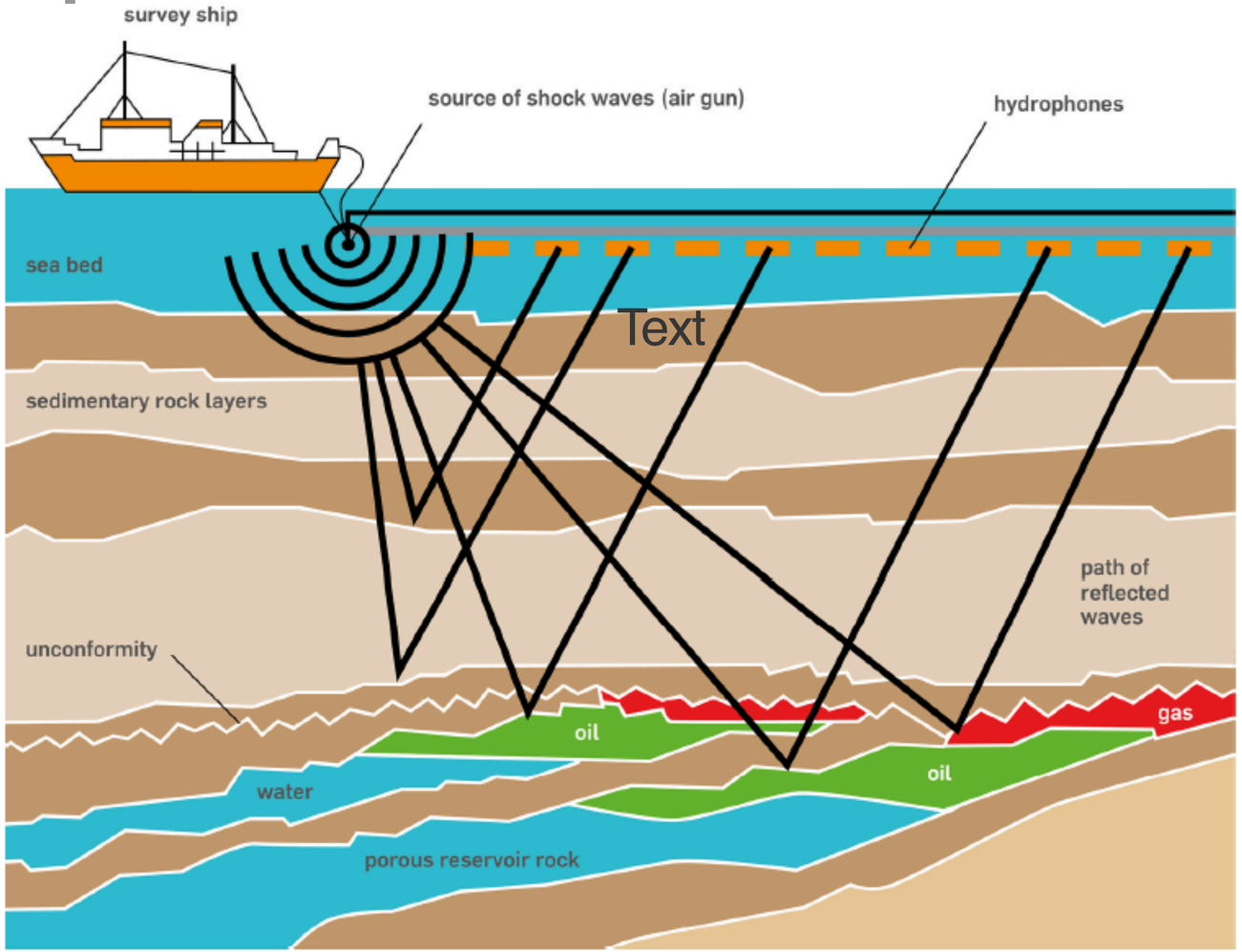
Domain-specific abstractions for large-scale geophysical inverse problems: Part I

Mathias Louboutin, Fabio Luporini, Navjot Kurjeka, Philipp Witte, Felix Herrma
and Gerard Gorman

SLIM
University of British Columbia

Wave-equation based geophysical exploration

Physical problem



http://www.open.edu/openlearn/science-maths-technology/science/environmental-science/earths-physical-resources-petroleum/content-section-3.2.1

Mathematical problem

$$\underset{\mathbf{m}}{\text{minimize}} \quad \frac{1}{2}|| \mathbf{P}_r \mathbf{A}^{-1}(\mathbf{m}) \mathbf{P}_s^T \mathbf{q} - \mathbf{d}||_2^2$$

(Virieux and Operto, 2009)

$\mathbf{m}$: squared slowness

$\mathbf{d}$: field recorded data

$\mathbf{A}(\mathbf{m})$: discretized wave-equation

$\mathbf{q}$: source term

$\mathbf{P}_r$: projection onto the receivers locations

$\mathbf{P}_s$: projection onto the source location

Challenges

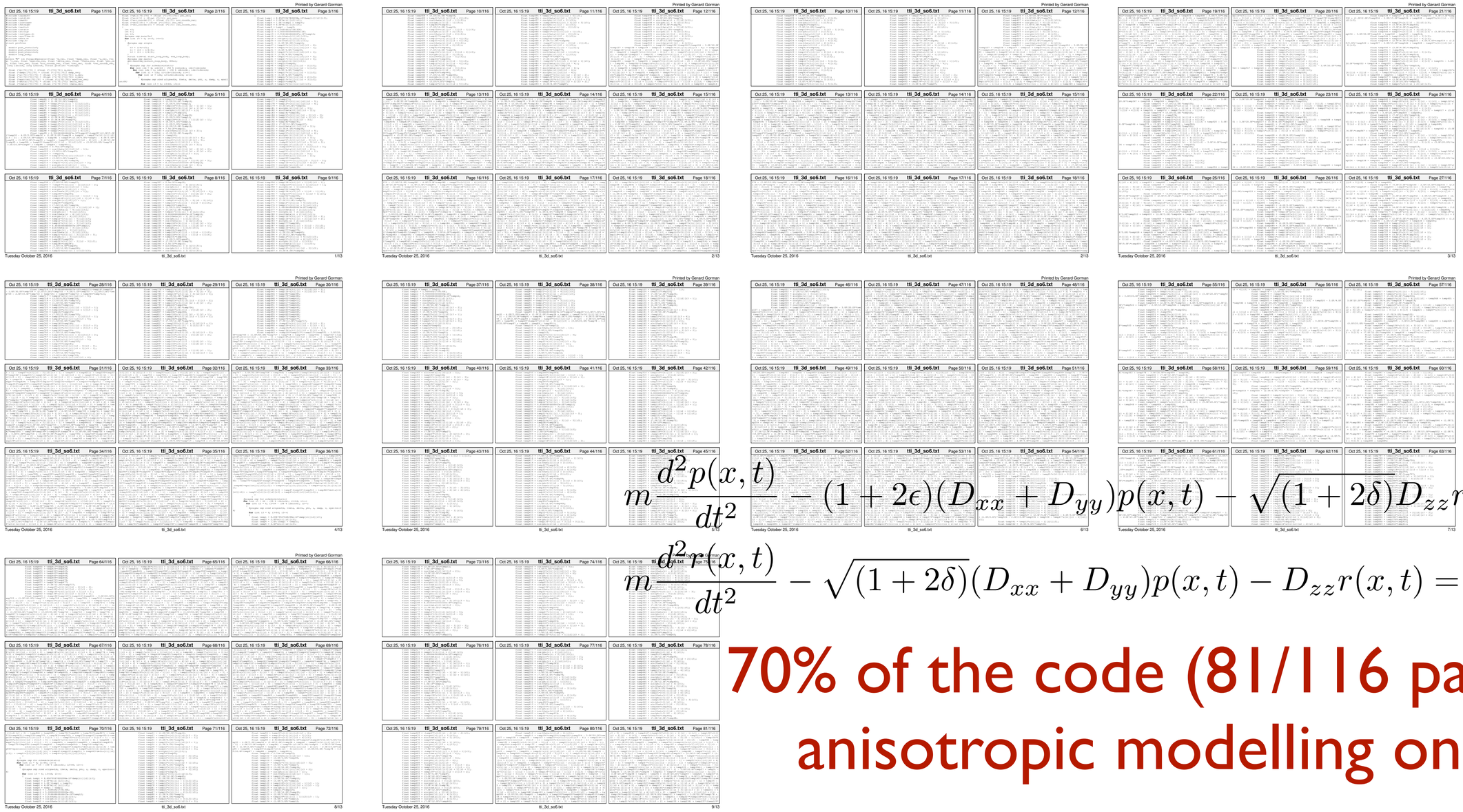
- Problem sizes are huge:
  - Seismic surveys consist of tens of thousands of individual experiments
  - Model wave propagation over thousands of time steps in large domains
  - Typical size of modeling matrix → $\mathbf{A}(\mathbf{m}) \in \mathbb{R}^{n \times n}, n = \mathcal{O}(1e13)$
- Multiple/complex representations of the physics
- Simulation for inversion
  - Adjoints, gradients
- Complex simulation codes
- Needs scalable, flexible, performant and portable discretization

Raising the level of abstraction

$$m\frac{\partial^2 u}{\partial t^2} + \eta\frac{\partial u}{\partial t} - \Delta u = 0$$

```
void kernel(…) {
  …
  <impenetrable code with crazy
   performance optimizations>
  …
}
```

Complexity of the code

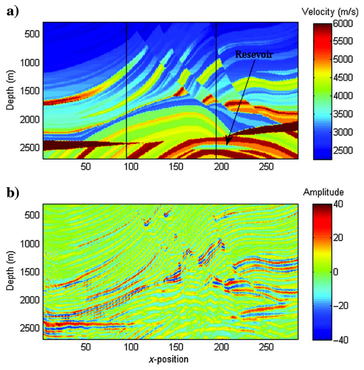


$$m\frac{d^2p(x,t)}{dt^2} - (1+2\epsilon)(D_{xx}+D_{yy})p(x,t) - \sqrt{(1+2\delta)}D_{zz}r(x,t) = q,$$

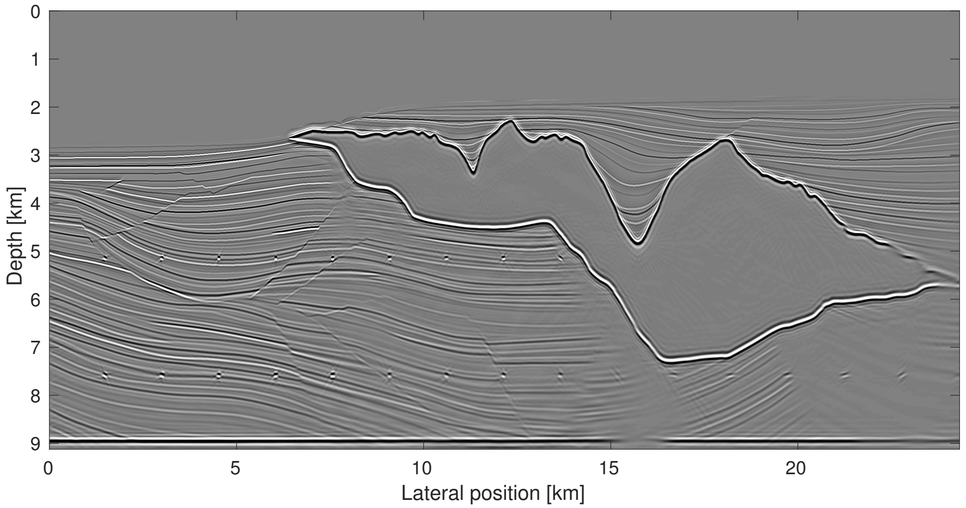
$$m\frac{d^2r(x,t)}{dt^2} - \sqrt{(1+2\delta)}(D_{xx}+D_{yy})p(x,t) - D_{zz}r(x,t) = q,$$

70% of the code (81/116 pages)

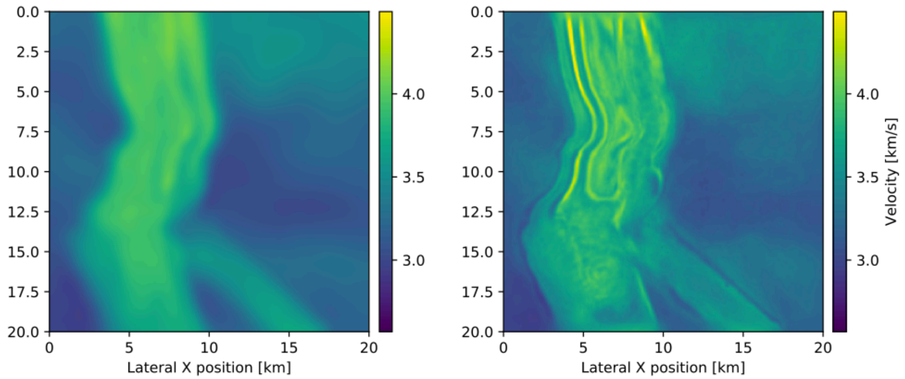
anisotropic modelling only



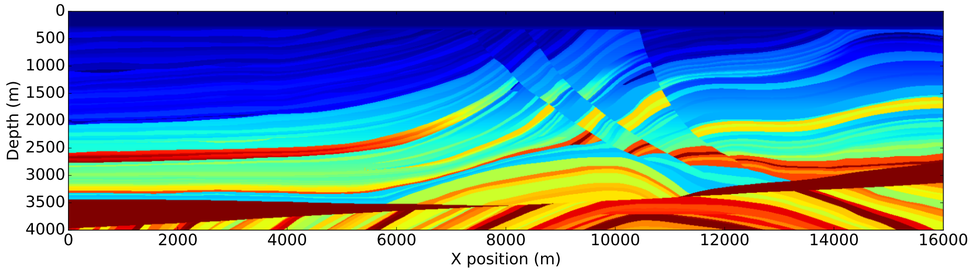
Tiny marmousi
62k gridpoints
1.5MFlop/time-step



Sigsbee
2.2M gridpoints
56MFlop/time-step

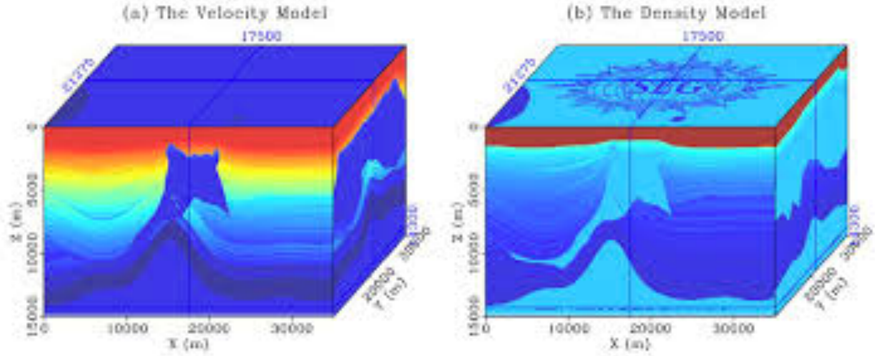


3D overthrust
222M gridpoints
6GFlop/time-step



Full marmousi
640k gridpoints
15MFlop/time-step

Patiently wait for result



SEAM
2.2G gridpoints
56GFlop/time-step

Design motivation

- Stencil codes:
  - Time consuming
  - Complex
  - Architecture dependent

F. Luporini, M. Lange, M. Louboutin, N. Kukreja, J. Hückelheim, C. Yount, P. Witte, P. H. J. Kelly, G. J. Gorman, and F. J. Herrmann.
Architecture and performance of Devito, a system for automated stencil computation.

Proposed solution

DEVITO – Domain specific language for stencil-based finite difference code generation for PDEs w/ explicit time stepping in Python using SymPy.

https://www.devitoproject.org

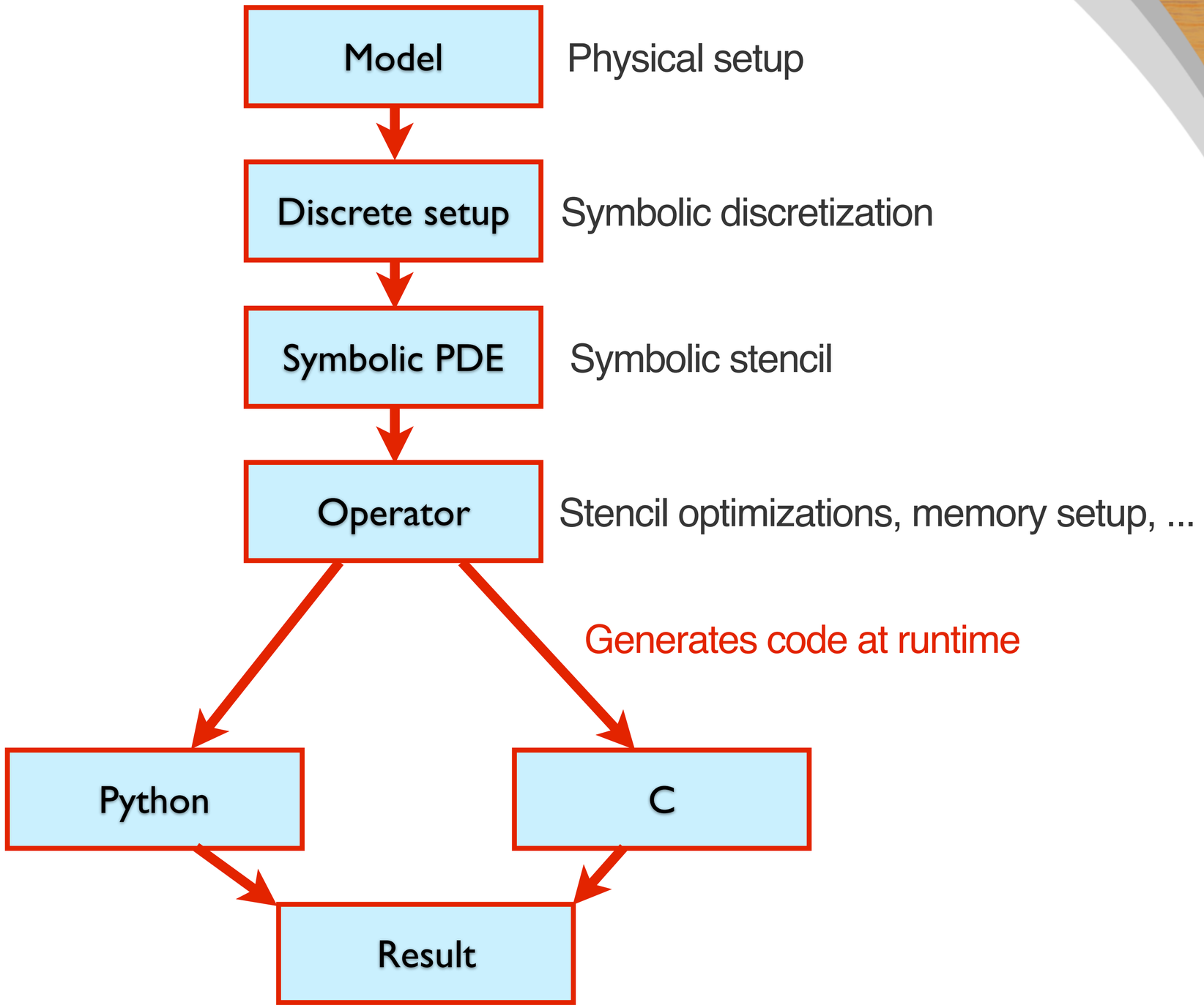
Finite-difference DSL

- Separation of Concerns:
  - Geophysicists focus on physics
  - Computer scientists focus on software
  - Mathematicians focus on numerical analysis

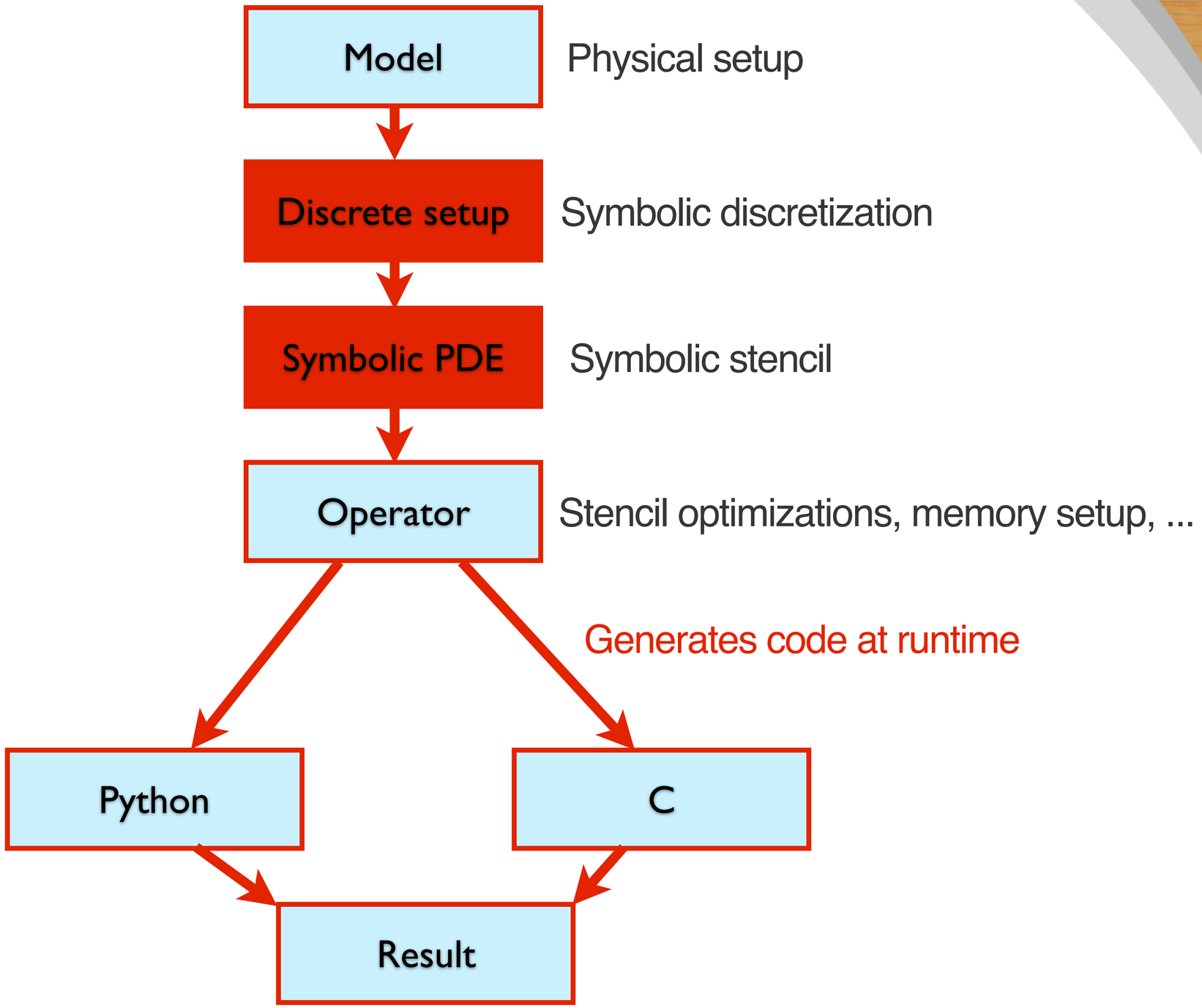
Devito

Michael Lange, Navjot Kukreja, Mathias Louboutin, Fabio Luporini, Felippe Vieira Zacarias, Vincenzo Pandolfo, Paulius Velesko, Paulius Kazakas, and Gerard Gorman,

"Devito: Towards a generic finite difference DSL using symbolic python", in 6th Workshop on Python for High-Performance and Scientific Computing, 2016, p. 67-75.



Devito


Model
Physical setup
Discrete setup
Symbolic discretization
Symbolic PDE
Symbolic stencil
Operator
Stencil optimizations, memory setup, ...
Generates code at runtime
Python
C
Result

Devito for simulation

Symbolic mathematics for wave-equation based exploration geophysics

Scalar acoustic wave-equations

$$\frac{1}{c^2}\frac{d^2p(x,t)}{dt^2} - \Delta p(x,t) = 0$$

Acoustic isotropic

$$\frac{1}{\rho c^2}\frac{d^2p(x,t)}{dt^2} - \nabla.(\frac{1}{\rho}\nabla p(x,t)) = 0$$

Acoustic isotropic with density

$$m\frac{d^2p(x,t)}{dt^2} - (1+2\epsilon)(G_{\bar{x}\bar{x}} + G_{\bar{y}\bar{y}})p(x,t) - \sqrt{(1+2\delta)}G_{\bar{z}\bar{z}}r(x,t) = q,$$

$$m\frac{d^2r(x,t)}{dt^2} - \sqrt{(1+2\delta)}(G_{\bar{x}\bar{x}} + G_{\bar{y}\bar{y}})p(x,t) - G_{\bar{z}\bar{z}}r(x,t) = q,$$

Acoustic anisotropic
Vertical transverse isotropic (VTI)

$$m\frac{d^2p(x,t)}{dt^2} - (1+2\epsilon)(D_{xx} + D_{yy})p(x,t) - \sqrt{(1+2\delta)}D_{zz}r(x,t) = q,$$

$$m\frac{d^2r(x,t)}{dt^2} - \sqrt{(1+2\delta)}(D_{xx} + D_{yy})p(x,t) - D_{zz}r(x,t) = q,$$

Acoustic anisotropic
Tilted transverse isotropic (TTI)

Symbolic discretization

Symbolic object with finite-difference discretization attributes

```
u =TimeFunction(name="u", grid=grid,
                time_order=self.t_order,
                space_order=self.s_order)
```

is a symbolic object with derivatives

```
u.dx, u.dy, u.dz, u.dx2, ..., u.laplace,
```

```
In[69]: u.dx
Out[69]:  -u(t - s, x - 3*h, y, z)/(60*h) + 3*u(t - s, x - 2*h, y, z)/(20*h) - 3*u(t - s, x - h, y, z)/(4*h) +
 3*u(t - s, x + h, y, z)/(4*h) - 3*u(t - s, x + 2*h, y, z)/(20*h) + u(t - s, x + 3*h, y, z)/(60*h)
```

Symbolic wave-equations

Acoustic

$$0 \quad = \quad m\frac{d^2u(x,t)}{dt^2} \quad - \quad \Delta u(x,t) \quad + \ \mathrm{damp}\frac{du(x,t)}{dt}$$

```
eqn = m * u.dt2 - u.laplace + damp * u.dt
```

Acoustic 4th order in time

$$0 \quad = \quad m\frac{d^2u(x,t)}{dt^2} \quad - \quad \Delta u(x,t) \quad - \quad \frac{dt^2}{12}\Delta\left(\frac{1}{m}\Delta u(x,t)\right) \quad + \ \mathrm{damp}\frac{du(x,t)}{dt}$$

```
eqn = m * u.dt2 - u.laplace - s**2/12 * u.laplace2(1/m)+ damp * u.dt
```

Worked example

Acoustic modelling

Wave-equation setup

$$\mathbf{m}(\mathbf{x})\frac{\mathbf{u}(\mathbf{x},\mathbf{t}+\Delta t)-2\mathbf{u}(\mathbf{x},t)+\mathbf{x},\mathbf{t}-\Delta t)}{\Delta t^2}-\Delta\mathbf{u}(\mathbf{x},\mathbf{t})=0$$

```
equation = m * u.dt2 - u.laplace + damp * u.dt
```

Absorbing boundary condition

$\mathbf{u}$: discretized wavefield

$\mathbf{m}$: discretized square slowness

Δ : discretized Laplacian

```
u = TimeFunction(name="u",grid=model.grid,
                 time_dim=nt,time_order=time_order,
                 space_order=spc_order,
                 save=save)

m = Function(name="m", grid=model.grid)


Lap = u.laplace
```

Stencil

$$\mathbf{u}(\mathbf{x}, \mathbf{t} + \Delta\mathbf{t}) = \frac{\Delta\mathbf{t}^2}{\mathbf{m}(\mathbf{x})}(2\mathbf{u}(\mathbf{x}, \mathbf{t}) - \mathbf{u}(\mathbf{x}, \mathbf{t} - \Delta\mathbf{t}) + \Delta\mathbf{u}(\mathbf{x}, \mathbf{t}))$$

`u.forward` = `solve(equation, u.forward)`

```
stencil = Eq(u.forward, solve(equation, u.forward))
```

Forward operator

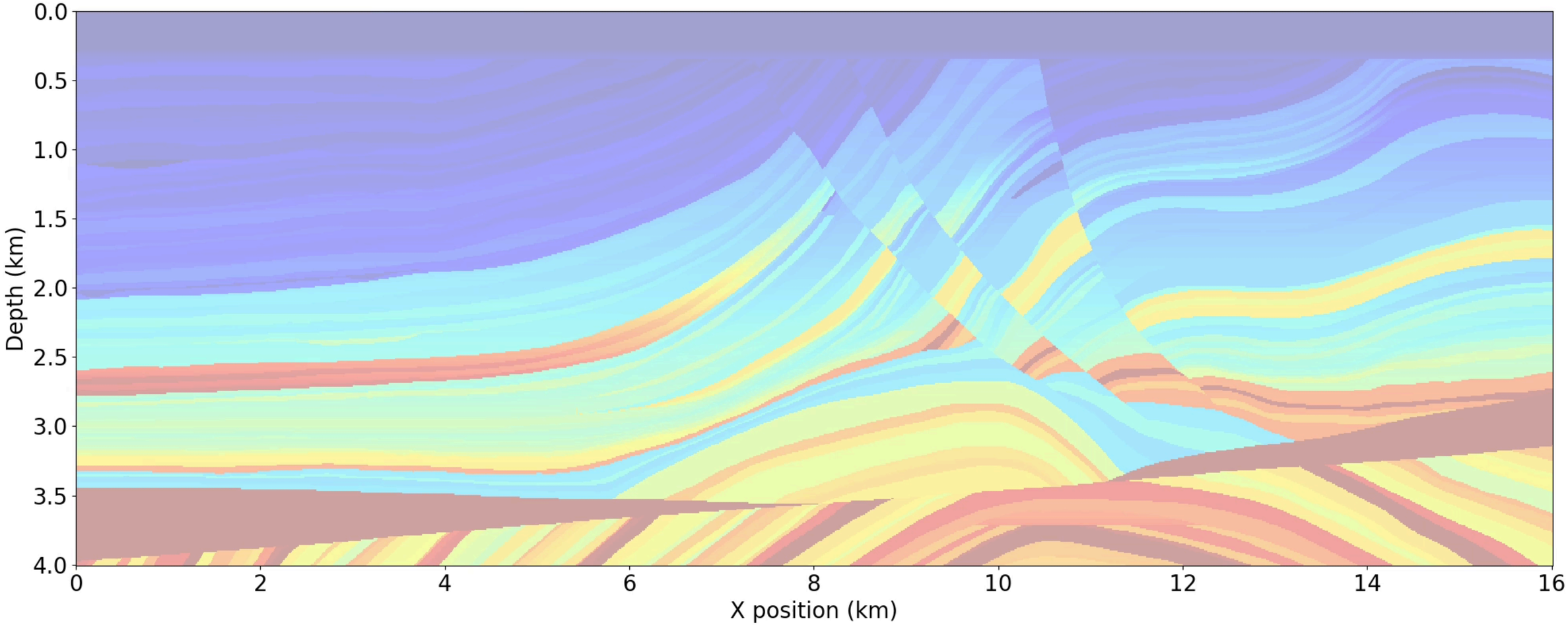
```
# Source term
src_eq = src.inject(field=u.forward, expr=src * s**2 / m)

# Insert source and receiver terms post-hoc
rec_term = rec.interpolate(expr=u)
```

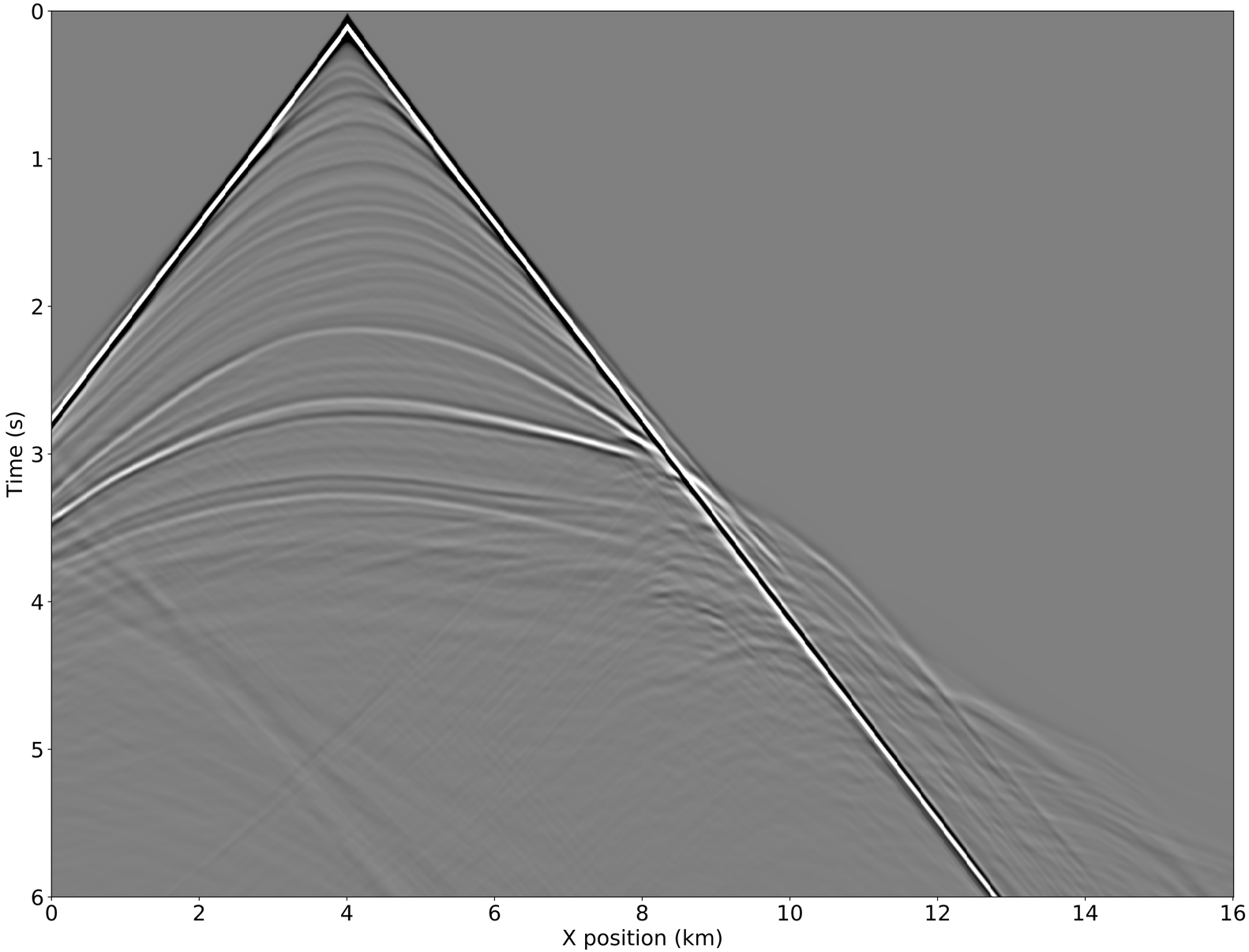
Forward operator

```
# Create a forward operator
Operator(stencil + src_expr + rec_expr, subs=subs, **kwargs)
```

```
(rec, u) = Acoustic.Forward()
```

```
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <string>
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
  double loop_body;
  double kernel;
} ;
struct flops
{
  long long loop_stencils_a;
  long long loop_body;
  long long kernel;
} ;
extern "C" int ForwardOperator(double *m_vec, double *u_vec, double *damp_vec, double *src_vec, float
*src_coords_vec, double *rec_vec, float *rec_coords_vec, long i1block, struct profiler *timings, struct flops *flops)
{
  double (*m)[280] = (double (*)[280]) m_vec;
  double (*u)[280][280] = (double (*)[280][280]) u_vec;
  double (*damp)[280] = (double (*)[280]) damp_vec;
  double (*src)[2] = (double (*)[2]) src_vec;
  float (*src_coords)[2] = (float (*)[2]) src_coords_vec;
  double (*rec)[101] = (double (*)[101]) rec_vec;
  float (*rec_coords)[2] = (float (*)[2]) rec_coords_vec;
  {
    struct timeval start_kernel, end_kernel;
    gettimeofday(&start_kernel, NULL);
    int t0;
    int t1;
    int t2;
    ;
    for (int i3 = 0; i3<3; i3+=1)
    {
      flops->kernel += 2.000000;
      {
        {
          t0 = (i3)%(3);
          t1 = (t0 + 1)%(3);
          t2 = (t1 + 1)%(3);
        }
        struct timeval start_loop_body, end_loop_body;
        gettimeofday(&start_loop_body, NULL);
        {
          for (int i1b = 1; i1b<279 - (278 % i1block); i1b+=i1block)
            for (int i1 = i1b; i1<i1b+i1block; i1++)
            {
              #pragma GCC ivdep
```


0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Depth (km)
0
2
4
6
8
10
12
14
16
X position (km)



Shot record

Measurements at the receiver locations

Devito for inversion

Adjoint PDE

Gradients

Adjoint state gradient

FWI objective

$$\Phi(\mathbf{m}) = \frac{1}{2}|| \mathbf{P}_r \mathbf{A}^{-1}(\mathbf{m}) \mathbf{P}_s^T \mathbf{q} - \mathbf{d}||_2^2$$

with gradient with respect to $\mathbf{m}$

$$\nabla\Phi(\mathbf{m}) = -(\frac{d^2\mathbf{u}}{d\mathbf{t}^2})^T \mathbf{A}(\mathbf{m})^{-T} \mathbf{P}_r^T (\mathbf{P}_r \mathbf{A}(\mathbf{m})^{-1} \mathbf{P}_s \mathbf{q} - \mathbf{d})$$

requires adjoint wave-equation

Discretization for inversion

Extend symbolic discretization to adjoints

```
first_derivative(u,  dim=x, side=centered, order=spc_order, matvec=transpose)
```

CRITICAL for non even order derivatives (anti-symmetric stencil)

Not required for acoustic (self adjoint system)

Stencil

$$\mathbf{v}(\mathbf{x}, \mathbf{t} - \Delta\mathbf{t}) = \frac{\Delta\mathbf{t}^2}{\mathbf{m}(\mathbf{x})}(2\mathbf{v}(\mathbf{x}, \mathbf{t}) - \mathbf{v}(\mathbf{x}, \mathbf{t} + \Delta\mathbf{t}) + \Delta\mathbf{v}(\mathbf{x}, \mathbf{t}))$$

`u.backward` = `solve(equation, u.backward)`

```
stencil = Eq(u.backward, solve(equation, u.backward))
```

Adjoint operator

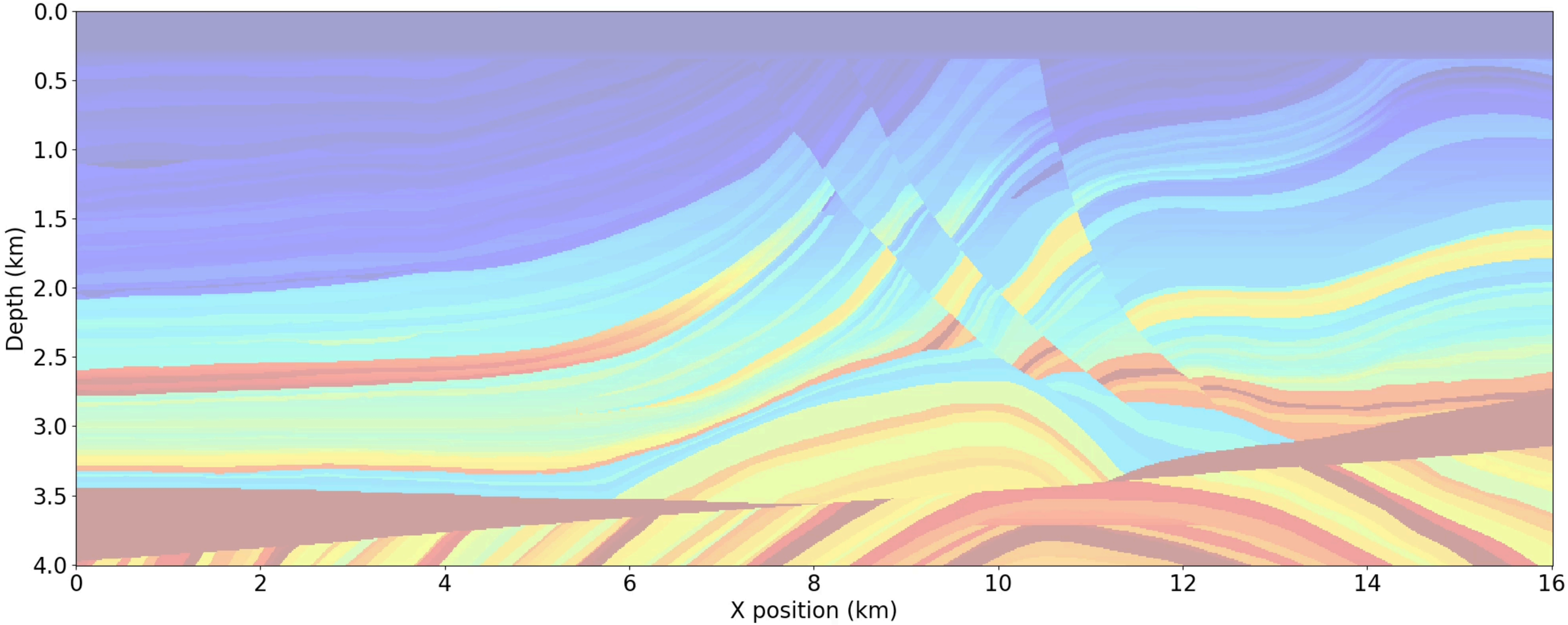
```
# Source term
src_eq = rec(field=v.backward, expr=rec * s**2 / m)

# Insert source and receiver terms post-hoc
rec_term = source.interpolate(expr=u)
```

Adjoint operator

```
# Create a forward operator
Operator(stencil + src_expr + rec_expr, subs=subs, **kwargs)
```

```
(srca, v) = Acoustic.Adjoint()
```

```
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <string>
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
  double loop_body;
  double kernel;
} ;
struct flops
{
  long long loop_stencils_a;
  long long loop_body;
  long long kernel;
} ;
extern "C" int AdjointOperator(double *m_vec, double *v_vec, double *damp_vec, double *srca_vec, float
*srca_coords_vec, double *rec_vec, float *rec_coords_vec, long i1block, struct profiler *timings, struct flops
*flops)
{
  double (*m)[280] = (double (*)[280]) m_vec;
  double (*v)[280][280] = (double (*)[280][280]) v_vec;
  double (*damp)[280] = (double (*)[280]) damp_vec;
  double (*srca)[2] = (double (*)[2]) srca_vec;
  float (*srca_coords)[2] = (float (*)[2]) srca_coords_vec;
  double (*rec)[101] = (double (*)[101]) rec_vec;
  float (*rec_coords)[2] = (float (*)[2]) rec_coords_vec;
  {
    struct timeval start_kernel, end_kernel;
    gettimeofday(&start_kernel, NULL);
    int t0;
    int t1;
    int t2;
    ;
    for (int i3 = 0; i3<3; i3+=1)
    {
      flops->kernel += 2.000000;
      {
        {
          t0 = (i3)%(3);
          t1 = (t0 + 1)%(3);
          t2 = (t1 + 1)%(3);
        }
        struct timeval start_loop_body, end_loop_body;
        gettimeofday(&start_loop_body, NULL);
        {
          for (int i1b = 1; i1b<279 - (278 % i1block); i1b+=i1block)
            for (int i1 = i1b; i1<i1b+i1block; i1++)
            {
```


0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Depth (km)
0
2
4
6
8
10
12
14
16
X position (km)

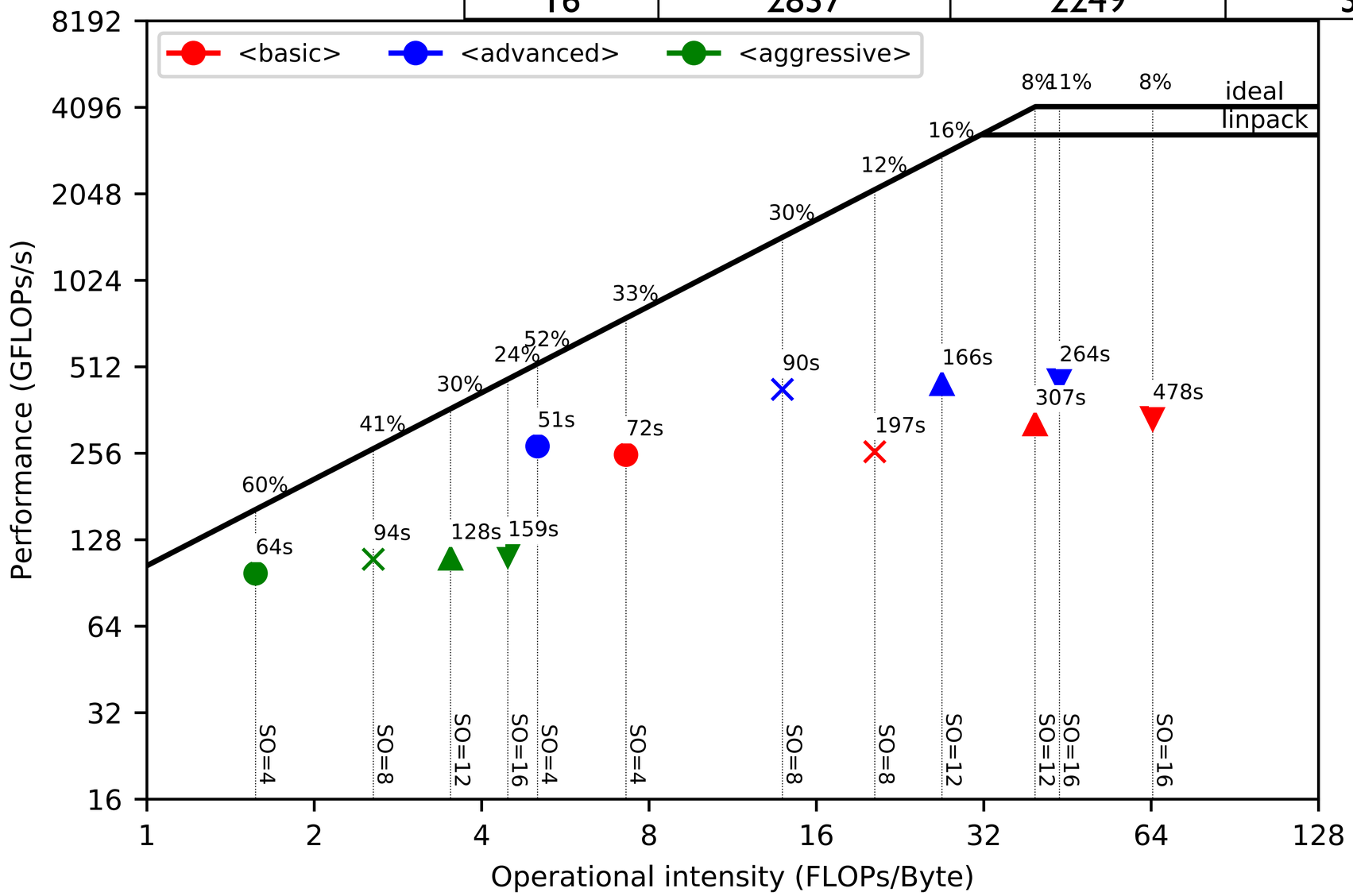
Adjoint state requirement

- How to handle forward wavefield saves?
  - Checkpointing
    Griewank, A., Walther, A.: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. ACM Trans. Math. Software 26, 19–45 (2000)
  - Time reversal
    McMechan, G. A., 1983, Migration by extrapolation of time-dependent boundary values*: Geophysical Prospecting, 31, 413–420.
    Mittet, R., 1994, Implementation of the kirchhoff integral for elastic waves in staggeredgrid modeling schemes: GEOPHYSICS, 59, 1894–1901.
    Raknes, E. B., and W. Weibull, 2016, Efficient 3d elastic full-waveform inversion using wavefield reconstruction methods: Geophysics, 81, R45–R55.
  - Subsampling
    Louboutin, M., and F. J. Herrmann, 2015, Time compressively sampled full-waveform in- version with stochastic optimization. (submitted to the SEG conference).

Computational performance

3D TTI performance w/roofline models:
- 512x512x512 grid points
- 1000ms propagation (416 time steps)

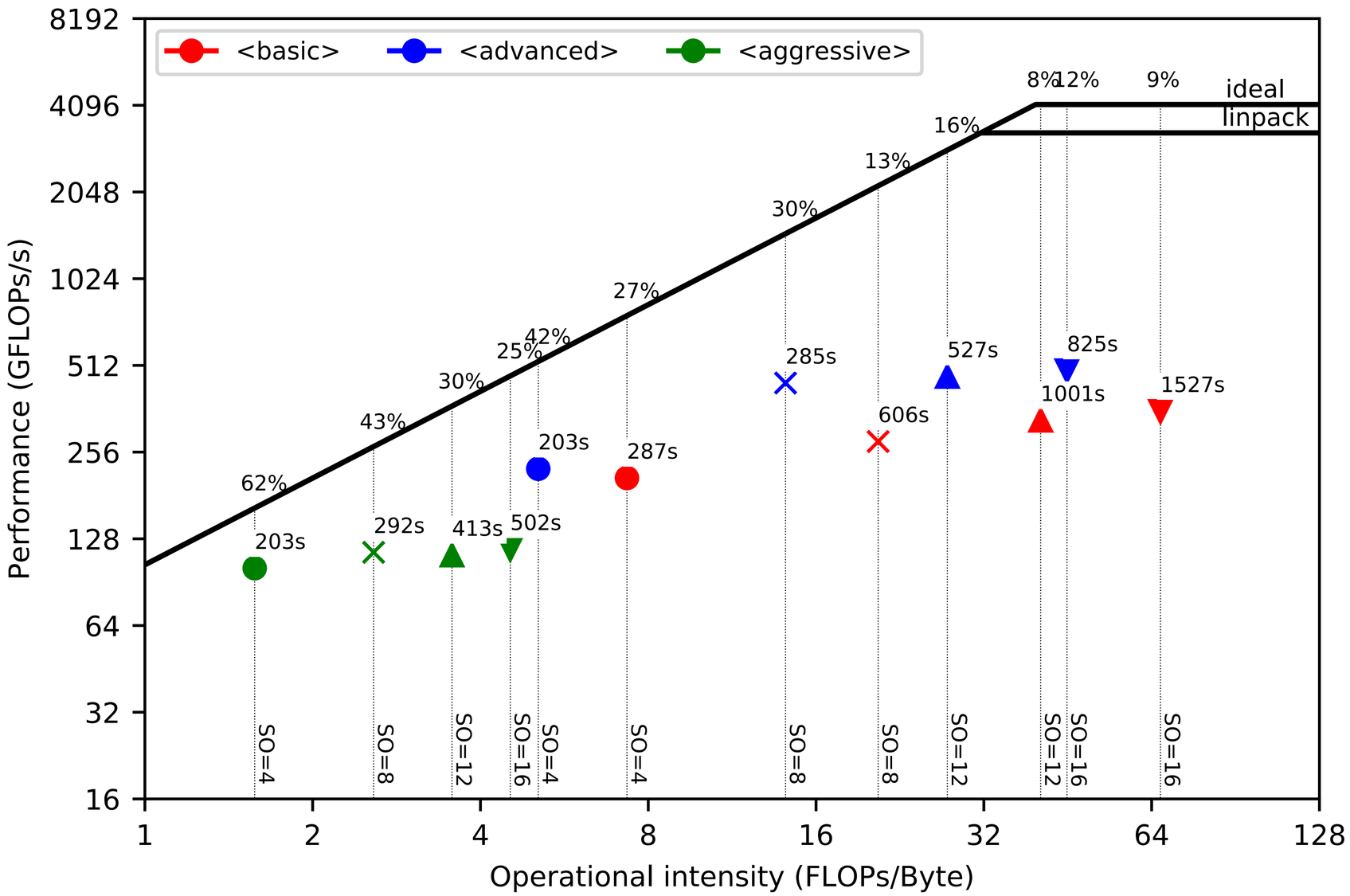
SO	Flops basic	Flops advanced	Flops aggressive
2	299	260	102
4	857	707	168
8	1703	1370	234
16	2837	2249	300



3D TTI performance:

- 768x768x768 grid points
- 1000ms propagation (416 time steps)

We scale linearly!



Summary

- Flexible physics with a simple finite-difference interface
  - Weeks, months of development time saved
  - Write your own physics
- Minimal coding required for geophysicists/mathematicians
  - Domain specialists only focus on their own problem
  - Improves collaborations with a high-level common ground
- Simulation for inversion with adjoint-aware discretization
- Adjoints are inherently hard, specially for complicated physics
- Advantages of code generation (performance, system and architecture portability)

Observations

Highly abstracted JIT compiler w/ pathways to

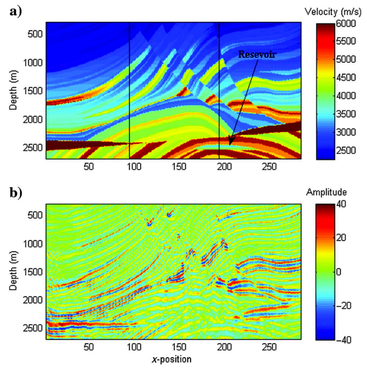
- C, MPI+OpenMP+C, CUDA, MPI+CUDA...
- backend w/ YASK implemented (3 X speed up on Xeon Phi)
- backend w/ OPS library for CPU-GPU(+MPI)

Take home message: getting the abstraction right is key!

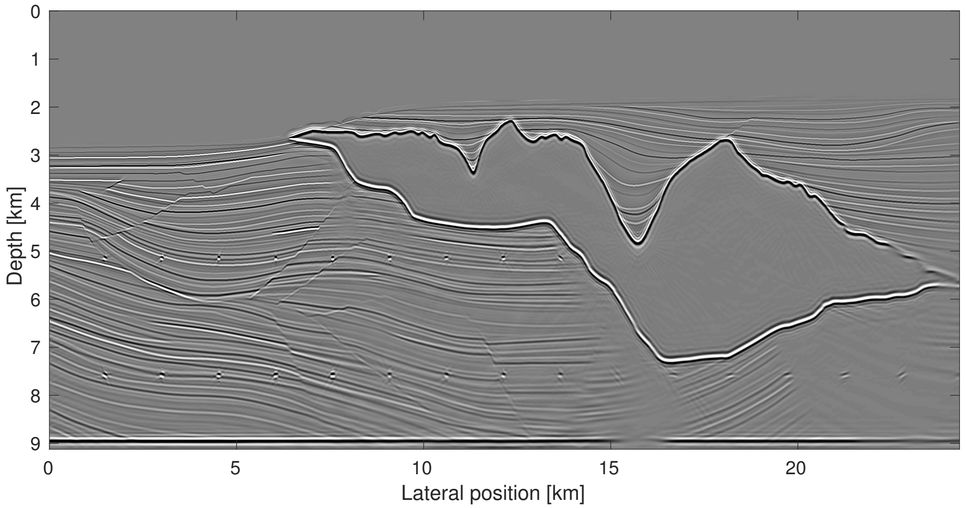
- highly productive environment w/ flexibility w.r.t. discretization (stencil)
- connection w/ linear algebra
- parallel IO & access to meta data
- intuitive data parallelism to work w/ multiple instances

https://github.com/intel/yask

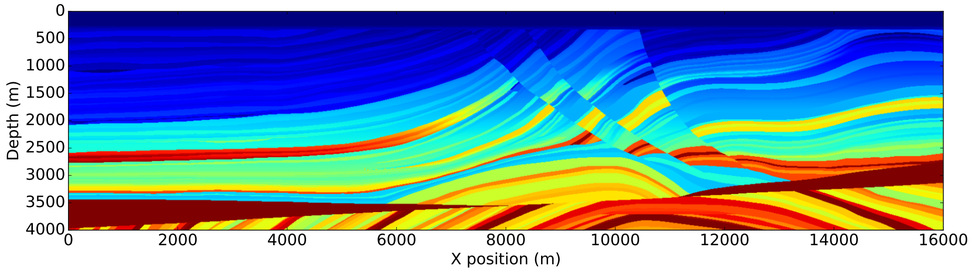
https://github.com/OP-DSL/OPS



Tiny marmousi
62k gridpoints
1.5MFlop/time-step

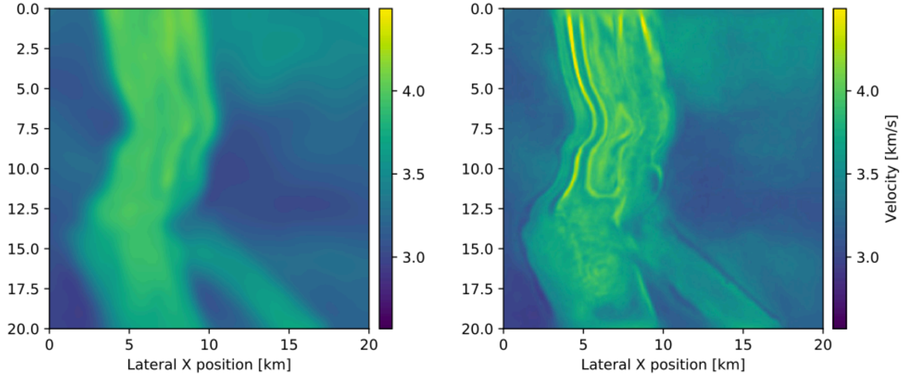


Sigsbee
2.2M gridpoints
56MFlop/time-step

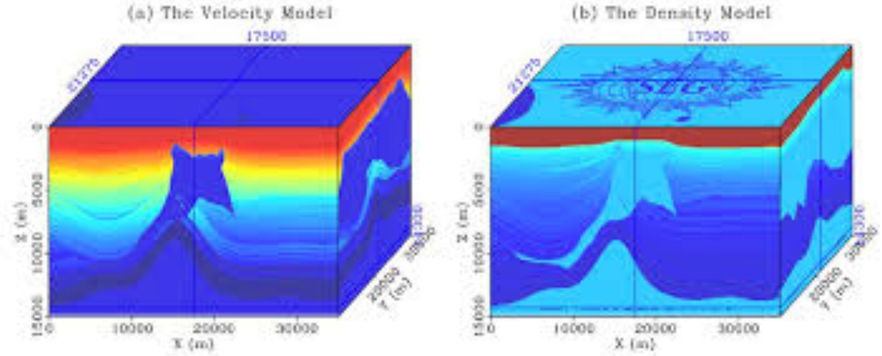


Full marmousi
640k gridpoints
15MFlop/time-step

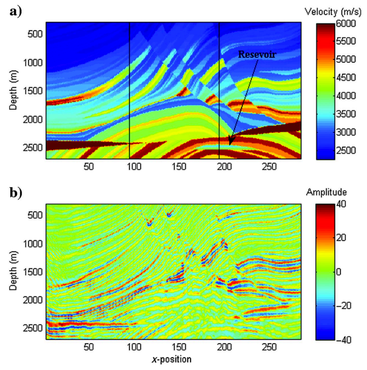
Patiently wait for result



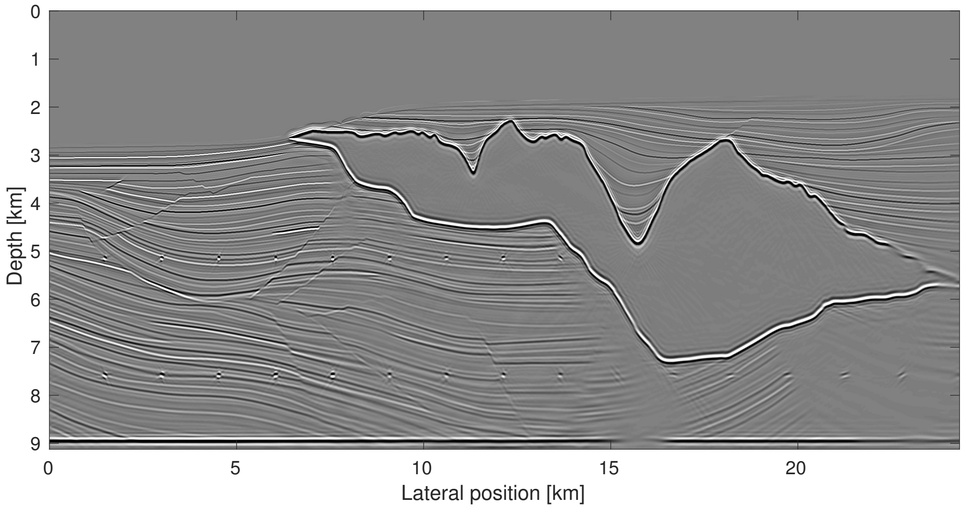
3D overthrust
222M gridpoints
6GFlop/time-step



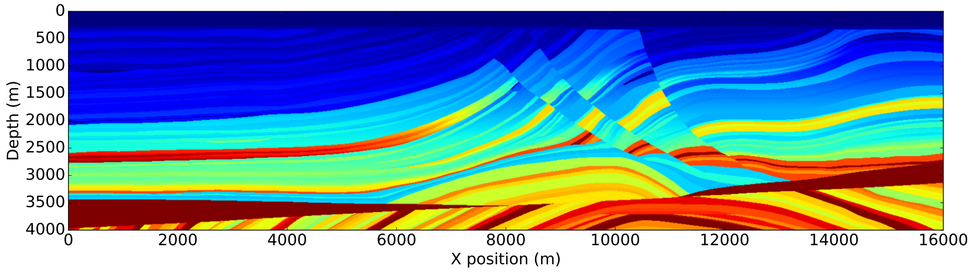
SEAM
2.2G gridpoints
56GFlop/time-step



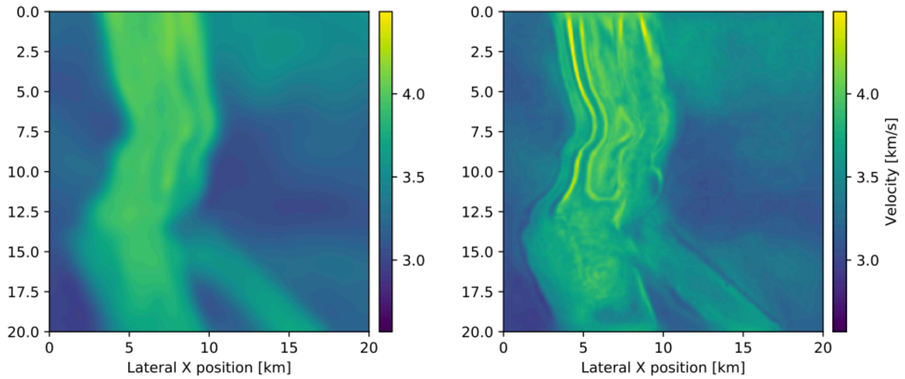
Tiny marmousi
62k gridpoints
1.5MFlop/time-step



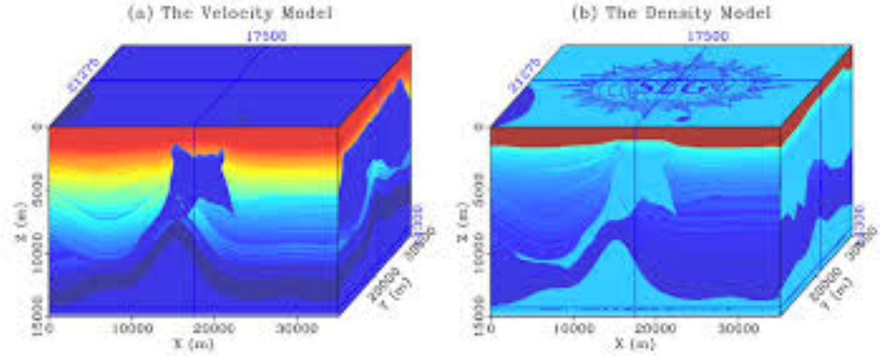
Sigsbee
2.2M gridpoints
56MFlop/time-step



Full marmousi
640k gridpoints
15MFlop/time-step



3D overthrust
222M gridpoints
6GFlop/time-step



SEAM
2.2G gridpoints
56GFlop/time-step

Future work

- Abstraction for the boundary conditions
  - How accurate?
  - How expensive?
- Automatic adjoints
- FD for source/receivers (i.e dipole sources)

Domain-specific abstractions for large-scale geophysical inverse problems

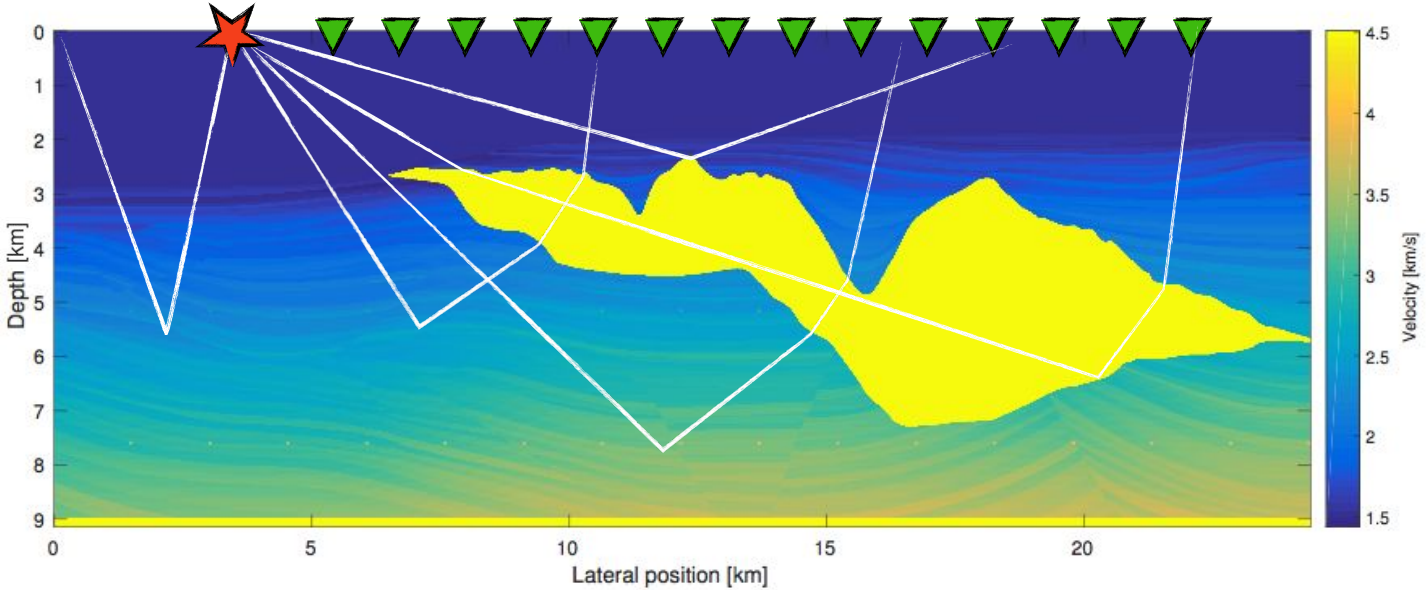
Philipp A. Witte, Mathias Louboutin and Felix J. Herrmann

SLIM
Georgia Institute of Technology

Motivation

Use Geophysics to understand the earth

- invert for subsurface parameters, e.g. velocity, density, porosity



Motivation

Computational challenges of seismic inverse problems:

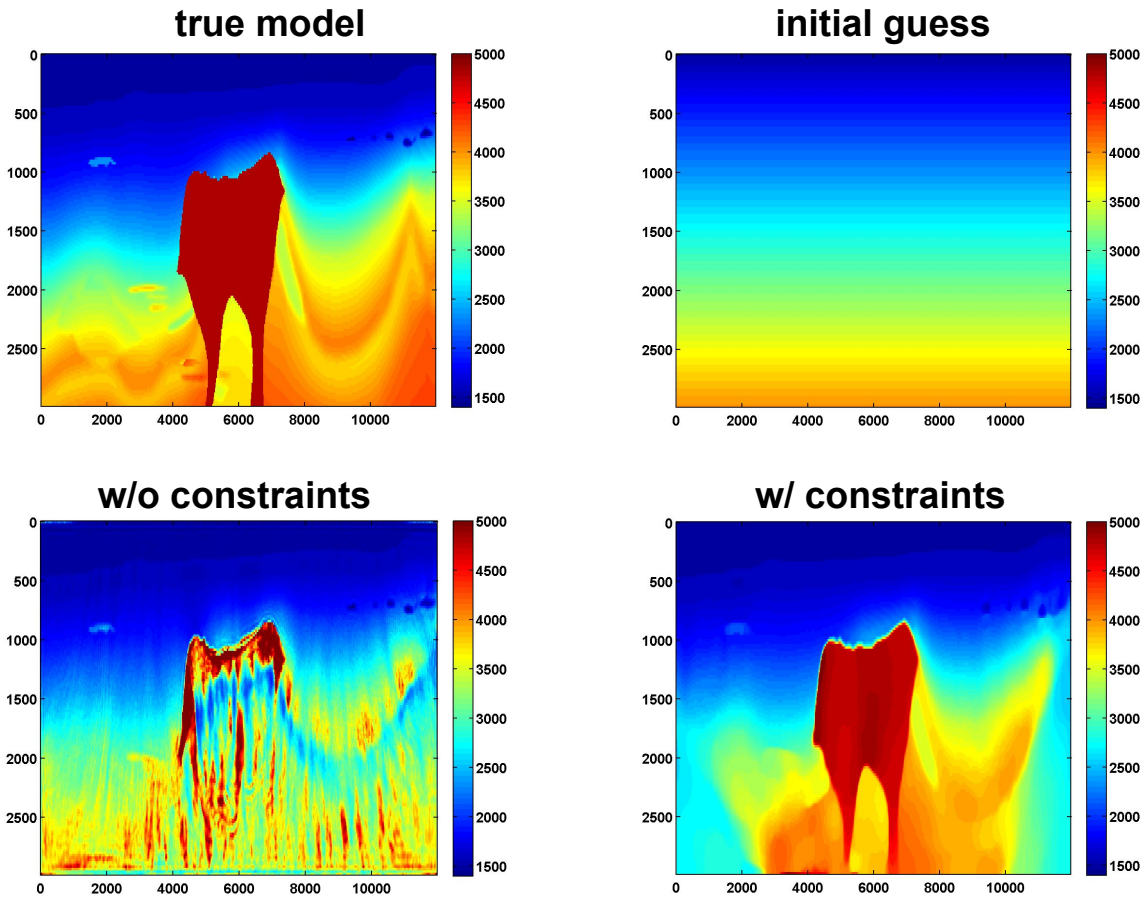
- Up to of 1e7 (2D) and 1e10 (3D) unknown variables
- Observations of several magnitudes larger (several TB of data)
- Large number of observed data samples (10k or more)
- Propagate wavefields over thousands of time steps (like 10k layer residual network with single/few channels)
- Many floating point operations per grid point/time step (typically 8 or 16 point stencils)

Motivation

Mathematical challenges of seismic inverse problems:

- Data: contains many physical effects unaccounted for in physical models (“noise”)
- Nonlinear problems often ill-posed and non-convex
- Require regularization/constraints
- Linear problems badly conditioned and inconsistent
- Require (physically-motivated) pre-conditioning
- Due to large computational cost: can only afford few iterations (between 20 and 50)

Motivation



(Esser et al., 2015)

Motivation

Software for seismic inverse problems:

- Needs efficient PDE solver w/ correct adjoint pairs, gradients etc.
- Large-scale parallelization (cluster/cloud) w/ resilience to hardware failures
- Manage large seismic data volumes and meta data
- Allow implementation of complex algorithms (2nd order methods, constraints/penalties, linear solvers, preconditioners, etc.)
- Use for testing/research and full scale production

Motivation

The reality of seismic inversion codes:

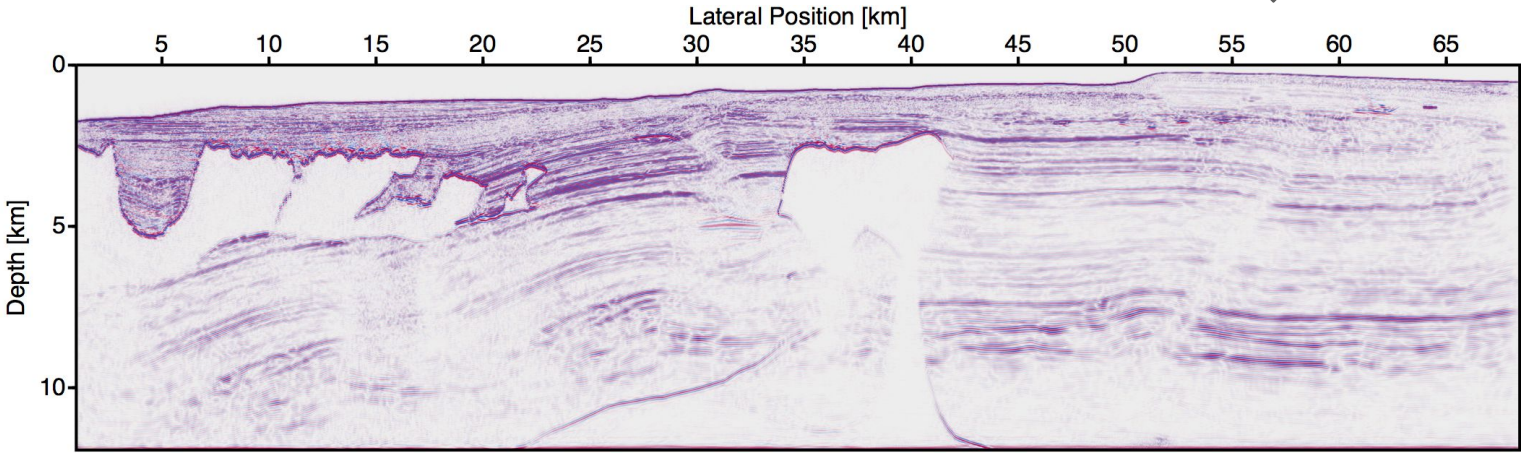
- Academic packages in Python, MATLAB, etc., do not scale to realistic problem sizes
- Software in O&G companies: “state secrets”, no open-source software or collaboration w/ academia
- Written in C or FORTRAN + developed over decades
- Mixing of PDE solvers, I/O, parallelization, data processing + algorithms
- Modifying individual parts often impossible (different PDE/stencil order, add line search, etc.)
- Performance-optimization by separate HPC experts w/o considerations for correct adjoints and gradients

Motivation

Utilize power of abstractions:

- Code that closely follows underlying math
- Inspired by software packages from ML + linear algebra (TensorFlow/PyTorch, Trilinos, Rice Vector Library)

```
for j=1:n
    r = J*x - d_obs
    g = J'*r
    x = x - alpha*g
end
```



Motivation

The Julia Devito Inversion (JUDI) framework:

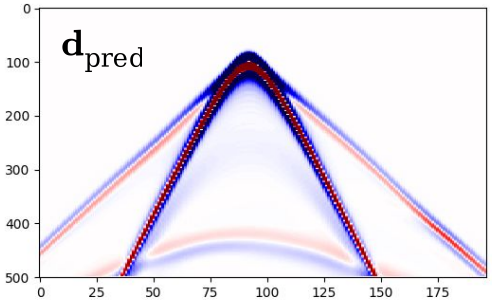
- High-level Julia package build on top of Devito
- Matrix-free linear operators and abstract data vectors
- Formulate algorithms in terms of linear algebra
- Julia provides resilience on cluster-like environments

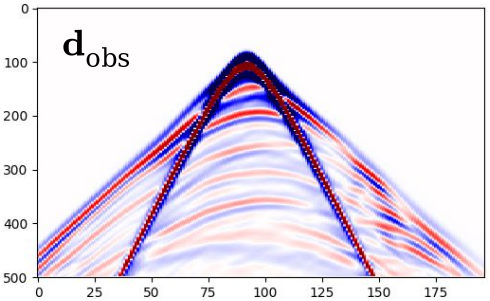
Non-linear seismic inversion

Full waveform inversion

- recover discrete parametrization of subsurface (velocity, density, etc.)
- minimize misfit between observed and modeled data:

$$\Phi(\mathbf{m}) = \frac{1}{2}\|\mathbf{d}_{\text{pred}} - \mathbf{d}_{\text{obs}}\|_2^2$$



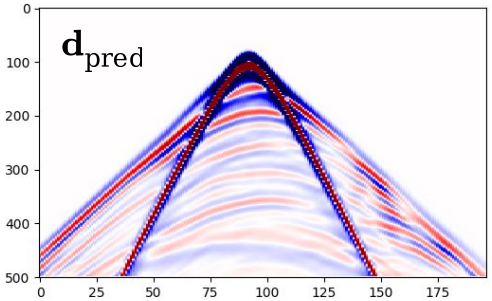


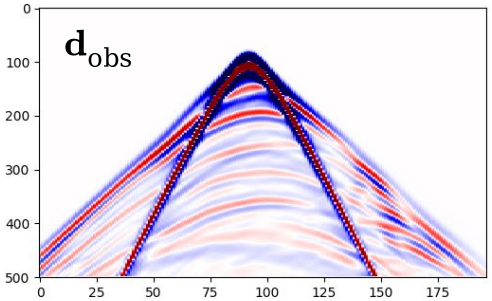
Non-linear seismic inversion

Full waveform inversion

- recover discrete parametrization of subsurface (velocity, density, etc.)
- minimize misfit between observed and modeled data:

$$\Phi(\mathbf{m}) = \frac{1}{2}\|\mathbf{d}_{\text{pred}} - \mathbf{d}_{\text{obs}}\|_2^2$$



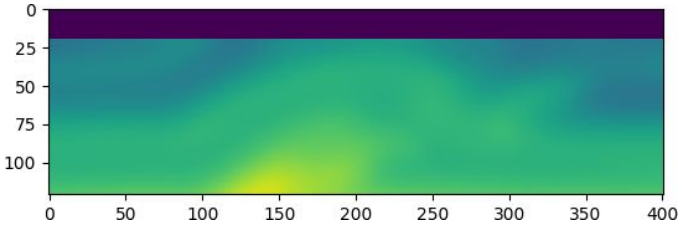


Non-linear seismic inversion

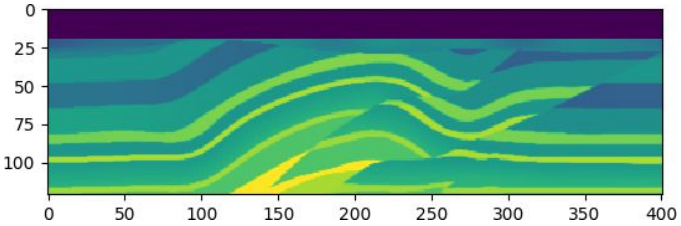
Full waveform inversion

- recover discrete parametrization of subsurface (velocity, density, etc.)
- minimize misfit between observed and modeled data:

$$\Phi(\mathbf{m}) = \frac{1}{2}\|\mathbf{d}_{\text{pred}} - \mathbf{d}_{\text{obs}}\|_2^2$$



$\mathbf{m}$



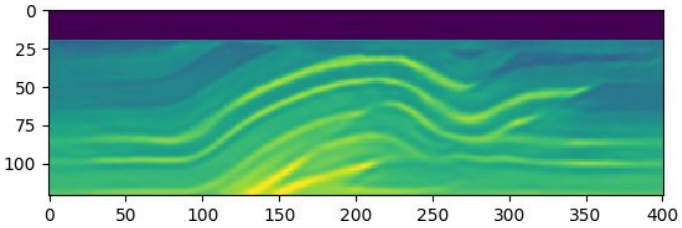
$\mathbf{m}_{\text{true}}$

Non-linear seismic inversion

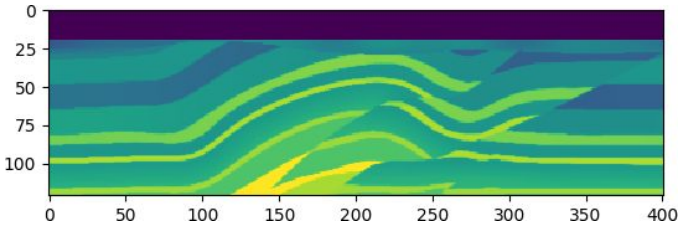
Full waveform inversion

- recover discrete parametrization of subsurface (velocity, density, etc.)
- minimize misfit between observed and modeled data:

$$\Phi(\mathbf{m}) = \frac{1}{2}\|\mathbf{d}_{\text{pred}} - \mathbf{d}_{\text{obs}}\|_2^2$$



$\mathbf{m}$



$\mathbf{m}_{\text{true}}$

Non-linear seismic inversion

Full waveform inversion

- Sensitivities of modeling operator

$$\Phi(\mathbf{m}) = \frac{1}{2}\|\mathbf{P}_r\mathbf{A}(\mathbf{m})^{-1}\mathbf{P}_s^\top\mathbf{q} - \mathbf{d}_{\text{obs}}\|_2^2$$

$$\mathbf{J} = \frac{\partial}{\partial\mathbf{m}}\left(\mathbf{P}_r\mathbf{A}(\mathbf{m})^{-1}\mathbf{P}_s^\top\mathbf{q}\right)$$

- Gradient of FWI objective function

$$\mathbf{g} = \mathbf{J}^\top\left(\mathbf{d}_{\text{pred}} - \mathbf{d}_{\text{obs}}\right)$$

Non-linear seismic inversion

Full waveform inversion

- Sensitivities of modeling operator

$$\Phi(\mathbf{m}) = \frac{1}{2}\|\mathbf{P}_r\mathbf{A}(\mathbf{m})^{-1}\mathbf{P}_s^\top\mathbf{q} - \mathbf{d}_{\text{obs}}\|_2^2$$

$$\mathbf{J} = \frac{\partial}{\partial\mathbf{m}}\left(\mathbf{P}_r\mathbf{A}(\mathbf{m})^{-1}\mathbf{P}_s^\top\mathbf{q}\right)$$

```
d_pred = Pr*A_inv*adjoint(Ps)*q
f = .5f0*norm(d_pred - d_obs)^2
g = adjoint(J)*(d_pred - d_obs)
```

- Gradient of FWI objective function

$$\mathbf{g} = \mathbf{J}^\top\left(\mathbf{d}_{\text{pred}} - \mathbf{d}_{\text{obs}}\right)$$

Example 1: Gradient descent

Runnable Julia code:

```
# Main loop
for j=1:maxiter

    # get fwi objective function value and gradient
    i = randperm(d_obs.nsrc)[1:batchsize]
    f, g = fwi_objective(model, q[i], d_obs[i])

    # linesearch
    step = backtracking_linesearch(model, f, g, varargs...)

    # Update model and projection
    model.m = proj(model.m + reshape(step, model.n))
end
```

alternatively:

- Change update rule (e.g. conjugate gradients, etc.)
- Implement more complex algorithms (L-BFGS, Gauss-Newton, etc.)

Example 2: Gauss-Newton method

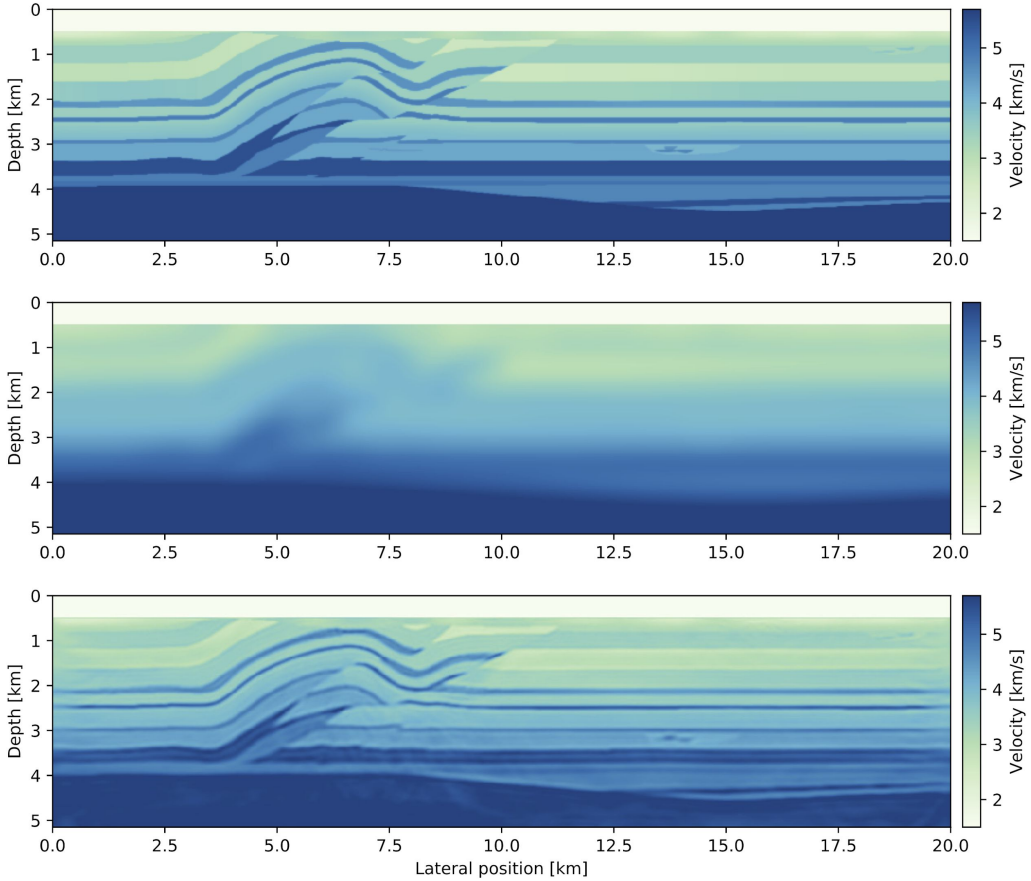
Runnable Julia code:

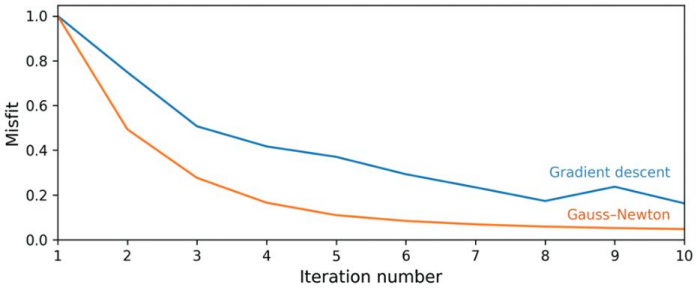
```
# Main loop
for j=1:maxiter

    # Model predicted data
    d_pred = Pr*A_inv*Ps'*q

    # GN update direction
    p = lsqr(J, d_pred - d_obs; maxiter=10)

    # Update model
    model.m = model.m - reshape(p, model.n)
end
```

Gauss-Newton subproblems:

- Pass matrix-free linear operator to third party solvers
- LSQR from Julia *IterativeSolvers.jl* package
- Overload necessary operations in *lsqr* for JUDI operators/vectors

Example 2: Gauss-Newton method





Example 3: Spectral projected GD

What if we want to use more complicated algorithms?

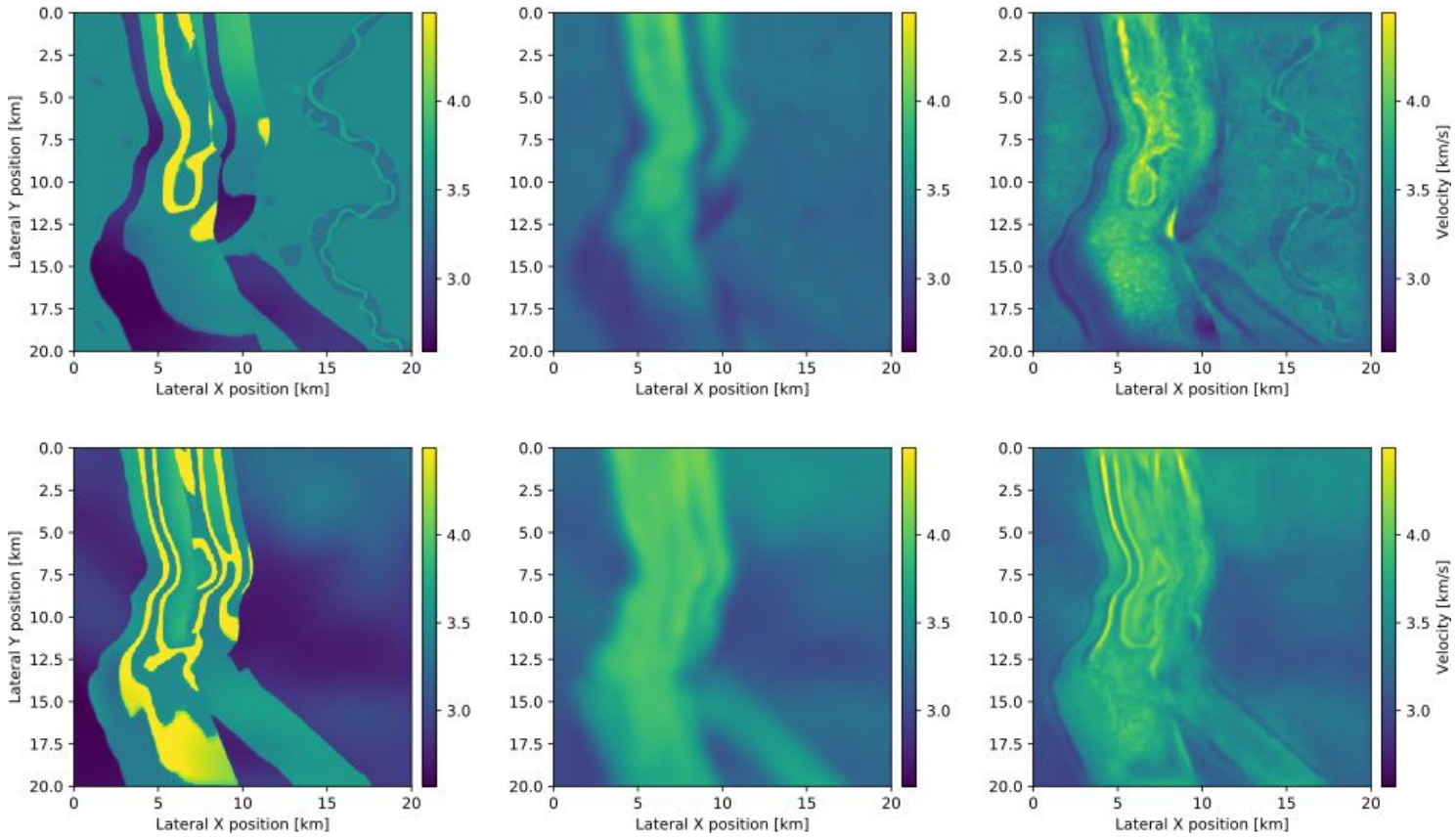
- Pass misfit functions 3rd-party optimization libraries
- Access to large variety of optimization methods
- E.g. *minConf, NLopt.jl, Optim.jl*

Does this scale to large problems?

- Case study w/ 3D Overthrust velocity model
- 801 x 801 x 207 grid points + PML (222 million unknowns)
- 100 billion data points, over 1.5 TB of data
- spectral-projected gradient algorithm from minConf library
- 15 iterations w/ batchsize of 1080 per iteration

(Schmidt et al., 2009, Johnson et al. 2017, White et al., 2017)

Example 3: Spectral projected GD



z = 400 m

z = 800 m

Conclusions

Seismic inverse problems are extremely challenging:

- Mathematically + computationally complex
- Abstractions + separation of concerns to manage complexity
- Testing/Research in high-level languages
- Automatic performance optimizations