tred

Initial functionality (version 0.0.2)

Brett Viren

February 4, 2025

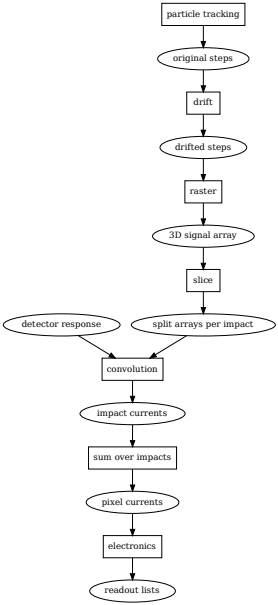
Summary of status

Almost “full” chain from **depos** (not yet steps) to **induced currents**, caveats:

- Current tests use a small number of hard-coded depos.
- Hard-wire use of NDLarSim’s response.
- Induced currents still in block, chunked form.
- No readout model yet.

Other features:

- CPU or GPU execution, function or computation graph
- partial data file input suport, simple output file



Tred has CPU/GPU Support

32xCPU is currently faster than 1xGPU!

```
$ uv run tred plots graph
INFO 0.0009701251983642578 construct (graph.py:runit)
INFO 7.390975952148438e-05 to device (graph.py:runit)
INFO 0.00019550323486328125 make depos (graph.py:runit)
INFO 0.059168100357055664 get response (graph.py:runit)
INFO 3.183227777481079 run sim (graph.py:runit)
INFO 2.2133476734161377 save data (graph.py:runit)
INFO 5.4569830894470215 TOTAL (graph.py:runit)
INFO FINISHED CPU (graph.py:plots)

INFO 0.00016307830810546875 construct (graph.py:runit)
INFO 0.27073025703430176 to device (graph.py:runit)
INFO 0.0058634281158447266 make depos (graph.py:runit)
INFO 0.07225227355957031 get response (graph.py:runit)
INFO 4.62535548210144 run sim (graph.py:runit)
INFO 2.3531875610351562 save data (graph.py:runit)
INFO 7.327552080154419 TOTAL (graph.py:runit)
INFO FINISHED CUDA (graph.py:plots)
```

This is a small job but I feel it is too slow! No optimization yet.

CPU/GPU programming with PyTorch

Temporary tensors

Must be careful to set device of a temporary tensor from an input.

```
def f(ten):
    tmp = torch.tensor(..., device=ten.device)
```

Computation graph

Computation graph takes care of “constant” tensors.

```
class Drifter(nn.Module):
    def __init__(self, diffusion, ...):
        self.register_buffer("diffusion", diffusion)

drifter = Drifter(...)
drifter.to('cuda')
```

Input data

Must move data by hand, eg in `tred/plots/graph.py`:

```
response = get_ndlarsim().to(device=device)
depos = make_depos(device) # put on device internally
```

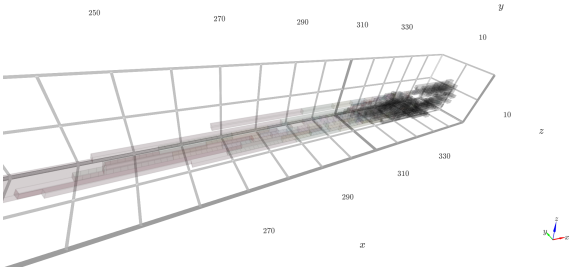
PyTorch Datasets do NOT put data on the device.

Some visualization of one chunk with k3d-jupyter

```
$ ssh -L 8888:localhost:8888 wcgpu0
$ uv run --with k3d --with jupyter jupyter notebook
### locally:
$ firefox http://localhost:8888/tree?token=<token>
```

```
# In jupyter notebook web page
import numpy
from tred.plots import k3dj
fp = numpy.load("full.npz")
meas = fp['meas_data']

# pick chunk with large currents
k3dj.voxels(m[3,:,6000:])
```



Colors did not capture, dark region is near end / collection.

Exercising the code

No “real” main program yet. Using “plot” functions to drive the functionality for now.

```
$ tred plots full
$ tred plots graph
```

- **full** hard-wired, low-level function calls.
- **graph** hard-wired, high-level Torch `nn.Module` computation graph.

Neither actually make “plots” yet, but do dump out a `.npz` file with inputs, results and some intermediate tensors.
Actual code is in `tred/plots/{full,graph}.py`.

Some work needed (mostly for Yousen)

Steps

1. Add `tred/raster/steps.py` with Q_{eff} function
2. Add `import` of the Q_{eff} function to `tred/graph.py` as `raster_steps()` and delete the dummy there. It is called by the `Raster` node. No other changes needed.
3. Add `make_steps()` like `make_depos()` in `tred/plots/graph.py` and a variant of `runit()` (for testing).

General

- Algorithm validation is needed! Though, I have been checking as I go.
- Understanding and optimizing CPU/GPU and RAM usage needed.

Some work needed (mostly for me)

User friendly

- Make PyTorch `Dataset` versions of current “data loaders” (depo/steps).
- Add a user configuration.

Optimization

- Understanding and optimizing CPU/GPU and RAM usage needed.
- Insinuate a “dynamic rebatching” between computation graph nodes.
  - If optimization work calls for it.