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
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

**ACCELERATORS/CO-PROCESSORS**

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

https://github.com/ai2cm/pace
Pace

Fortran

Python

https://github.com/ai2cm/pace
Example Conversion

**Fortran**

```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, 1
   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
end subroutine del2_cubed
```

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

**Pace**

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

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

(Submitted as SC’22 paper!)
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

GFDL
NASA
CSCS
MeteoSwiss
ETH Zürich
SPCL

UNIVERSITY OF WASHINGTON
This slide intentionally left blank
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
Testing - Fortran to Python

1. Write Inputs
2. Fortran Code
3. Write Outputs
4. Read Inputs
5. Python Code
6. Read and Compare Outputs

https://github.com/ai2cm/pace
Development and Testing

Read Inputs

Larger block of Code

Error Growth

Read and compare Outputs

https://github.com/ai2cm/pace
**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)*

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

Productivity

Generality

Performance

C++ / Fortran

DSL
Separation of Concerns

PDEs

\[ \nabla^2 u \]

Grid

Discretization

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

for \( i = 1, ni \)
  for \( j = 1, nj \)
    for \( k = 1, nk \)
      \[ u[i+1,j,k] - 2u[i,j,k] + u[i-1,j,k] \]

Implementation

Optimization
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
<table>
<thead>
<tr>
<th>Authors</th>
<th>Scheme</th>
<th>Status</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mikael</td>
<td>GFDL Cloud Microphysics Scheme</td>
<td>Ported</td>
<td></td>
</tr>
<tr>
<td>Chris</td>
<td>GFS scale-aware EDMF PBL and Free Atmospheric Turbulence Scheme</td>
<td>Ported</td>
<td></td>
</tr>
<tr>
<td>Vera</td>
<td>GFS Sea Ice Scheme</td>
<td>Ported</td>
<td></td>
</tr>
<tr>
<td>Nadjia</td>
<td>GFS SAS-based Mass-Flux Scheme for Shallow convection (sa-MF)</td>
<td>Ported</td>
<td></td>
</tr>
<tr>
<td>Nicolai</td>
<td>GFS Noah Land Surface Model</td>
<td>Ported</td>
<td></td>
</tr>
<tr>
<td>Langwen</td>
<td>GFS RRTMG</td>
<td>Ported</td>
<td></td>
</tr>
</tbody>
</table>

Check it out on GitHub! https://github.com/ai2cm/physics_standalone
Just-in-time (JIT) compilation

**Fortran**

- `file.f90`
- `compile` to `a.out`
- `configure` to `input.nml` and `job.slurm`
- `submit` to `run`

**Python DSL**

- `file.py`
- `configure` to `input.nml` and `job.slurm`
- `submit` to `run`

**Cache**

The diagram also shows a cache symbol, indicating that the system may cache compiled files for faster access.