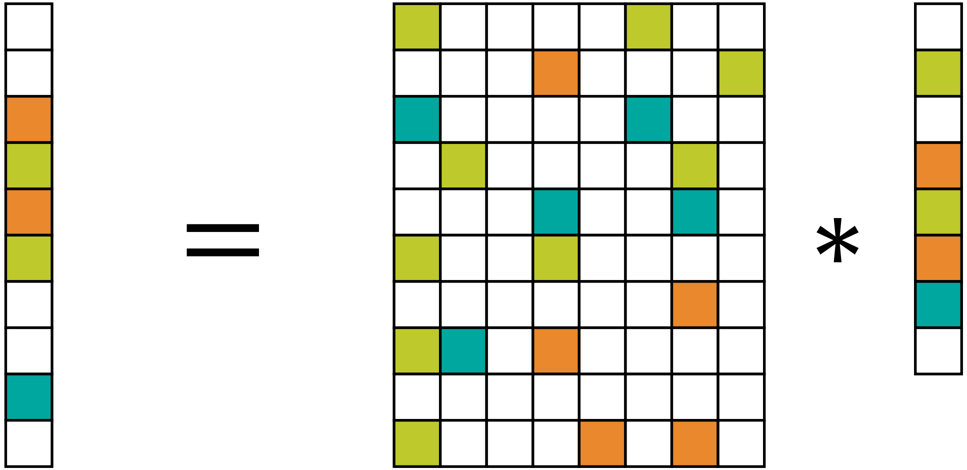
Leveraging GPUs for matrix-free optimization with PyLops

M. Ravasi

Fifth EAGE Workshop on HPC for Upstream

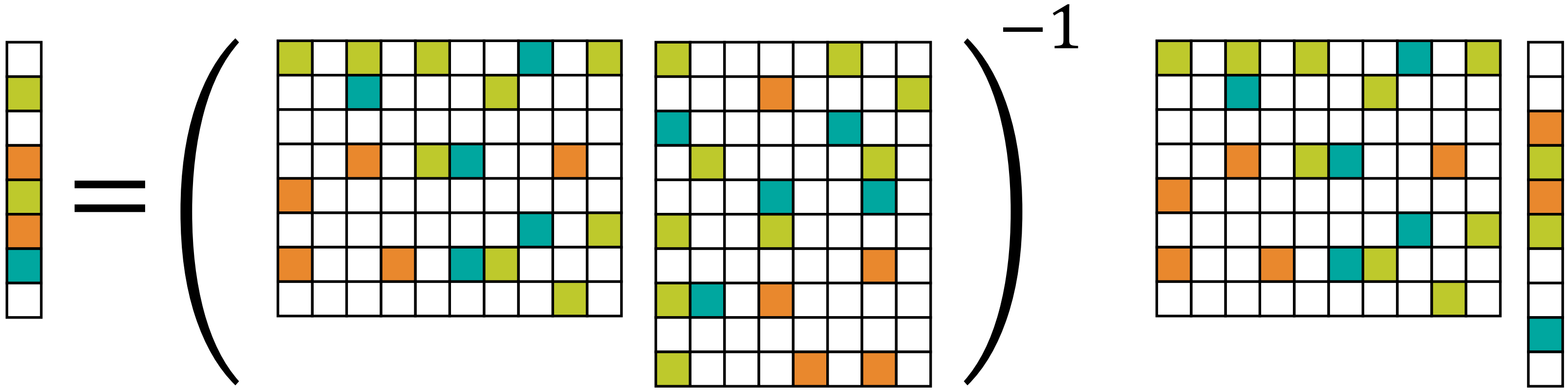
Inverse problems (pen and paper)



$$\mathbf{d} = \mathbf{G} * \mathbf{m}$$

Observations | Modelling matrix | Model

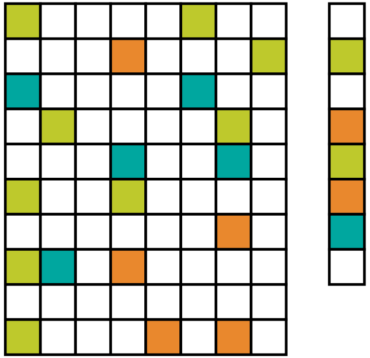
Inverse problems (pen and paper)



$$\hat{\mathbf{m}} \qquad \underbrace{\mathbf{G}^{\mathbf{H}} \qquad \mathbf{G} \qquad \mathbf{G}^{\mathbf{H}}}_{\text{Pseudo-inverse}} \qquad \mathbf{d}$$

Est. model

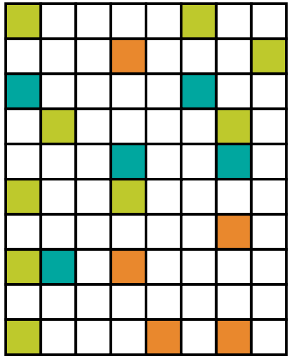
Inverse problems (a computer perspective)



Gm

Very expensive

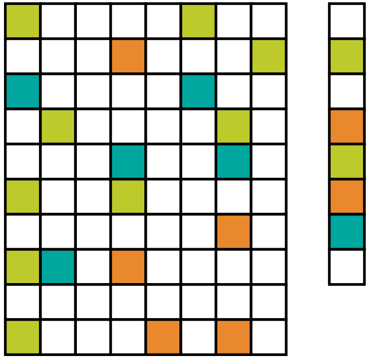
&



G

Very large

Inverse problems (a computer perspective)



$$\mathbf{Gm}$$

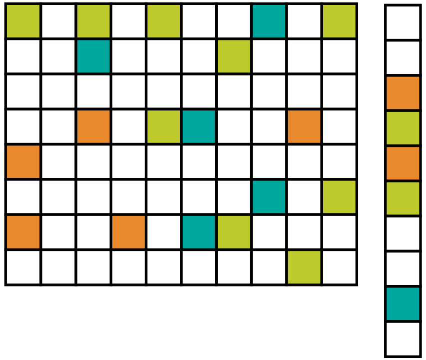
```
# Forward
def matvec(m):

    ---------
    ---------
    ---------
    ---------

    return y
```

$$\mathbf{y = Gm}$$

Inverse problems (a computer perspective)



$$\mathbf{G}^H\mathbf{d}$$

```
# Adjoint
def rmatvec(d):

    ---------
    ---------
    ---------
    ---------

    return x
```

$$\mathbf{x} = \mathbf{G}^H\mathbf{d}$$

Inverse problems (a computer perspective)

```
# Forward
def matvec(m):

    --------
    --------
    --------
    --------

    return y
```

$$\mathbf{y} = \mathbf{G}\mathbf{m}$$

&

```
# Adjoint
def rmatvec(d):

    --------
    --------
    --------
    --------

    return x
```

$$\mathbf{x} = \mathbf{G}^{\mathbf{H}}\mathbf{d}$$

<u>Matrix-free optimization</u>

Matrix-free inverse problems

$$J = \|\mathbf{d} - \mathbf{G}\mathbf{m}\|_p + R(\mathbf{m})$$

$\mathbf{d}$

Modelling

```
matvec
rmatvec
```

$\longrightarrow$

Solver

```
CG
CGLS
LSQR
ISTA
Split-Bregman
Proximal
```

$\longrightarrow$ $\hat{\mathbf{m}}$

Matrix-free inverse problems with PyLops

An ecosystem of tools to ease research in inverse problems

-> +50 Operators (CPU + GPU)

-> Least-squares, L1 (sparsity), and Proximal solvers

-> Sparsity transforms (e.g., FFT, FFTN, DWT, Curvelet, Seislet)

-> PyTorch integration for Autograd

-> Dask integration for distributed operators

Matrix-free inverse problems with PyLops

An ecosystem of tools to ease research in inverse problems

-> +50 Operators (CPU + **GPU**)

-> Least-squares, L1 (sparsity), and Proximal solvers

-> Sparsity transforms (e.g., FFT, FFTN, DWT, Curvelet, Seislet)

-> PyTorch integration for Autograd

-> Dask integration for distributed operators

Matrix-free inverse problems with PyLops

Basic operators

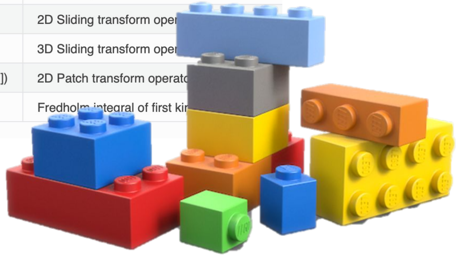
`MatrixMult` (A[, dims, dtype])	Matrix multiplication.
`Identity` (N[, M, dtype])	Identity operator.
`Zero` (N[, M, dtype])	Zero operator.
`Diagonal` (diag[, dims, dir, dtype])	Diagonal operator.
`Restriction` (M, iava[, dims, dir, dtype])	Restriction (or sampling) operator.
`Regression` (taxis, order[, dtype])	Polynomial regression.
`LinearRegression` (taxis[, dtype])	Linear regression.
`CausalIntegration` (N[, dims, dir, sampling, …])	Causal integration.
`Spread` (dims, dimsd[, table, dtable, fh, …])	Spread operator.
`Flip` (N[, dims, dir, dtype])	Flip along an axis.
`Symmetrize` (N[, dims, dir, dtype])	Symmetrize along an axis.
`VStack` (ops[, dtype])	Vertical stacking.
`HStack` (ops[, dtype])	Horizontal stacking.
`Block` (ops[, dtype])	Block operator.
`BlockDiag` (ops[, dtype])	Block-diagonal operator.

Smoothing and derivatives

`Smoothing1D` (nsmooth, dims[, dir, dtype])	1D Smoothing.
`Smoothing2D` (nsmooth, dims[, nodir, dtype])	2D Smoothing.
`FirstDerivative` (N[, dims, dir, sampling, dtype])	First derivative.
`SecondDerivative` (N[, dims, dir, sampling, dtype])	Second derivative.
`Laplacian` (dims[, dirs, weights, sampling, dtype])	Laplacian.

Signal processing

`Convolve1D` (N, h[, offset, dims, dir, dtype, …])	1D convolution operator.
`Convolve2D` (N, h, dims[, offset, nodir, …])	2D convolution operator.
`ConvolveND` (N, h, dims[, offset, dirs, …])	ND convolution operator.
`Interp` (M, iava[, dims, dir, kind, dtype])	Interpolation operator.
`Bilinear` (iava, dims[, dtype])	Bilinear interpolation operator.
`FFT` (dims[, dir, nfft, sampling, real, …])	One dimensional Fast-Fourier Transform.
`FFT2D` (dims[, dirs, nffts, sampling, dtype])	Two dimensional Fast-Fourier Transform.
`FFTND` (dims[, dirs, nffts, sampling, dtype])	N-dimensional Fast-Fourier Transform.
`DWT` (dims[, dir, wavelet, level, dtype])	One dimensional Wavelet operator.
`DWT2D` (dims[, dirs, wavelet, level, dtype])	Two dimensional Wavelet operator.
`Seislet` (slopes[, sampling, level, kind, …])	Two dimensional Seislet operator.
`Radon2D` (taxis, haxis, pxaxis[, kind, …])	Two dimensional Radon transform.
`Radon3D` (taxis, hyaxis, hxaxis, pyaxis, pxaxis)	Three dimensional Radon transform.
`ChirpRadon2D` (taxis, haxis, pmax[, dtype])	2D Chirp Radon transform
`ChirpRadon3D` (taxis, hyaxis, hxaxis, pmax[, …])	3D Chirp Radon transform
`Sliding1D` (Op, dim, dimd, nwin, nover[, …])	1D Sliding transform operator.
`Sliding2D` (Op, dims, dimsd, nwin, nover[, …])	2D Sliding transform oper
`Sliding3D` (Op, dims, dimsd, nwin, nover, nop)	3D Sliding transform oper
`Patch2D` (Op, dims, dimsd, nwin, nover, nop[, …])	2D Patch transform operat
`Fredholm1` (G[, nz, saveGt, usematmul, dtype])	Fredholm integral of first ki



Matrix-free inverse problems with PyLops

Gemv, Gemm → MatrixMult

Element-wise prod. → Diagonal

Basic operators

`MatrixMult` (A[, dims, dtype])	Matrix multiplication.
`Identity` (N[, M, dtype])	Identity operator.
`Zero` (N[, M, dtype])	Zero operator.
`Diagonal` (diag[, dims, dir, dtype])	Diagonal operator.
`Restriction` (M, iava[, dims, dir, dtype])	Restriction (or sampling) operator.
`Regression` (taxis, order[, dtype])	Polynomial regression.
`LinearRegression` (taxis[, dtype])	Linear regression.
`CausalIntegration` (N[, dims, dir, sampling, …])	Causal integration.
`Spread` (dims, dimsd[, table, dtable, fh, …])	Spread operator.
`Flip` (N[, dims, dir, dtype])	Flip along an axis.
`Symmetrize` (N[, dims, dir, dtype])	Symmetrize along an axis.
`VStack` (ops[, dtype])	Vertical stacking.
`HStack` (ops[, dtype])	Horizontal stacking.
`Block` (ops[, dtype])	Block operator.
`BlockDiag` (ops[, dtype])	Block-diagonal operator.

Smoothing and derivatives

Convolution, correlation:

`Smoothing1D` (nsmooth, dims[, dir, dtype])	1D Smoothing.
`Smoothing2D` (nsmooth, dims[, nodir, dtype])	2D Smoothing.
`FirstDerivative` (N[, dims, dir, sampling, dtype])	First derivative.
`SecondDerivative` (N[, dims, dir, sampling, dtype])	Second derivative.
`Laplacian` (dims[, dirs, weights, sampling, dtype])	Laplacian.

Signal processing

`Convolve1D` (N, h[, offset, dims, dir, dtype, …])	1D convolution operator.
`Convolve2D` (N, h, dims[, offset, nodir, …])	2D convolution operator.
`ConvolveND` (N, h, dims[, offset, dirs, …])	ND convolution operator.
`Interp` (M, iava[, dims, dir, kind, dtype])	Interpolation operator.
`Bilinear` (iava, dims[, dtype])	Bilinear interpolation operator.
`FFT` (dims[, dir, nfft, sampling, real, …])	One dimensional Fast-Fourier Transform.
`FFT2D` (dims[, dirs, nffts, sampling, dtype])	Two dimensional Fast-Fourier Transform.
`FFTND` (dims[, dirs, nffts, sampling, dtype])	N-dimensional Fast-Fourier Transform.
`DWT` (dims[, dir, wavelet, level, dtype])	One dimensional Wavelet operator.
`DWT2D` (dims[, dirs, wavelet, level, dtype])	Two dimensional Wavelet operator.
`Seislet` (slopes[, sampling, level, kind, …])	Two dimensional Seislet operator.
`Radon2D` (taxis, haxis, pxaxis[, kind, …])	Two dimensional Radon transform.
`Radon3D` (taxis, hyaxis, hxaxis, pyaxis, pxaxis)	Three dimensional Radon transform.
`ChirpRadon2D` (taxis, haxis, pmax[, dtype])	2D Chirp Radon transform
`ChirpRadon3D` (taxis, hyaxis, hxaxis, pmax[, …])	3D Chirp Radon transform
`Sliding1D` (Op, dim, dimd, nwin, nover[, …])	1D Sliding transform operator.
`Sliding2D` (Op, dims, dimsd, nwin, nover[, …])	2D Sliding transform oper
`Sliding3D` (Op, dims, dimsd, nwin, nover, nop)	3D Sliding transform oper
`Patch2D` (Op, dims, dimsd, nwin, nover, nop[, …])	2D Patch transform operat
`Fredholm1` (G[, nz, saveGt, usematmul, dtype])	Fredholm integral of first kir

Convolve1D, Convolve2D, ConvolveND: Convolution

FFT, FFT2D, FFTND: Fourier Transform

ChirpRadon2D, ChirpRadon3D: Fourier Transform + Element-wise prod.

CPU-based PyLops

Vector containers and basic operations

Scientific operations (e.g., eigenvalues, solvers, conv/corr)

Accelleration on CPUs

Efficient FFTs

Wavelet transform

Curvelet transform

GPU-based PyLops

✗ Lots of code to write

✗ Missing (or immature) scientific routines – eg FFT

✓ AD for free

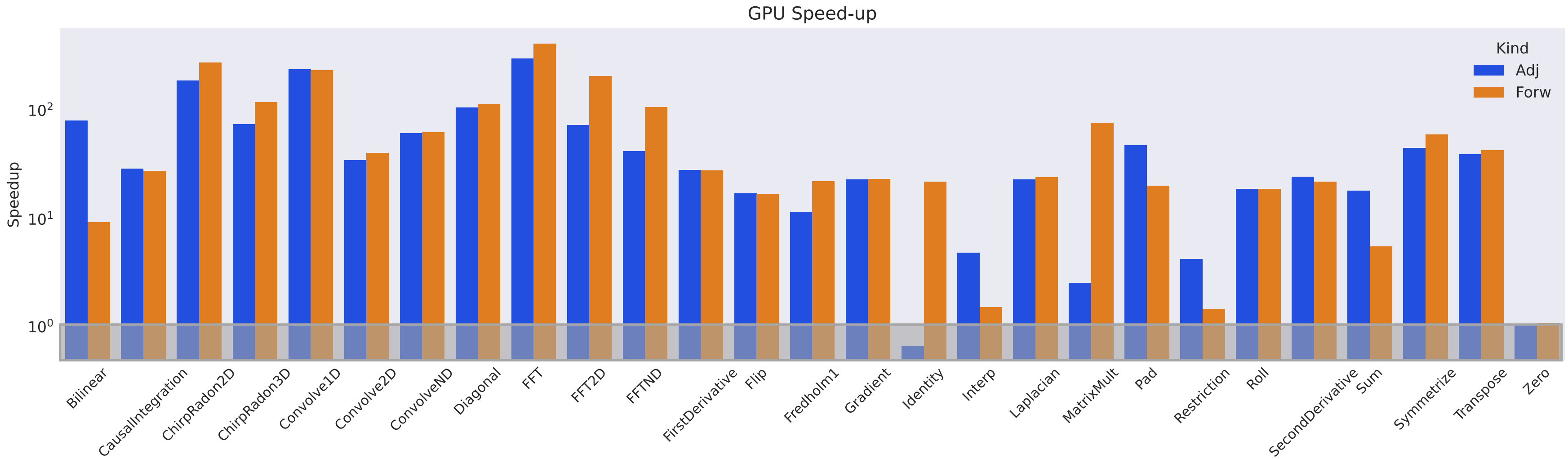
CuPy

cuSignal

✓ Little code to write (same API as NumPy/SciPy)

✗ No AD *(not true!)*

GPU-based PyLops, is it worth the effort?



CPU: Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz with 20 cores

GPU: Tesla V100 GPU.

The road towards GPU integration

→ **Seamless integration for old and new users (*pylops.utils.deps*)**

```
import os
from importlib import util

cupy_enabled = util.find_spec("cupy") is not None and \
               int(os.getenv('CUPY_PYLOPS', 1)) == 1
cusignal_enabled = util.find_spec("cusignal") is not None and \
                   int(os.getenv('CUSIGNAL_PYLOPS', 1)) == 1
```

```
from pylops.utils import deps

if deps.cupy_enabled:
   ----
```

The road towards GPU integration

→ NumPy/SciPy and CuPy/CuSignal interplay (*pylops.utils.backend*)

```
def get_array_module(x):
    if deps.cupy_enabled:
        return cp
    else:
        return np
```

```
ncp = get_array_module(x)
ncp.X
```

Use ncp for either numpy or cupy

```
from cupyx.scipy.linalg import
block_diag as cp_block_diag

def get_block_diag(x):
    if not deps.cupy_enabled:
        return block_diag
    if cp.get_array_module(x) == np:
        return block_diag
    else:
        return cp_block_diag
```

```
y = get_block_diag(x)
```

Map cupyx and scipy routines

The road towards GPU integration

→ New solvers (*pylops.optimization.solvers*)

- SciPy solvers are not compatible with CuPy
- CuPy solvers do not allow LinearOperator

→

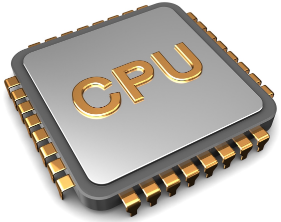
Native CPU/GPU solvers

L2: CG, CGLS, LSQR

L1: OMP, IRLS, ISTA, FISTA, Split-Bregman

Proximal: PyProximal (WIP :GPU backend)

The user experience




```
# Model
x = np.*(nt)

# Operator
G = Convolve1D(nt, g)

# Data
y = G * x

# Inverse
xinv = sp.sparse.linalg.*(G, y)
```

```
# Model
x = cp.*(nt)

# Operator
G = Convolve1D(nt, cp.asarray(g))

# Data
y = G * x

# Inverse
xinv = pylops.optimization.*(G, y)
```

ChirpRadon operator

Forward: $\mathcal{R}\big(f(x,y,t)\big) = \mathcal{F}^{-1}\Big(\big((\mathcal{F}(f)K^*) * K\big)K^*\Big)$

Inverse: $\mathcal{R}^{-1}\Big(f\big(p_x,p_y,\tau\big)\Big) = \mathcal{F}^{-1}\Big(\big((\mathcal{F}(f)K) * K^*\big)K|\omega|\Big)$

F. Andersson and J. Robertsson, 2019, Fast τ–p transforms by chirp modulation, Geophysics.

ChirpRadon operator

FFT/IFFT Element-wise prod.

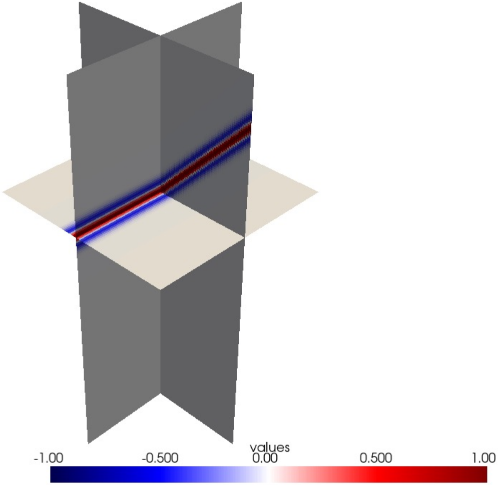
Forward: $\mathcal{R}\big(f(x,y,t)\big) = \mathcal{F}^{-1}\Big(\big(\big(\mathcal{F}(f)K^*\big) * K\big)K^*\Big)$

Conv (FFT+Elp+IFFT)

Inverse: $\mathcal{R}^{-1}\Big(f\big(p_x,p_y,\tau\big)\Big) = \mathcal{F}^{-1}\Big(\big(\big(\mathcal{F}(f)K\big) * K^*\big)K|\omega|\Big)$

F. Andersson and J. Robertsson, 2019, Fast τ−p transforms by chirp modulation, Geophysics.

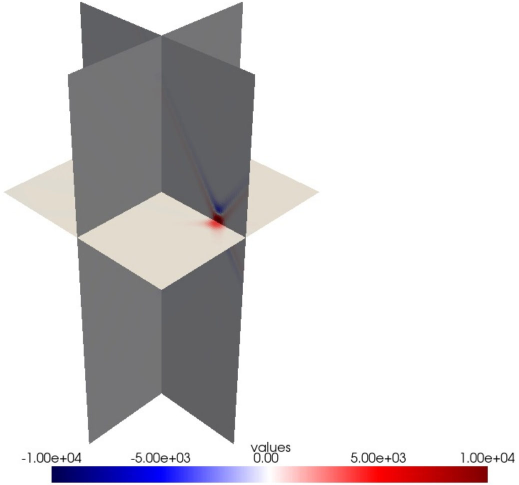
ChirpRadon operator



[201 x 101 x 101]

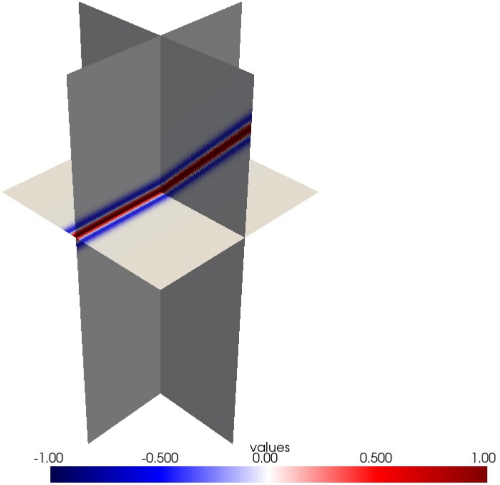
$$\xrightarrow{\mathcal{R}}$$

$$\xleftarrow{\mathcal{R}^{-1}}$$



[201 x 101 x 101]

ChirpRadon operator



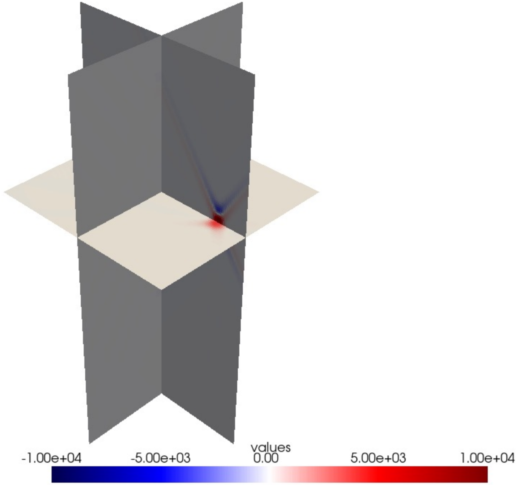
[201 x 101 x 101]

NumPy 1.75 s

CuPy 1.64 ms

1.75 s NumPy

1.71 ms CuPy



[201 x 101 x 101]

PhaseShift operator

Forward: $\mathcal{R}\big(f(x, y, t)\big) = \mathcal{F}^{-1}(\mathcal{F}(f)\Phi)$

Adjoint: $\mathcal{R}^{H}\Big(f\big(k_x, k_y, f\big)\Big) = \mathcal{F}^{-1}(\mathcal{F}(f)\Phi^{*})$

PhaseShift operator

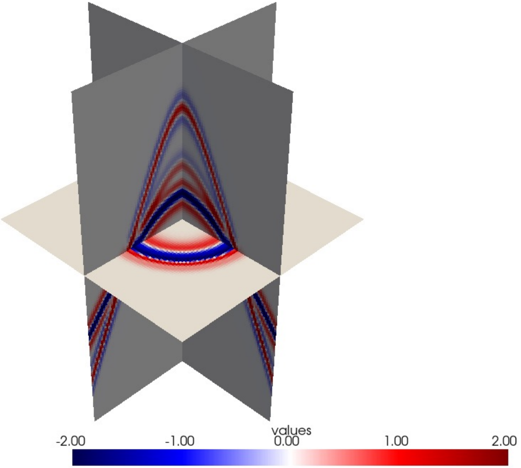
3DFFT/3DIFFT

Forward: $\mathcal{R}\big(f(x,y,t)\big) = \mathcal{F}^{-1}(\mathcal{F}(f)\Phi)$

Element-wise prod.

Adjoint: $\mathcal{R}^{H}\Big(f\big(k_x, k_y, f\big)\Big) = \mathcal{F}^{-1}(\mathcal{F}(f)\Phi^{*})$

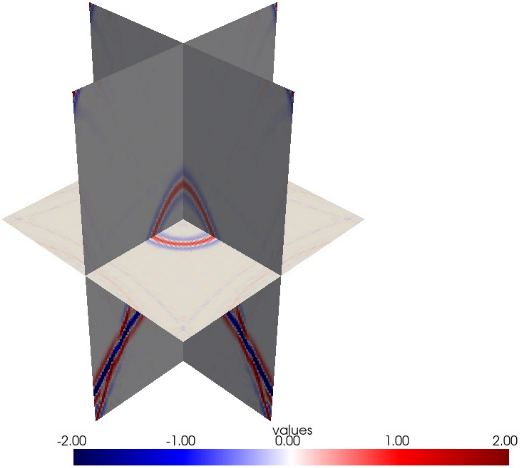
PhaseShift operator



[151 x 101 x 101]

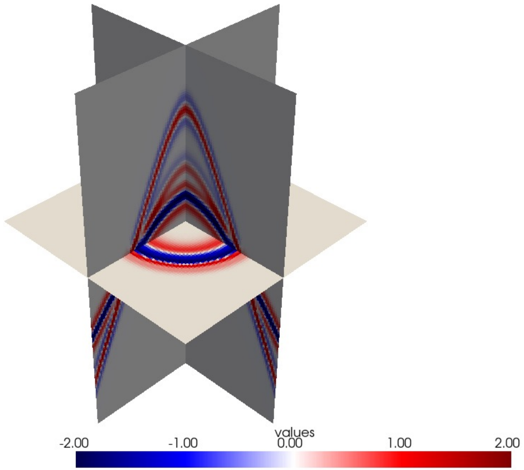
$\xrightarrow{\mathcal{R}}$

$\xleftarrow{\mathcal{R}^H}$



[151 x 101 x 101]

PhaseShift operator



[151 x 101 x 101]

NumPy 0.8 s

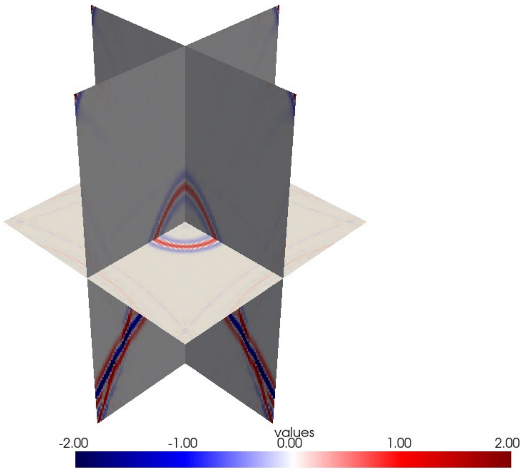
CuPy 1.4 ms

→

0.8 s NumPy

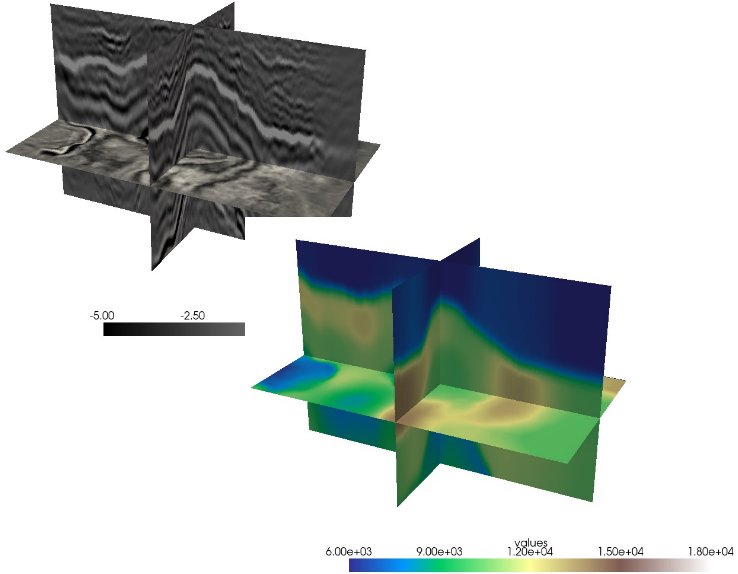
1.4 ms CuPy

←



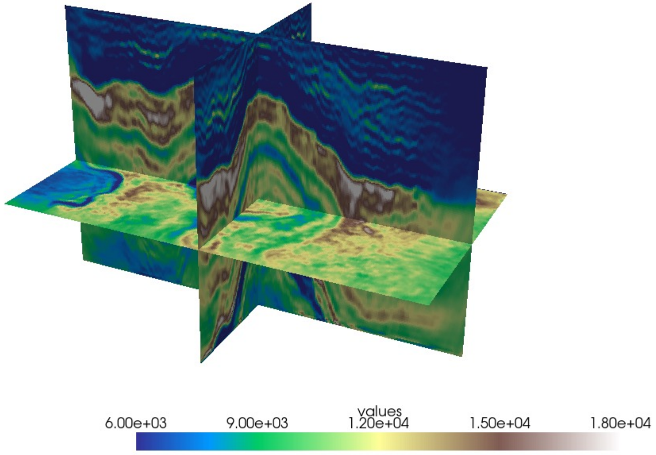
[151 x 101 x 101]

Seismic inversion



NumPy 44.3 s

CuPy 3.4 ms

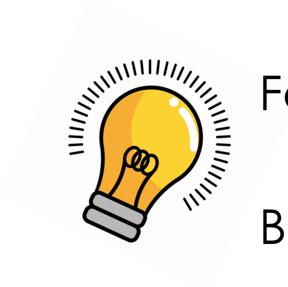


[201 x 361 x 200]

[201 x 361 x 200]

What about AD?

Linear operator: $\mathbf{G}$



Forward: $\mathbf{Gm}$ `def matvec(m)`

Backward: $\mathbf{G^H d}$ `def rmatvec(d)`

```
class TorchOperator(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, forw, adj):
        ctx.forw = forw
        ctx.adj = adj
        --------
        # apply forward operator
        y = ctx.forw(x)
        --------
        return y

    @staticmethod
    def backward(ctx, y):
        --------
        # apply adjoint operator
        x = ctx.adj(y)
        --------
        return x, None, None, None, None
```

https://github.com/PyLops/pylops-gpu/blob/master/pylops_gpu/TorchOperator.py

What about AD?

DLPack

DLPack is a specification of tensor structure to share tensors among frameworks.

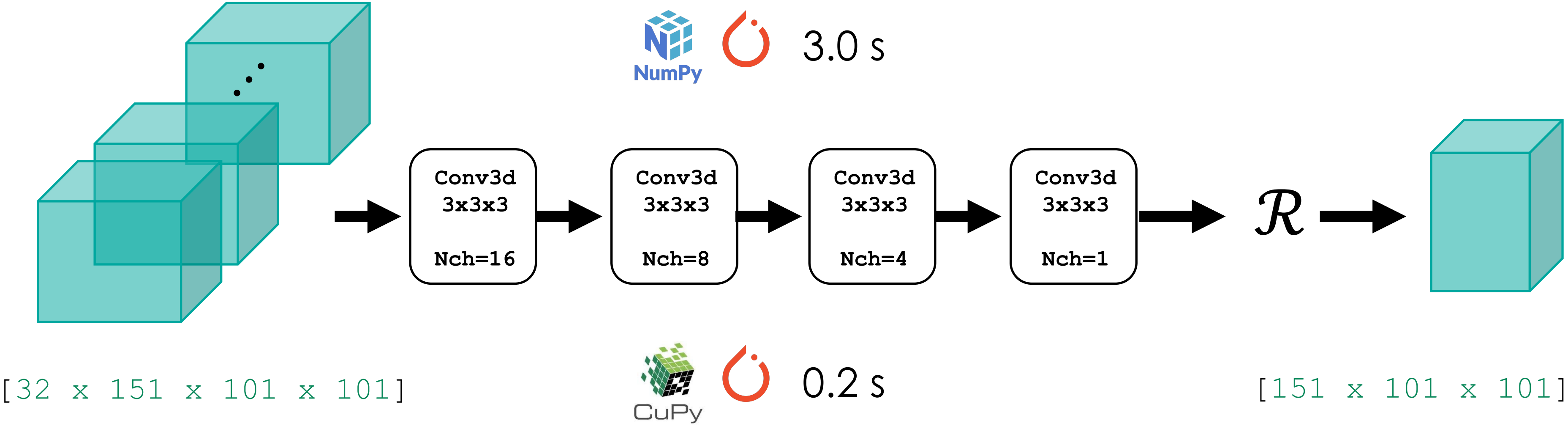
CuPy supports importing from and exporting to DLPack data structure (`cupy.fromDlpack()` and `cupy.ndarray.toDlpack()`).

```
# pass torch array x to cupy
x = cp.fromDlpack(to_dlpack(x))

# pass cupy array y to torch
y = from_dlpack(y.toDlpack())
```

→ Seamless integration between CuPy and PyTorch

What about AD?



From: Ravasi, 2021, Preconditioning Seismic Processing problems with Neural Networks, SEG Annual. Accepted

Lessons learned

-> Matrix-free optimization can greatly benefit from GPU computing

-> Very mature Python ecosystem (CuPy, cusignal, PyTorch)

-> PyLops porting to GPUs allows seamless integration with ML frameworks (e.g. PyTorch) with no compromise on geophysical operators

https://github.com/PyLops/pylops

https://github.com/PyLops/pylops_eagehpc2021

Thank you for listening!

Deep imaging group.

King Abdullah University of Science and Technology