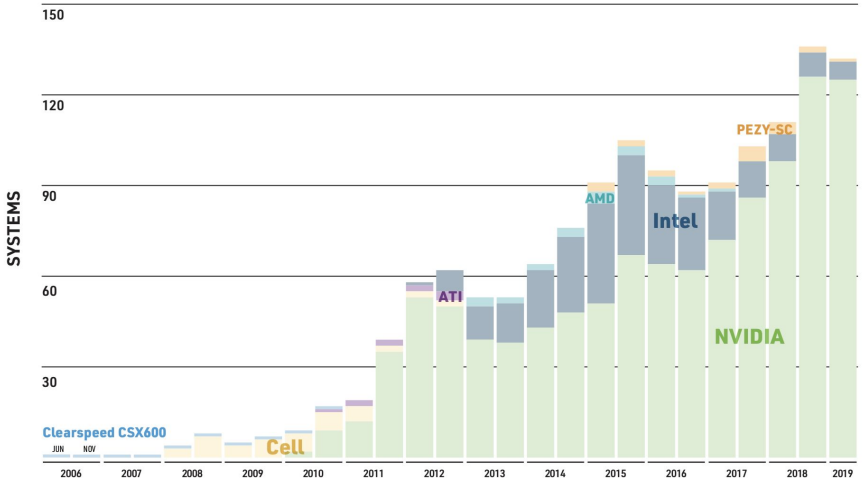
Pace: A Python-Based Implementation of FV3GFS for GPU and CPU Supercomputers

Oliver Elbert, Johann Dahm, Florian Deconinck, Jeremy McGibbon, Tobias Wicky, Elynn Wu, Oliver Fuhrer, Lucas Harris, Rusty Benson

AI2

The Future of HPC

- Power constraints lead to increased use of accelerators (GPUs, FPGAs, etc.)
- Architectures are rapidly changing



- Climate models will need to adapt to new hardware

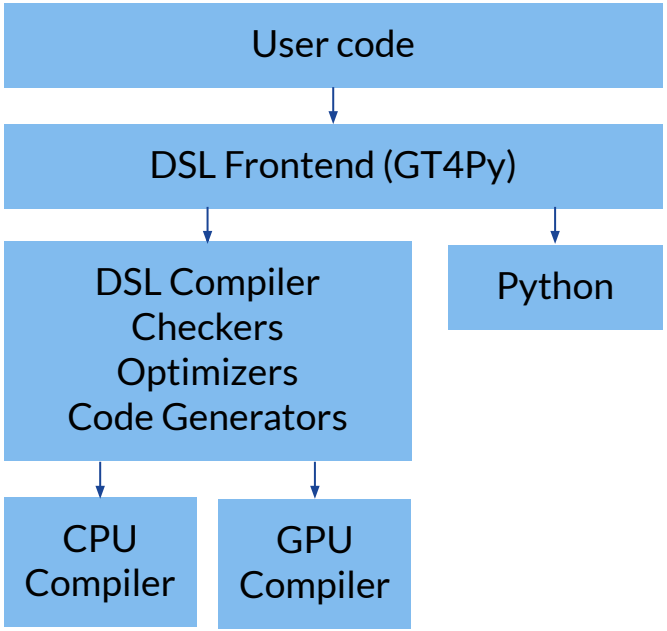
#	System	Nodes	Power [MW]	Rmax [PFlop/s]	Chip Technology
1	Frontier	9,472	21.1	1,102.0	1 x AMD, **4 x 250X**
2	Fugaku	158,976	29.9	442.0	1 x A64FX
3	Lumi	2,650*	2.9	151.9	1 x AMD, **4 x 250X***
4	Summit	4,608	10.1	148.6	2 x Power9, **6 x V100**
5	Sierra	4,474	7.4	94.6	2 x Power9, **4 x V100**
6	TaihuLight	40,960	15.4	93.0	**1 x SW26010**
7	Perlmutter	1,536	2.5	64.6	1 x AMD, **4 x A100**
8	Selene	560	2.6	63.5	2 x AMD, **8 x A100**
9	Tianhe-2A	16,000	18.5	61.4	2 x Xeon, **3 x Xeon Phi**
10	Adastra	338*	0.92	46.1	1 x AMD, **4 x 250X***

GT4Py: GridTools for Python

- Python has a growing presence in the atmospheric science community

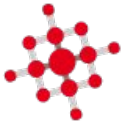
- Python too slow for execution on a supercomputer, but effective as the ‘front end’ to a DSL that generates efficient code

  https://github.com/GridTools/gt4py



Motivations

- One codebase for CPU and GPU
- Better optimizations than traditional compiler
- Separate optimization logic from science logic
  - No re-used variables, no 1000-line functions
- Easier to use code base



GridTools

https://github.com/ai2cm/pace

Ai2

Pace

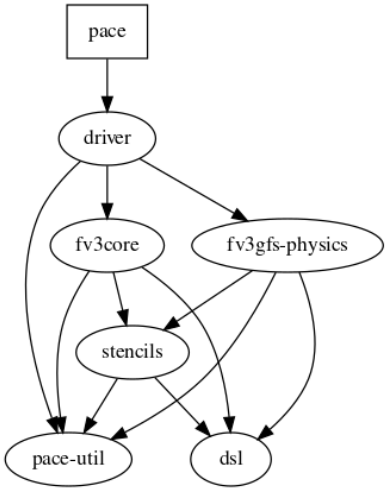
Fortran

UFS
UNIFIED FORECAST SYSTEM

GFDL FV3

GFS
Global Forecast System

Python



https://github.com/ai2cm/pace

Ai2

Example Conversion

Fortran

```
subroutine del2_cubed(q, cd, del6_v, del6_u, rarea, grid)

real :: fx(is:ie+1, js,je), fy(is:ie, js:je+1)
!$OMP parallel do default(none) shared(km, q,&
!$OMP is,ie,js,je, & cd) &
!$OMP private(fx, fy)
do k = 1, km
  do j = js, je
    do i = is, ie + 1
      fx(i,j) = del6_v(i,j) * ( q(i-1,j,k) - q(i,j,k) )
    enddo
  enddo

  do j = js, je + 1
    do i = is, ie
      fy(i,j) = del6_u(i,j) * ( q(i,j-1,k) - q(i,j,k) )
    enddo
  enddo

  do j = js, je
    do i = is, ie
      q(i,j,k) = q(i,j,k) + cd * rarea(i,j) * (
          fx(i,j) - fx(i+1,j) + fy(i,j) - fy(i,j+1) )
    enddo
  enddo
enddo
...

end subroutine del2_cubed

call del2_cubed(q, cd, del6_v, del6_u, rarea, grid)
```

Pace

```
@gtscript.function
def delx(q, weight):
    return weight * (q[-1, 0, 0] - q)

@gtscript.function
def dely(q, weight)
    return weight * (q[0, -1, 0] - q)

@gtscript.stencil(backend='numpy')
def del2_cubed(q:field, rarea:field, del6_v:field, del6_u:field, cd:float):
    with computation(PARALLEL), interval(...):
        fx = delx(q, del6_v)
        fy = dely(q, del6_u)
        q = q + cd * rarea * (fx - fx[1, 0, 0] + fy - fy[0, 1, 0])

del2_cubed(q, del6_u, del6_v rarea, cd,
           origin=grid.compute_origin(), domain=grid.compute_domain())
```

- Horizontal loops removed, schedule removed
- Index offsets instead of absolute indices
- No explicit storage statements for temporary variables
- Overhead-free, reusable functions -- inlining
- Less code
- No explicit parallelism or data storage layout
- Escaping into straight Python is possible

Python Development

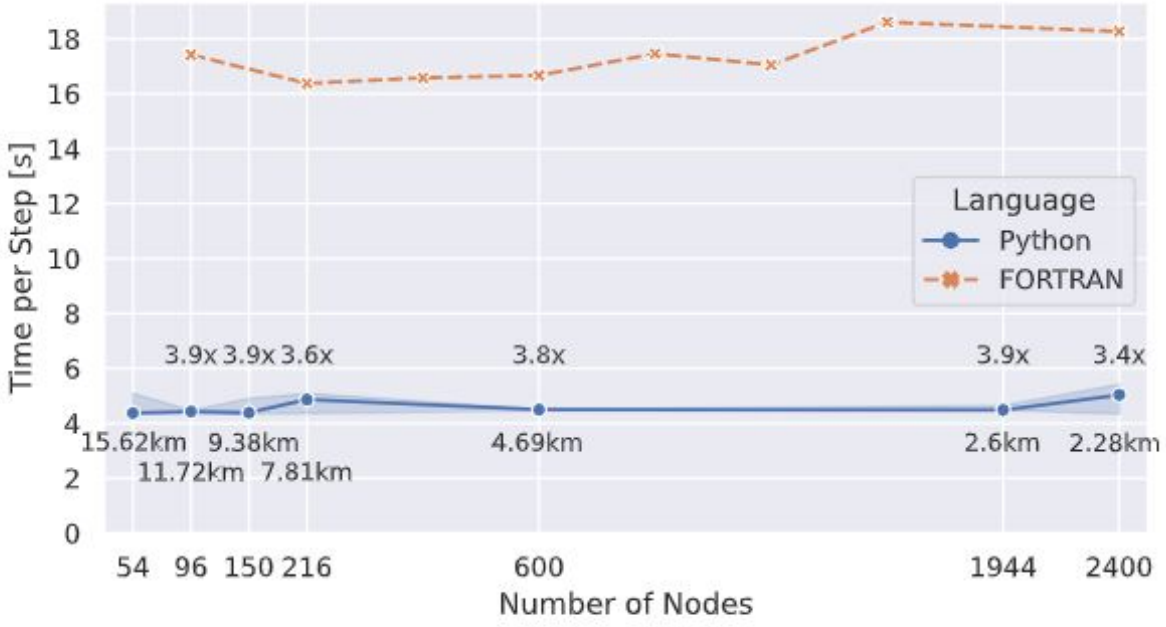
Debugging in Python:

- pdb is amazing (built-in to Python)
- Access to Python tooling when debugging
- instead of adding print statements, add plotting or netcdf dumping

```
import pdb;pdb.set_trace()
self._ut_main(
    self._utmp,
    uc,
    v,
    self._cosa_u,
    self._rsin_u,
    utc,
)
import xarray as xr
xr.Dataset(
    data_vars={
        "uc": xr.DataArray(uc),
        "vc": xr.DataArray(vc),
        "utc": xr.DataArray(utc),
    },
).to_netcdf("d2a2c_vect_0.nc")
```

- Easy to run and test single module or subtract dycore from itself

Performance



(Submitted as SC'22 paper!)

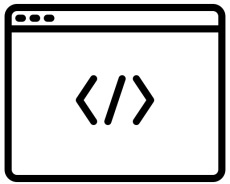
Benchmark on Piz Daint supercomputer at CSCS (Intel Haswell, NVIDIA P100)

CPU performance still roughly 2x slower than Fortran

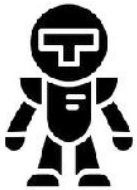
Simulation throughput of 0.12 SYPD at 2.6 km grid spacing

Separation of concerns in action

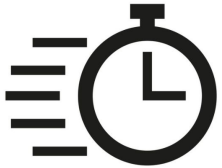
6
weeks of
work



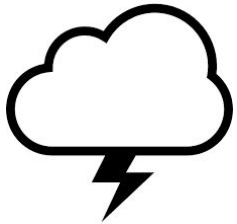
10
performance
code revisions



4
performance
developer



1.8x to 3.5x
speed up
over Fortran



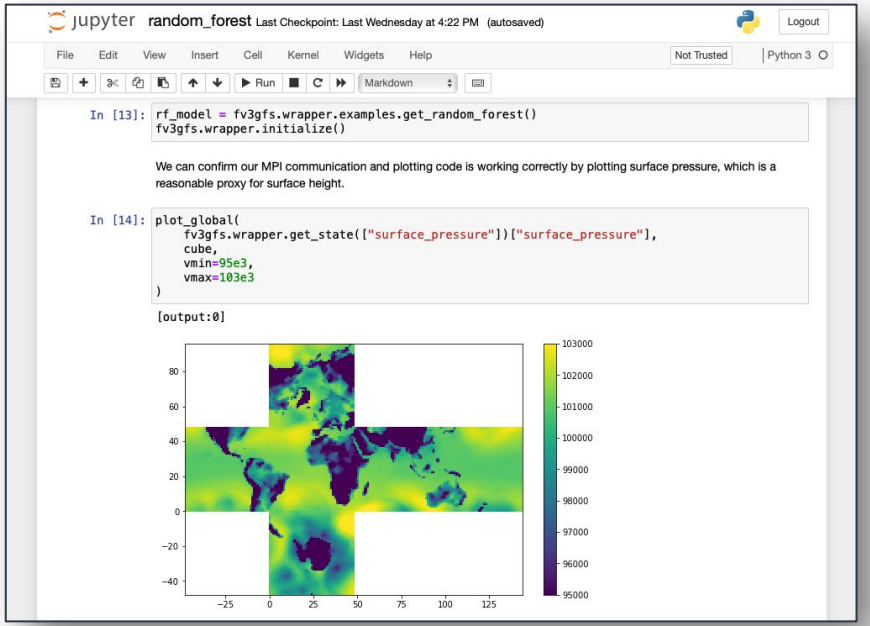
0
model
code revision

Benefits

- Performance
- One codebase for model
- Simple code
- Focus on numerics, not optimization
- Python ecosystem
- Ease of development and debugging

- Public!

https://github.com/ai2cm/pace



Thanks!

The AI2 DSL Team

Oliver Elbert

Johann Dahm

Florian Deconinck

Oliver Fuhrer

Jeremy McGibbon

Tobias Wicky

Elynn Wu

Partners

Geophysical Fluid Dynamics Laboratory GFDL NOAA

NASA

CSCS

MeteoSwiss

ETH zürich

SPCL

UNIVERSITY of WASHINGTON

AI2

This slide intentionally left blank

Solutions

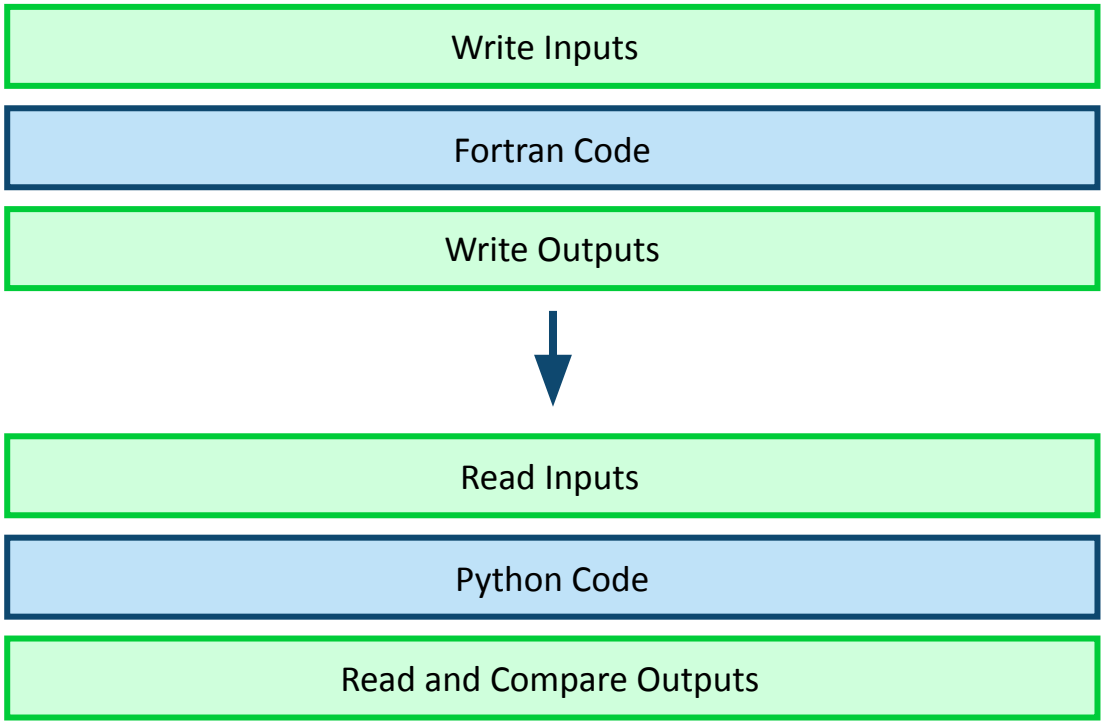
- Rewrite for backend (e.g. CUDA, AMD-HIP)
  - Multiple hardware-specific codebases
- Compiler Directives
  - Increases code complexity
- How to ‘future proof’ for the next hardware advancement?
- Domain Specific Languages

```
!---- Local automatic arrays
!$acc present ( pa1c,pa1f,pa2c,pa2f,pa3c,pa3f                    ) &
!$acc present ( ztu1,ztu2,ztu3,ztu4,ztu5,ztu6,ztu7,ztu8,ztu9     ) &
!---- Module arrays
!$acc present ( cobi,coali,cobti                                 )

!$acc parallel
!$acc loop gang vector(32) collapse(2)
DO j3 = ki3sc, ki3ec+1
  DO j1 = ki1sc, ki1ec
    pflfd(j1,j3) = pbbr(j1,j3)
    pflcd(j1,j3) = 0.0_dp
  ENDDO
ENDDO
!$acc end parallel

#ifdef _OPENACC
  !$acc parallel
  !$acc loop gang vector(32)
  DO j1 = ki1sc, ki1ec
    CALL coe_th_gpu(pduh2oc (j1,ki3sc), pduh2of (j1,ki3sc), &
                    pduco2  (j1,ki3sc), pduo3   (j1,ki3sc), &
                    palogp  (j1,ki3sc), palogt  (j1,ki3sc), &
                    podsc   (j1,ki3sc), podsf   (j1,ki3sc), &
                    podac   (j1,ki3sc), podaf   (j1,ki3sc), &
                    pbsfc   (j1,ki3sc), pbsff   (j1,ki3sc), &
                    kspec   , kh2o    , kco2    , ko3     , &
                    pa1c(j1), pa1f(j1), pa2c(j1),           &
                    pa2f(j1), pa3c(j1), pa3f(j1)            )
  ENDDO
  !$acc end parallel
#else
  CALL coe_th ( pduh2oc,pduh2of,pduco2 ,pduo3  ,palogp ,palogt , &
                podsc  ,podsf  ,podac  ,podaf  ,pbsfc  ,pbsff  , &
                ki3sc  ,kspec  ,kh2o   ,kco2   ,ko3    ,       &
                ki1sd  ,ki1ed  ,ki3sd  ,ki3ed  ,ki1sc  ,ki1ec  , &
                ldebug_coe_th  ,jindex ,                        &
                pa1c   ,pa1f   ,pa2c   ,pa2f   ,pa3c   ,pa3f)
#endif
```

Testing - Fortran to Python

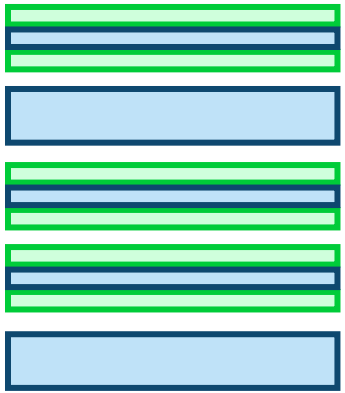


https://github.com/ai2cm/pace

Development and Testing

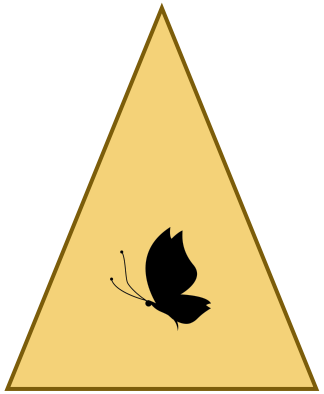
Read Inputs

Larger block of Code



Read and compare Outputs

Error Growth



https://github.com/ai2cm/pace

Ai2

DYAMOND Project

(Satoh et al., 2017; Stevens et al., 2019)

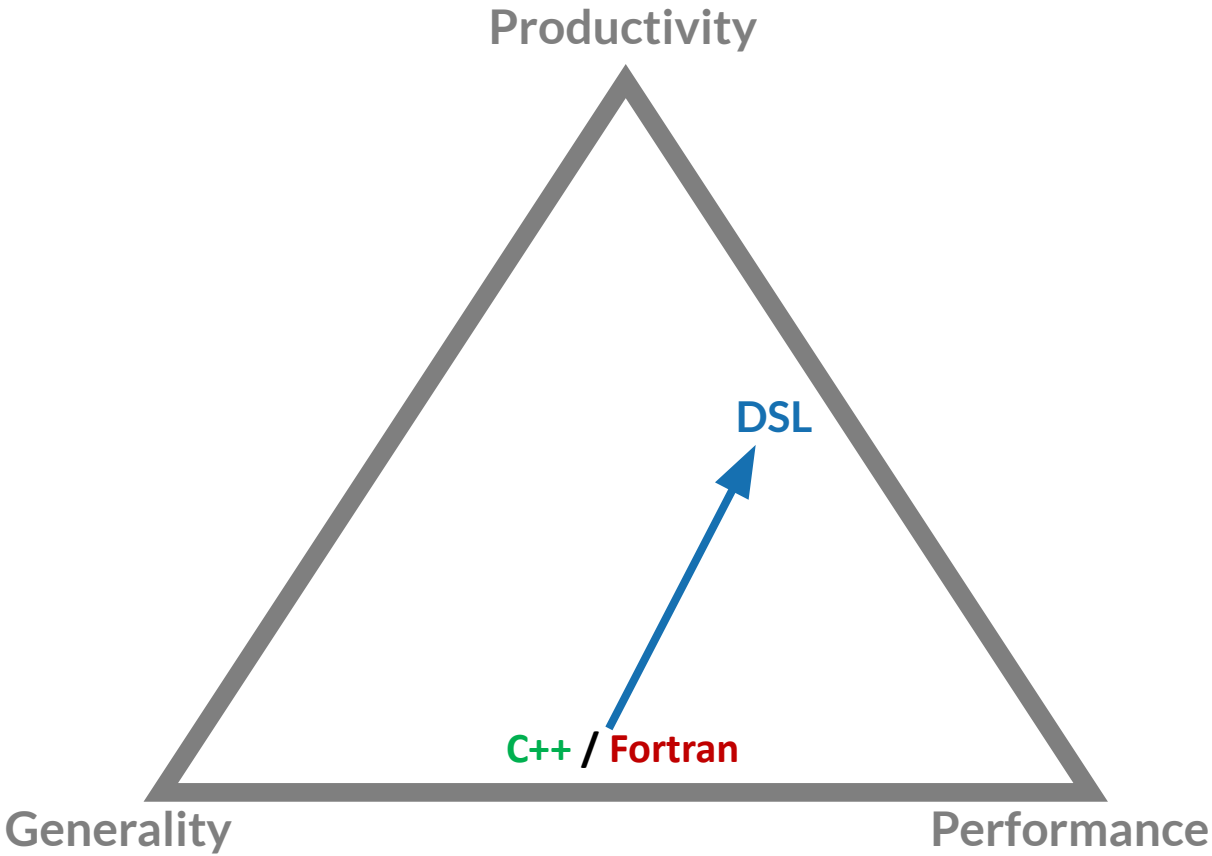
adapted from Stevens et al., 2019

All models simulated 40 days beginning Aug 1, 2016 00Z

*Projected SDPD @ 1km accounts for horizontal grid size, number of levels, timestep, processor type, missing coupling and assumes ideal scalability onto full UK MetOffice supercomputer (6720 nodes, Cray XC40, Intel Xeon Broadwell, #27)

Model	Grid	Scheme	SDPD	Δt	Nodes	Cores	Processor	SDPD*
ARPEGE-NH	Gaussian red	SI/SL	**2.6**	100 s	300	7,200	Intel Xeon	**1.8**
FV3GFS	Cube, C-D	HEVI, split	**19**	4.5 s	384	13,824	Intel Xeon	**9.1**
GEOS-5	Cube, C-D	HEVI, split	**6.2**	3.75 s	512	20,480	Intel Xeon	**2.2**
ICON	Icosah, C	HEVI, split	**6.1**	4.5 s	540	12,960	Intel Xeon	**4.0**
IFS	Octah, spectral	Hydrostatic	**124**	240 s	360	12,960	Intel Xeon	**12**
MPAS	Voronoi, C	HEVI, split	**3.5**	10 s	256	9,216	Intel Xeon	**1.1**
NICAM	Icosah, A	HEVI, split	**2.6**	10 s	640	2,560	NEC SX-ACE	**-**
SAM	Lat-Lon, C	Anelastic	**6.0**	7.5 s	128	4,608	Intel Xeon	**0.013**

DSL Approach

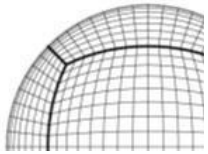


Separation of Concerns

PDEs

$$\nabla^2 u$$

Grid



Discretization

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

Implementation

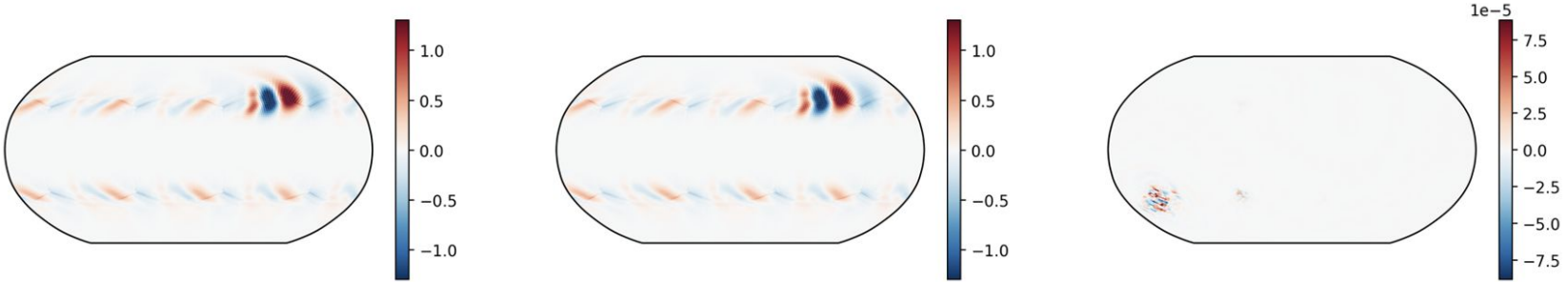
```
for i = 1, ni
  for j = 1, nj
    for k = 1, nk
      u[i+1,j,k] – 2 u[i,j,k] + u[i-1,j,k]
```

Optimization

```
!$acc parallel present(u)
!$acc loop vector collapse(3)
for i = 1, ni
  for j = 1, nj
    for k = 1, nk
      u[i+1,j,k] – 2 u[i,j,k] + u[i-1,j,k]
```

Dynamical Core (FV3)

Baroclinic instability testcase (Jablonowski and Williamson 2006)
6 day surface temperature anomaly [K]



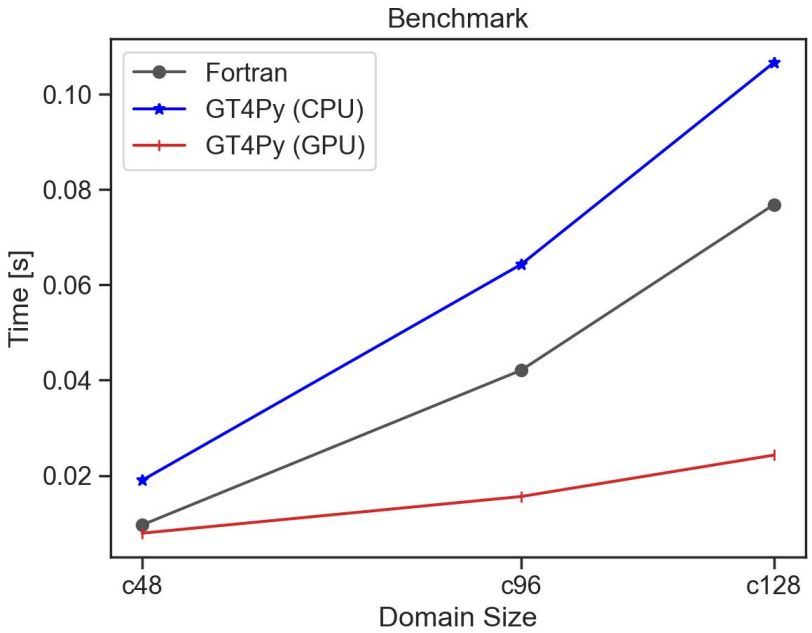
Performance

Also have GFS physics parameterizations ported, encouraging performance

Initial port of Microphysics scheme at c128 (no optimization):

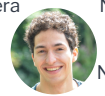
- GPU: 3.2x faster than Fortran
- CPU: 1.4x slower

Again, plenty of speedup on the table



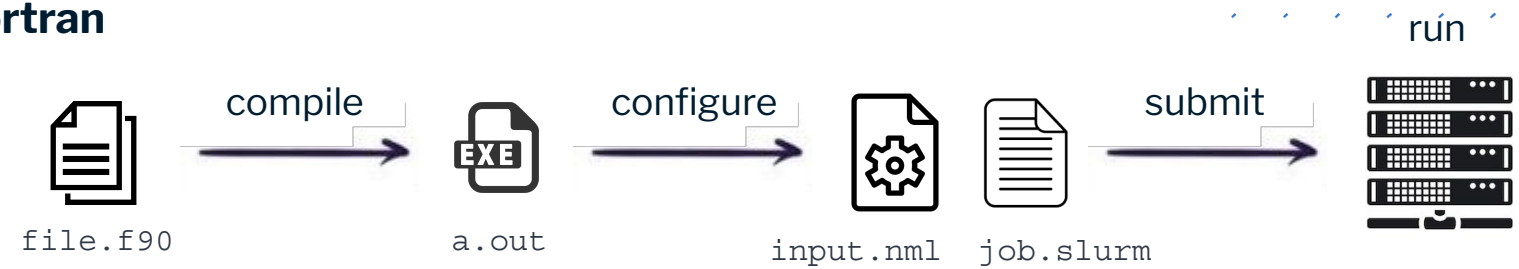
Physical Parameterizations

Check it out on GitHub!
https://github.com/ai2cm/physics_standalone

	Microphysics	PBL &Turbulence	Sea-Ice	Shallow Convection	LSM	Radiation
Authors	 Mikael	 Chris	 Vera  Nadja  Nicolai	 Mikael  Chenwei  Langwen	 Safira	 Andrew
Scheme	GFDL Cloud Microphysics Scheme	GFS scale-aware EDMF PBL and Free Atmospheric Turbulence Scheme	GFS Sea Ice Scheme	GFS SAS-based Mass-Flux Scheme for Shallow convection (sa-MF)	GFS Noah Land Surface Model	GFS RRTMG
Status	Ported	Ported	Ported	Ported	Ported	Ported

Just-in-time (JIT) compilation

Fortran



Python DSL