of all trades — master of some?

GPU-accelerated ocean modelling in Python

Dion Häfner
Roman Nuterman
Markus Jochum
Niels Bohr Institute, University of Copenhagen, Denmark
dion.haefner@nbi.ku.dk
Higher-level geophysical models

- Same code needs to run on CPU & GPU
- Your time is more valuable than your computer’s time
- Interfaceability with external libraries & ML
- People seem to enjoy it
- Your students are using Windows

https://dionhaefner.github.io/2021/04/higher-level-geophysical-modelling/
Problems with Fortran are getting recognized … and hopefully addressed

https://ondrejcertik.com/blog/2021/03/resurrecting-fortran/

RESURRECTING FORTRAN

March 12, 2021

We have finished the first chapter on a long journey of resurrecting Fortran: forming an initial core community of developers and users with enough momentum around the new Fortran website and projects at fortran-lang.org. There is still have a long way to go to, but this was a necessary (although not sufficient) achievement and I feel this was a once in a lifetime event. The timing was right, we were ready, and we also got lucky. In this blog post I have tried to write up the details about the pivotal moments from my perspective. I have asked Milan Curcic who has been a co-founder of this effort to do the same (please see his blog post), more on that below. Brad Richardson has also written about his perspective.

This has been the most unusual experience for me, nothing like what I anticipated, but two years ago things seemed to fall into place and set up just right. I had to give it everything I had in order to give it a proper chance. I took the bull by the horns.

This blog post is long, but you might find it useful to get ideas how to form other such communities.

BACKGROUND

I have been using Fortran on and off for a long time, but only started using it as my main language around 2010 thanks to John Pask, my advisor at LLNL at a time, who introduced me to modern Fortran and I saw that it was a superior choice for high performance numerical code than Python - my main choice until that time. I immediately realized there is not much information about modern Fortran online, so I started the fortran90.org website in 2012 to collect information about the
In the meantime:
High-performance ocean modelling in Python

Array operations
- NumPy
- JAX

Solvers
- petsc4py

Communication
- mpi4py
- mpi4jax

GPU

MPI

PETSc
What is Veros?
The versatile ocean simulator.

- Translation of PyOM2 (Fortan ocean model) to Python
- Runs on your laptop, gaming PC (GPU), or cluster
- Full-fledged primitive equation model
- Idealized and realistic setups
- 140 model variables, 100 settings

https://github.com/team-ocean/veros
Fortran to Python

do i=js_pe-1,je_pe

Search:

\[ \text{do } (w) = ( (w | [-+])+, (w | [-+])+ )/ \]

Replace:

\[ \text{for } \1 \text{ in range}(\2): \]

for i in range(is_pe-1, ie_pe):
Python to NumPy
Most of it is straightforward

Fortran

```fortran
subroutine get_tke_surface_correction
  ! explicit loop in Fortran, modify state in-place
  use main_module
  tke_surf_corr = 0.0
  do j=js_pe,je_pe
    do i=is_pe,ie_pe
      if (tke(i,j,nz,taup1) < 0.0) then
        tke_surf_corr(i,j) = -tke(i,j,nz,taup1)*0.5*dzw(nz) / dt_tke
        tke(i,j,nz,taup1) = 0.0
      endif
    enddo
  enddo
```

NumPy

```python
import numpy as np

def get_tke_surface_correction(tke, dzw, taup1, dt_tke):
  # use a boolean mask instead of `if` statement
  # and return changed arrays
  tke_surf_corr = np.zeros(tke.shape[:2])
  mask = tke[2:-2, 2:-2, -1, taup1] < 0.0
  tke_surf_corr[2:-2, 2:-2] =
    -tke[2:-2, 2:-2, -1, taup1] * 0.5 * dzw[-1] / dt_tke
    * mask
  tke[2:-2, 2:-2, -1, taup1] += mask
  return tke_surf_corr, tke
```

(>90% of code)
Now with JAX support!

Ingredients:

- Change data model so all kernels are pure functions
- Adapt index update syntax (JAX arrays are immutable)
- Write mpi4jax so we can use MPI calls from compiled functions
Demo time
Modern high-performance computing in Python

Simple to write

```python
@partial(jax.jit, static_argnums=(1,))
def enforce_boundaries(arr, grid, token=None):
    # ... etc.
    recv_arr, token = mpi4jax.sendrecv(
        send_arr, recv_arr,
        source=recv_proc, dest=send_proc, comm=mpi_comm,
        token=token,
    )
    arr = arr.at[recv_idx].set(recv_arr)
```

Scales

Runs on any hardware

... but is it fast?

https://mpi4jax.readthedocs.io/en/latest/shallow-water.html
With JAX, our pure Python ocean model Veros is close to Fortran performance on CPU clusters. It also has almost perfect weak scaling on GPUs, using at least 3x less energy.

We can even integrate a global 0.1° eddy-resolving setup on a single Google Cloud GPU instance with 16 NVIDIA A100 GPUs. The performance is equivalent to at least 2000 CPU cores running a Fortran model.
It works well on CPU

Scaling with CPU cores
Wall time per 1M grid cells (lower is better)

Relative speedup (higher is better)
It works well on GPU

Scaling with CPU cores
Wall time per 1M grid cells (lower is better)

Strong scaling with GPUs
Wall time per 1M grid cells (lower is better)

Weak scaling with GPUs
Wall time per 1M grid cells (lower is better)
High-resolution setups on GPU

- 1 cloud VM
- 16 A100 GPUs
- 600GB memory
- 1 model year/day (equivalent to > 2000 Fortran processes)
- ~10x less energy usage
Open problem
Separating physics from numerics

vs.flux_east[1:-2, 2:-2, :] = \[
0.25 * (vs.u[1:-2, 2:-2, :, vs.tau] + vs.u[2:-1, 2:-2, :, vs.tau]) \]
* (vs.utr[1:-2, 2:-2, :] + vs.utr[1:-2, 2:-2, :])

Abstraction

API Draft

vs.flux_east.update(
on('t', vs.u[...], vs.tau]) * on('t', vs.utr)
So, what else can we do with it?
JAX code is also natively differentiable!

```python
>>> salt = 35.
>>> temp = 12.
>>> press = 100.
>>> get_dHdT(salt, temp, press)
DeviceArray(0.18120956, dtype=float32)
>>> import jax
>>> jax.grad(get_dHdT, argnums=(0,))(salt, temp, press)
(DeviceArray(0.00208719, dtype=float32),)
```
Integration with machine learning
Parameter estimation

Integration with machine learning
Emulation & sub-grid scale parameterization

Integration with machine learning
Can we use models and ML to make discoveries?

*There are many things we don’t understand in earth system models.*

- What are necessary / sufficient causes of events like El Niño?
- How does information propagate between distant regions?
- Are there macro-scale anomalies that we are not aware of?
- Can we search for tipping points?
- Which knobs do we need to turn to eliminate known biases?

Machine learning might be able to help with these!
Thank you for listening!
