JAX: writing Performant Portable GPU code in Python.

NUG Meeting 2022: Performance Portability Panel

Nestor Demeure¹
¹ National Energy Research Scientific Computing Center, Berkeley CA, United-states
Who am I?

I am a [NESAP Postdoctoral Researcher at NERSC](#) with a focus on high performance computing, numerical accuracy and artificial intelligence.

I specialize in helping teams of researchers make use of high performance computing environments.

I am currently working to help port the [TOAST software framework](#) to the new Perlmutter supercomputer and, in particular, port it to graphic processors (GPU).
Can we have good GPU performance, portability and productivity?
Introducing JAX

High-level introduction to JAX
**What is JAX?**

**JAX** is a Python library to write code that can run in parallel on:

- CPU,
- GPU (Nvidia and AMD),
- TPU,
- etc.

Developed by Google as a building block for deep-learning frameworks. Seeing wider use in numerical applications including:

- **Molecular dynamics**, 
- **computational fluid dynamics**, 
- **ocean simulation**.
What does JAX look like?

It has a Numpy-like interface:

```python
from jax import random
from jax import numpy as jnp

key = random.PRNGKey(0)
x = random.normal(key, shape=(3000, 3000), dtype=jnp.float32)

y = jnp.dot(x, x.T)  # runs on GPU if available
```
How does JAX work?

Calls a *just-in-time compiler* when you execute your function with a new problem size:
JAX’s limitations

- Compilation happens just-in-time, at runtime, easily amortized on a long running computation
- Input sizes must be known to the tracer, padding, masking and recompiling for various sizes
- Loops and tests are limited inside JIT sections, JAX provides replacement functions
- No side effects and no in-place modifications, one gets used to it, it actually helps with correctness
- Focus on GPU optimizations rather than CPU, there is growing attention to the problem
Is it worth it?
Case study

Porting the TOAST codebase to GPU
**TOAST** is a large Python application used to study the cosmic microwave background.

It is made of pipelines distributed with MPI and composed of C++ kernels parallelized with OpenMP.

Kernels use a **wide variety of numerical methods** including random number generation, linear algebra and fast fourier transforms.

We ported one pipeline to GPU, from C++ to Numpy to JAX.
Porting the code (x7 reduction in lines of code)
Performance per kernel (up to x17 speed-up)

- stage_requirements_to_device
- finalize
- template_offset_apply_diag_precond
- template_offset_add_to_signal
- pointing_detector
- build_noise_weighted
- pixels_healpix
- stokes_weights_IQU
- template_offset_project_signal

Cumulative runtime in seconds:

- OpenMP
- JAX GPU
Overview

Should you use JAX in your project?
I believe JAX is in a **sweet spot for research and complex numerical codes**:

- Focus on the semantic, leaves optimization to the compiler,
- single code base to deal with CPU and GPUs,
- immutable design is actually *nice* for correctness,
- easy to use numerical building blocks inside kernels.
Should you use JAX?

- Your code is written in **Python**,
- your code can be written with **Numpy**,
- your array sizes are **not too dynamic**,
- single-thread CPU is an **acceptable fallback** in the absence of GPU.
Thank you!

ndemeure@lbl.gov