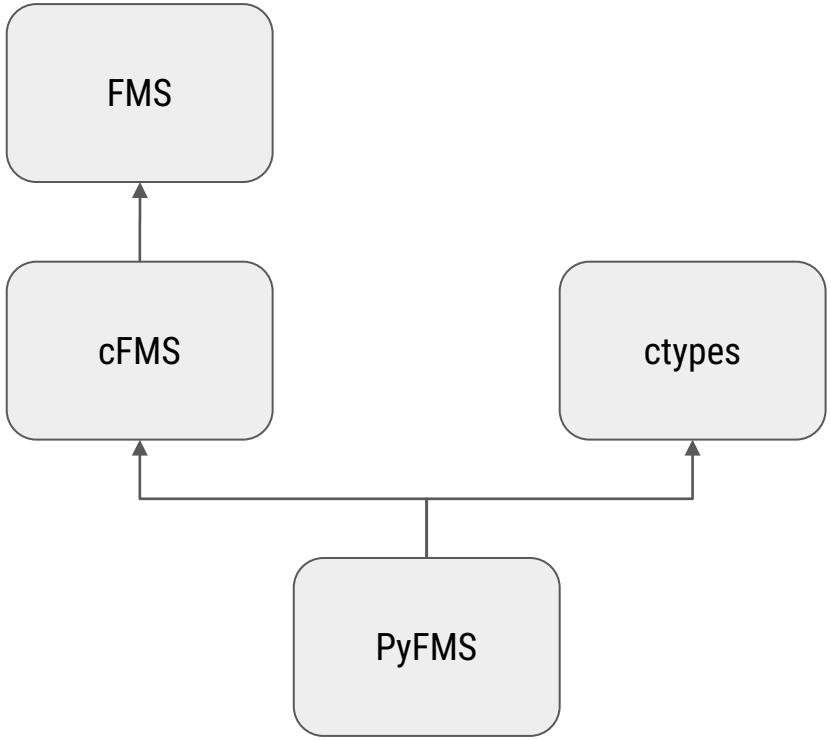
PyFMS

Our long-term goal to bring Pace and GT4Py models fully into the GFDL modeling ecosystem will require Pace to use the Flexible Modeling System (FMS). Because FMS is not directly in the computational path, we can interface to it without the need for porting the logic to GPU.

Instead, we are constructing PyFMS, a Python interface to FMS. This will allow Pace to couple to other GFDL model components such as MOM6 and provide direct support of nested and stretched grids targeted at high-resolution configurations, e.g. T-SHiELD and C-SHiELD.

The development of PyFMS will also allow us to support future models and products written in Python.

By building a C-layer (cFMS) between the Fortan and Python, we provide a mechanism for calling FMS primitives directly from NDSL.



Summary and Future Plans

Pace is a performance-portable version of SHiELD, with science-readiness within FY25, allowing us to unlock new scales of modeling for atmospheric and meteorological research with the goal of improving forecasts and projections. As we work towards the goal of using Pace for scientific research, we will continue to enhance Pace components to support more numerical motifs and further model development, while also improving the experience of modeling in Python.

The NDSL middleware layer is a powerful platform for model development and can be used to create DSL versions of other atmospheric models within GFDL and NOAA broadly, letting them take full advantage of leadership-class supercomputers.

PyFMS will bring Pace fully within the GFDL modeling system. Achieving this milestone will allow us to seamlessly couple Pace to MOM6 and also facilitates using PyFV3 as the dynamical core for the next generation of GFDL models (i.e. AM5).

Projects to support SHiELD's full breadth of capabilities in Pace are underway. These projects will enable the use of nested and stretched grids, regional modeling, as well as the duo-grid halo projection.

Collaborations and Funding Sources

PyFV3 and NDSL are being co-developed with NASA GSFC's Advanced Science and Technology Group, who are using PyFV3 and the NDSL in the GEOS model.

GFDL is a leading member of the GT4Py and DaCe communities, and we are collaborating with ETH Zurich on their development.

NOAA's Global Systems Laboratory has begun work on GT4Py ports of the Community Common Physics Package and we are supporting them and their efforts.

The Pace project is funded by the Global Nest Initiative and the Software Engineering for Novel Architectures project.