ML4Seismic Open Source Software environment

Mathias Louboutin et. al.

October 22, 2021

SLIM
Georgia Institute of Technology

ML4Seismic

Georgia Tech College of Computing
School of Computational Science and Engineering

Georgia Tech College of Sciences
School of Earth and Atmospheric Sciences

Georgia Tech College of Engineering
School of Electrical and Computer Engineering

Released to public domain under Creative Commons license type BY (https://creativecommons.org/licenses/by/4.0)
Copyright (c) 2021, Mathias Louboutin (Georgia Tech)

Some history

SINBAD 2006-2017, UBC

- inversion, data processing, acquisition design
- (parallel) Matlab code
- commercially adopted & in production
  (e.g. seismic data acquisition w/ Compressive Sensing)

Closed proprietary software framework stifled innovation due to

- restricted access to commercial (Matlab) code
- issues w/ scalability on parallel clusters
- limited uptake by sponsors

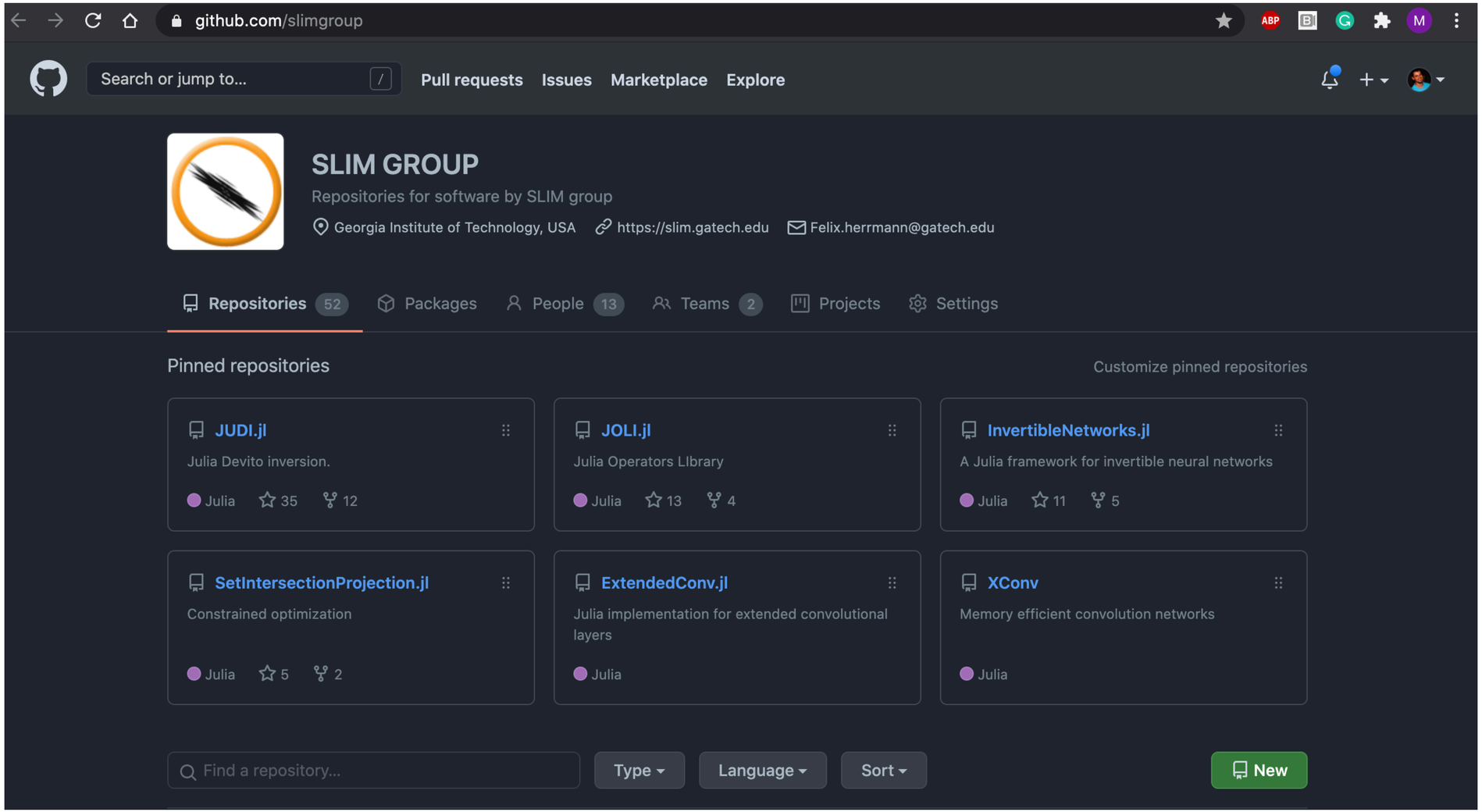
Moved to an open-source software (OSS) model in the Cloud.

ML4Seismic – Software

To improve our software stack, we

- improve access via Open Source (MIT license)
- build on years of experience
- leverage hyperscale serverless Cloud computing
- employs high-level abstractions to combat complexity
- are engaged in several (international) collaborations

Open source – https://github.com/slimgroup



Demos

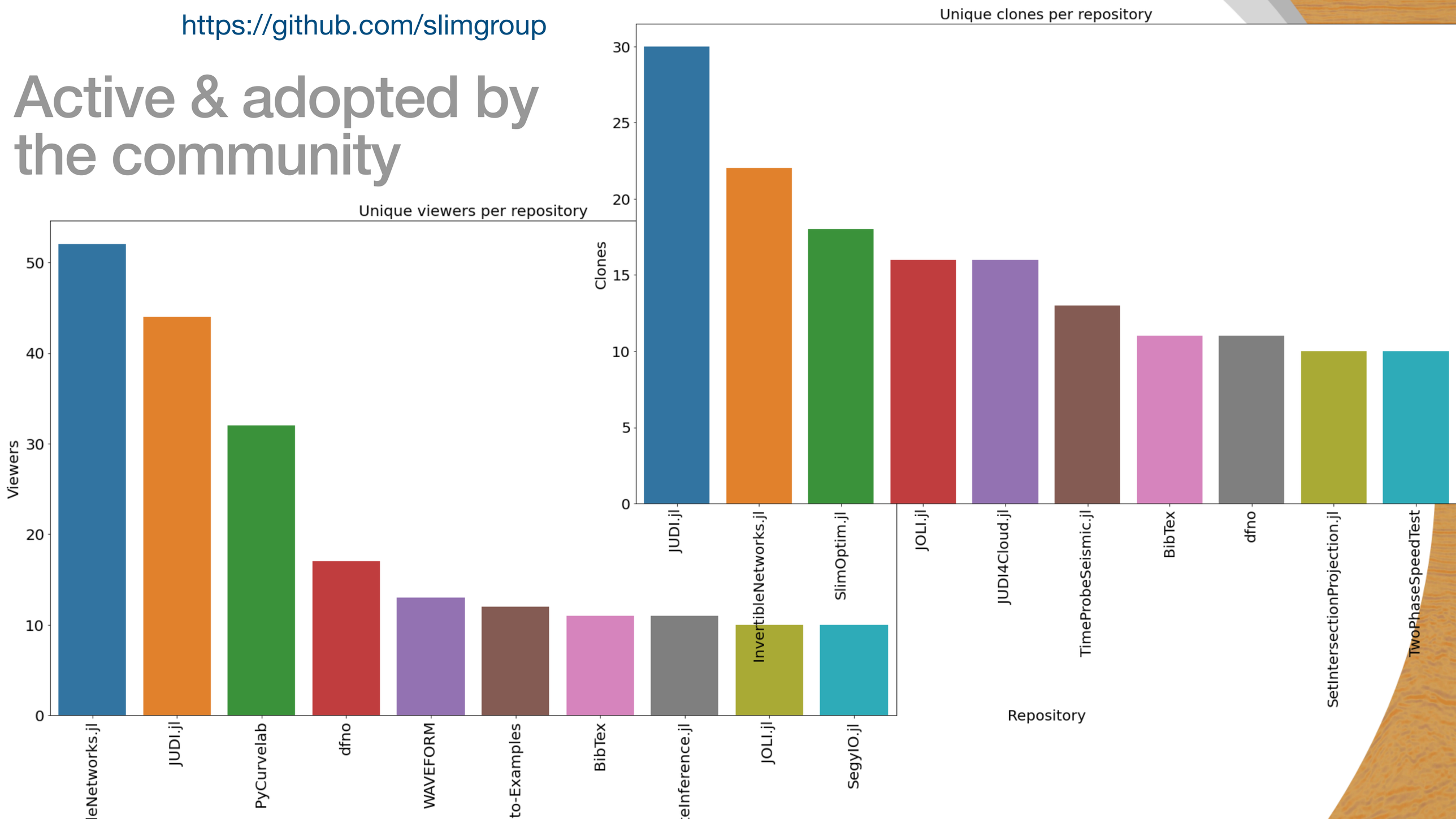
Afternoon breakout session:

- ML4Seismic software environment preinstalled
- showcase code development on simple problems
- allows for evaluation of software
- platform to interact w/ users

https://ml4shub.eastus.cloudapp.azure.com

https://github.com/slimgroup

Active & adopted by the community



[1] P. A. Witte, M. Louboutin, N. Kukreja, F. Luporini, M. Lange, G. G. Gorman and Felix J. Herrmann, 2019, *A large-scale framework for symbolic implementations of seismic inversion algorithms in Julia*, Geophysics, 84, 3.
[2] M. Louboutin, M. Lange, F. Luporini, N. Kukreja, P. A. Witte, F. J. Herrmann, P. Velesko and G. J. Gorman, 2019, Devito 3.1.0: an embedded domain-specific language for finite differences and geophysical exploration: Geoscientific model development, 12, 3.
[3] F. Luporini, M. Lange, M. Louboutin, N. Kukreja, J. Hückelheim, , C. Yount, P. A. Witte, P. H. J. Kelly, F. J. Herrmann and G. J. Gorman, 2019, Architecture and performance of Devito, a system for automated stencil computation, arXiv preprints.

High-level Abstractions for Seismic Inversion & ML

JUDI.jl & JUDI4Cloud.jl:[1]

- matrix-free linear operators and abstract data (SEGY) containers
- parallel I/O based on look-up tables, parallelism (OMP + soon MPI)
- interface to Julia's machine learning package Flux.jl
- *https://github.com/slimgroup/JUDI.jl/* (MIT license)

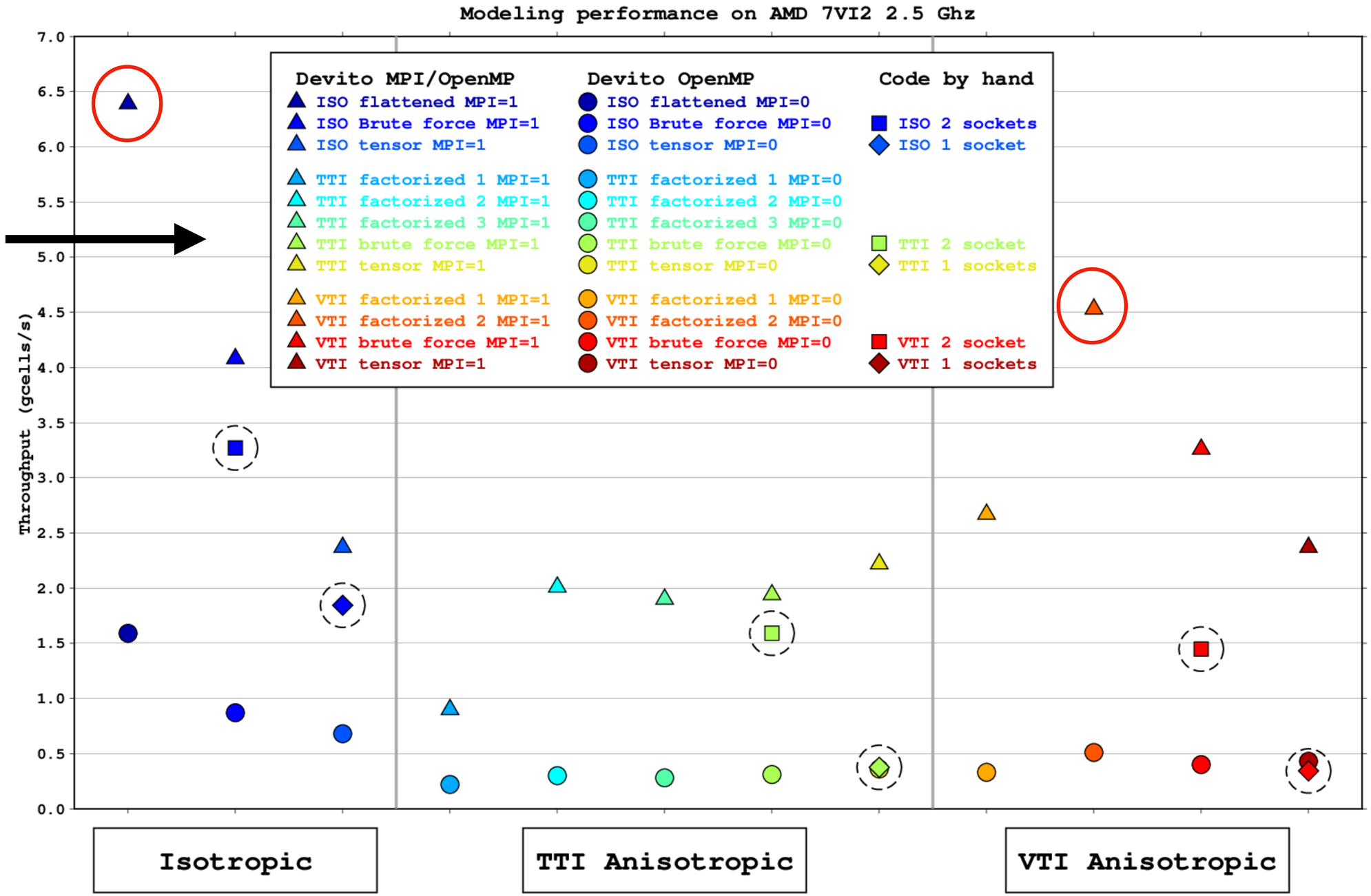
Devito*:[2][3] *Devito is a joint project between Imperial & Georgia Tech with Mathias one of the two main developers.

- a domain-specific language for finite-differences in Python
- code generation with automated performance optimizations
- model parallelism (MPI, multithreading)
- *https://github.com/devitocodes/devito* (MIT license)
- Adopted by industry (CVX, BP, DUG, …)

InvertibleNetworks.jl:

- building blocks for invertible neural networks/normalizing flows
- scalable & gradients derived by hand
- *https://github.com/slimgroup/InvertibleNetworks.jl* (MIT license)

State-of-the-art performance



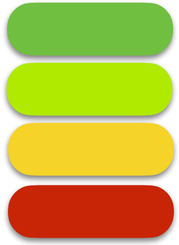
Devito – seismic

Louboutin, Mathias, et al. "Devito (v3. 1.0): an embedded domain-specific language for finite differences and geophysical exploration." *Geoscientific Model Development* 12.3 (2019): 1165-1187.
Lange, Michael, et al. "Devito: Towards a generic finite difference dsl using symbolic python." *2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*. IEEE, 2016.

GPU support roadmap

- Support for multiple target languages
  - OpenMP, OpenACC
  - potentially: CUDA, HIP, SYCL, …
- Unreliability of the target languages' software stack
- Multi-GPU support:
  - Make it possible to run different shots on different GPUs
  - Single-node multi-GPU via domain decomposition
  - Multi-node multi-GPU via domain decomposition
- Data movement (optimized)
- Data streaming (optimized)
- Kernel performance (best so far: 27 GPOINTS on iso-acoustic O(2, 8))

Legend:



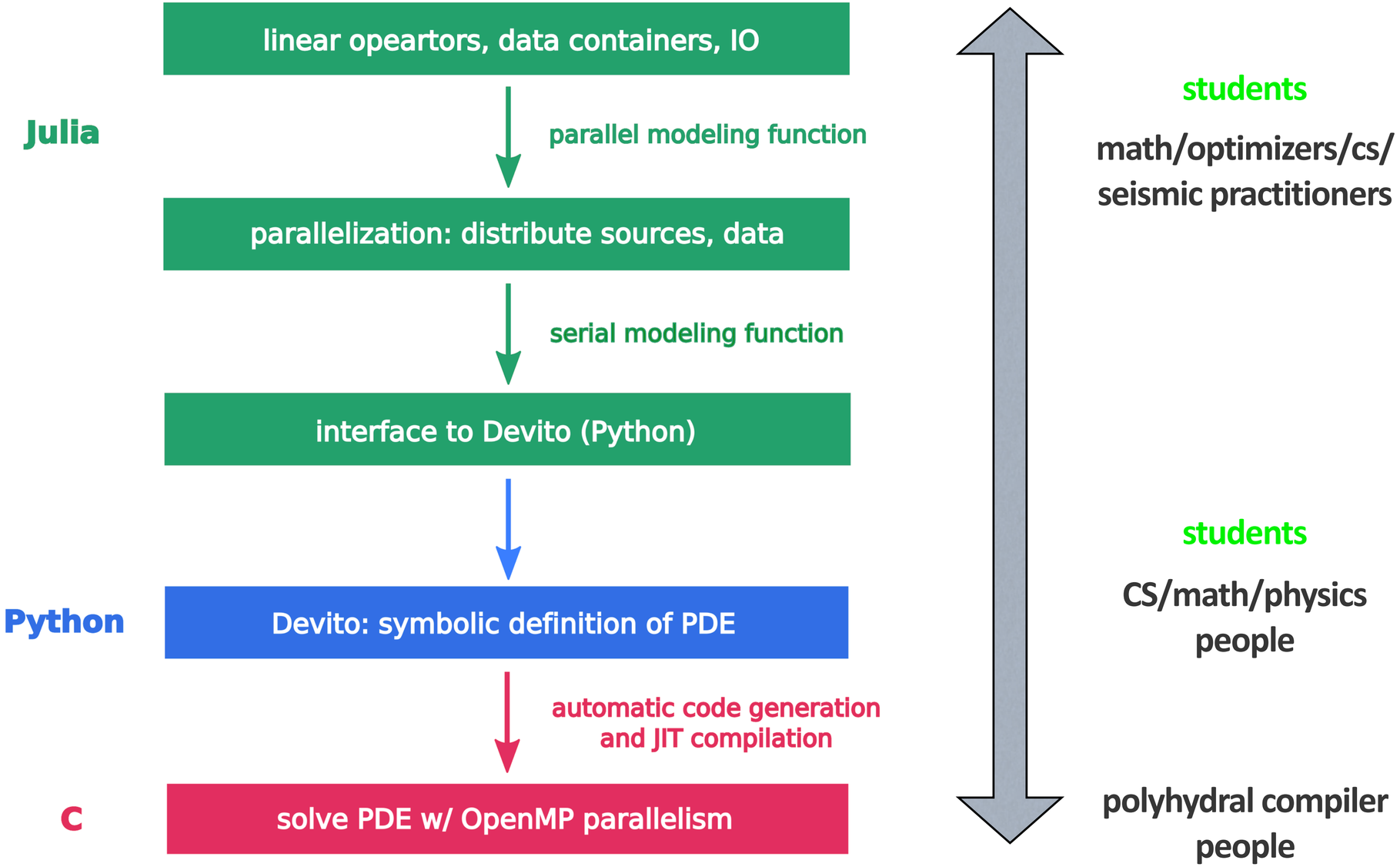
Done

Nearly done

In progress

Potentially later this year

JUDI – true vertical integration



[1] P. A. Witte, M. Louboutin, F. Luporini, G. G. Gorman and Felix J. Herrmann, 2019, *Compressive least-squares migration with on-the-fly Fourier transforms* , Geophysics, 84, 5.

Example : Compressive seismic imaging

Least-squares migration as an elastic net:[1]

$$\underset{\delta\mathbf{m}}{\text{minimize}} \quad \lambda \left\| \mathbf{C}\,\delta\mathbf{m} \right\|_1 + \frac{1}{2} \left\| \mathbf{C}\,\delta\mathbf{m} \right\|_2^2$$

$$\text{subject to:} \quad \sum_{i=1}^{n_s} \sum_{j=1}^{n_f} \left\| \mathbf{M}_l^{-1} \mathbf{J} \mathbf{M}_r^{-1}\,\delta\mathbf{m} - \mathbf{M}_l^{-1} \bar{\mathbf{d}}_{ij} \right\| \leq \sigma$$

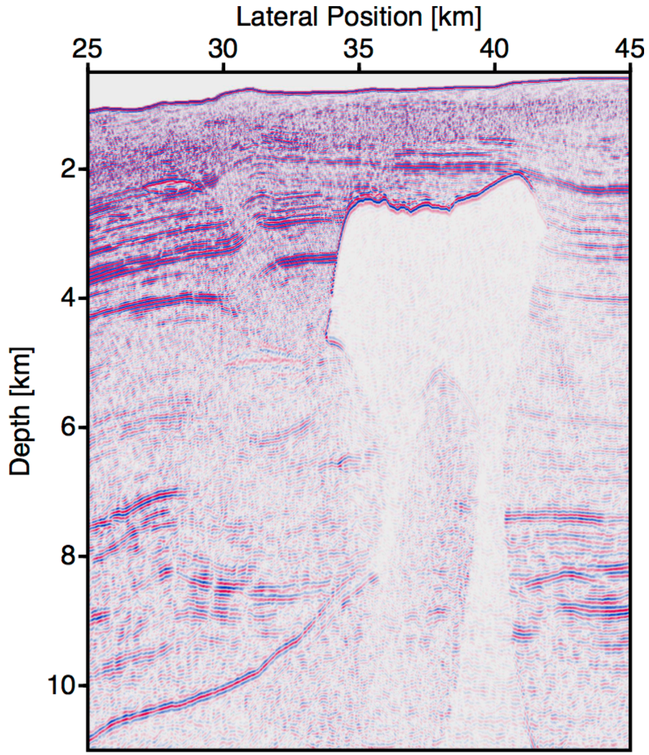
Solve via the linearized Bregman method:

- at each iteration: random subsets of sources + frequencies
- memory per source: $\mathcal{O}(\bar{n}_f)$
- compressive sensing: no. of samples $\sim$ no. of grid points, non-zero entries

Compressive seismic imaging

Linearized Bregman method with JUDI:

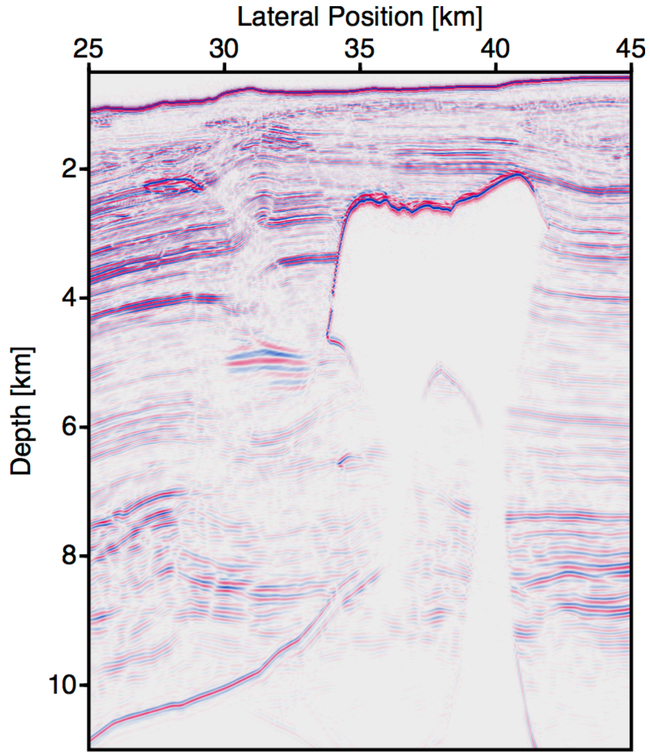
```
for j=1:maxiter

    # Compute residual and gradient
    i = randperm(d_obs.nsrc)[1:batchsize_source]
    select_frequencies!(J, batchsize_freq)
    r = Ml*J[i]*Mr*x - Ml*d_obs[i]
    g = Mr'*J[i]'*Ml'*proj_l2(r)

    # Residual and function value
    res[j] = norm(r, 2)
    fval[j] = λ*norm(C*z, 1) + .5f0*norm(C*z, 2)^2

    # Update variables
    global z -= α*g
    global x = C*soft_thresholding(C*z, λ)
end
```



Compressive seismic imaging

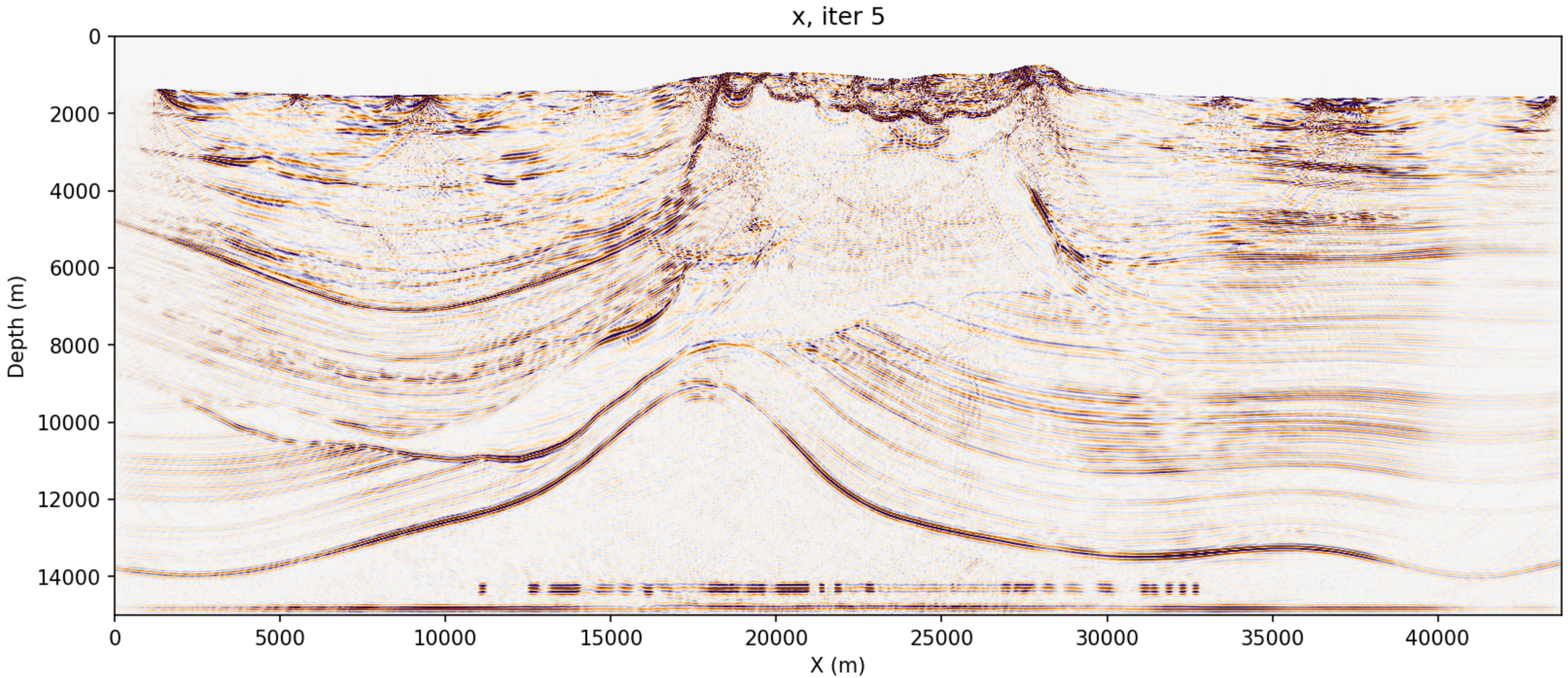
Linearized Bregman method with JUDI:

```
for j=1:maxiter

    # Compute residual and gradient
    i = randperm(d_obs.nsrc)[1:batchsize_source]
    select_frequencies!(J, batchsize_freq)
    r = Ml*J[i]*Mr*x - Ml*d_obs[i]
    g = Mr'*J[i]'*Ml'*proj_l2(r)

    # Residual and function value
    res[j] = norm(r, 2)
    fval[j] = λ*norm(C*z, 1) + .5f0*norm(C*z, 2)^2

    # Update variables
    global z -= α*g
    global x = C*soft_thresholding(C*z, λ)
end
```



Ultra-long offset SEG workshop

– 5 iterations 4 shots each



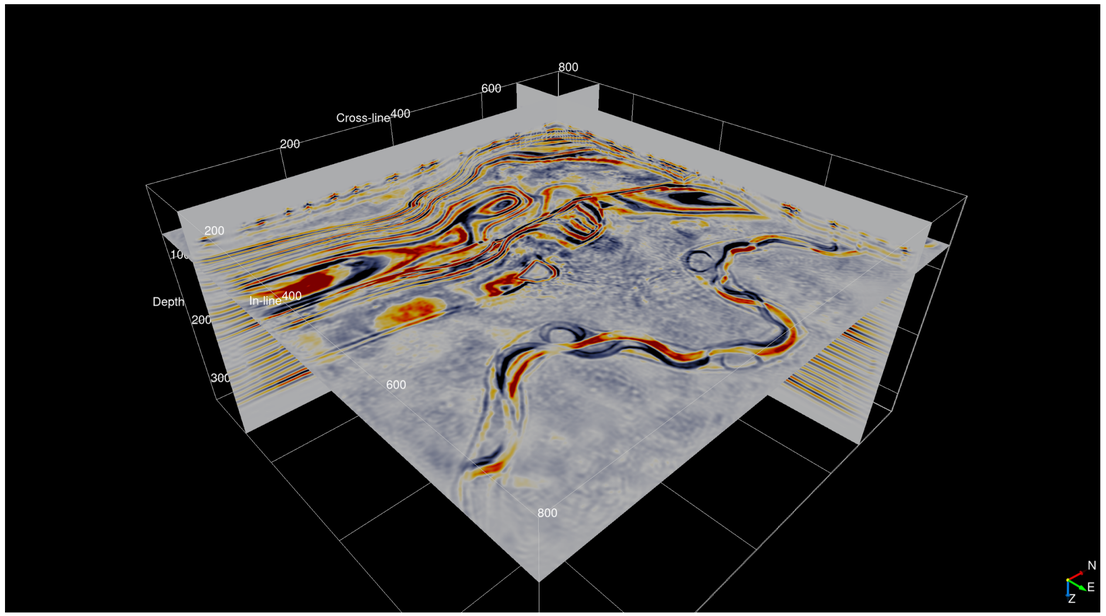
Witte, Philipp A., et al. "An event-driven approach to serverless seismic imaging in the cloud." *IEEE Transactions on Parallel and Distributed Systems* 31.9 (2020): 2032-2049.

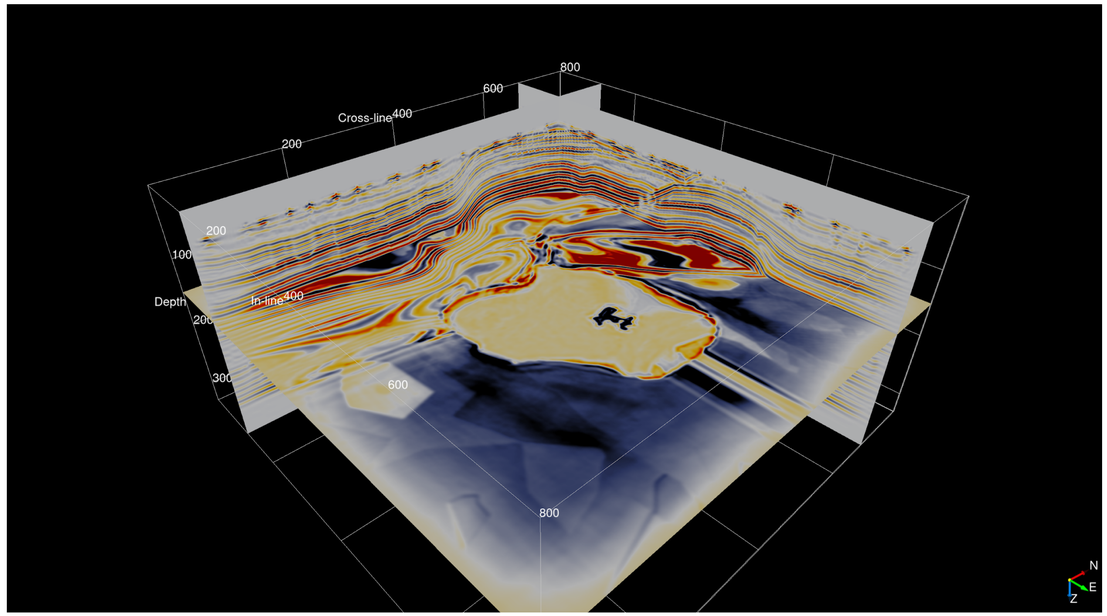
SLIM
ML4Seismic

Serverless seismic imaging on Azure

“Small” 3D Imaging case study w/ Devito DSL + JUDI

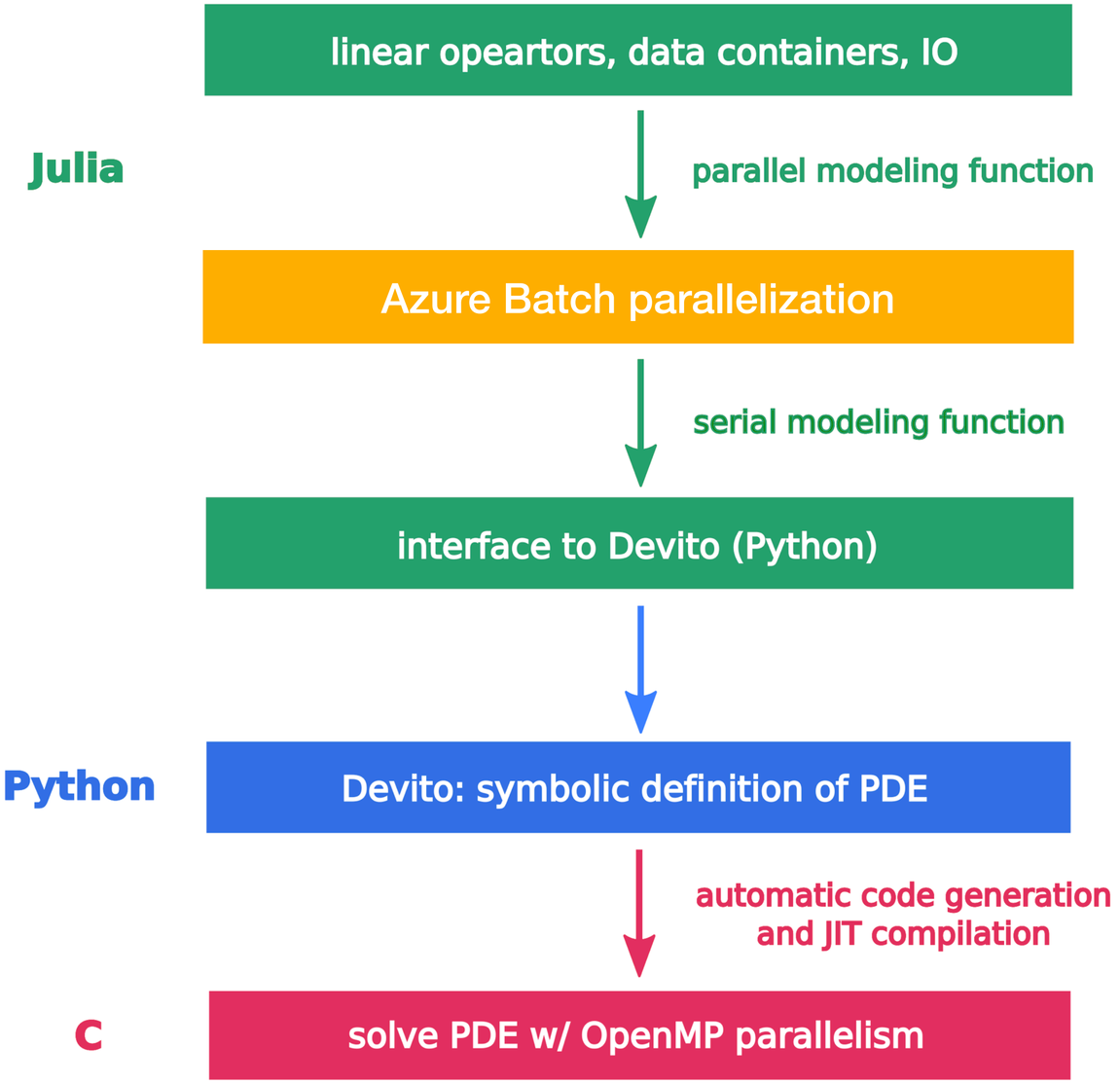
- Data set: 1,500 source locations (~**2.1 TB** data)
- Model: 10 x 10 x 3.325 km (**270 million** unknowns)
- PDE: tilted transversely isotropic (TTI) wave equation, **3,500** time steps
- **Cost: < 10,000$** on 100 E64/E64s instances (2 VMs per gradient with MPI)
- **Peak performance: 140 TFLOPs**
- **cost single image is comparable to training a large NN**

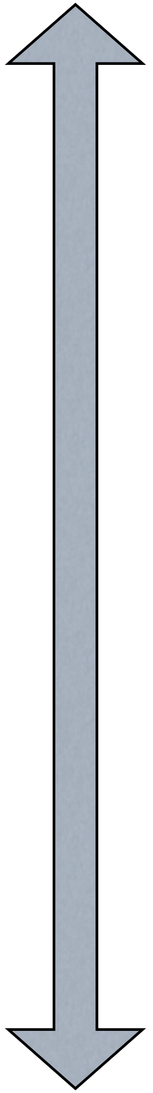




JUDI4Cloud

Simple swap-in





students

math/optimizers/cs/
seismic practitioners

students

CS/math/physics
people

polyhydral compiler
people

Clusterless implementation on Azure

JUDI operators in Azure Batch w/ JUDI4Cloud.jl

```
JUDI4Cloud.init_clusterless(n_instances; n_julia_per_instance=2)
# Setup operators
Pr = judiProjection(info, recGeometry)
F = judiModeling(info, model; options=opt)
F0 = judiModeling(info, model0; options=opt)
Ps = judiProjection(info, srcGeometry)
J = judiJacobian(Pr*F0*adjoint(Ps), q)

# Nonlinear modeling
dobs = Pr*F*adjoint(Ps)*q
```

← Setup Azure Batch (see previous talk)

← Run on Azure & gather results

Once again, abstractions pay off:

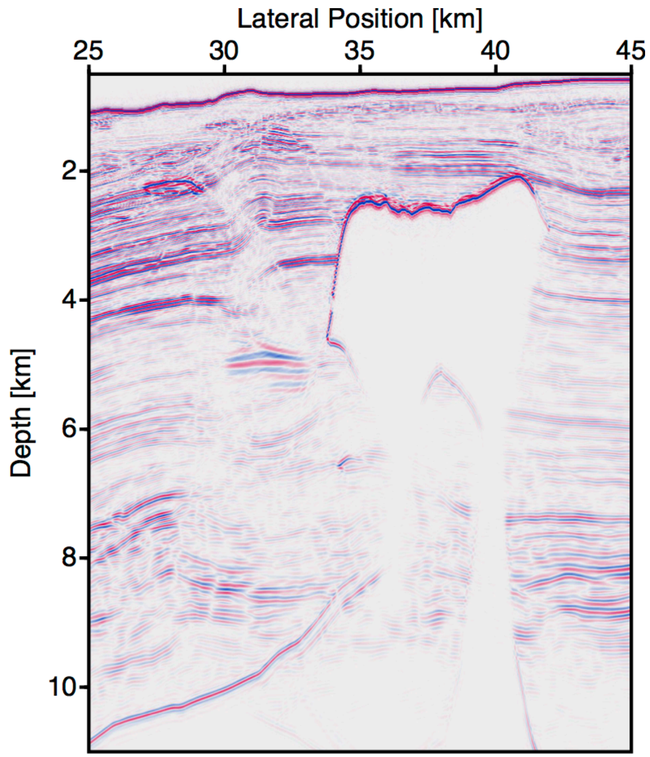
- no need to refactor code
- readable codes

Compressive seismic imaging

Linearized Bregman method with JUDI:

Now runs on Azure

```
for j=1:maxiter

    # Compute residual and gradient
    i = randperm(d_obs.nsrc)[1:batchsize_source]
    select_frequencies!(J, batchsize_freq)
    r = Ml*J[i]*Mr*x - Ml*d_obs[i]
    g = Mr'*J[i]'*Ml'*proj_l2(r)

    # Residual and function value
    res[j] = norm(r, 2)
    fval[j] = λ*norm(C*z, 1) + .5f0*norm(C*z, 2)^2

    # Update variables
    global z -= α*g
    global x = C*soft_thresholding(C*z, λ)
end
```



JUDI

the Julia Devito Inversion framework

Additional advantages:

- Devito propagators usable as standalone python
- SEGY I/O
- Extended source imaging (rank 1 source $\mathbf{q}[t]\mathbf{w}[\mathbf{x}]$)
- GPU offloading, simple ENV-variable change
- Low-memory gradients with randomized trace estimation (See Tue)

Future work: expose in JUDI

- Devito's MPI domain decomposition
- AD support

Designed for Interoperability

Combine JOLI matrix-free linear operators w/ COFII's Jets

```
using JOLI, JetPack, Jets
# 1D real DFT with 1000 samples
J = joDFT(1000; DDT=Float64, RDT=Float64)
# Jet operator for circular shift by 25 samples
jop = JopCircShift(JetSpace(Float64, 1000), 25);

a = randn(1000)
# Parenthesis needed to prevent julia to compute J*jop first
a = randn(1000);b = J*(jop*a); c = jop'*(J'*b);
# dot test
@show dot(b, b),    dot(c, a),  dot(b, b) - dot(c, a), dot(b, b) / dot(c, a)
# 490.00330785, 490.003307859, 1.1368683772161603e-13, 1.0000000000000002)
```

Once again, abstractions pay off:

- no need to refactor code
- readable codes

Seamless integration JUDI & COFII $\Longrightarrow$ super fast developments cycle = innovation.

https://github.com/slimgroup/ConstrainedFWIExamples/blob/master/notebooks/02_constr_fwi_jetpack.ipynb
https://github.com/ChevronETC/Examples/blob/main/50_fwi/11_constrained_fwi_pqn.ipynb

Mored advanced integration

synergetic

COFII propagator

SetIntersectionProjection.jl
Projection onto intersection of constraints

SlimOptim.jl
quasi-newton

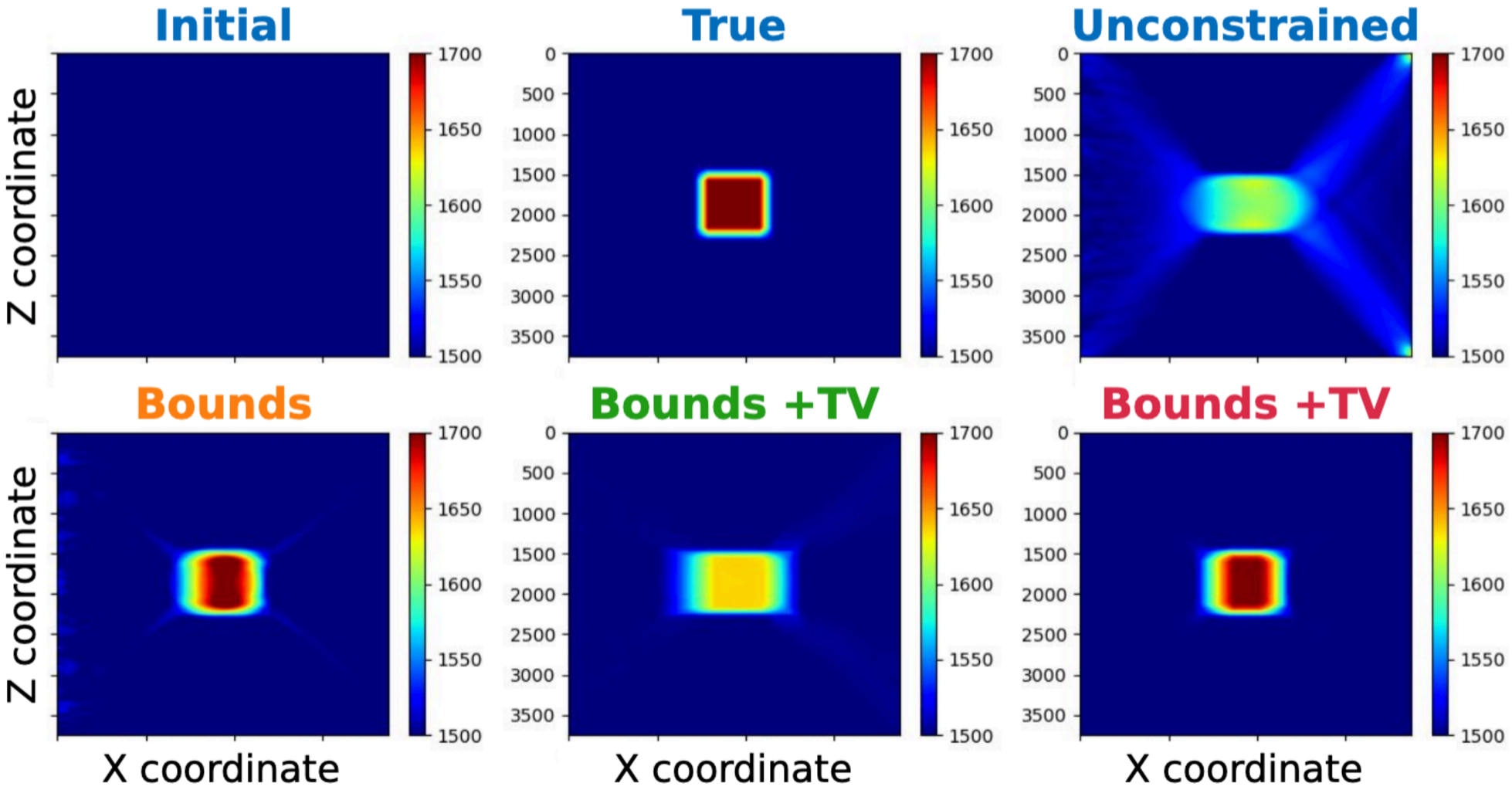
```
sol = pqn(g, vec(v2), prj, options_pqn);
```

```
Running PQN...
Number of L-BFGS Corrections to store: 10
Spectral initialization of SPG: 1
Maximum number of SPG iterations: 10
SPG optimality tolerance: 1.00e-06
SPG progress tolerance: 1.00e-07
PQN optimality tolerance: 1.00e-05
PQN progress tolerance: 0.00e+00
Quadratic initialization of line search: 0
Maximum number of iterations: 25
input to PARSDMM is feasible, returning
p.gscale = 163.19803f0
 Iteration   FunEvals  GradEvals Projections     Step Length    Function Val        Opt Cond
input to PARSDMM is feasible, returning
         0          0          0          0     0.00000e+00     5.47293e+06     1.00000e+01
input to PARSDMM is feasible, returning
input to PARSDMM is feasible, returning
input to PARSDMM is feasible, returning
input to PARSDMM is feasible, returning
         1          3          3          6     1.00000e+00     5.28328e+06     9.59644e+00
```

https://github.com/ChevronETC/Examples/blob/main/50_fwi/11_constrained_fwi_pqn.ipynb

Mored advanced integration

constrains from SLIM & wave propagators from CVX's COFII



COFII vs SLIM

	SLIM	COFII	Compatible
Language	julia, python	julia, C++	yes
Software philosophy	Linear Algebra	Jets	yes (see above)
Propagators	Devito	Legacy C++, potential migration to Devito	NA
IO	SEGY (via SegyIO), no cloud support yet	JavaSeis CloudSeis	not yet
Cloud	Azure Batch (via AzureClusterlessHPC) fully remote (local desktop/laptop)	Azure Virtual Machine ScaleSet Custom cloud cluster manager Azure Network only	no
Cloud storage	OpenVDS (future plan) Cloud native OSDU compliant	Blob manager	yes
OSS	Yes, all	Yes, transitioning	Yes

ML Software

***InvertibleNetworks.jl*: A Julia package for invertible CNNs**

- Manually implemented derivatives and Jacobians that make use of invertibility
- Unit testing (adjoints, gradients, invertibility)
- *https://github.com/slimgroup/InvertibleNetworks.jl* (MIT license)
- GPU support through CUDA.jl

Main features:

- Additive and affine invertible coupling layers (e.g. as in NICE, Glow)
- Invertible recurrent inference machines (i-RIM)
- Hyperbolic networks
- Hierarchical invertible neural transport (HINT) and conditional HINT
- 1 x 1 convolutions, wavelet squeezing, etc.
- Integration of Flux.jl layers
- AD with ChainRules.jl

InvertibleNetworks.jl

Class structure:

```
struct CouplingLayerGlow <: NeuralNetLayer
    C::Conv1x1
    RB::ResidualBlock
    logdet::Bool
    forward::Function
    inverse::Function
    backward::Function
end
```

Usage:

```
# Create Layer
LayerGlow = CouplingLayerGlow(nx, ny, nchannel, nhidden, batchsize)

# Operations
Y = LayerGlow.forward(X)
X_ = LayerGlow.inverse(Y)
ΔX, X_ = LayerGlow.backward(ΔY, Y)
```

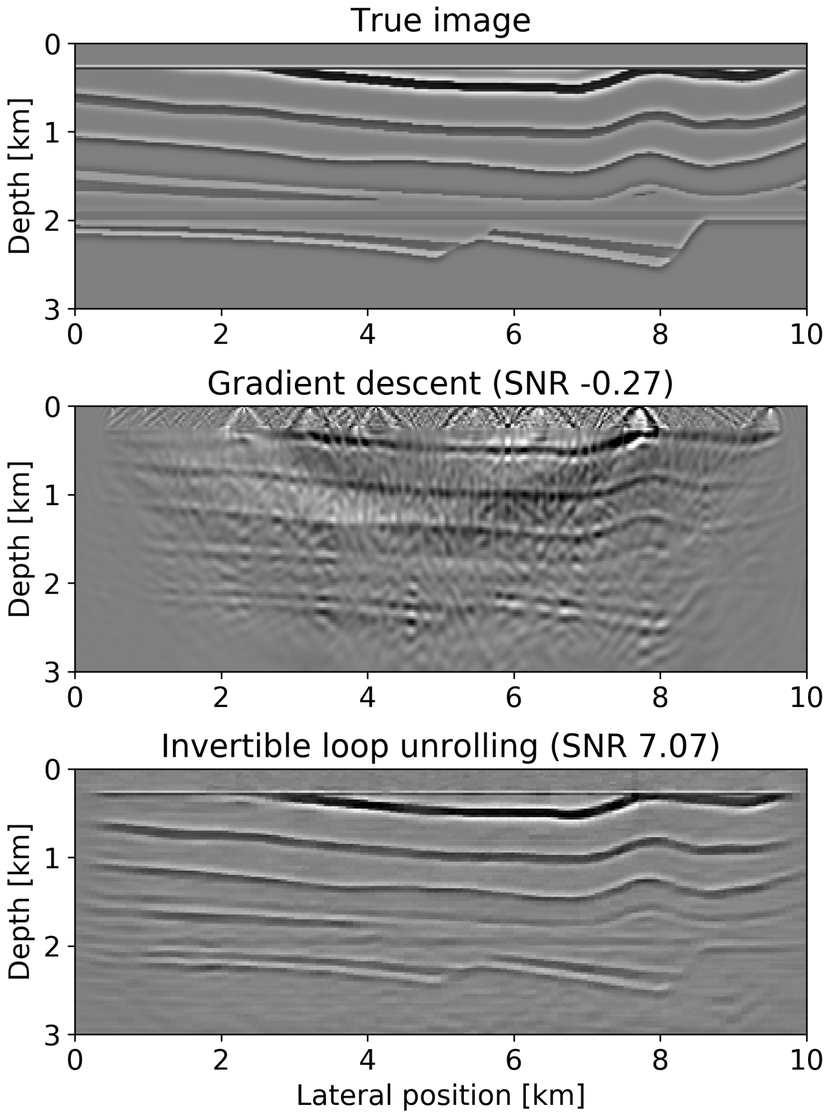
Loop unrolling revisited

Invertible version of loop-unrolled network:

- Inspired by i-RIM and Real NVP
- Invertible + tractable logdet

Migration-demigration

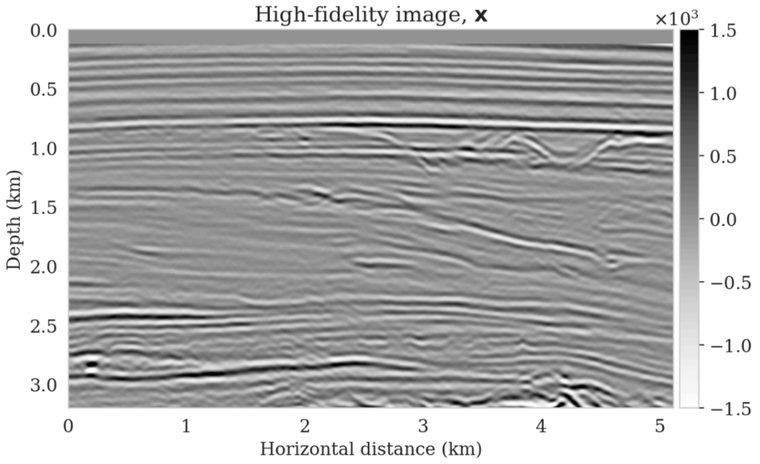
1. **function** $\mathcal{G}(\mathbf{J}, \mathbf{d})$
2. $\quad \mathbf{x} = 0$
3. $\quad$ **for** $j = 1, ..., n$
4. $\quad\quad \mathbf{x} = \mathbf{Q}\mathbf{x}$
5. $\quad\quad \mathbf{x}'_1 = \mathbf{x}_1$
4. $\quad\quad \mathbf{g} = \mathbf{J}^\top\left(\mathbf{J}\mathbf{x}'_1[1] - \mathbf{d}\right)$
5. $\quad\quad \mathbf{s}', \mathbf{t} = \mathrm{NN}\left([\mathbf{g}, \mathbf{x}'_1[2{:}\mathrm{end}]]\right)$
6. $\quad\quad \mathbf{s} = \sigma\left(\mathbf{s}'\right)$
6. $\quad\quad \mathbf{x}'_2 = \mathbf{x}_2 \odot \mathbf{s} + \mathbf{t}$
3. $\quad\quad \mathbf{x} = \mathbf{Q}^\top\mathbf{x}'$
12. $\quad$ **end**
13. $\quad$ return $\mathbf{x}$
14. **end**

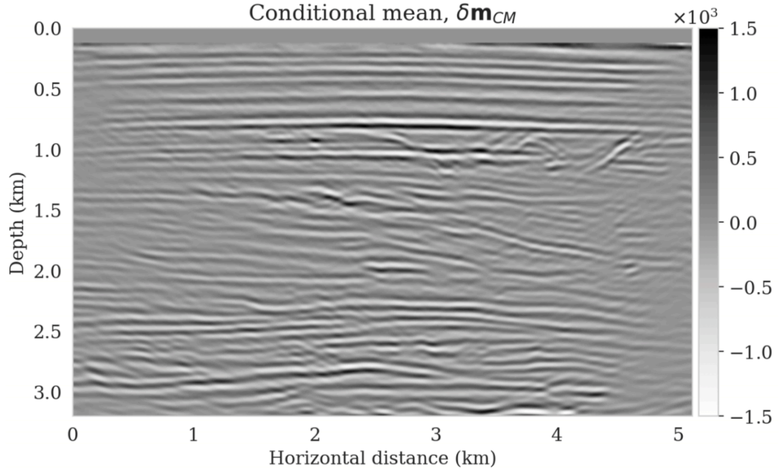


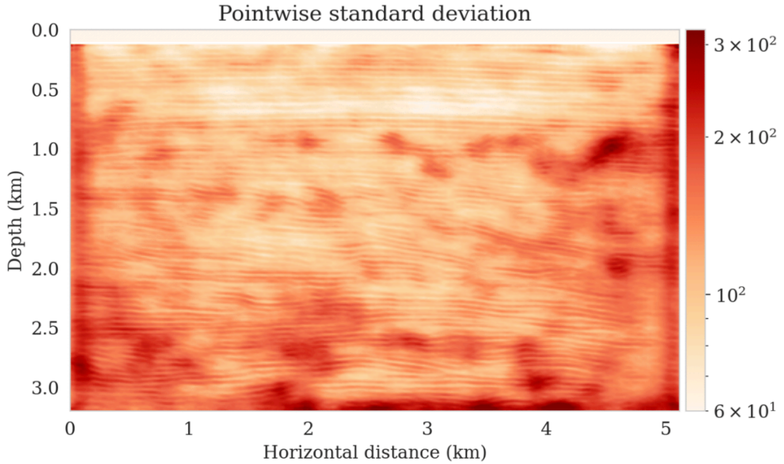
Uncertainty Quantification

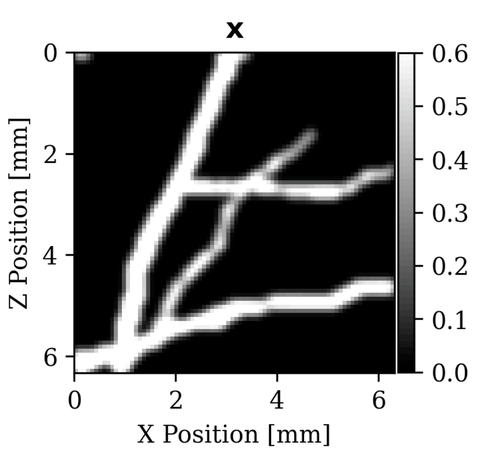
SLIM
ML4Seismic

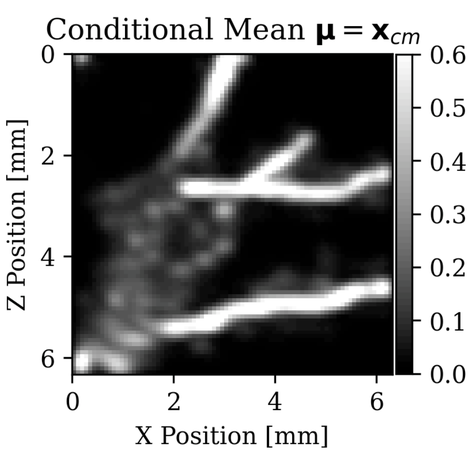
Examples:

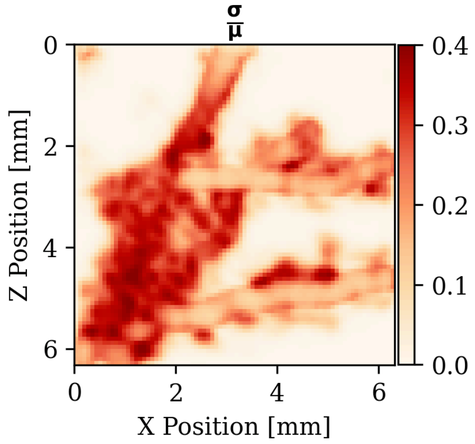


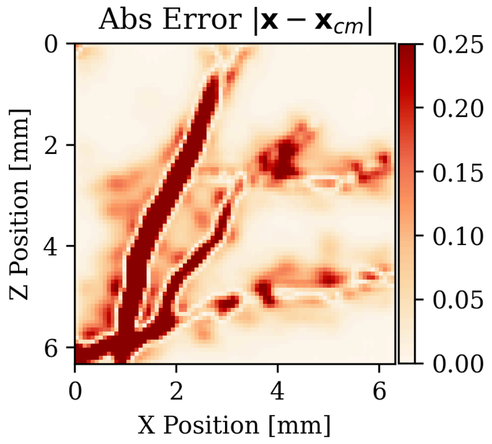












Siahkoohi, Ali, and Felix J. Herrmann. "Learning by example: fast reliability-aware seismic imaging with normalizing flows." *arXiv preprint arXiv:2104.06255* (2021).

Orozco, Rafael, et al. "Photoacoustic imaging with conditional priors from normalizing flows." *NeurIPS 2021 Workshop on Deep Learning and Inverse Problems*. 2021.

Uncertainty Quantification

Posterior sampling implemented with conditional layers in InvertibleNetworks.jl

```
struct ConditionalLayerHINT <: NeuralNetLayer
    CL_X::CouplingLayerHINT
    CL_Y::CouplingLayerHINT
    CL_YX::CouplingLayerBasic
    C_X::Union{Conv1x1, Nothing}
    C_Y::Union{Conv1x1, Nothing}
    logdet::Bool
    is_reversed::Bool
end
```

First, train with joint distribution target then sample from posterior distribution:

```
Zy_fixed = H.forward_Y(y_obs);
post_sampling_size = 50
Zx = randn(Float32, nx, ny, n_in, post_sampling_size)
X_post = H.inverse(Zx, Zy_fixed.*ones(Float32, nx, ny, n_in, post_sampling_size))[1];

#statistics over pixels in batch
mean(X_post; dims=4)
std(X_post; dims=4)
```

Kruse, Jakob, et al. "HINT: Hierarchical invertible neural transport for density estimation and Bayesian inference." *arXiv preprint arXiv:1905.10687* (2019)

Next steps

Scaling of ML in 3D

Integration w/ OLIVES & generative classifiers

Further integration AD tools to combine wave-equation solvers w/ ML

Integration of Julia's new AD tool when available

- leverages JIT compiler
- offers more control → scales well

This research was carried out with the support of Georgia Research Alliance and partners of the ML4Seismic consortium.