Pace: A GPU-enabled implementation of FV3GFS Using GT4Py

Oliver Elbert, NOAA GFDL

Johann Dahm, Eddie Davis, Florian Deconinck, Rhea George, Jeremy McGibbon, Tobias Wicky, Elynn Wu, Christopher Kung, Tal Ben-Nun, Lucas Harris, Linus Groner, and Oliver Fuhrer
## The Future of HPC

### 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, <strong>4 x 250X</strong></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>6.02</td>
<td>309.1</td>
<td>1 x AMD, <strong>4 x 250X</strong>*</td>
</tr>
<tr>
<td>4</td>
<td>Leonardo</td>
<td>3,456</td>
<td>7.4</td>
<td>238.7</td>
<td>1 x Xeon, <strong>4 x A100</strong></td>
</tr>
<tr>
<td>5</td>
<td>Summit</td>
<td>4,608</td>
<td>10.1</td>
<td>148.6</td>
<td>2 x Power9, <strong>6 x V100</strong></td>
</tr>
<tr>
<td>6</td>
<td>Sierra</td>
<td>4,474</td>
<td>7.4</td>
<td>94.6</td>
<td>2 x Power9, <strong>4 x V100</strong></td>
</tr>
<tr>
<td>7</td>
<td>TaihuLight</td>
<td>40,960</td>
<td>15.4</td>
<td>93.0</td>
<td><strong>1 x SW26010</strong></td>
</tr>
<tr>
<td>8</td>
<td>Perlmutter</td>
<td>1,536</td>
<td>2.6</td>
<td>70.9</td>
<td>1 x AMD, <strong>4 x A100</strong></td>
</tr>
<tr>
<td>9</td>
<td>Selene</td>
<td>560</td>
<td>2.6</td>
<td>63.5</td>
<td>2 x AMD, <strong>8 x A100</strong></td>
</tr>
</tbody>
</table>
Approaches

- Rewrite model for hardware backend
- Adapt code with compiler directives
- Write model in a DSL
GridTools for Python

Write model code in Python using GT4Py library

Have access to Python environment for infrastructure, development, testing, etc…

DSL compiler translates code to C++/CUDA and optimizes code for hardware target

[Diagram: User code → DSL Frontend (GT4Py) → DSL Compiler
Checkers
Optimizers
Code Generators

Python

CPU Compiler

GPU Compiler]

github.com/GridTools/gt4py
The Pace Model

GT4Py port of FV3 + v2 of GFDL cloud microphysics

Rest of physics routines ported, not optimized or integrated yet

Contains infrastructure and utilities needed to run simulations, test code, profile performance…

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

Dahm et al. 2023:

https://gmd.copernicus.org/articles/16/2719/2023/
Comparing to Fortran

Moist baroclinic instability integrated for 9 days

Results match fairly well given arithmetic changes

Plotted: 850 mbar temperature
Performance

~3.6x speedup over Fortran on P100 GPUs, ~8.6x on A100 GPUs

CPU performance tuning coming soon, currently ~25% of Fortran speed

Ben-Nun et al. 2022:

https://dl.acm.org/doi/abs/10.5555/3571885.3571982

Take Advantage of Python

Python ecosystem available for model development

Can use Jupyter notebooks to run the model, or individual model components

Ex: run tracer advection for 10 steps and plot the results
Ongoing Development

Latest GFDL Cloud Microphysics
Integrate and optimize more physics parameterizations
CPU performance
Additional Capabilities

Put Pace to work!
Thank You!

Former AI2 DSL Team

Oliver Elbert  
Oli Fuhrer  
Johann Dahm

Tobias Wicky  
Rhea George  
Eddie Davis

Jeremy McGibbon  
Elynn Wu  
Florian Deconinck

Collaborators

AI2  
CSCS  
NASA  
UNIVERSITY of WASHINGTON  
ETH  
MeteoSwiss
This slide intentionally left blank
GPUs vs CPUs

CPU:
Complex instructions done in serial

```
<table>
<thead>
<tr>
<th>7</th>
<th>2</th>
<th>1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>5</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4.9e7</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>21</td>
</tr>
</tbody>
</table>
```

"Add 3 to column 1, then divide the first row by 7, then multiply elements 1, 3, 7, 12, and 13 by 4..."

GPU:
Simple instructions done in parallel

```
<table>
<thead>
<tr>
<th>7</th>
<th>2</th>
<th>1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>5</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4.9e7</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>21</td>
</tr>
</tbody>
</table>
```

"Multiply all these by 5"

"Ok!"
GT4Py Stencils

Instead of looping through x, y, z define a function that gets moved around the grid and applied to all the points you want

Pass 3D fields as arguments

```python
@gtscript.stencil(backend=backend)
def laplacian(
    in_field: FloatField,
    out_field: FloatField
):
    with computation(PARALLEL), interval(...):
        out_field = (-4.0 * in_field + in_field[1, 0, 0] + in_field[0, 1, 0] + in_field[-1, 0, 0] + in_field[0, -1, 0])
```
Stencils run on all horizontal indices in parallel.

Can set vertical order and boundaries within stencil through computation and interval.

FORWARD to go from $k=0$ to $k_{end}$, BACKWARD to go in reverse.

Interval slices like numpy.

Use offsets instead of absolute indices $i-1$ instead of $i=13$.

Can’t write with offsets.
Comparing Code

**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
     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)

**GT4P**

```python
@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, **OMP** 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
DaCe

GT4Py Pipeline

- General Analysis
- Per-target Optimization

User Code

Compilation Toolchain

Generated Code

GT4Py Stencils

- NumPy
- C++
- CUDA

Stencils

Per-target Optimization

Compilation Toolchain

User Code

Generated Code

GT4Py Pipeline

- General Analysis
- Per-target Optimization

User Code

Compilation Toolchain

Generated Code
import dace

@dace.program
def somethingelse(x):
    return x * 5

@dace.program
def example(A: dace.float64[20, 20]):
    B = somethingelse(A)
    C = A + A
    B += C
    return np.dot(B, A)
Running the Dycore

Python runfile.py

Import dynamical core, state setup modules

Load namelist from file, initialize grid, state, and dycore

Then simple loop over timesteps.

Full access to Python ecosystem for performance monitoring, postprocessing, etc.
Object Orientation

Most stencils live inside classes

- Preserves temporary storages
- Stencils compile at init-time
- Simple organization

__init__ creates an object of the class, handles stencil compilation, etc.

__call__ means objects are called like functions
Object Orientation

Objects create objects inside their constructors

Ex: fvtp2d uses xppm

```
self.x_piecewise_parabolic_inner = XPiecewiseParabolic(
    stencil_factory=stencil_factory,
    dxa=grid_data.dxa,
    grid_type=grid_type,
    iord=ord_inner,
    origin=idx.origin_compute(add=(0, -idx.n_halo, 0)),
    domain=idx.domain_compute(add=(1, 1 + 2 * idx.n_halo, 1)),
)
```

```
self.x_piecewise_parabolic_inner(q, crx, self._q_x_advected_mean)
```
Suite of tests to make sure code matches Fortran as Pace builds up
Serialbox
Handy tool from GridTools team
Extracts data from C++, Python, Fortran codes
For Fortran: !$ser statements
Save data or execute code
Python preprocessor replaces !$ser with #ifdef SERIALIZE
Compile Fortran with -DSERIALIZE and run

```fortran
!$ser savepoint XPPM-In
!$ser data_kbuff k=k k_size=nz qx=q_i cx=crx
call xppm(fx, q_i, crx(is,js), ord_ou, is,ie,isd,iel, js,je,jsd,jed, &
    npx,npy, gridstruct%dx, gridstruct%nested, gridstruct%grid_type, lim_fac,regional)
!$ser savepoint XPPM-Out
!$ser data_kbuff k=k k_size=nz xflux=fx

!$ser verbatim bdt=dt_atmos/real(abs(p_split))
```