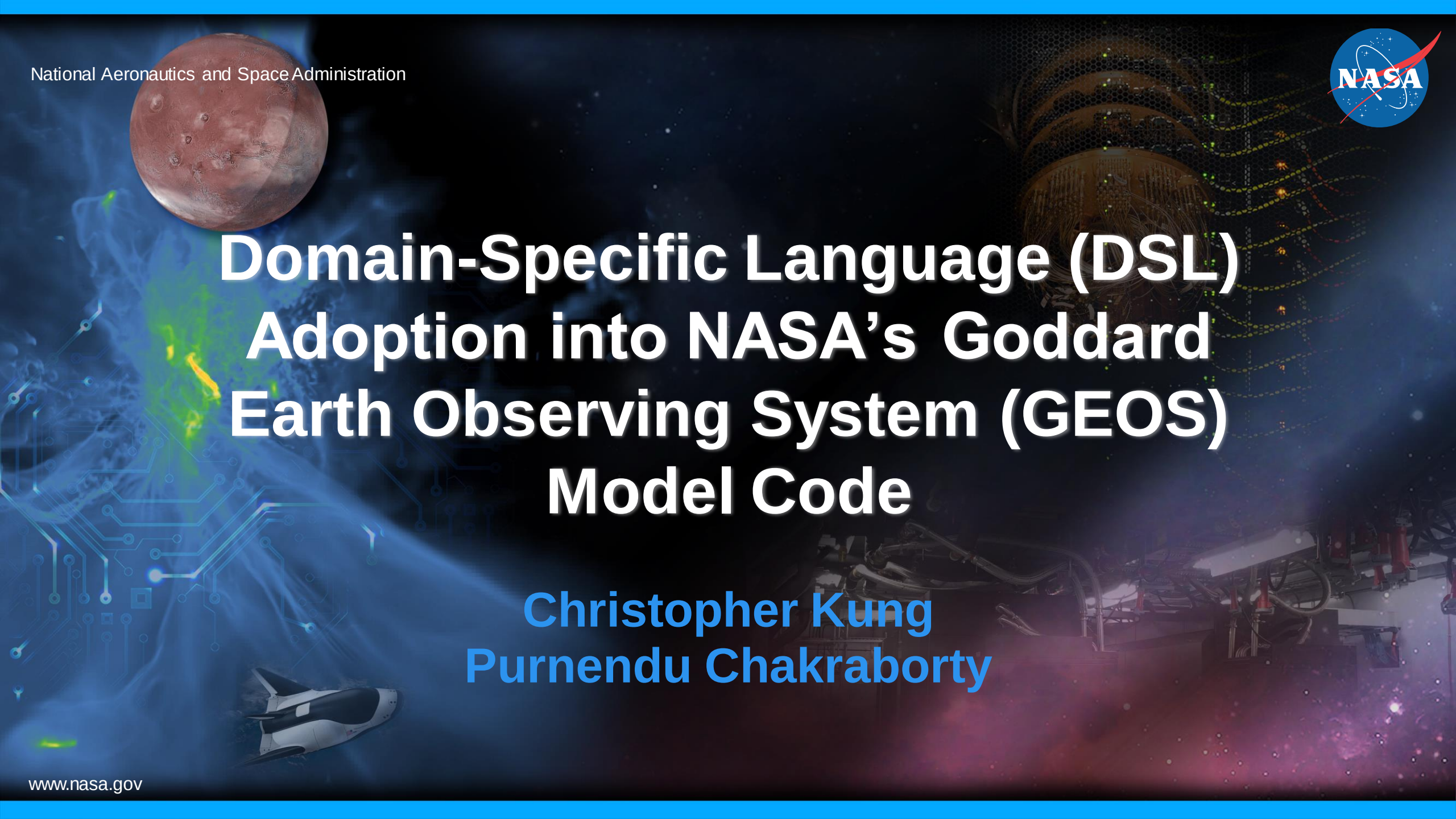
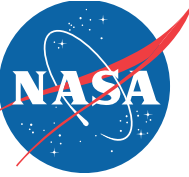


The background is a complex collage of space-related imagery. It includes a large, reddish-brown planet (Mars) in the upper left, a blue and green nebula or galaxy structure on the left, a white space shuttle in the lower left, and a detailed view of a satellite or spacecraft component on the right. The overall theme is space exploration and technology.

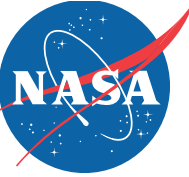
Domain-Specific Language (DSL) Adoption into NASA's Goddard Earth Observing System (GEOS) Model Code

**Christopher Kung
Purnendu Chakraborty**



Outline

- **Motivation**
- **DSL Integration Strategy**
- **Validation / Benchmarking via “gtFV3 + Held-Suarez”**
- **Ported physics packages to GPU via OpenACC**
- **Lessons Learned**
- **Next Steps**



Outline

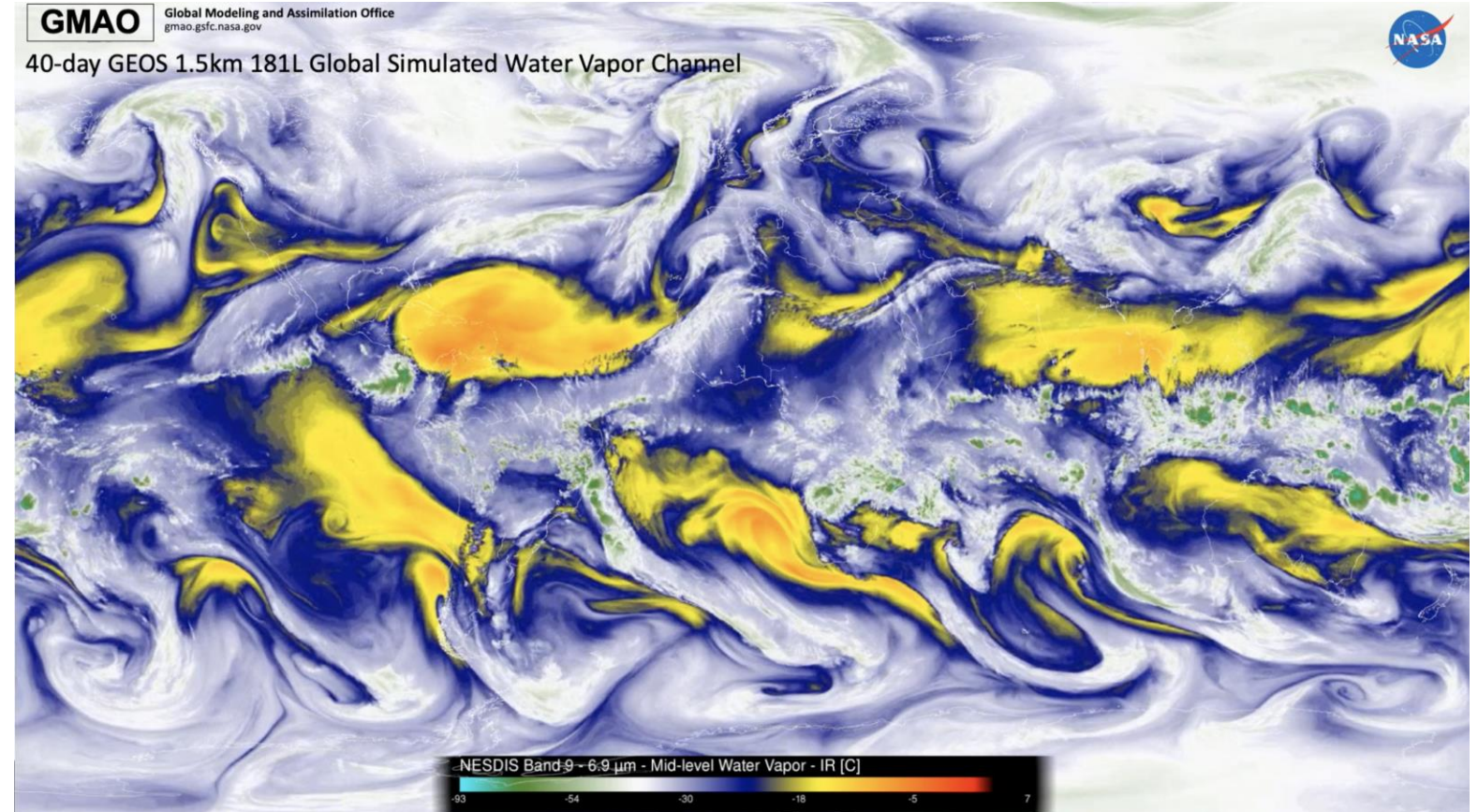
- **Motivation**
- DSL Integration Strategy
- Validation / Benchmarking via “gtFV3 + Held-Suarez”
- Ported physics packages to GPU via OpenACC
- Lessons Learned
- Next Steps

Next-Generation Goddard Earth Observing System (GEOS) code

Goal : Develop the next generation GEOS model that enables an increase in resolution and scalability to meet future NASA Global Modeling and Assimilation Office (GMAO) requirements.

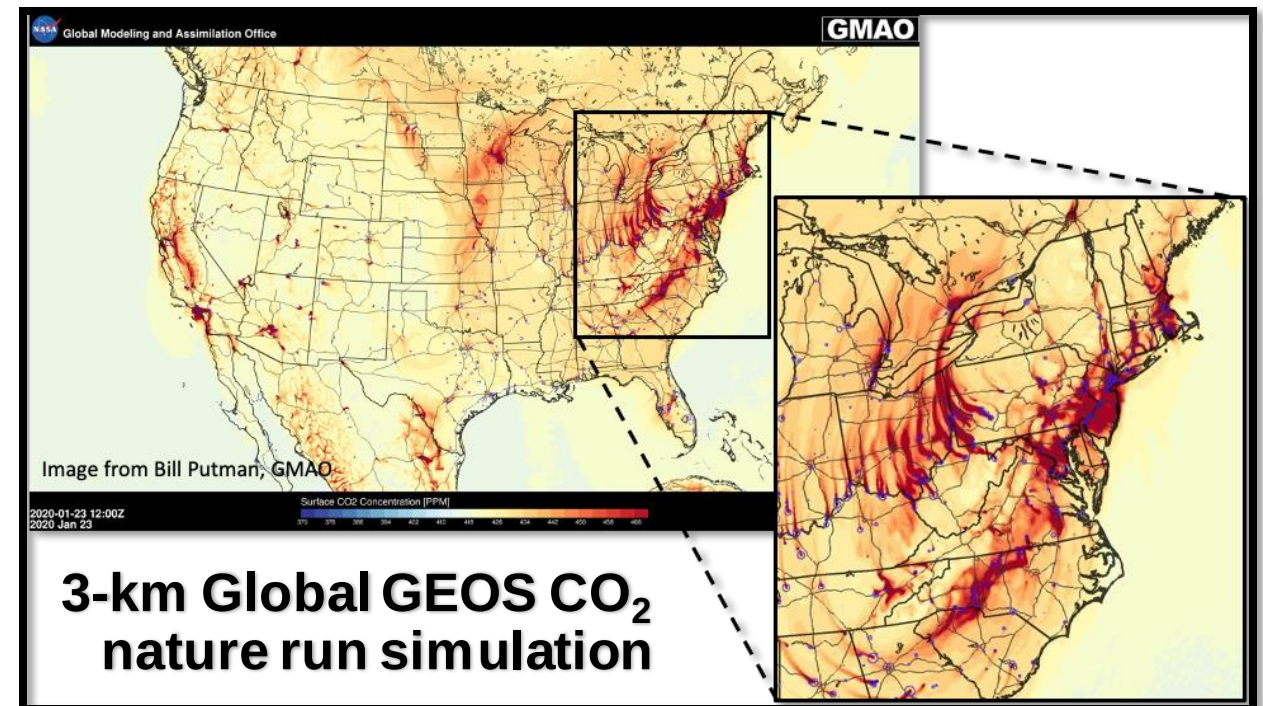
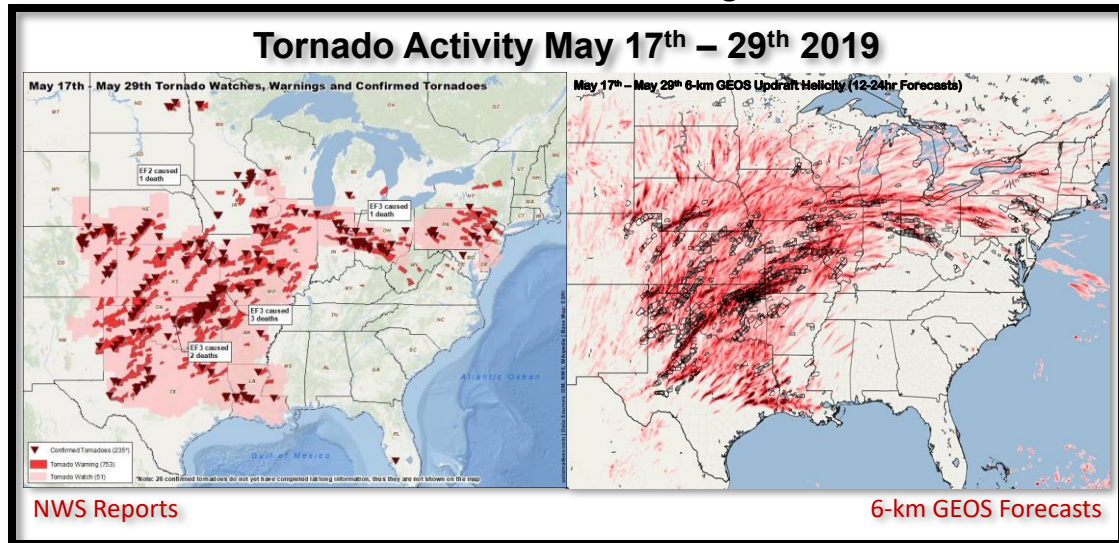
Team

- Developers : Purnendu Chakraborty, Chris Kung
- Support : GMAO Software Integration Team, NASA Center for Climate Simulation (NCCS)
- Management : Bill Putnam, Dan Duffy, Craig Pelissier
- External Collaborators : Allen Institute for AI (AI2), NVIDIA

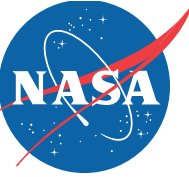


GEOS Future Requirements

- **Production support for Coupled Data Assimilation**
 - Numerical Weather Prediction: 6-km Atmosphere Model (ATM) coupled to 6-km Ocean Model (OCN)
 - Sub-Seasonal to Seasonal Predictions: 25-km ATM coupled to 25-km OCN (Ensemble members creating 30- to 90-day predictions)
 - Reanalysis: 12-km ATM coupled to 12-km OCN
- **Ultra-high resolution global Nature Runs for Climate Observing System Simulation Experiments**
 - Coupled 3-km ATM and 3-km OCN
 - Carbon and chemistry: 3- to 1-km ATM
 - Global convection resolving weather: < 1 km ATM



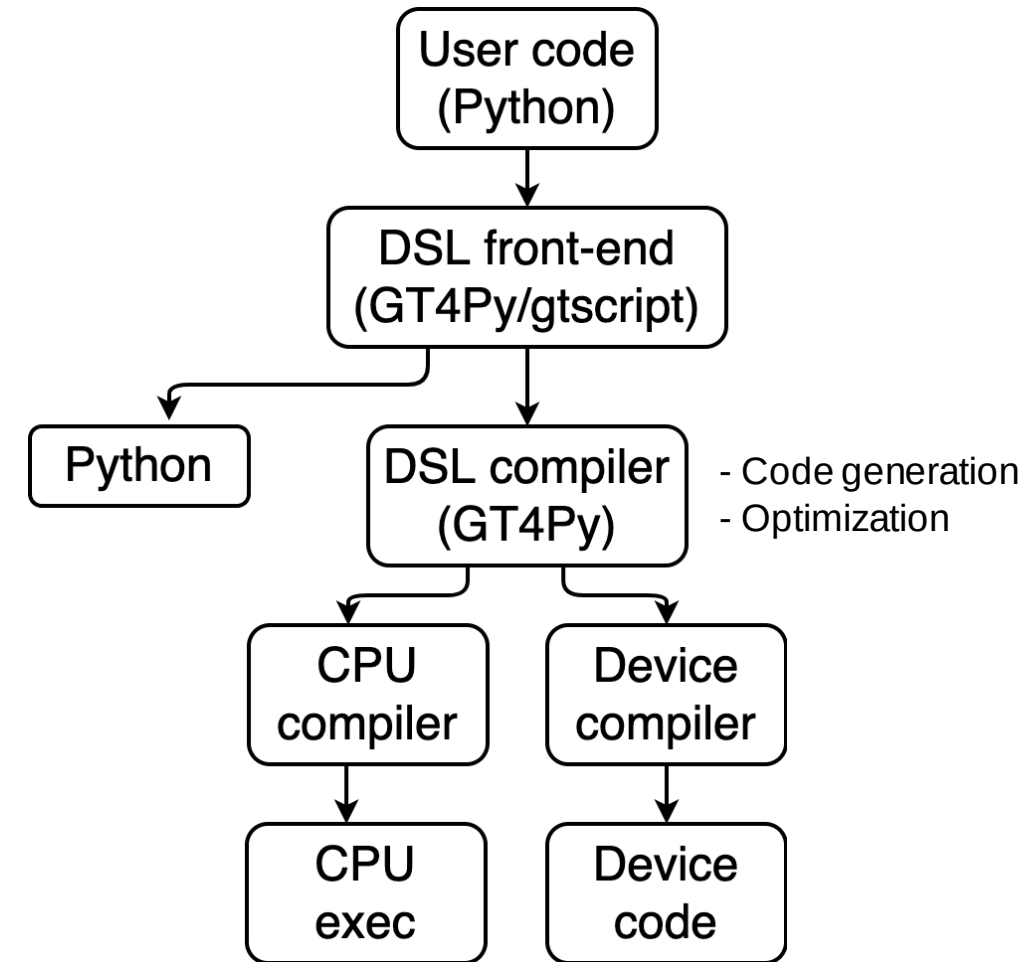
Approach / Strategy



- **Leverage heterogeneous architectures via Domain Specific Language (DSL) and OpenACC**
 - Recognize that accelerator-based heterogeneous systems provide a promising platform to meet future GEOS requirements.
- **DSL : Programming language specialized to a particular domain**
 - DSL adoption creates an opportunity to create portable and scalable code across multiple platforms, including traditional CPU systems and accelerator-based systems, by abstracting away details of the computing architecture
 - For GEOS, we utilize the GridTools for Python (GT4Py) DSL
- **Incorporate a GT4Py port of Geophysical Fluid Dynamics Laboratory's (GFDL) dynamical core (FV3) created by AI2 into GEOS**
 - GT4Py-ported FV3 (gtFV3) is a non-hydrostatic code used for global storm-resolving modeling
 - Leverage DSL without extensive investment in code rewriting / refactoring

GT4Py

- Open-source project under active development by ETH Zurich/Swiss National Supercomputing Centre (CSCS) and AI2
 - <https://github.com/Gridtools/gt4py>
- On-going collaboration with MeteoSwiss, ECMWF, and others
- Part of the GridTools ecosystem of tools and libraries for weather and climate
- GT4Py is a domain-specific library that exposes a domain-specific language (GTScript) to express stencil computations
- Utilize multiple backends (OpenMP on CPUs, CUDA on GPUs) with same codebase
- SC22 Session on Thursday at 1:30 PM : **Productive Performance Engineering for Weather and Climate Modeling with Python**



GT4Py : Code Sample

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

```

GT4py

```

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

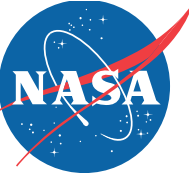
@gtscript.function
def dely(q, weight):
    return weight * (q[0, -1, 0] - q)

@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())

```

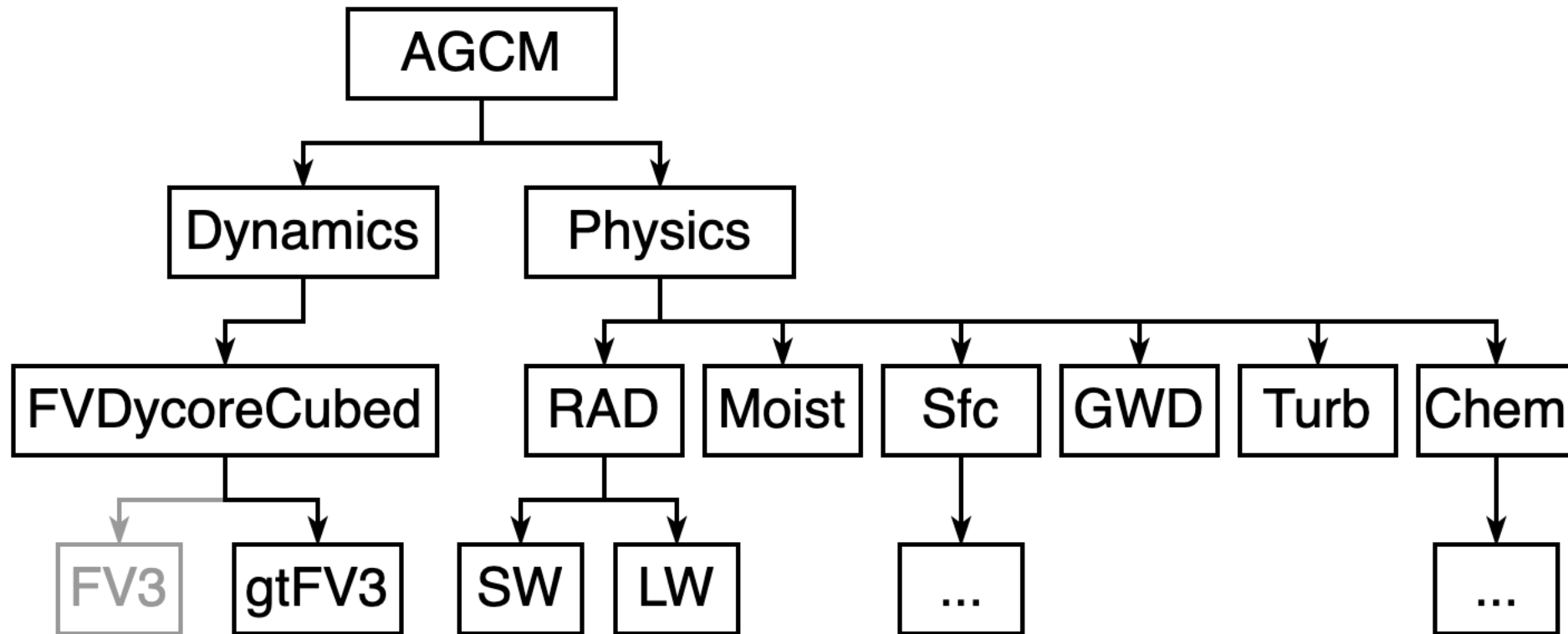
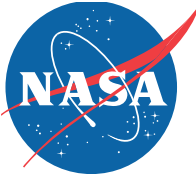
- Horizontal **loops** removed; **schedule** removed
- Index offsets instead of absolute indices
- Overhead-free, reusable **functions** – inlining
- Can utilize other Python packages
- Less code



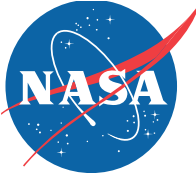
Outline

- Motivation
- **DSL Integration Strategy**
- Validation / Benchmarking via “gtFV3 + Held-Suarez”
- Ported physics packages to GPU via OpenACC
- Lessons Learned
- Next Steps

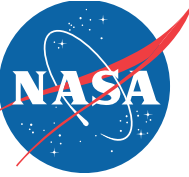
GEOS Atmospheric GCM



Integration Options : gtFV3 into GEOS



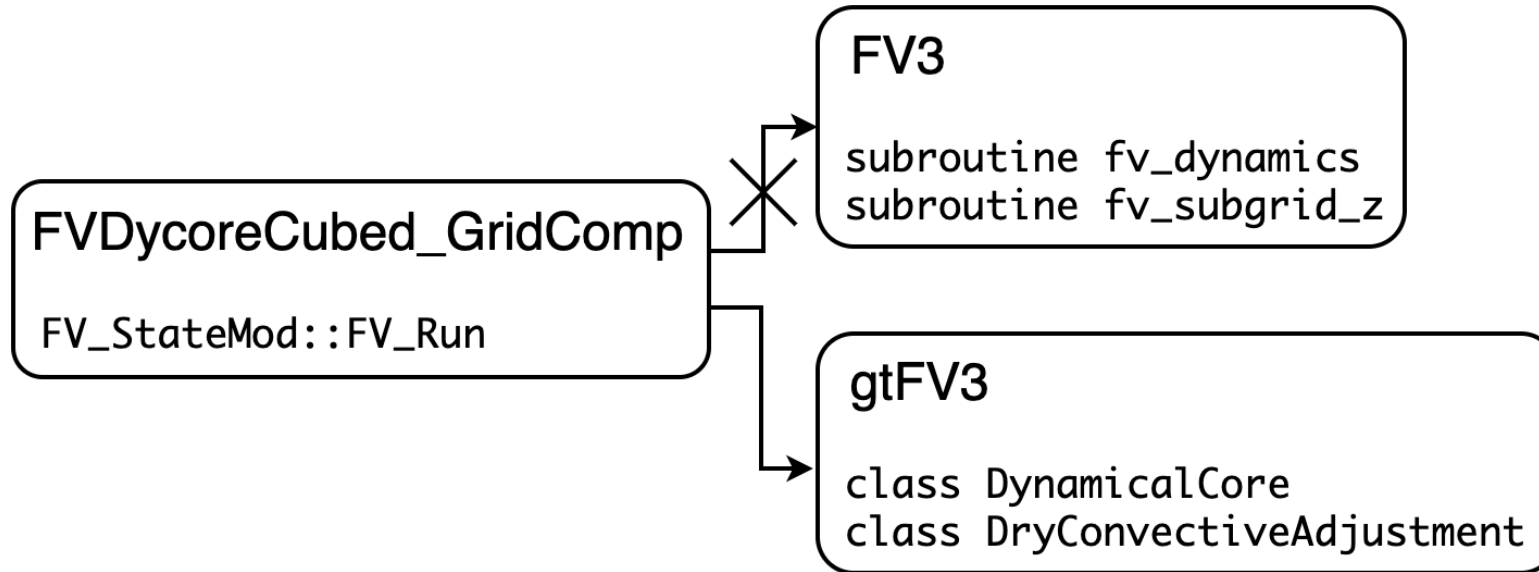
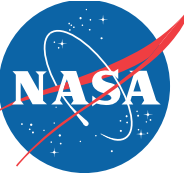
- **gtFV3: A Python package that provides the following**
 - Replacements for Fortran kernels in the form of GT4Py stencils
 - Replacements for top-level FV3 routines
 - subroutine fv_dynamics → class DynamicalCore
 - subroutine fv_subgrid_z → class FVSubgridZ
 - subroutine fv_tp_2d → class FiniteVolumeTransport
- **Considered three possible integration paths**
 1. Replace Fortran kernels with Python stencils
 - Drawback : Creating Fortran-Python interfaces for each stencil is time consuming
 2. Replace Fortran kernels with generated (optimized) C++ code
 - Drawback : Fragile, generated API can change with GT4Py development/change
 3. Replace top-level FV3 routines with gtFV3 counterparts
 - Avoids the drawbacks of 1 and 2 and leverages GT4Py more robustly
 - Integration path of choice



Integration Challenges

- **Fortran / Python Interfacing**
 - C Foreign Function Interface for Python (CFFI) was chosen for its simplicity to call Python functions from Fortran
- **Integrate gtFV3 into GEOS build system**
- **gtFV3 includes a subset of FV3 features – requires careful curation of runtime environment**
- **Code differences between the GEOS and gtFV3**
 - gtFV3 is based on GFDL FV3
- **Lack of debugging support in our mixed language environment**
- **Data layout differences between GEOS and gtFV3**
 - Requires data transpose
 - Requires data translation
- **Precision differences between GEOS and gtFV3**

gtFV3 Integration into GEOS



In FV_StateMod::FV_Run, replace

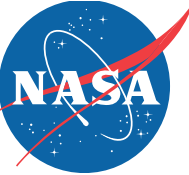
`call fv_dynamics(npx, ..., u, ...)`

with

`call geos_gtfv3_interface_f(comm, npx, ..., u, ...)`

Once integrated, any GEOS configuration using FVDycoreCubed component should be able to run either FV3 or gtFV3

GEOS - gtFV3 Interface via CFFI



Fortran

```
interface
  subroutine geos_gtfv3_interface_f(comm, npx, ..., u, ...) bind(c, name='geos_gtfv3_interface_c')
    ...
  end subroutine geos_gtfv3_interface
end interface
```

C

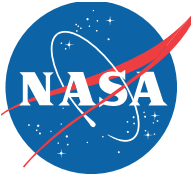
```
void geos_gtfv3_interface_c(MPI_Fint comm_f, int npx, ..., float* u, ...) {
  // comm_f -> comm_c
  geos_gtfv3_interface_py(comm_c, npx, ..., u, ...);
}
```

Python

```
def geos_gtfv3_interface_py(comm_c, npx, ..., u_ptr, ...):
  # comm_c -> comm_py
  geos_gtfv3(comm_py, npx, ..., u_ptr, ...)
```

Top level Python function that interacts directly with gtFV3

GEOS - gtFV3 Interface : First Attempt



← Called by Fortran at every time step

```
def geos_gtfv3(comm, npx, ..., u_ptr, ..., dx_ptr, ...):

    # Set backend: gtx86/gtcuda
    backend = 'gtx86'
    fv3core.set_backend(backend)

    # Wrap Fortran array as NumPy array (no copy here)
    u = fort_to_numpy(ffl, u_ptr, (ied-isd+1, jed+1-jsd+1, npz))
    ...

    # Message input and grid data (local transpose + float32->float64)
    input_data = {'u': u.transpose().astype(np.float64), ...} # u, v, w, mfxd, ...
    grid_data = {'dx': dx.transpose().astype(np.float64), 'dy': ...}

    # Translate input and grid data for uniform indexing
    grid = fv3core.testing.TranslateGrid(grid_data, comm.Get_rank()).python_grid()
    driver_object = fv3core.testing.TranslateFVDynamics([grid])
    state = driver_object.state_from_inputs(input_data)

    # Instantiate DynamicalCore class
    dycore = fv3core.DynamicalCore(...)

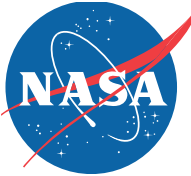
    # Run a single step
    dycore.step_dynamics(state, ...)

    # Un-translate 'state' and copy data back
```

GEOS → gtFV3
"data massaging"
- Transpose
- Translate
- Single → double
- Instantiate dycore

gtFV3 → GEOS
"data massaging"

GEOS - gtFV3 Interface : Second Attempt



```
def geos_gtfv3(comm, npx, ..., u_ptr, ..., dx_ptr, ...):  
  
    # Set backend: gtx86/gtcuda  
    backend = 'gtx86'  
    fv3core.set_backend(backend)  
  
    # Wrap Fortran array as NumPy array (no copy here)  
    u = fort_to_numpy(ffr, u_ptr, (ied-isd+1, jed+1-jsd+1, npz))  
    ...  
  
    # Massage input and grid data (local transpose + float32->float64)  
    input_data = {'u': u.transpose().astype(np.float64), ...} # u, v, w, mfxd, ...
```

Initialization
(executed once)

```
    grid_data = {'dx': dx.transpose().astype(np.float64), 'dy': ...}  
    # Translate input and grid data for uniform indexing  
    grid = fv3core.testing.TranslateGrid(grid_data, comm.Get_rank()).python_grid()  
    driver_object = fv3core.testing.TranslateFVDynamics([grid])  
  
    # Instantiate DynamicalCore class  
    dycore = fv3core.DynamicalCore(...)
```

```
    state = driver_object.state_from_inputs(input_data)
```

```
    # Run a single step  
    dycore.step_dynamics(state, ...)
```

```
    # Un-translate 'state' and copy data back
```

Move initialization
to global scope

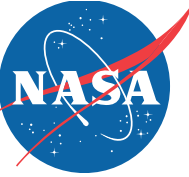
Profile : AGCM with gtFV3

- C180 (50 km), L91
- 24 (2×2×6) MPI ranks
- Single timestep (seconds)
- Setup
 - GFDL microphysics
 - Non-hydrostatic
 - Non-Adiabatic
- FV3
 - Single Precision
 - 44 Tracers
 - AMD EPYC 7402
- gtFV3
 - Double Precision
 - 9 Tracers
 - 4 NVIDIA A100 GPUs (6 ranks / GPU)

Rank	Time (sec)		
	FV3	gtFV3 [CPU]	gtFV3 [GPU]
0	7.287	21.152	4.189 (3.338)
1	7.294	21.134	4.204 (3.270)
2	7.306	21.130	4.199 (3.253)
3	7.298	21.162	4.215 (3.323)
4	7.294	21.117	4.200 (3.331)
...	...	...	...

~ 1 second per timestep

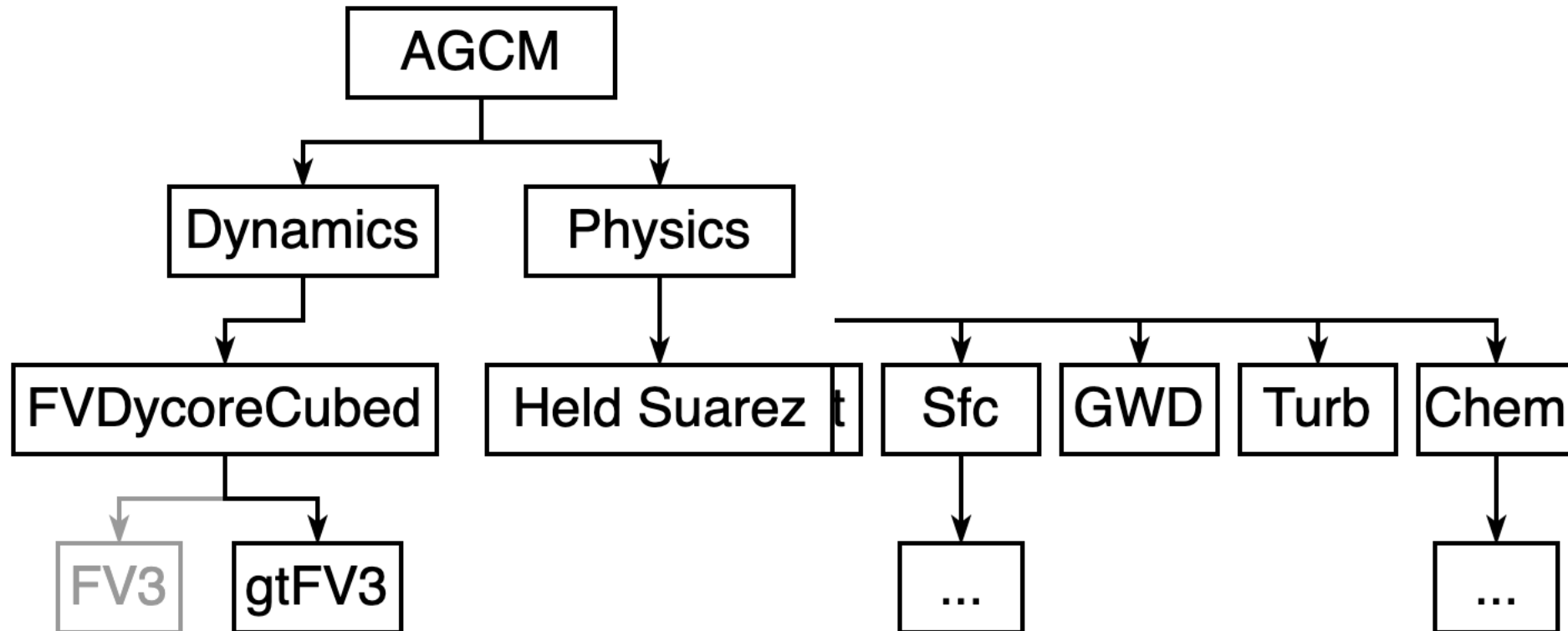
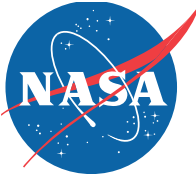
- host/device data transfers
- data array transposes
- data translation
- single ↔ double precision

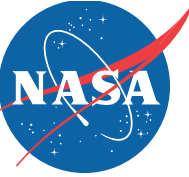


Outline

- Motivation
- DSL Integration Strategy
- **Validation / Benchmarking via “gtFV3 + Held-Suarez”**
- Ported physics packages to GPU via OpenACC
- Lessons Learned
- Next Steps

gtFV3 + Held-Suarez Physics



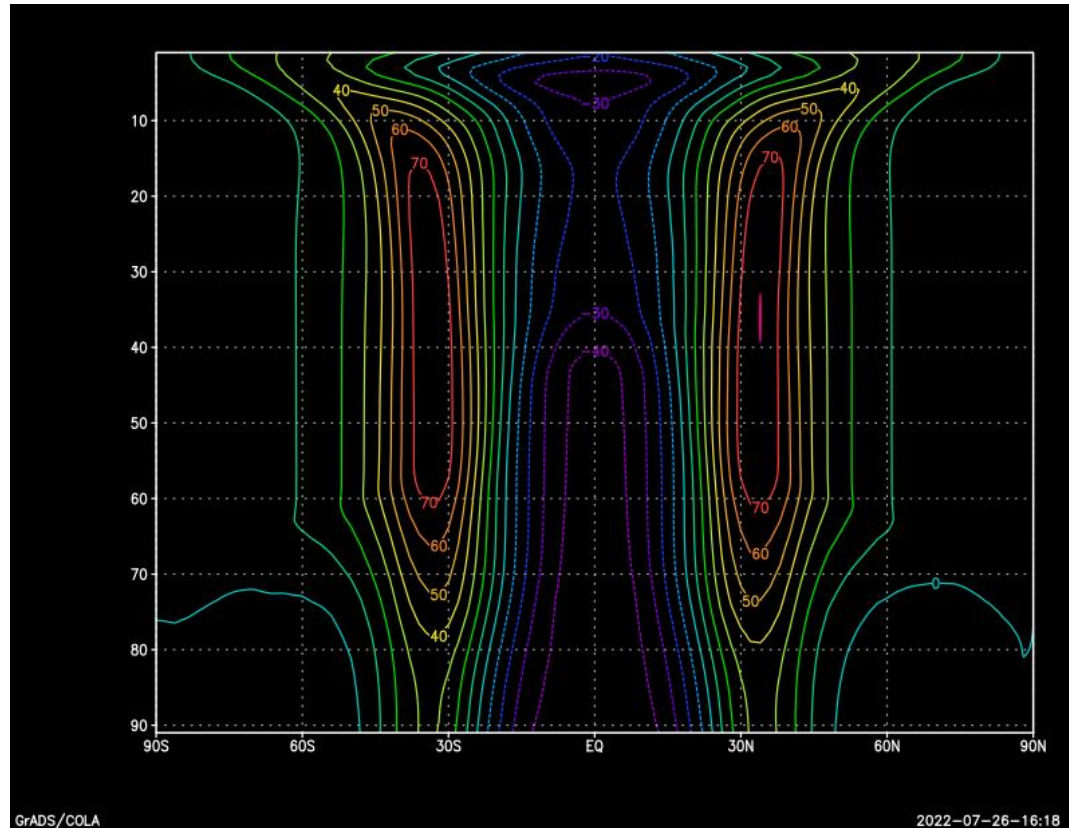


Held-Suarez Challenges

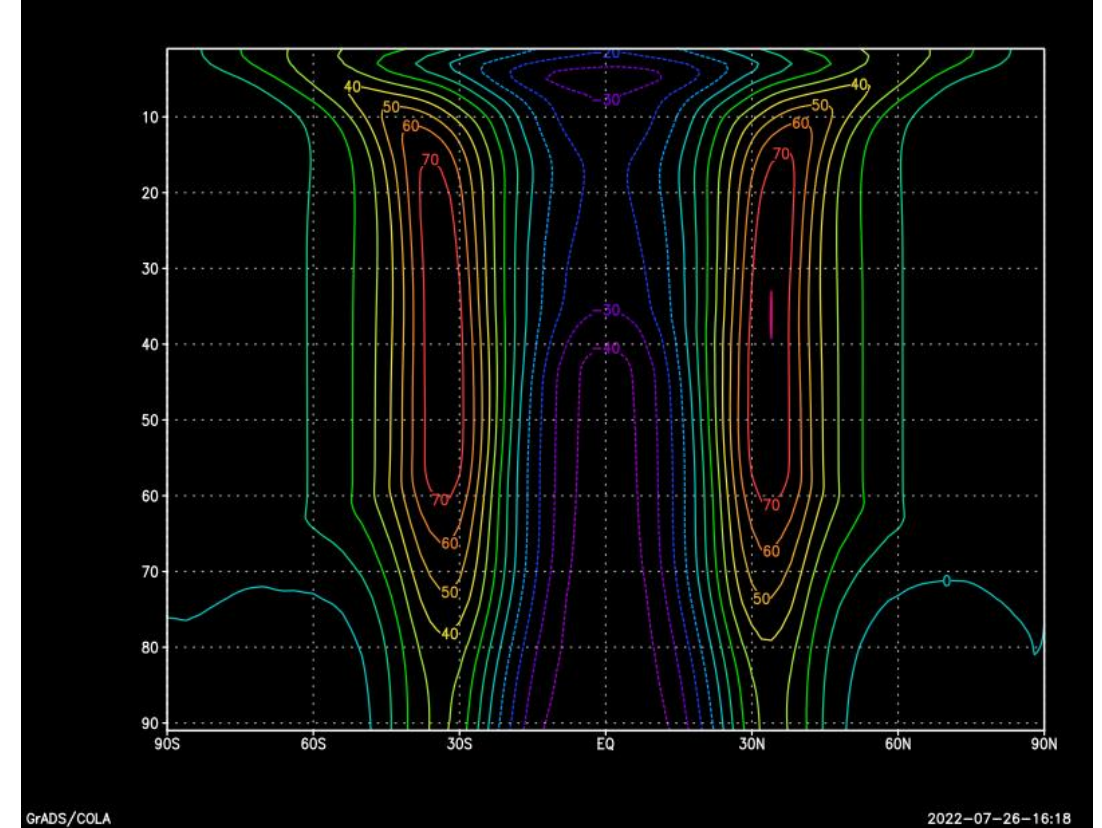
- **Integration (replace original FVDycoreCubed component with new)**
 - Resolving build issues, setting up the correct environment, etc.
 - Setting up a GEOS run environment to mimic a GFS run
- **Validation – code differences needed to be tracked down**
 - GEOS treats the sponge layer differently (n_sponge/n_zfilter)
 - GEOS remaps mass fluxes and Courant numbers
 - Hard-coded variables in gtFV3 to avoid reductions (cmax)
 - Some bugs (e.g. diss_est is not accumulated correctly)

FV3 – HS vs gtFV3 – HS

FV3 (CPU), HS (CPU)

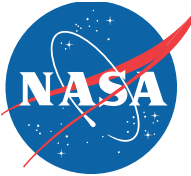


gtFV3 (GPU), HS (CPU)



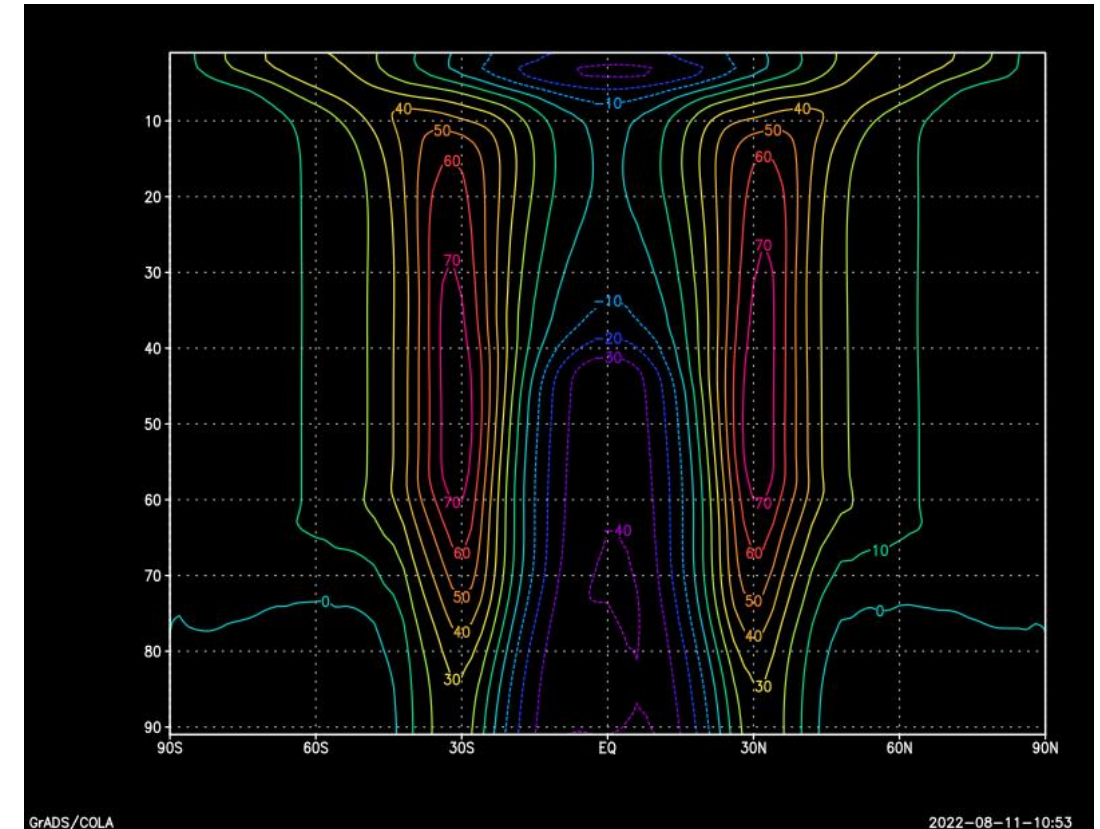
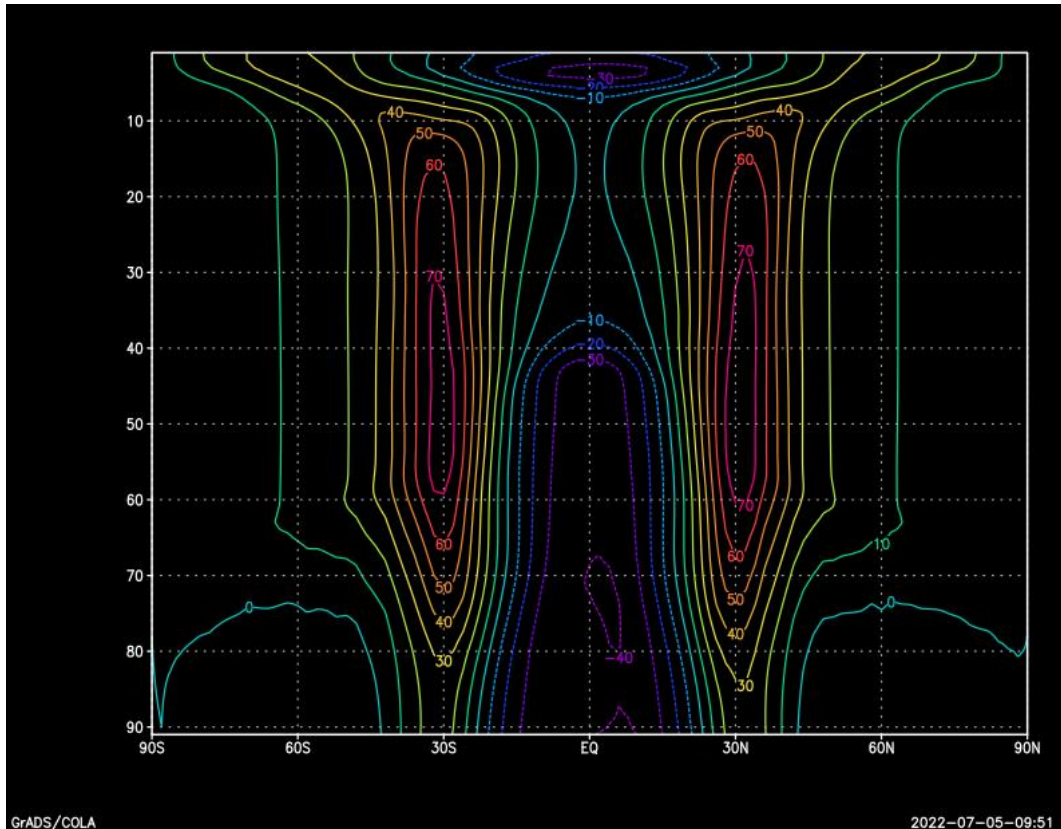
Zonal mean of U-winds averaged over 90 days

FV3 – HS vs gtFV3 – HS (OpenACC)

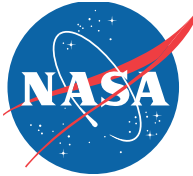


FV3 (CPU), HS (CPU)

gtFV3 (GPU), HS (GPU)



Zonal mean of U-winds averaged over 30 days



Held-Suarez Benchmarking: CPUs vs GPUs

FV3 (AMD EPYC 7402)

- C180, L91
- Timestep: 450s
- 6 sockets (3 nodes)
- 144 ranks
- **Single** precision
- Time/timestep: **~1.4s**

FV3 (Intel Cascade Lake)

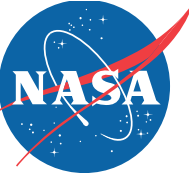
- C180, L91
- Timestep: 450s
- 6 sockets (3 nodes)
- 120 ranks
- **Single** precision
- Time/timestep: **~1.6s**

gtFV3 (NVIDIA A100)

- C180, L91
- Timestep: 450s
- 6 GPUs (1.5 nodes)
- 6 ranks (1 rank/GPU)
- **Double** precision
- Time/timestep: **~3.9s (2.9s)**

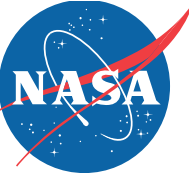
~ 1 second per timestep

- host/device data transfers
- data array transposes
- data translation
- single ↔ double precision



Outline

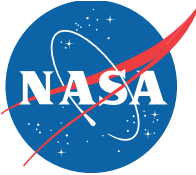
- Motivation
- DSL Integration Strategy
- Validation / Benchmarking via “gtFV3 + Held-Suarez”
- **Ported physics packages to GPU via OpenACC**
- Lessons Learned
- Next Steps



OpenACC for Physics

- Leverage OpenACC to port physics packages onto GPUs
- **Why OpenACC?**
 - Development remains in Fortran
 - Flexibility : GT4Py "coding style" is based on stencil computations, making it less flexible if a computational algorithm doesn't map into a stencil.
- **What about "proprietary-ness"?**
 - OpenACC may not be the final solution since it's (mostly) NVIDIA-centric and OpenACC compilers (generally) are not available for non-NVIDIA GPUs.
 - Note: Gfortran supports OpenACC-offloading on AMD GPUs, but this has not been tested.
 - The OpenACC implementation can guide development of an OpenMP-offloading implementation due to the relatively similar feature set of both programming models.

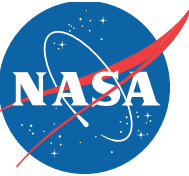
OpenACC-Ported Physics



- **General Porting Methodology**

- Create a CPU-based standalone physics code that verifies with input and output data generated from GEOS runs that call the physics schemes
- Apply OpenACC to parallelizable code sections within the CPU-based standalone code
- Verify that data from OpenACC standalone code matches GEOS generated data from CPUs

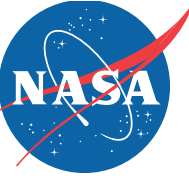
Physics Scheme Ported	Compilers verified with OpenACC	GPU-computed results verified against GEOS CPU-generated data
Held-Suarez	nvfortran gfortran	✓
GWD	nvfortran gfortran	✓
Turbulence (run_edmf, run_shoc, beljaars, vtrilu, vtrisolvesurf, update_moments, postlock, prelim)	nvfortran	✓
Combined Turbulence	nvfortran	✓



Turbulence Performance

- **Standalone compute time listed**
 - Hardware : AMD EPYC 7402 CPU and NVIDIA A100 GPU
 - Note : CPU runtimes are single core performance
 - No data transfer time factored in

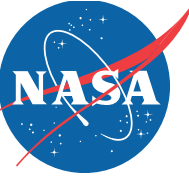
Physics Scheme	c90 Runtime on CPU (Intel Compiler) [sec]	c90 Runtime on GPU (NVIDIA Compiler) [sec]	Speedup on GPU		c180 Runtime on CPU (Intel Compiler) [sec]	c180 Runtime on GPU (NVIDIA Compiler) [sec]	Speedup on GPU
BELJAARS	4.27E-03	8.20E-05	52.09		1.70E-02	1.34E-04	126.87
POSTLOCK	1.28E-01	2.04E-03	62.65		4.70E-01	3.30E-03	142.42
PRELIMS	2.48E-02	2.45E-04	100.99		8.35E-02	4.68E-04	178.48
RUN_SHOC	7.61E-02	1.72E-02	4.42		2.98E-01	1.82E-02	16.37
UPDATE_MOMENTS	9.07E-03	1.81E-03	5.00		3.58E-02	3.03E-03	11.81
VTRILU	3.20E-04	1.40E-04	2.29		1.93E-03	1.97E-04	9.78
VTRISOLVESURF	6.50E-04	1.05E-04	6.18		1.79E-03	1.22E-04	14.74
Total	2.43E-01	2.17E-02	11.21		9.08E-01	2.55E-02	35.65
GWD					2.13712	5.21E-02	40.98



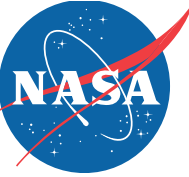
Outline

- Motivation
- DSL Integration Strategy
- Validation / Benchmarking via “gtFV3 + Held-Suarez”
- Ported physics packages to GPU via OpenACC
- **Lessons Learned**
- Next Steps

DSL Lessons Learned



- **Lack of debugging tools that can simultaneously examine both Fortran and Python codebases**
 - Totalview mentioned that the feature to examine Fortran and Python codes simultaneously will be coming soon
- **DSL creates additional “developmental distance” between the application developer and compiler during the development process, which potentially introduces additional points of error/failure**
 - Application Developer ↔ DSL ↔ Performance Portability ↔ Compiler Engineer
- **Need to monitor the future development/support of GT4Py**
 - GT4Py is still under active development, and APIs may change
- **gtFV3 + Held-Suarez repo : <https://github.com/GEOS-ESM/GEOSgcm/tree/pchakrab/feature/integrate-gtfv3-held-suarez-oacc>**



Outline

- Motivation
- DSL Integration Strategy
- Validation / Benchmarking via “gtFV3 + Held-Suarez”
- Ported physics packages to GPU via OpenACC
- Lessons Learned
- **Next Steps**

Next Steps

- **Aquaplanet**

- GT4Py Dynamics (gtFV3)
- Physics schemes (Gravity Wave Drag [100% complete], Moist [0% Complete], Seale [Porting not needed], Turbulence [100% Complete], Radiation [100% Complete]) ported to OpenACC

- **Data Movement Optimization**

- Refactor code to generally reduce need to transfer data between host and device
- Explore how GEOS' Modeling Analysis and Prediction Layer (MAPL) can account for GPUs
 - Need support to share device memory between components
- Support from GT4Py to have device pointers exposed

- **Additional GT4Py Optimizations**

- Work with AI2 and CSCS

