FV3 on GPUs

Two approaches to accelerate global cloud-resolving modeling

Lucas Harris
with Rusty Benson and Oli Fuhrer

and many others from GFDL, GMAO, Vulcan/AI^2, NVIDIA, etc.

Motivation: FV3-based GCRMs

NASA Goddard and GFDL pioneered US GCRMs in the mid 2000s

- NASA GEOS and GFDL X-SHiELD lead US contributions to the international DYAMOND intercomparison, phases 1 and 2
  - 40-day simulations at 3-km (C3072) resolutions
  - Great TCs, both #s and structure (Judt et al., JMSA, 2021)
  - X-SHiELD: Year-round simulation in progress
  - GEOS: Experimental 1.5-km run also submitted

These are useful prototypes for future weather prediction and climate modeling systems, and powerful demonstrations of model capabilities.

GFDL GCRM Performance

	Model	Grid/levels	Machine	# Cores	Performance	Source
2013	FV3 (dycore only)	C2560, 32 levels	Argonne BlueGene/Q	1048K (threads)	87 SDPD 24 min/d	https://www.alcf.anl.gov/files/ANL_ALCF_TM_14_1.pdf
2015	FV3 (dycore only)	C3072, 127 levels	NERSC Edison	110K	144 SDPD 10 min/d	NGGPS AVEC Phase I report
2019	X-SHiELD	C3072, 79 levels	NOAA Gaea	55K	65 SDPD 22 min/d	My testing
2020	X-SHiELD	C3072, 79 levels	MSU Orion	36K	55 SDPD 26 min/d	Rusty Benson

SDPD = Simulated days per day

FV3 + FMS, with MPI and OpenMP

Tastes Great! Less Filling!

Convection "resolving" models

I'm old

Deep convective plumes not resolved
until Δx ~ O(250 m)

Bryan et al. (2003, JAS), Bryan and Morrison (2012, MWR), Jeevanjee (2017, JAMES), Shi et al. (2019, JAMES), lots more

At Δx = 3–4 km:

- ➢ Continental convection kinda sorta resolved?
- ➢ Tropical convection barely represented
- ➢ Shallow convection, definitely not
- ➢ Turbulent eddies haha
- ➢ Orography always benefits

We can do better. But higher resolutions need better physics.

Moving forward: < 3-km efforts

- Lots of work outside of the US:
  - UKMO: 2.2-km and 1.1-km CONUS, 100 m London
  - ECMWF: 1.4-km global (hydrostatic) nature run
  - MeteoSwiss COSMO: 1.1-km central Europe
  - Japan NICAM: 800-m GCRM
- Some US efforts:
  - 1.5-km NAM Fire Weather and HWRF inner nest
  - NSSL 1-km WoF; other NSSL and CAPS 1-km experiments
  - Various NCAR experiments (including LES prediction)

Open question: how does explicit deep convection affect synoptic and planetary circulations??

Previously, on GPUs

- FV3's modular design, applying atomic stencils to 3D arrays, fits well to GPUs, but some reorganization of code needed
  - NASA GEOS ported to GPUs using CUDA Fortran
    Hydrostatic GEOS was 2–5x faster than CPUs *socket-to-socket*
  - Institute for Atmospheric Physics (Beijing) has ported SHiELD to CUDA-C
    Nonhydrostatic model is 6x faster than on CPUs *socket-to-socket*

https://slideplayer.com/slide/7775285/

- Success, but difficult to maintain:
  - Continual changes to GPU hardware and compilers
  - No GPU programming standards: CUDA only really works for NVIDIA GPUs
  - No large-scale GPU systems to work with in NOAA or NASA
  - <u>**Have to keep "feeding the dog"**</u>: Need a really big compute problem
    - Few "standard" weather or climate problems will benefit enough to justify the port

WARNING: 1 CPU to 1 GPU comparisons may cause dizziness, embarrassment

Method I
Porting with ACC

NVIDIA Hackathons

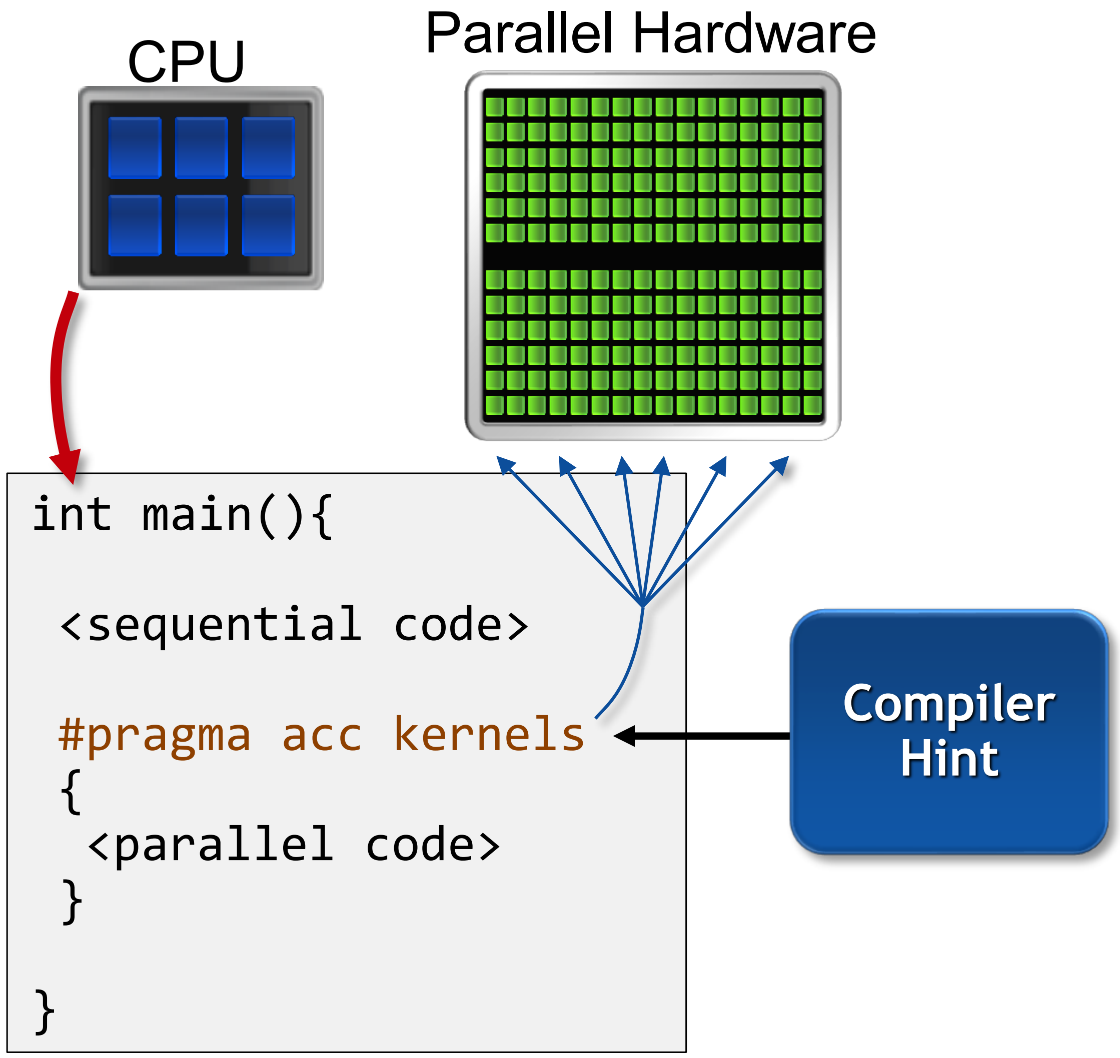
- NVIDIA sponsored a Hackathon at Princeton University in 2019. Several GFDL and Princeton employees created kernels of routines in FV3 and began ACC optimization of 1D advection operator and vertical semi-implicit solver
- In the 2020 NOAA Hackathon a shallow-water routine and the vertical remapping were ported by a GFDL-Princeton-Vulcan Team.
- Since advection and vertical remapping are frequently used in FV3 for many purposes these are key areas to accelerate.
- Porting was significantly aided by mentors from NVIDIA and Lawrence Berkeley National Laboratory

ACC Porting

OpenACC works like OpenMP does for multi-threading. Directives are added to existing code to tell the compiler how to parallelize a block and which data to move between GPU and CPU.

Pros: easy to learn, more portable than CUDA

Cons: Not a standard (yet; may merge with OpenMP), may choke on complex loops, often still requires some code-rearrangement.



GPU Results

- Need to use a sufficiently large problem (eg. 96x96x91 as in GCRM) to take advantage of GPU parallelism.
- Advection operator: 50x speedup 1 CPU ⇒ 1 GPU
  - **WARNING**: No optimization for this CPU, no MPI
- fv_mapz: 47x speedup, despite challenging double-loop
  - Multi-tracer remapping adds lots of parallelism
- c_sw shallow-water routine: 3.8x speedup
  - Complex routine that needs more careful thought
- Further speedup can be done by better overlapping computes and copies and taking advantage of asynchronous calculation

ACC Lessons

- Branches for upwinding and monotonicity do not degrade performance as feared.
- In-loop conditions (eg. edge handling) do need copying and do give a performance hit
- Some code reorganization may be necessary to reduce copying and increase parallelism. May need to think carefully about how best to do this.
- Key areas can also be re-written in CUDA if necessary.

Method II:
GT4py Domain-Specific Language

All results courtesy the Vulcan/AI^2 Climate Modeling DSL Team

Domain-Specific Language (DSL)

- A DSL is a language tailored for a specific purpose. The domain scientist specifies the algorithm layout and fundamental operations.
  - Domain-specific knowledge → Domain-specific optmizations
- A special DSL compiler with several “backends” creates the codes optimizing for a specific computing architecture
  - Chooses memory layout, parallelism, compute order, etc.
- Goals:
  - Improved productivity by domain scientist without needing to learn the ins-and-outs of code optimization
  - Performance portability between systems without code re-writes: only a new backend is needed

GT4py

GridTools

- Joint open-source effort with CSCS and MeteoSwiss
- Domain scientists write *stencils* of operations in Python, which is then compiled by different backends
    - x86 CPU, NVIDIA GPU
    - Parallelism, looping, data structures etc. specified by backend, not in Python
- Using Python leverages vast array of tools and libraries, permitting integration with visualization and Jupyter notebooks
    - Vulcan FV3GFS Python wrapper tutorial at GFDL in January 2021
    - See McGibbon et al. 2021, GMDD
- GT4py being considered by MPI/DWD for ICON and ECMWF for FVM—but unstructured grid solvers are harder to port

Original FV3 Fortran 90

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

real :: fx(is:ie+1, js,je), fy(is:ie, js:je+1)

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)
```

DSL Port of Routine

```
@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())
```

UFS and GT4py

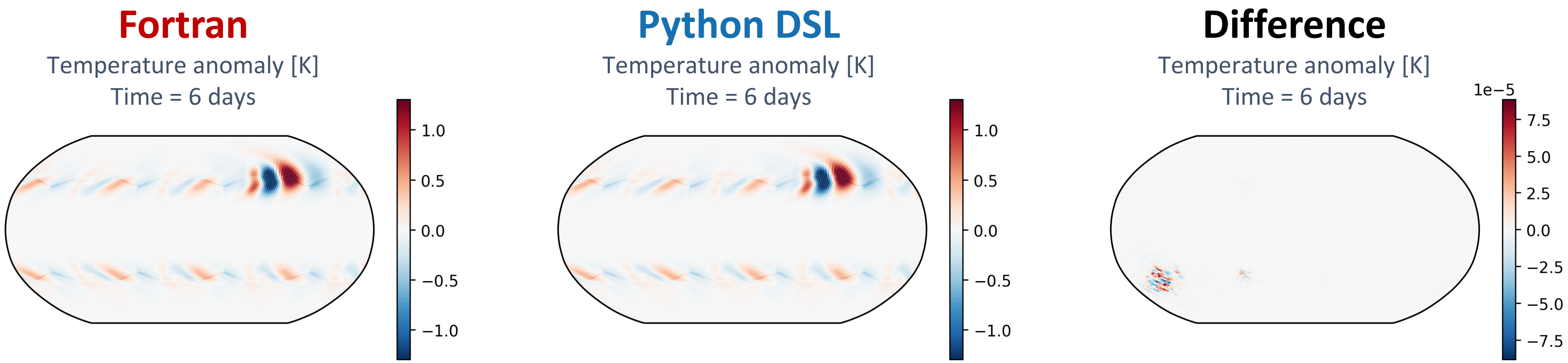
A multi-agency international public-private-academic partnership

- Vulcan has committed a team (8 scientists and engineers) to porting FV3 and the GFS Physics into GT4py
  - DSL Training held at GFDL in November 2020
- NASA Goddard has committed resources to GT4py development of FV3 and GEOS
- GFDL supports and advises development, two Vulcan embeds
- CSCS and MeteoSwiss develop back-end in tandem with UFS implementation

FV3 in GT4py

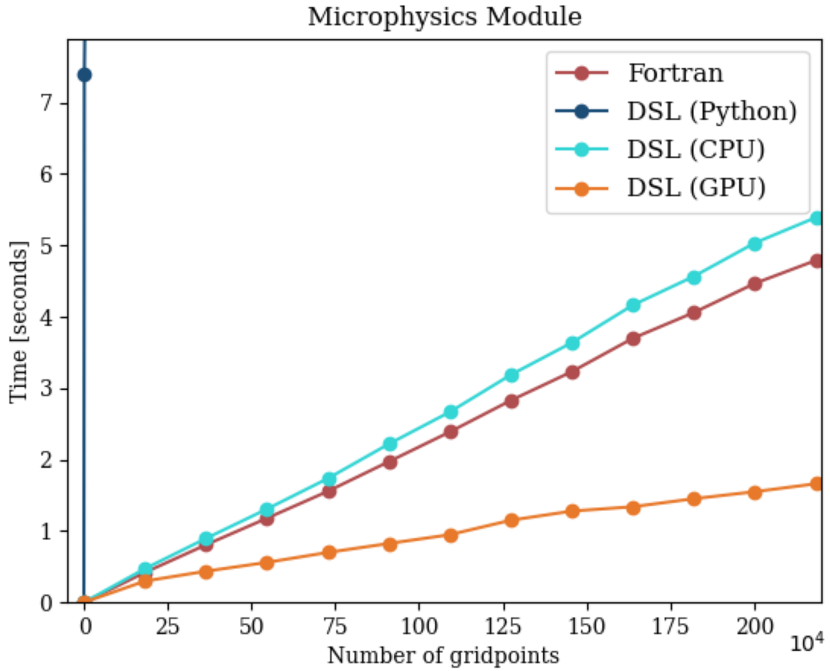
- FV3 has been ported into GT4py and validates answers vs. Fortran
- Next step: optimization
  - Want to eliminate as much slow Python as possible
  - All operations reading/writing prognostic variables must run on GPU

    Data transfer to CPU is slow
- New features have been added to DSL (language and backend) to support cubed-sphere edge handling, caching and fusing stencils, improving compiler code translation, etc.



GFS Physics in GT4py

- ETH Zurich students working to port GFS Physics packages into GT4py
- GFDL Microphysics, TKE-EDMF, sea ice routines ported; RRTM this summer
- Already seeing substantial single-node speedups on Piz Daint
  - 12-core Intel Xeon vs. NVIDIA Tesla V100 (ie. a good comparison)



GFDL Microphysics in Fortran vs. GT4py

Concluding Thoughts

- The CPU MPI/OpenMP methods are no longer the only game in town, but they still serve us well
- GPUs can give us great performance for lower cost and less energy on big problems (GCRMs), but the landscape is constantly shifting
- FV3 has succeeded on GPUs, but how to ensure performance-portability?
- The GridTools/GT4py community gives us a lot of hope for performance portability.
    - Vulcan has made great progress—still work that needs to be done for FV3 and UFS performance portability