Modeling for inversion in exploration Geophysics

Mathias Louboutin
Ph.D Defense of dissertation
February 26th 2020

SLIM
Georgia Institute of Technology

Wave-equation based geophysical exploration

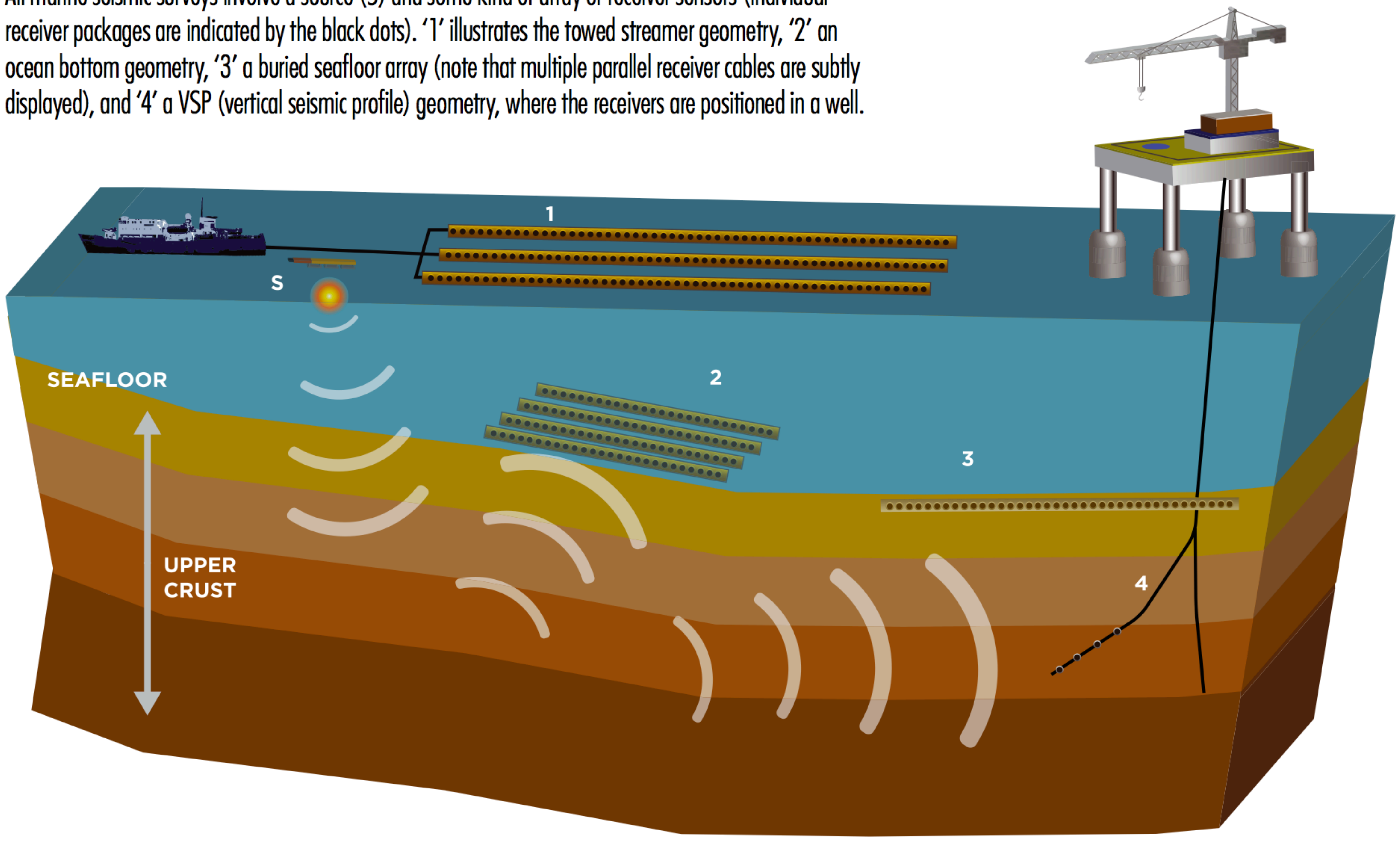
Seismic inversion

Infer 3D images from massive multi-experiment data:

- $\mathcal{O}(10^9)$ unknowns
- $\mathcal{O}(10^{15})$ datapoints
- propagate $\mathcal{O}(10^2)$ wavelengths

- >10k time steps
- acoustic only
- elastic much larger
- extreme compute & IO

All marine seismic surveys involve a source (S) and some kind of array or receiver sensors (individual receiver packages are indicated by the black dots). '1' illustrates the towed streamer geometry, '2' an ocean bottom geometry, '3' a buried seafloor array (note that multiple parallel receiver cables are subtly displayed), and '4' a VSP (vertical seismic profile) geometry, where the receivers are positioned in a well.



Equivalent to (computationally)

- 10000 Layers CNN
- 1000x1000x1000 input (ie HD gifs)
- Tens of iterations
- Thousands of source experiments (training pair)
- Interested in the parameters (physical meaning and interpretation) not the data fit only

Motivations and problem statement

Complex mathematics

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

Non-convex problem

- Requires complex algorithms

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 ➞ $\mathbf{A}(\mathbf{m}) \in \mathbb{R}^{n \times n}, n = \mathcal{O}(1e13)$

Gradient based optimisation

- Requires adjoints, gradients (and Hessian approximations)

Motivations and problem statement

Complex mathematics

Complex Physics with multiple representations

Scalar acoustic wave-equations

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

Acoustic isotropic

O(10) Flops/grid point

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

Acoustic isotropic with density

O(100) Flops/grid point

$$m(x)\frac{d^2p(x,t)}{dt^2} - (1+2\epsilon(x))H_{\bar{x}\bar{y}}p(x,t) - \sqrt{1+2\delta(x)}\ H_{\bar{z}}r(x,t) = 0,$$
$$m(x)\frac{d^2r(x,t)}{dt^2} - \sqrt{1+2\delta(x)}\ H_{\bar{x}\bar{y}}p(x,t) - H_{\bar{z}}r(x,t) = 0$$

Acoustic anisotropic

Tilted transverse isotropic (VTI)

O(1000) Flops/grid point

Challenges

Flexible representation of the physics

Flop expensive and complex implementations

Approximation (TTI) not always self-adjoint

- Derivation and
- implementation of the true adjoint necessary

Adjoints

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

Self-adjoint

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

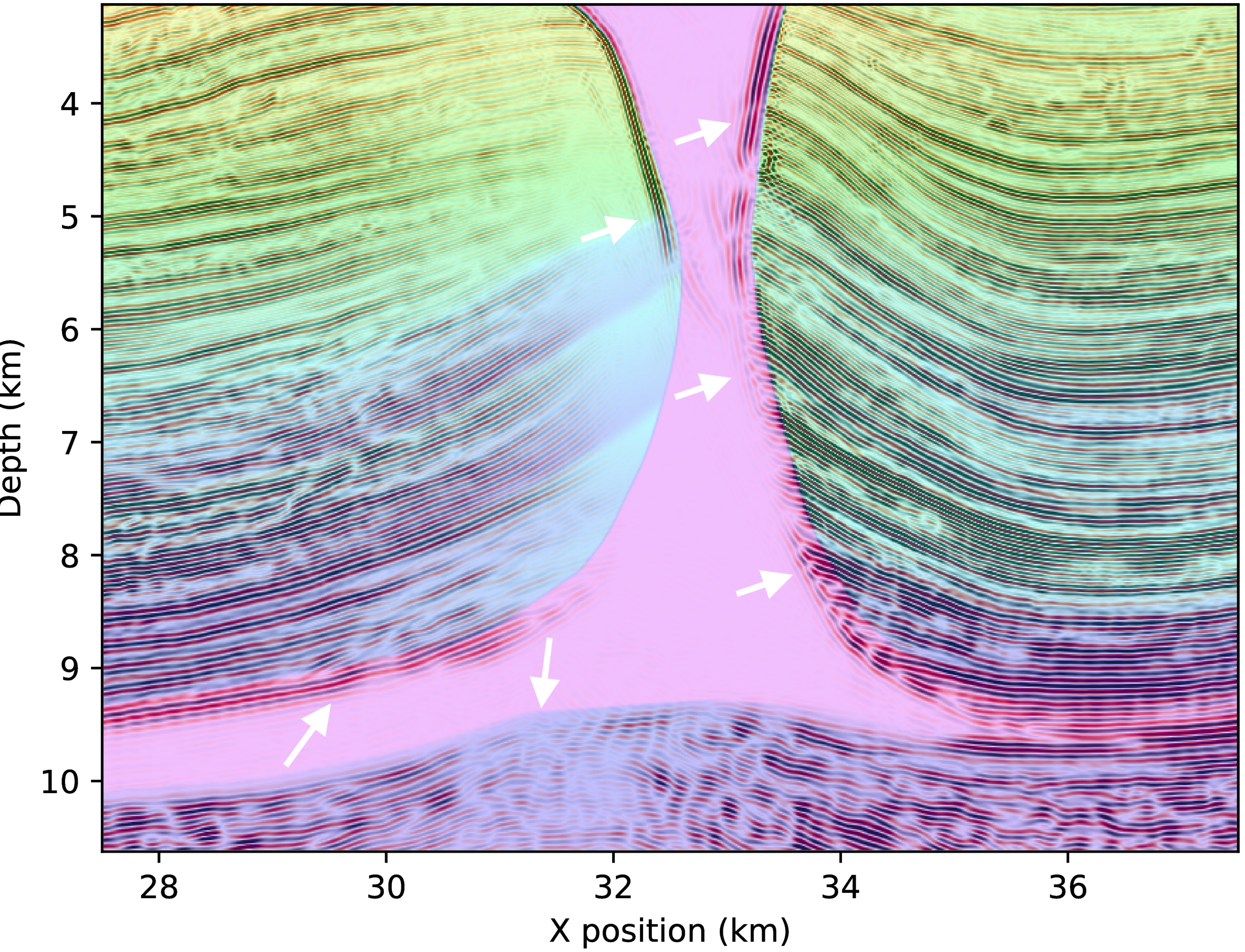
$$m(x)\frac{d^2 p(x,t)}{dt^2} - (1+2\epsilon(x))H_{\bar{x}\bar{y}}p(x,t) - \sqrt{1+2\delta(x)}\ H_{\bar{z}}r(x,t) = 0,$$
$$m(x)\frac{d^2 r(x,t)}{dt^2} - \sqrt{1+2\delta(x)}\ H_{\bar{x}\bar{y}}p(x,t) - H_{\bar{z}}r(x,t) = 0$$

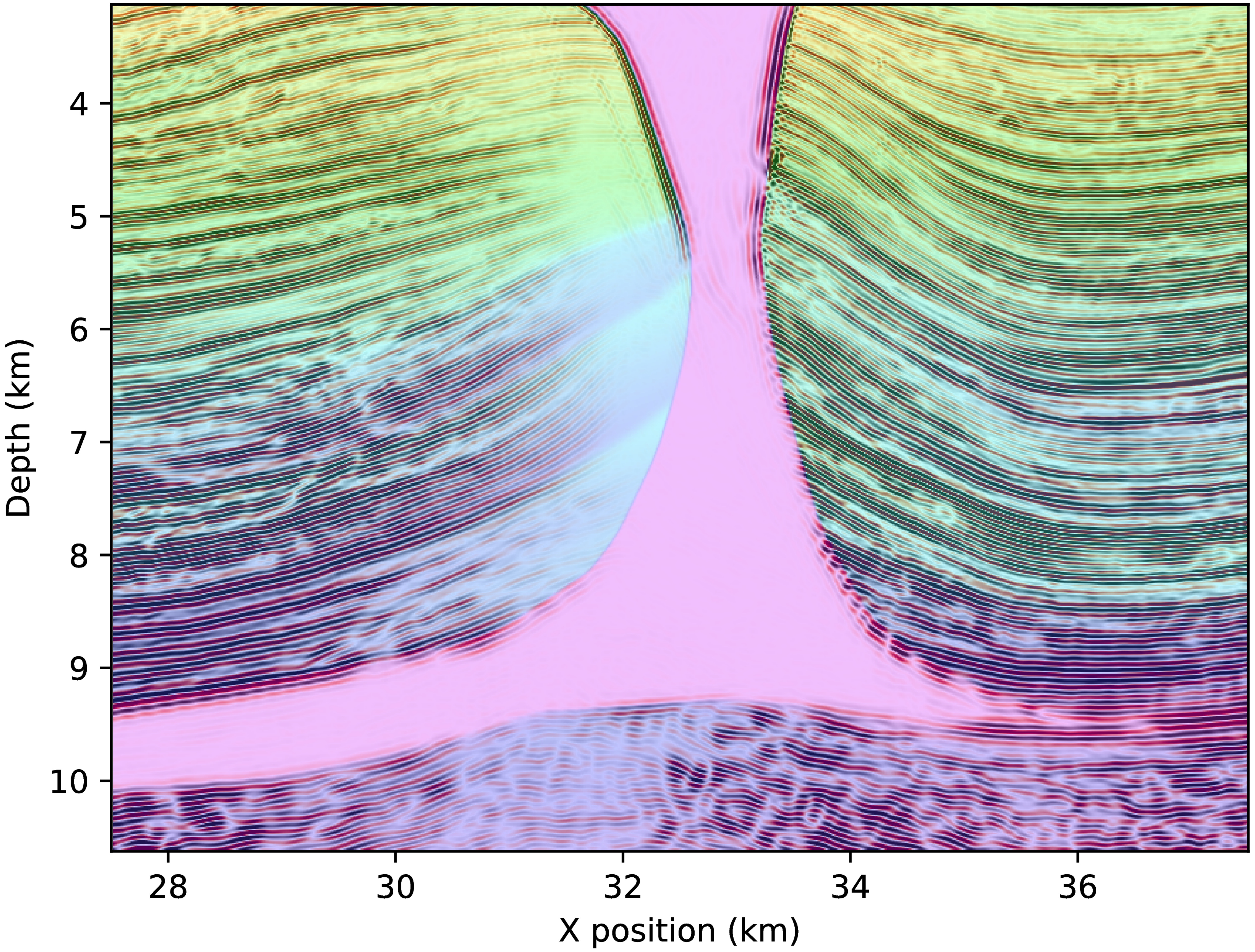
Not self-adjoint

$$m(x)\frac{d^2 p_a(x,t)}{dt^2} - H_{\bar{x}\bar{y}}(1+2\epsilon(x))p_a(x,t) - H_{\bar{x}\bar{y}}\sqrt{1+2\delta(x)}r_a(x,t) = 0,$$
$$m(x)\frac{d^2 r_a(x,t)}{dt^2} - H_{\bar{z}}\sqrt{1+2\delta(x)}p_a(x,t) - H_{\bar{z}}r_a(x,t) = 0$$

M. Louboutin, Witte, P. A., and Herrmann, F. J., "Effects of wrong adjoints for RTM in TTI media", in SEG Technical Program Expanded Abstracts, 2018, pp. 331–335

Effect of incorrect adjoint





Contributions

Complex Physics with multiple representations (Chapter 3 and 4)

- Separation of concern with Devito

Non trivial Performance evaluation (Chapter 2)

- The roofline model for performance prediction and evaluation

Enabling at-scale research and development with separation of concern

- M. Louboutin, Lange, M., Herrmann, F. J., Kukreja, N., and Gorman, G., "**Performance prediction of finite-difference solvers for different computer architectures**", Computers & Geosciences, vol. 105, pp. 148–157, 2017
- S. Williams, A. Waterman, D. Patterson, **"The Roofline model offers insight on how to improve the performance of software and hardware."**, communications of the ACM 52

SLIM

The Roofline model for PDEs

Chapter 2 Contributions:

- Performance prediction

- Solver design from predictions

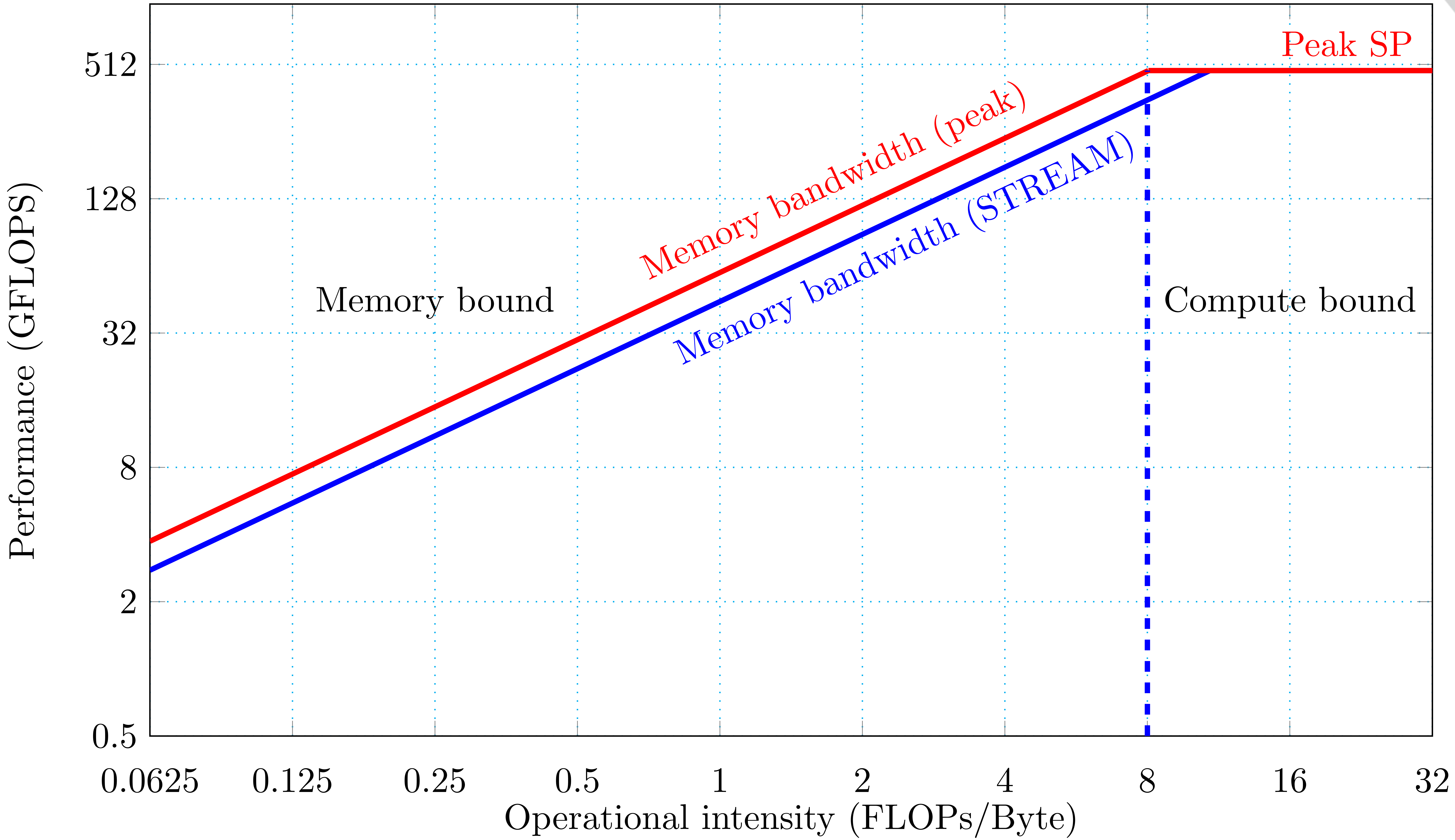
- Portable and absolute performance metric

Roofline model

Measure hardware level performance

No source to source comparison (“I achieved 200x speedup”)

Enables theoretical estimations of performance



Peak SP
Memory bandwidth (peak)
Memory bandwidth (STREAM)
Memory bound
Compute bound
Performance (GFLOPS)
512
128
32
8
2
0.5
0.0625
0.125
0.25
0.5
1
2
4
8
16
32
Operational intensity (FLOPs/Byte)

Operational intensity

$$\mathcal{OI}_{alg}(k) = \mathcal{F}_{kernel}(k)/\mathcal{B}_{global} = \frac{\mathcal{F}_{kernel}(k)}{4(l+s)}$$

Not all quantities are defined

Fixed

$\mathcal{OI}_{alg}(k)$: Operational Intensity

$\mathcal{B}_{global}$: Data movement

$\mathcal{F}_{kernel}(k)$: number of floating point operation (Add/Mul)

l : number of load/read

s : number of store/write

Operational intensity, example

$$f[x] = .5 * (f[x+1] + f[x-1])$$

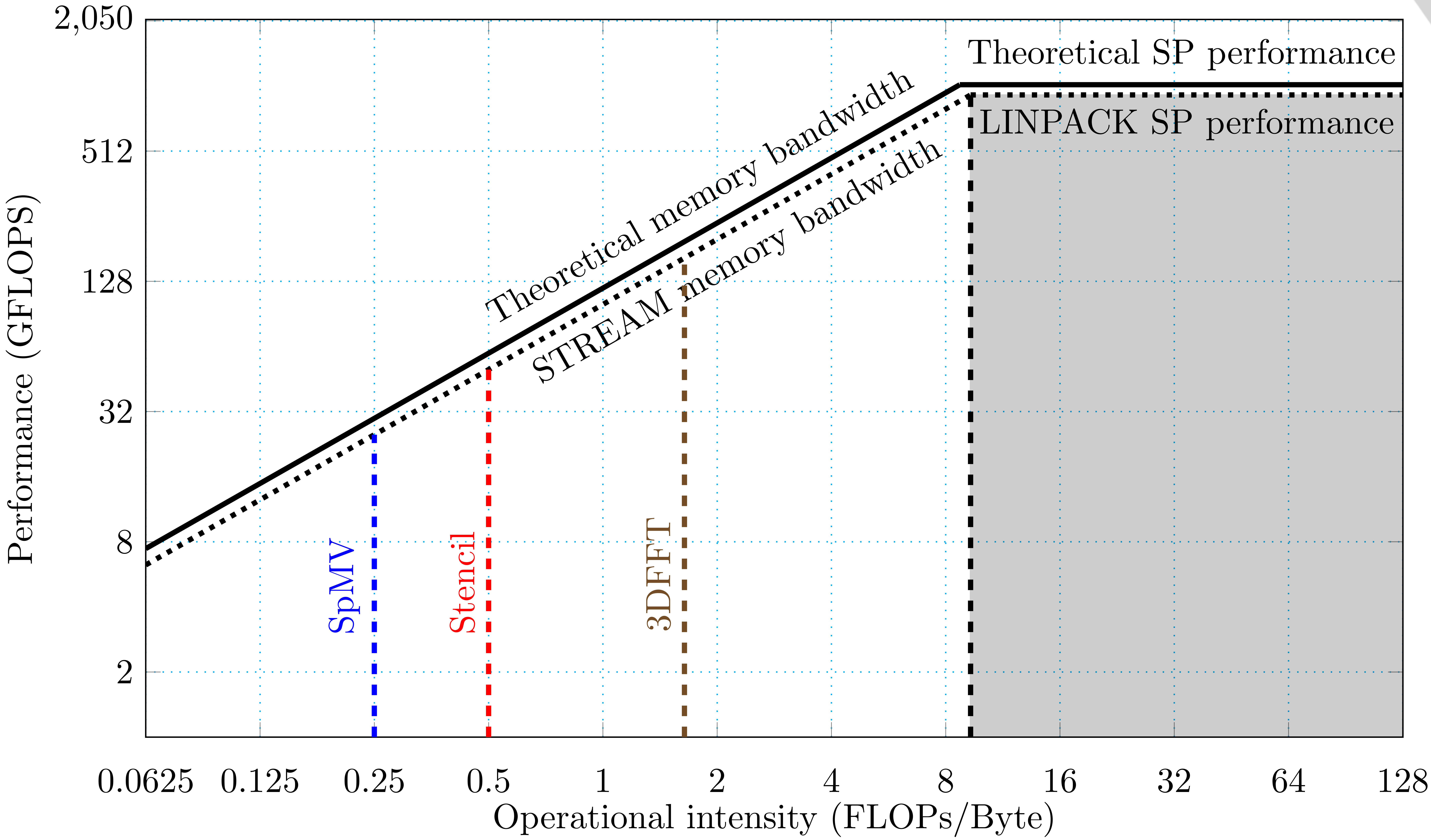
l : Two loads $f[x+1], f[x-1]$

s : One store $f[x]$

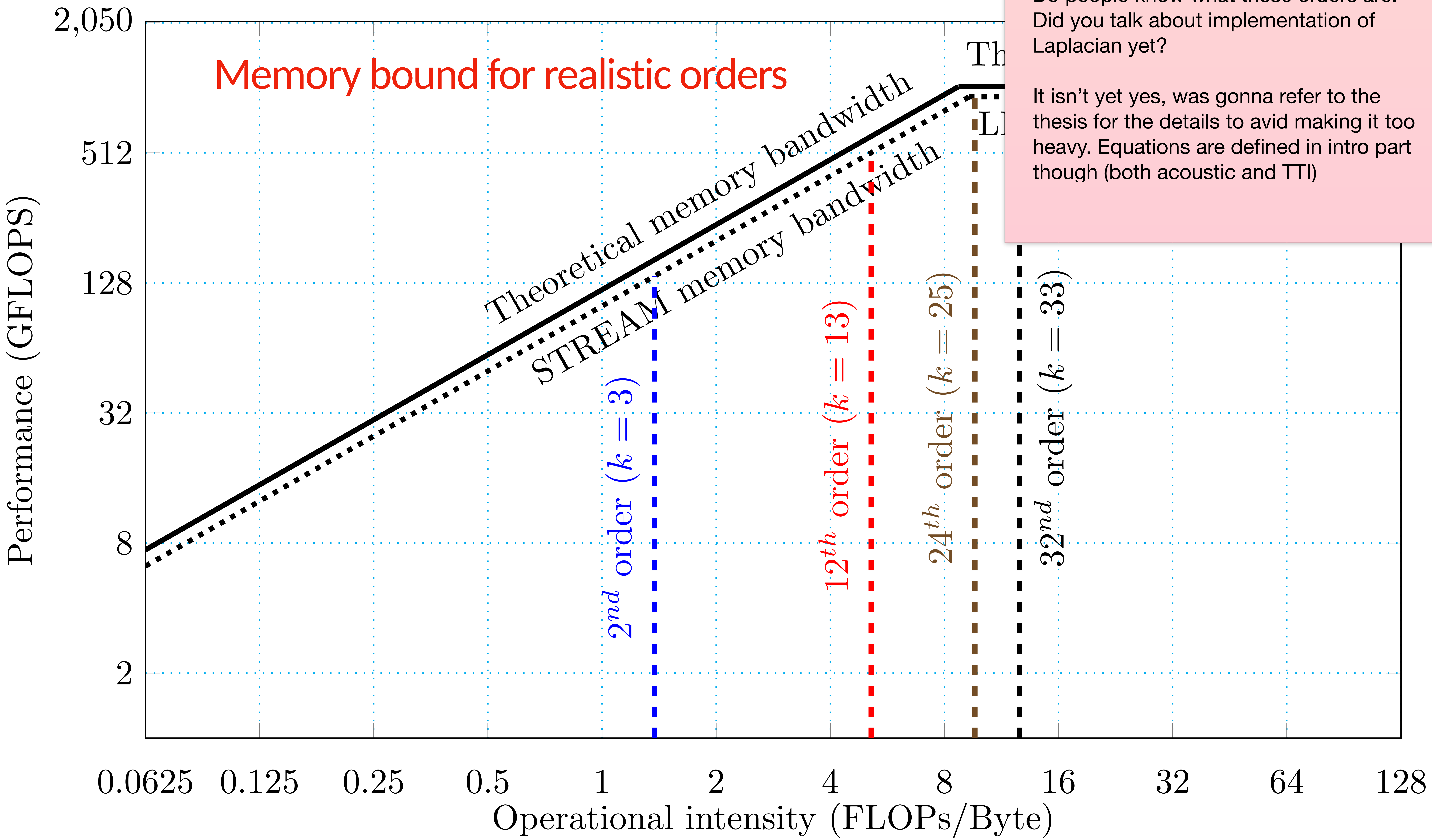
$\mathcal{F}_{kernel}(k)$: One addition + One multiplication

$$\mathcal{OI}_{alg}(k) = \frac{1+1}{4*1+2} = .17$$

Standard algorithms



Finite-differences estimates, acoustic

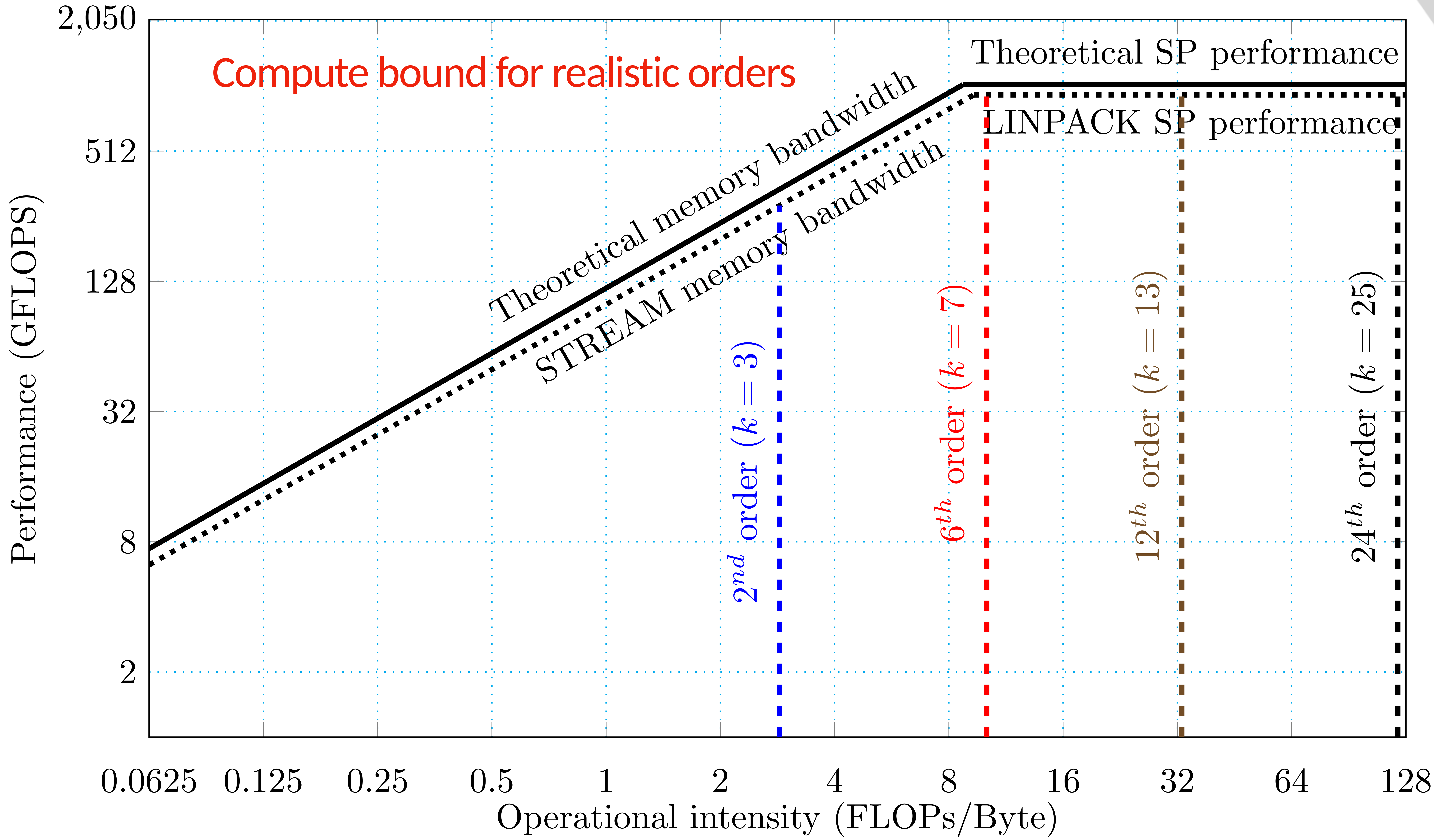


Do people know what these orders are. Did you talk about implementation of Laplacian yet?

It isn't yet yes, was gonna refer to the thesis for the details to avid making it too heavy. Equations are defined in intro part though (both acoustic and TTI)

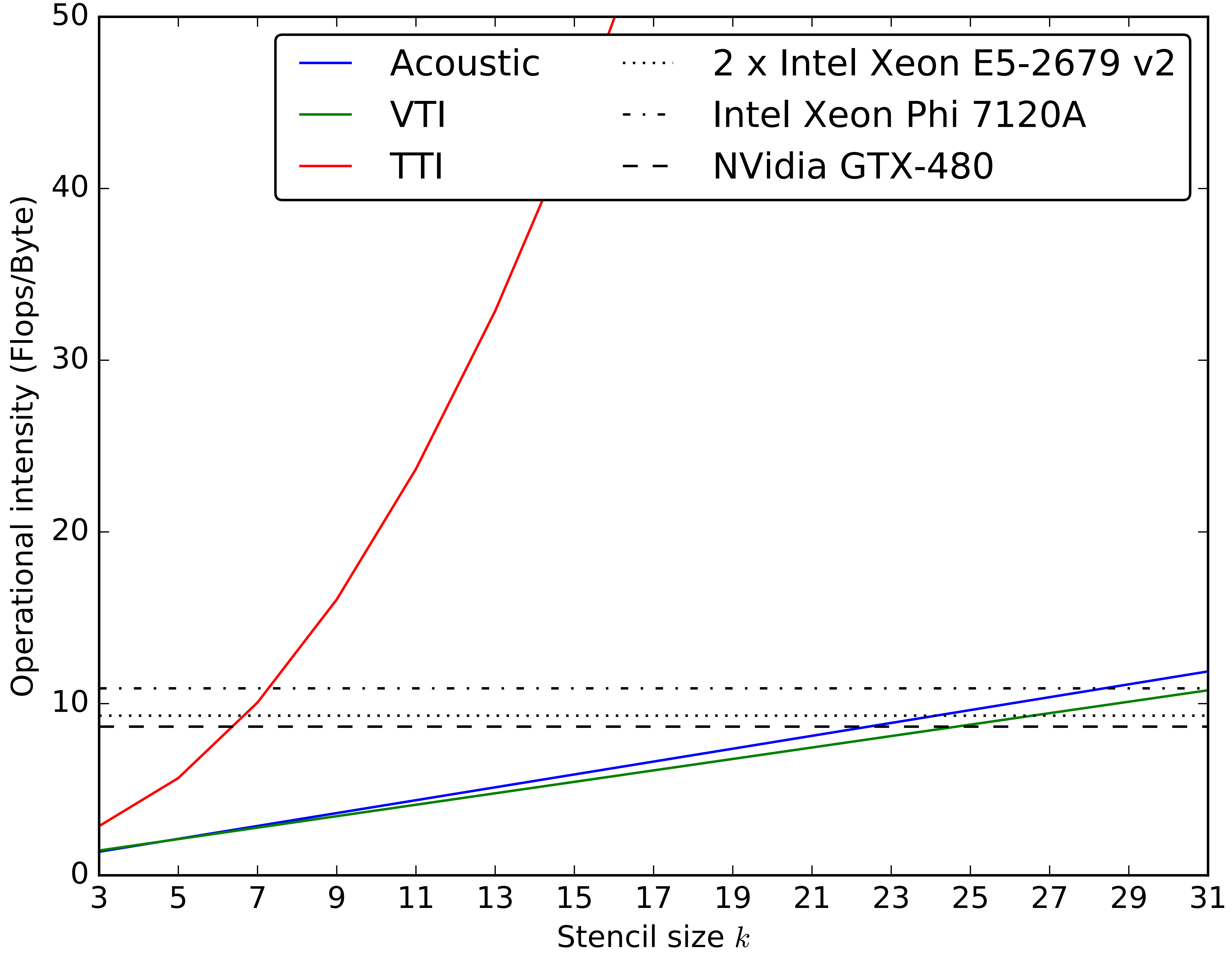
SLIM

Finite-differences estimates, TTI



Finite-differences

- Theoretical Operational Intensity
- Discretization dependent
- Physics dependent
- Architecture dependent



Conclusions

Performance prediction:

- can be estimated before implementation
- depends on wave-equation choice
- depends on discretization

One-fits-all not adapted to finite-difference codes

(Relative) Runtime not fully representative of performance

Needs flexible framework to take advantage of architecture

M. Louboutin, Lange, M., Luporini, F., Kukreja, N., Witte, P. A., Herrmann, F. J., Velesko, P., and Gorman, G. J., “**Devito: an embedded domain-specific language for finite differences and geophysical exploration**”
F. Luporini, Lange, M., Louboutin, M., Kukreja, N., Hückelheim, J., Yount, C., Witte, P. A., KELLY, P. A. U. L. H. J., Gorman, G. J., and Herrmann, F. J., “**Architecture and performance of Devito, a system for automated stencil computation**”

SLIM

Separation of concerns

Chapter 3 contributions:

- High level symbolic language
- Finite difference DSL

Chapter 4 contributions:

- Code generation
- Just-in-time compiler

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

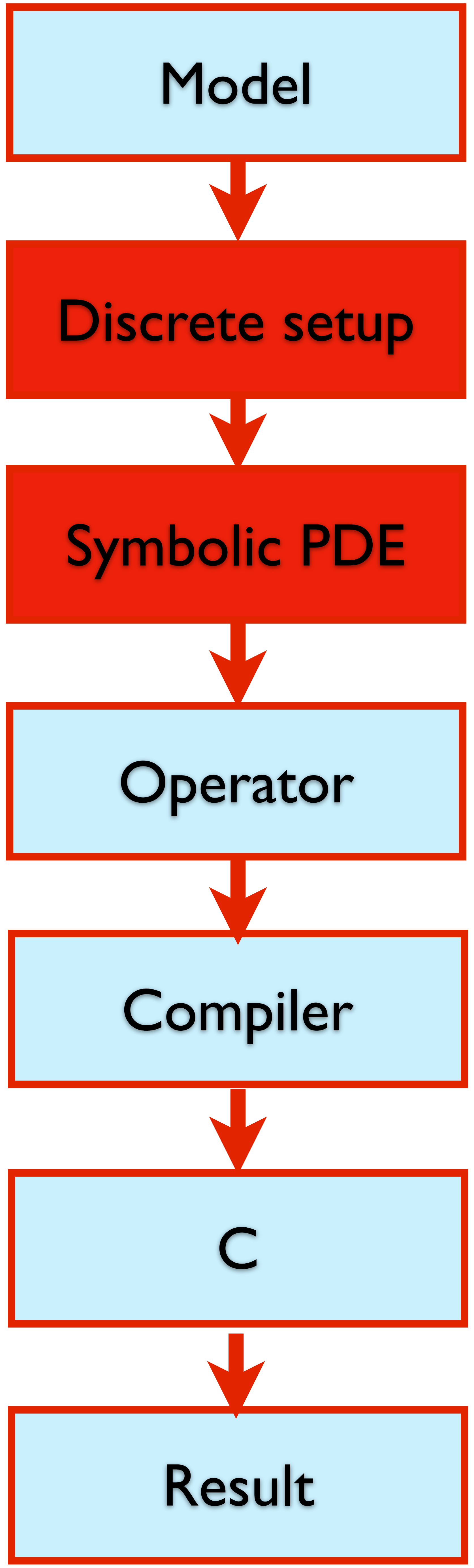
M. Louboutin, Lange, M., Luporini, F., Kukreja, N., Witte, P. A., Herrmann, F. J., Velesko, P., and Gorman, G. J., "**Devito: an embedded domain-specific language for finite differences and geophysical exploration**"

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

Model	Physical setup
↓	
Discrete setup	Symbolic discretization
↓	
Symbolic PDE	Symbolic stencil
↓	
Operator	Input/output processing and setup
↓	
Compiler	Stencil optimizations, memory setup, parallelisation, ..
↓	
C	Compile (gcc/clang/icc/…) and run
↓	
Result	

Chapter 3

M. Louboutin, Lange, M., Luporini, F., Kukreja, N., Witte, P. A., Herrmann, F. J., Velesko, P., and Gorman, G. J., “**Devito: an embedded domain-specific language for finite differences and geophysical exploration**”



Physical setup

Symbolic discretization

Symbolic stencil

Input/output processing and setup

Stencil optimizations, memory setup, parallelisation, ..

Compile (gcc/clang/icc/...) and run

Contributions

Design of the symbolic DSL for finite-differences

Implementation of a high-level user interface

Implementation and enabling of non standard operations

- Interpolation/injection
- Imaging condition

Design and implementation of finite-difference automation

Design and implementation of geophysical propagators

Implementation of inversion operators (adjoint/gradients/..)

Wave-equation setup

$$\mathbf{m}(x,y,z)\frac{d^2\mathbf{u}(t,x,y,z)}{dt^2} - \Delta\mathbf{u}(t,x,y,z) = 0$$

$$\mathbf{m}(x,y,z)\frac{d^2\mathbf{u}(t,x,y,z)}{dt^2} - \frac{d^2\mathbf{u}(t,x,y,z)}{dx^2} - \frac{d^2\mathbf{u}(t,x,y,z)}{dy^2} - \frac{d^2\mathbf{u}(t,x,y,z)}{dz^2} = 0$$

In Devito translates as:

```
u = TimeFunction(name="u",grid=grid, time_order=2,
                 space_order=spc_order)

m = Function(name="m", grid=model.grid)
equation = m * u.dt2 - u.laplace
```

Only parameter to change for a different spatial order

```
In [16]: equation = m * u.dt2 - u.laplace

In [17]: sympy.pprint(eq)
```

$$m(x, y, z)\cdot\frac{\partial^2}{\partial t^2}(u(t, x, y, z)) - \frac{\partial^2}{\partial x^2}(u(t, x, y, z)) - \frac{\partial^2}{\partial y^2}(u(t, x, y, z)) - \frac{\partial^2}{\partial z^2}(u(t, x, y, z))$$

Forward operator

Define the PDE

Solve symbolically for update rule

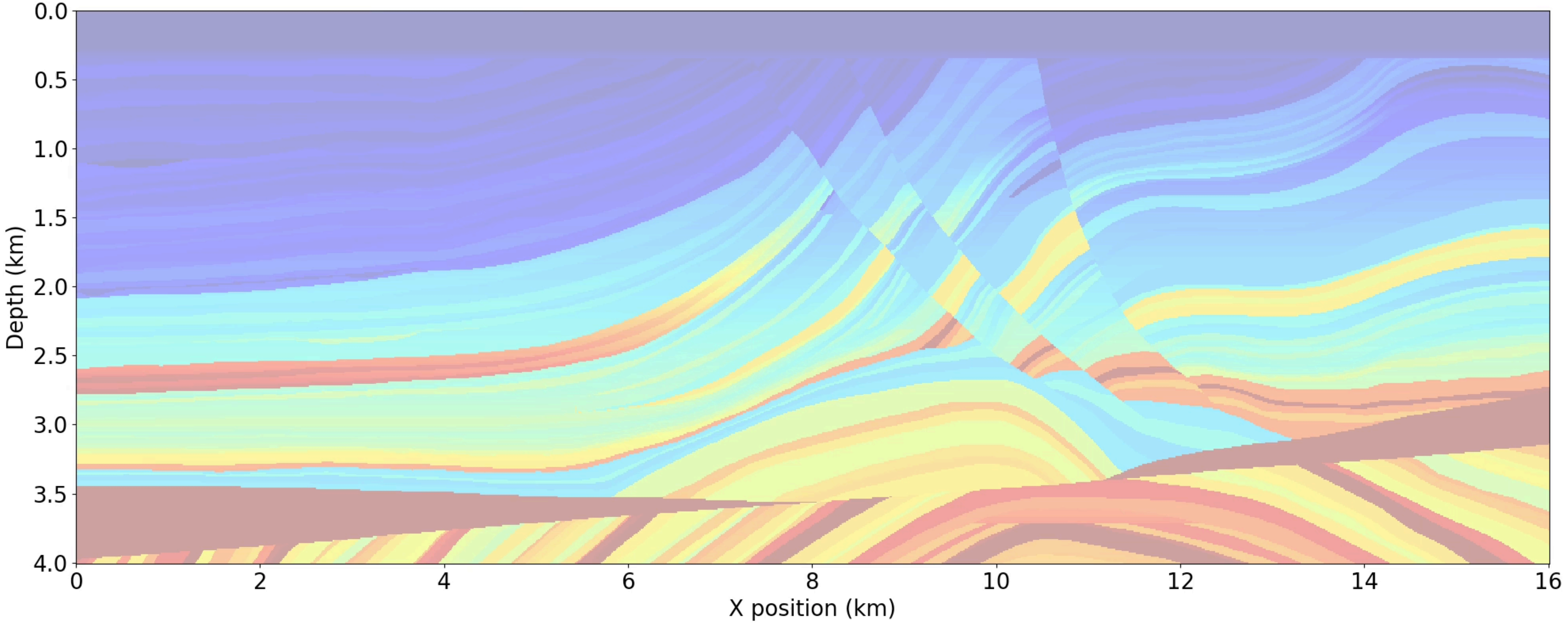
Create source injection and receiver sampling

Create Devito Operator

```
def forward(model, source, receiver,
     space_order=2):
    m, eta = model.m, model.damp
    # Allocate wavefield and auxiliary fields
    u = TimeFunction(name='u', grid=model.grid,
         time_order=2,
                    space_order=space_order)

    # Derive stencil from symbolic equation
    eqn = m * u.dt2 - u.laplace + eta * u.dt
    stencil = solve(eqn, u.forward)
    update_u = Eq(u.forward, stencil)

    # Source injection and receiver interpolation
    src = source.inject(field=u.forward,
         expr=src * dt**2 / m)
    rec = receiver.interpolate(expr=u)

    op = Operator([update_u] + src + rec,
         subs=model.spacing_map)
```

```
# Run the forward propagator op()
```

```
#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++)
            {
```


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)

Vectorial extension

Earth is elastic (better physics = better inversion)

Elastic is expensive

- Low velocities (half of acoustic) => 16 time bigger
- Nine fields
- Nine scalar equations (15 with viscosity)

Complex and tedious implementation

Vectorial extension

```
grid = Grid((1751, 2001, 1501), extent=(35000., 40000., 15000.))

# Elastic parameters
lam = Function(name="lam", grid=grid, space_order=0, is_parameter=True)
mu = Function(name="mu", grid=grid, space_order=0, is_parameter=True)
rho = Function(name="rho", grid=grid, space_order=0, is_parameter=True)

# Absorbing mask
damp = Function(name="damp", grid=grid, space_order=0, is_parameter=True)

# Stress and particle velocities
v = VectorTimeFunction(name="v", grid=grid, space_order=so, time_order=1)
tau = TensorTimeFunction(name="tau", grid=grid, space_order=so, time_order=1)

# symbol for dt
s = grid.time_dim.spacing

# Velocity stress formulation in its vectorial form
u_v = Eq(v.forward, damp * (v + s / rho * div(tau)))
u_t = Eq(tau.forward, damp *  (tau + s * (lam * diag(div(v.forward)) + mu * (grad(v.forward) + grad(v.forward).T))))
```

Without vectorial support

```
# Stress and particle velocities
~~v = VectorTimeFunction(name="v", grid=grid, space_order=so, time_order=1)~~
~~tau = TensorTimeFunction(name="tau", grid=grid, space_order=so, time_order=1)~~
```

9 fields

```
# Create symbols for forward wavefield, source and receivers
txx = TimeFunction(name='txx', grid=model.grid, staggered=stagg_xx, save=save, time_order=1, space_order=so)
tzz = TimeFunction(name='tzz', grid=model.grid, staggered=stagg_zz, save=save, time_order=1,  space_order=so)
tyy = TimeFunction(name='tyy', grid=model.grid, staggered=stagg_yy, save=save, time_order=1, space_order=so)
txz = TimeFunction(name='txz', grid=model.grid, staggered=stagg_xz, save=save, time_order=1, space_order=so)
txy = TimeFunction(name='txy', grid=model.grid, staggered=stagg_xy, save=save, time_order=1, space_order=so)
tyz = TimeFunction(name='tyz', grid=model.grid, staggered=stagg_yz, save=save, time_order=1, space_order=so)

# Create symbols for forward wavefield, source and receivers
vx = TimeFunction(name='vx', grid=model.grid, staggered=stagg_x, time_order=1, space_order=so)
vy = TimeFunction(name='vy', grid=model.grid, staggered=stagg_y, time_order=1, space_order=so)
vz = TimeFunction(name='vz', grid=model.grid, staggered=stagg_z, time_order=1, space_order=so)
```

```
# Velocity stress formulation in its vectorial form
~~u_v = Eq(v.forward, damp * (v + s / rho * div(tau)))~~
~~u_t = Eq(tau.forward, damp *  (tau + s * (lam * diag(div(v.forward)) + mu * (grad(v.forward) + grad(v.forward).T))))~~
```

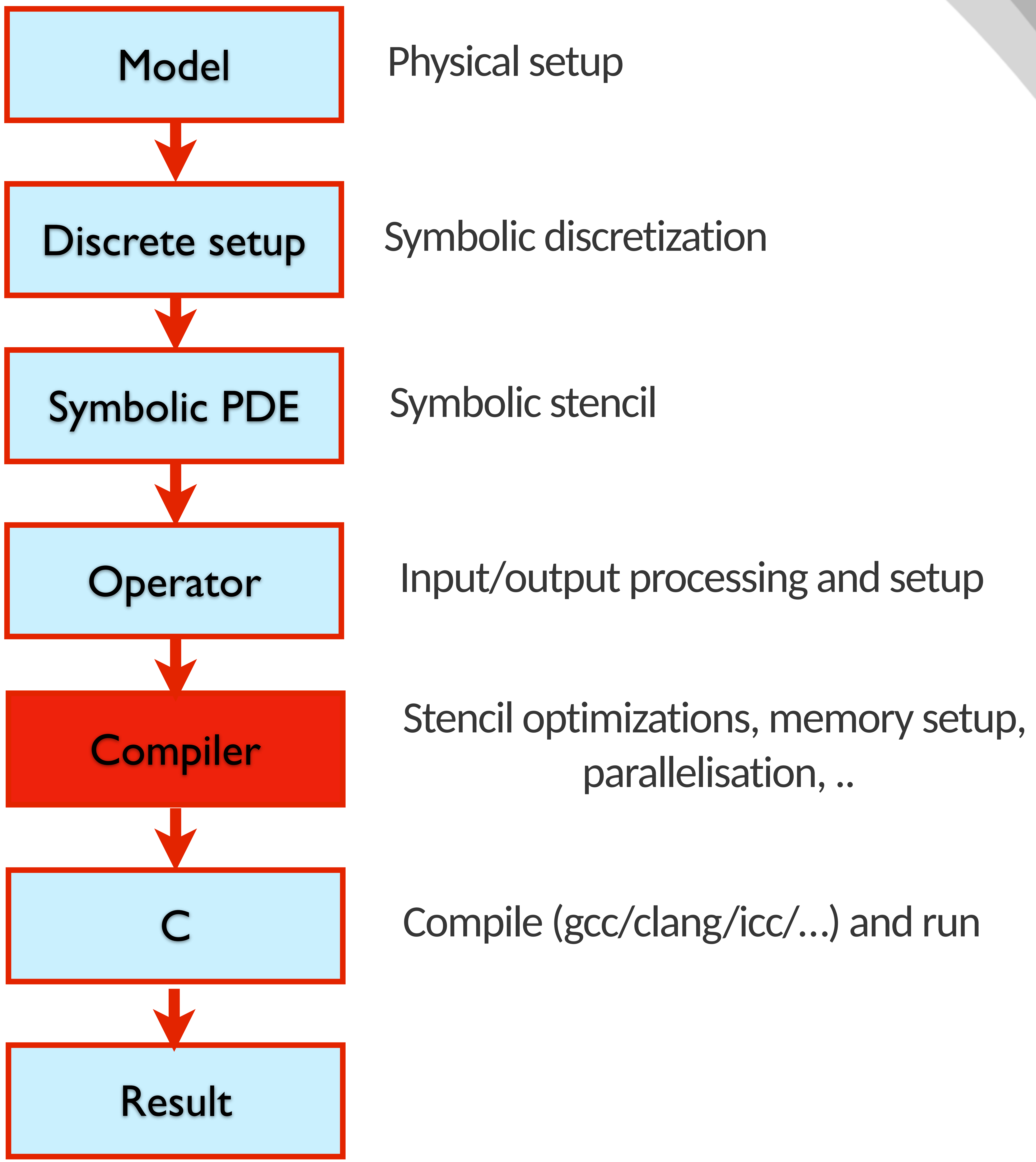
9 equations

```
u_vx = Eq(vx.forward, damp * vx - damp * s * ro * (txx.dx + txy.dy + txz.dz))
u_vy = Eq(vy.forward, damp * vy - damp * s * ro * (txy.dx + tyy.dy + tyz.dz))
u_vz = Eq(vz.forward, damp * vz - damp * s * ro * (txz.dx + tyz.dy + tzz.dz))

u_txx = Eq(txx.forward, damp * txx - damp * (l + 2 * mu) * s * vx.forward.dx - damp * l * s * (vy.forward.dy + vz.forward.dz))
u_tyy = Eq(tyy.forward, damp * tyy - damp * (l + 2 * mu) * s * vy.forward.dy - damp * l * s * (vx.forward.dx + vz.forward.dz))
u_tzz = Eq(tzz.forward, damp * tzz - damp * (l+2*mu)*s * vz.forward.dz - damp * l * s * (vx.forward.dx + vy.forward.dy))
u_txz = Eq(txz.forward, damp * txz - damp * mu * s * (vx.forward.dz + vz.forward.dx))
u_txy = Eq(txy.forward, damp * txy - damp * mu * s * (vy.forward.dx + vx.forward.dy))
u_tyz = Eq(tyz.forward, damp * tyz - damp * mu * s * (vy.forward.dz + vz.forward.dy))
```

v_x
v_z
$\tau_{xx} + \tau_{zz}$
τ_{xy}
Depth (m)
X (m)
0
500
1000
1500
2000
2500
3000
3500
0
2000
4000
6000
8000
10000
12000
14000
16000

Chapter 4

F. Luporini, Lange, M., Louboutin, M., Kukreja, N., Hückelheim, J., Yount, C., Witte, P. A., KELLY, P. A. U. L. H. J., Gorman, G. J., and Herrmann, F. J., “**Architecture and performance of Devito, a system for automated stencil computation**”



Contributions

Interfacing with the symbolic API

Testing framework and continuous integration

Performance evaluation and performance metric

Compiler components implementation

Symbolic manipulation and lowering

Devito compiler architecture

Analysis

Equation lowering

Preliminary transformations, such as

Constant folding
Index shifting (to account for halo and padding)
Subdimensions (to implement subdomains)

CORE specialization | OPS specialization | YASK specialization

Clustering

Group equations according to data dependencies

Symbolic optimization

Flop-reduction optimisation, such as:

Common sub-expressions elimination
Aliases detection and precomputation
Factorization

CORE specialization | OPS specialization | YASK specialization

Tree-fication

Abstract Syntax Tree (AST)

CORE specialization | OPS specialization | YASK specialization

AST analysis

Detect loop nest properties using advanced data dependence analysis, such as:

Parallel vs sequential
SIMD-vectorizability
Affine vs non-affine
Communication-avoiding opportunities

AST optimization

Loop nest optimization, such as

Loop blocking
SIMD vectorization
Distributed-memory parallelism (e.g., comp/comm overlap)
Shared-memory parallelism

CORE specialization | OPS specialization | YASK specialization

Finalization

Type and variable declarations, instrumentation for profiling, header files, globals, macros, …

CORE specialization | OPS specialization | YASK specialization

SLIM

Example step:
Data dependency analysis

Represents data (array) accesses as (labelled) vectors in $\mathbb{Z}^n$

$$u[t+2, x-3] \rightarrow \begin{bmatrix} t+2 \\ x-3 \end{bmatrix}$$

Currently uses a simple in-house framework

The Devito compiler relies on data dependence analysis for many tasks

- Inferring the iteration direction (i++ or i-- ?)
- Loop scheduling (convert list of equations into a tree of loops)
- Loop optimizations (e.g., discovery of parallel loops)

Example step:
topological sorting for maximal "loop fusion"

```
u[t+1,x] = F0(u[t,x], u[t-1,x], v[t,x], …)

u[t+1,s] = F1(u[t+1,u_coords[s]], …)

v[t+1,x] = F2(v[t,x], v[t-1,x], u[t,x], …)

v[t+1,s] = F3(u[t+1,v_coords[s]], …)
```

Only flow-dependences in the time (t) dimension!

Example step: Generalized Common Sub-expressions Elimination

```
a[t, i, j] = sin(phi[i, j]) + sin(phi[i-1, j-1]) + sin(phi[i+2, j+2])
```

Observations:

- Same operators (**sin**), same operands (**phi**), same indices (**i**, **j**)
- Linearly dependent index vectors (**[i, j]**, **[i-1, j-1]**, **[i+2, j+2]**)

```
B[i,j] = sin(phi[i,j])

a[t, i, j] = B[i,j] + B[i-1,j-1] + B[i+2,j+2]
```

Trade-off FLOPs/storage

Extract Time invariants

20 X flop reduction for TTI

FD order	Flops dse=noop	Flops dse=basic	Flops dse=advanced	Flops dse=aggressive
2	501	217	175	95
4	539	301	238	102
8	1613	860	653	160
16	**5489**	2839	2131	**276**

dse: Devito Stencil Engine, performs stencil optimisations

MPI support — for free

Python-level features:

- Domain decomposition based on MPI Cartesian grid abstraction.
- New package numpy4mpi: NumPy arrays automatically split and distributed according to domain decomposition.
- Parallel data slicing dealt with efficiently under the hood
- Sources/receivers (i.e. “sparse data”) distributed automatically.

MPI support — for free

C-level features:

- Generated code contains the MPI halo exchanges. Required halo exchanges identified through data dependence analysis.
- Optimizations (e.g., reshuffling halo updates, computation/ communication overlap) also exploit data dependence analysis.
- Data packing/unpacking is threaded for performance.

- M. Louboutin, Lange, M., Herrmann, F. J., Kukreja, N., and Gorman, G., "**Performance prediction of finite-difference solvers for different computer architectures**", Computers & Geosciences, vol. 105, pp. 148–157, 2017.DOI BibTeX
- F. Luporini, Lange, M., Louboutin, M., Kukreja, N., Hückelheim, J., Yount, C., Witte, P. A., KELLY, P. A. U. L. H. J., Gorman, G. J., and Herrmann, F. J., "**Architecture and performance of Devito, a system for automated stencil computation**"

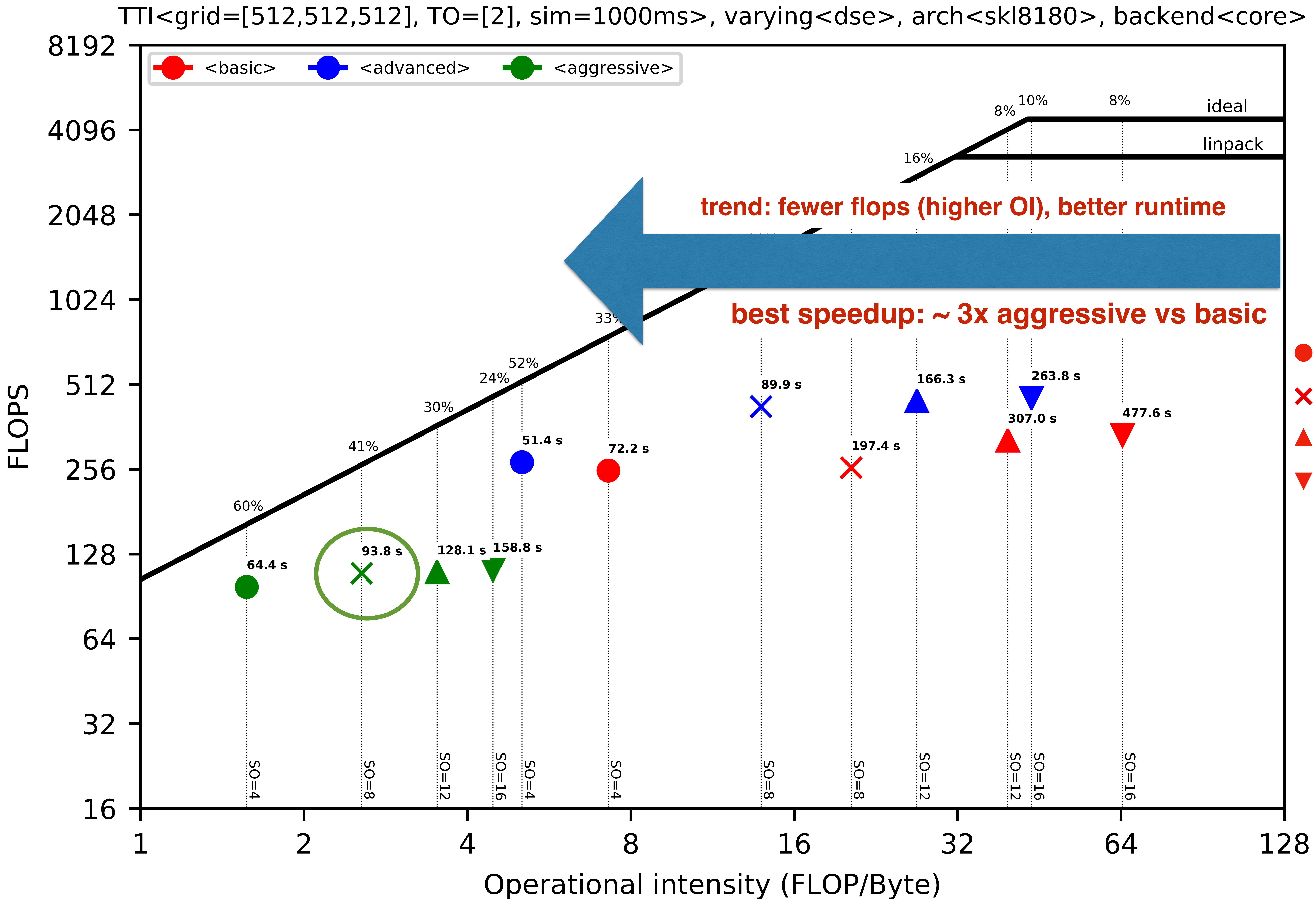
SLIM

Performance evaluation

Roofline analysis

Strong scaling

Single-socket — TTI on Skylake 8180 for multiple optimization levels

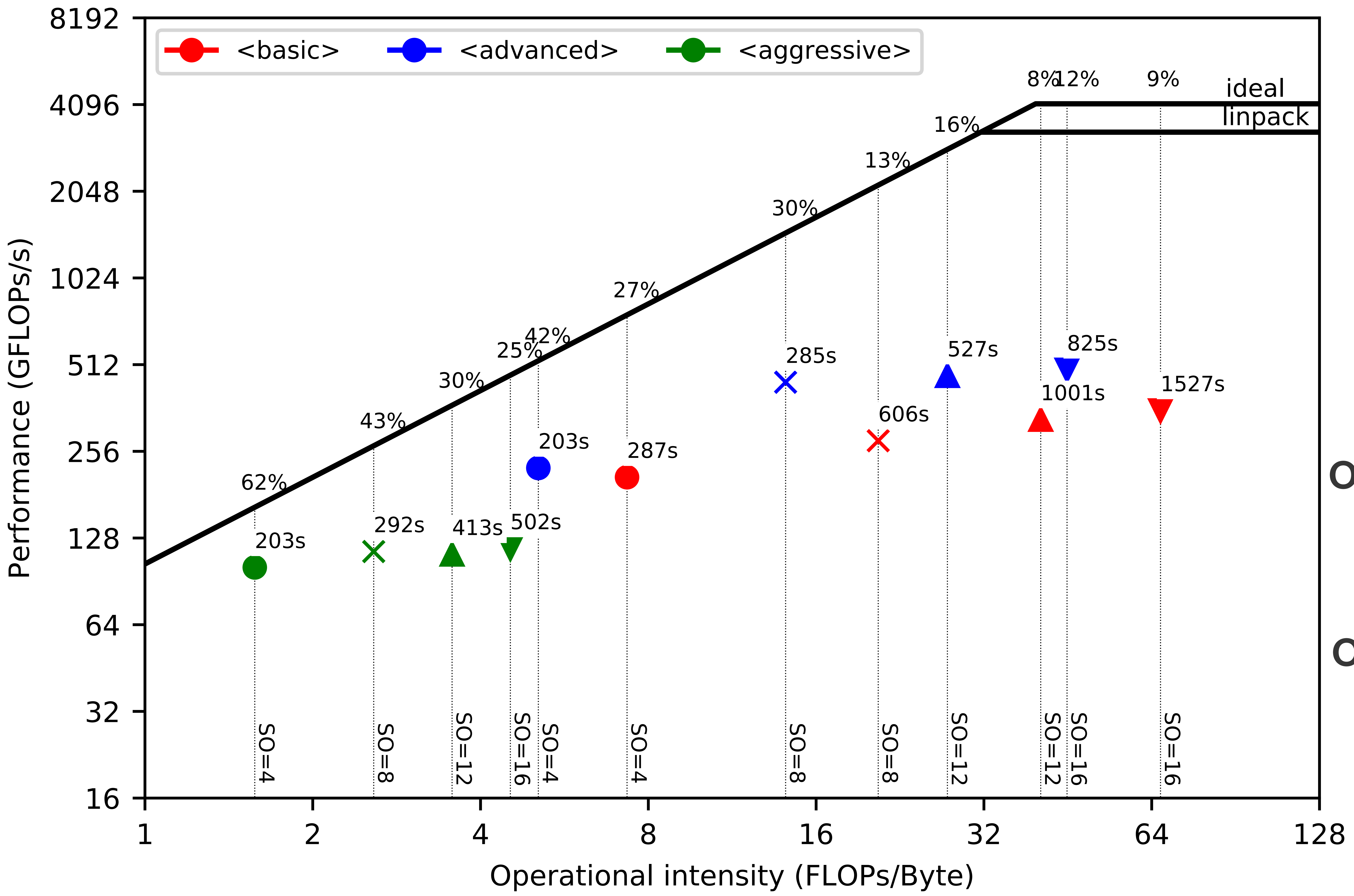


4th order FD
8th order FD
12th order FD
16th order FD

3D TTI performance:

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

Scales linearly!

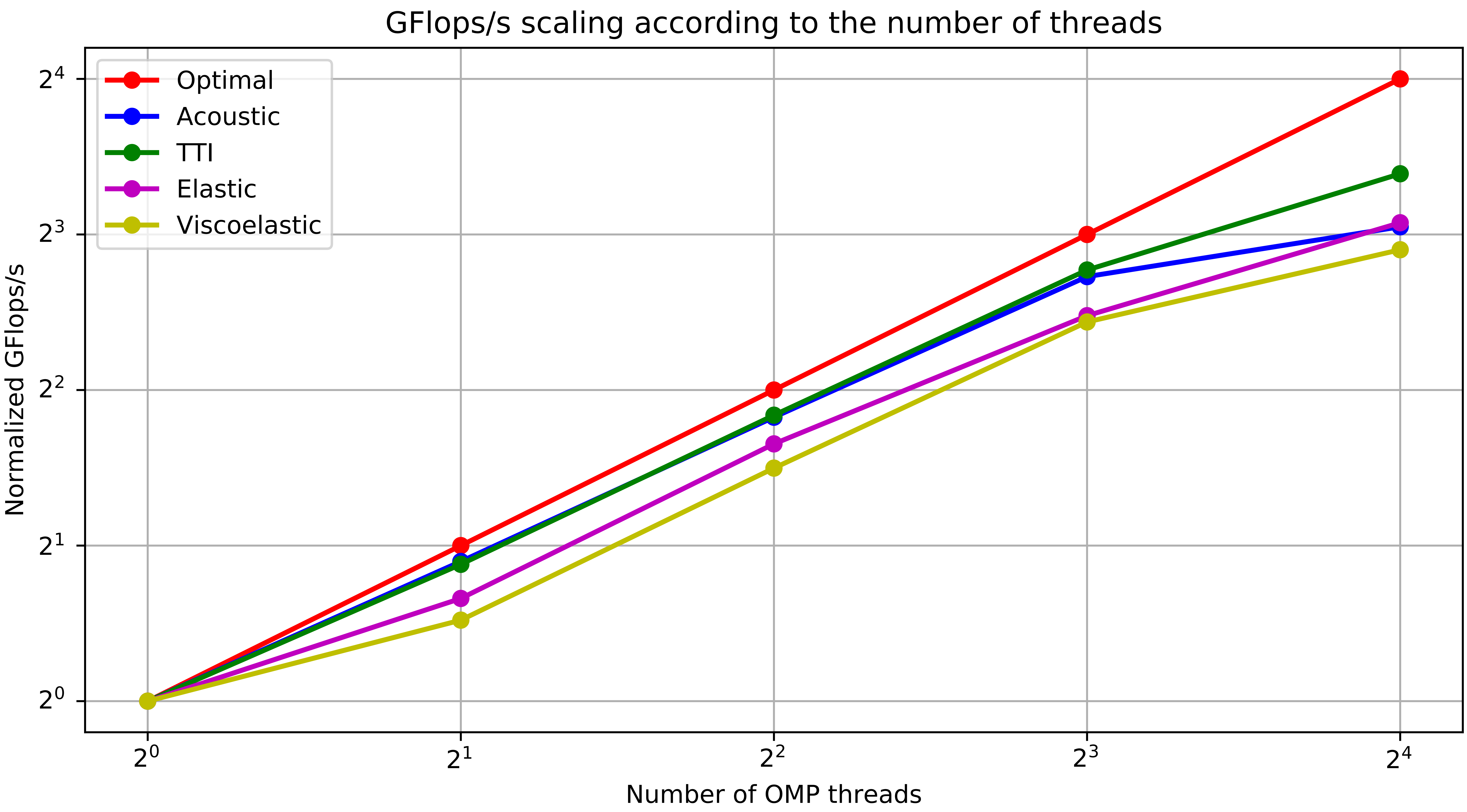


$512^3 \rightarrow 768^3$
3.4 X larger grid

Order 16: 72s → 287s
3.9 X slower

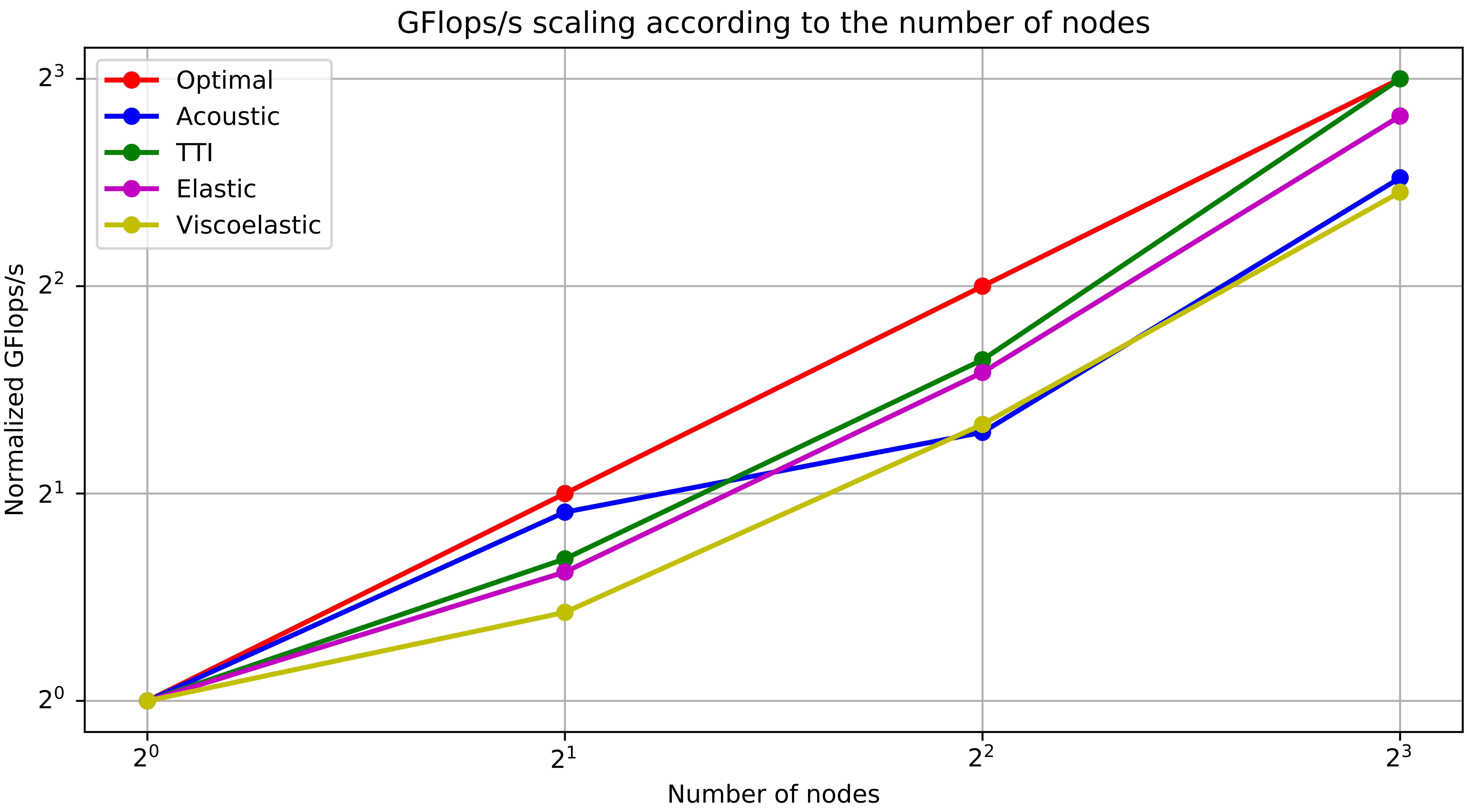
Order 8: 93s → 292s
3.2 X slower

OpenMP (multi-threading) strong scaling



MPI strong scaling

Plots are inconsistent style



Contributions

- Theoretical analysis of finite-difference solvers
- Flexible and portable performance metric
- Symbolic finite-difference DSL
- Enables research and development for large scale inverse problems
- Reproducible and verifiable research
- Just-in-time compiler for high-performance computing
- Hardware portability

Philipp A. Witte, Mathias Louboutin, Navjot Kukreja, Fabio Luporini, Michael Lange, Gerard J. Gorman, and Felix J. Herrmann, “**A large-scale framework for symbolic implementations of seismic inversion algorithms in Julia**”. 2018.

JUDI

JUDI – Domain specific language for linear algebra abstractions, data parallelism & meta data in Julia

https://github.com/slimgroup/JUDI.jl

JUDI – true vertical integration

Julia

linear opeartors, data containers, IO

↓ parallel modeling function

parallelization: distribute sources, data

↓ serial modeling function

interface to Devito (Python)

↓

Python

Devito: symbolic definition of PDE

↓ automatic code generation and JIT compilation

C

solve PDE w/ OpenMP parallelism

students

math/optimizers/cs/
seismic practitioners

students

CS/math/physics
people

polyhydral compiler
people

Example: LS-RTM w/ serial & parallel SGD

Algorithm 1 Preconditioned LS-RTM with SGD

for $j = 1$ to n **do**

$\mathbf{r}_j = \mathbf{M}_l^{-1}\mathbf{J}_{r(j)}\mathbf{M}_r^{-1}\mathbf{x}_j - \mathbf{M}_l^{-1}\delta\mathbf{d}_{r(j)}$

$\mathbf{g}_j = \mathbf{M}_r^{-\top}\mathbf{J}_{r(j)}^{\top}\mathbf{M}_l^{-\top}\mathbf{r}_j$

$t_j = \frac{\|\mathbf{r}_j\|^2}{\|\mathbf{g}_j\|^2}$

$\mathbf{x}_{j+1} = \mathbf{x}_j - t_j\mathbf{g}_j$

end for

```
# Stochastic gradient descent
batchsize = 10
niter = 32

for j=1:niter
   # Select batch
   idx = randperm(dD.nsrc)[1:batchsize]
   Jsub = subsample(J,idx)
   dsub = subsample(dD,idx)

   # Compute residual and gradient
   r = Ml*Jsub*Mr*x - Ml*dsub
   g = Mr'*Jsub'*Ml'*r

   # Step size and update variable
   t = norm(r)^2/norm(g)^2
   x -= t*g
end
```

3D TTI RTM on Azure

Synthetic model based on SEG Overthrust + Salt models:

- Domain: 10 x 10 x 3.325 km
- Grid: 881 x 881 x 347 (12.5 m grid + ABCs)
- **Wide-azimuth acquisition w/ 1,500 randomly distributed OBNs**
- 799 x 799 dense source grid (12.5 m)
- Anisotropic TTI models + density
- Used source-receiver reciprocity

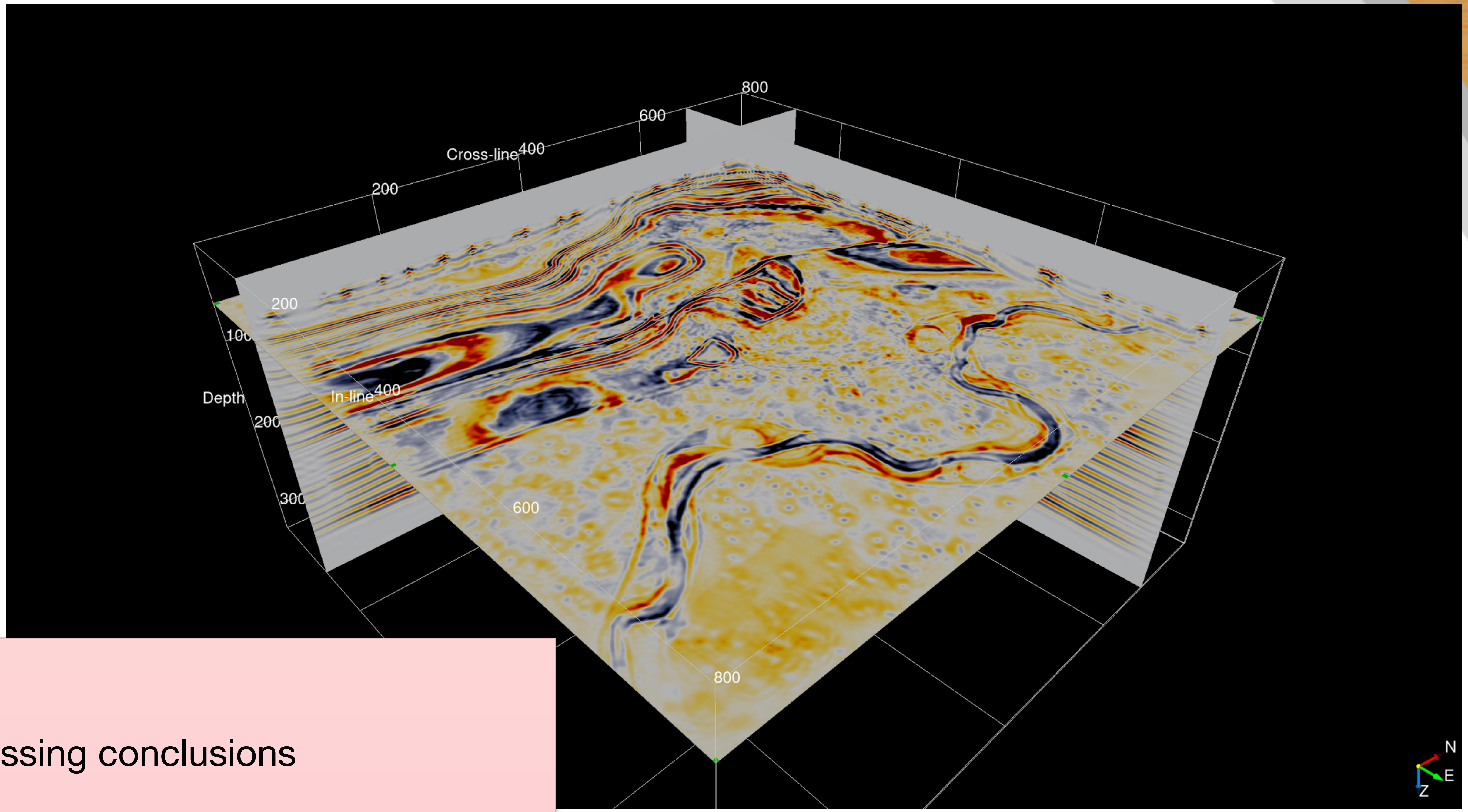
3D TTI imaging on Azure

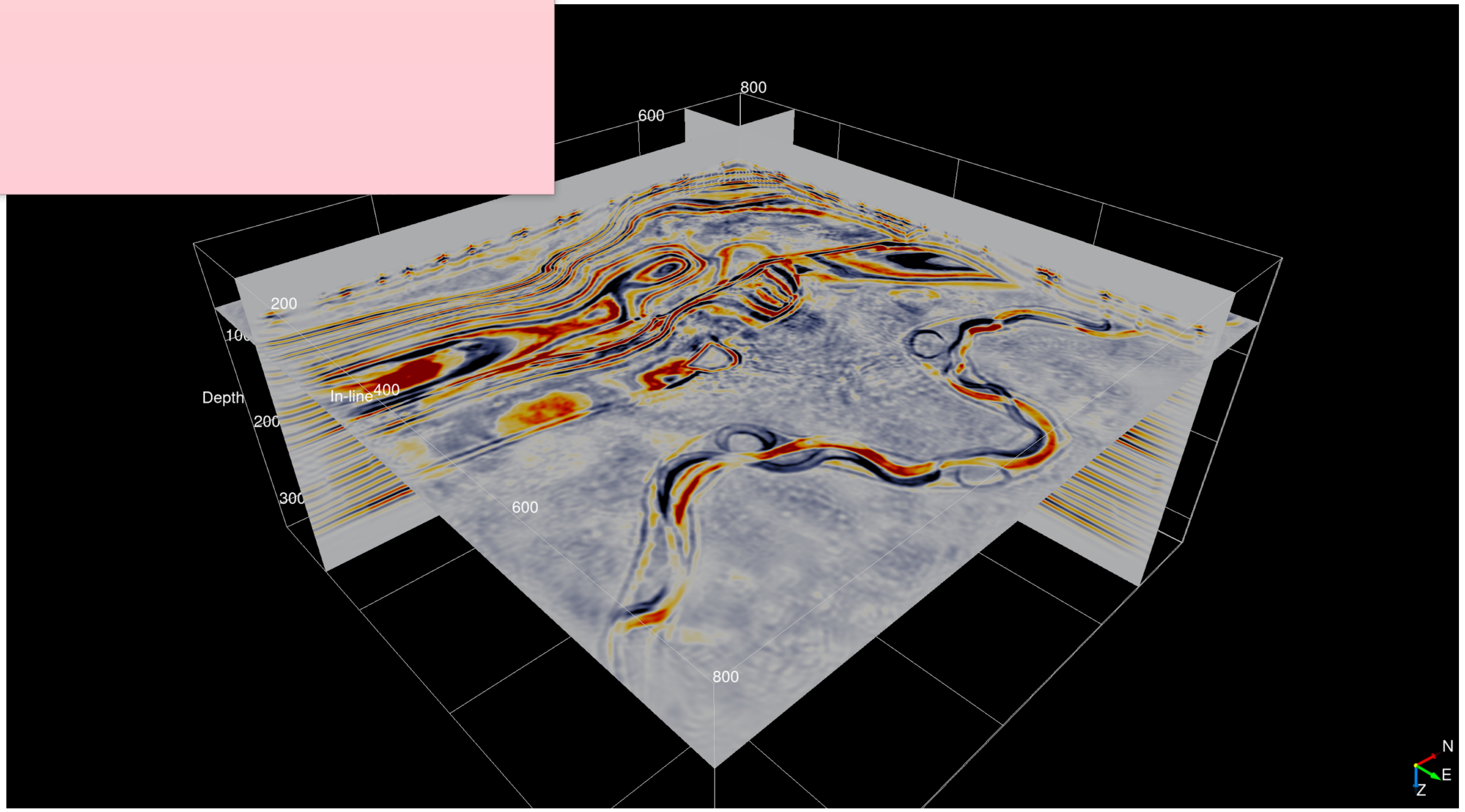
Industry scale imaging

Realistic physical representatio

Demonstrates Devito portability

I am missing conclusions





Conclusions

Modeling for inversion with separation of concern

- High level abstractions enable research and development

- Just-in-time compiler provides portability and reproducibility

- Roofline model gives rigorous and absolute benchmarks

P. A. Witte, Louboutin, M., Luporini, F., Gorman, G. J., and Herrmann, F. J., "**Compressive least squares migration with on-the-fly Fourier transforms**", in SIAM Conference on Computational Science and Engineering, 2019.BibTeX

G. Rizzuti, Louboutin, M., Wang, R., Daskalakis, E., and Herrmann, F. J., "**A dual formulation for time-domain wavefield reconstruction inversion**", in SIAM Conference on Mathematical & Computational Issues in the Geosciences, 2019.

A. Siahkoohi, Louboutin, M., Kumar, R., and Herrmann, F. J., "**Deep Convolutional Neural Networks in prestack seismic–-two exploratory examples**", in SEG Technical Program Expanded Abstracts, 2018, pp. 2196–2200.DOI BibTeX

M. Louboutin, Witte, P. A., and Herrmann, F. J., "**Effects of wrong adjoints for RTM in TTI media**", in SEG Technical Program Expanded Abstracts, 2018, pp. 331–335.DOI

S. Sharan, Kumar, R., Dumani, D. S., Louboutin, M., Wang, R., Emelianov, S., and Herrmann, F. J., "**Sparsity-promoting photoacoustic imaging with source estimation**", in 2018 IEEE International Ultrasonics Symposium (IUS), 2018, pp. 206–212.DOI BibTeX

M. Louboutin, Witte, P. A., Lange, M., Kukreja, N., Luporini, F., Gorman, G., and Herrmann, F. J., "**Full-Waveform Inversion - Part 2: adjoint modeling**", The Leading Edge, vol. 37, pp. 69–72, 2018.DOIBibTeX

P. A. Witte, Louboutin, M., Lensink, K., Lange, M., Kukreja, N., Luporini, F., Gorman, G., and Herrmann, F. J., "**Full-Waveform Inversion - Part 3: optimization**", The Leading Edge, vol. 37, pp. 142–145, 2018.DOIBibTeX

P. A. Witte, Louboutin, M., Luporini, F., Gorman, G. J., and Herrmann, F. J., "**Compressive least-squares migration with on-the-fly Fourier transforms**". 2018.BibTeX

P. A. Witte, Louboutin, M., Kukreja, N., Luporini, F., Lange, M., Gorman, G. J., and Herrmann, F. J., "**A large-scale framework for symbolic implementations of seismic inversion algorithms in Julia**". 2018.BibTeX

M. Louboutin, Lange, M., Luporini, F., Kukreja, N., Witte, P. A., Herrmann, F. J., Velesko, P., and Gorman, G. J., "**Devito: an embedded domain-specific language for finite differences and geophysical exploration**". 2018.BibTeX

F. Luporini, Lange, M., Louboutin, M., Kukreja, N., Hückelheim, J., Yount, C., Witte, P. A., KELLY, P. A. U. L. H. J., Gorman, G. J., and Herrmann, F. J., "**Architecture and performance of Devito, a system for automated stencil computation**". 2018.BibTeX

M. Louboutin, Witte, P. A., Lange, M., Kukreja, N., Luporini, F., Gorman, G., and Herrmann, F. J., "**Full-Waveform Inversion - Part 1: forward modeling**", The Leading Edge, vol. 36, pp. 1033–1036, 2017.DOIBibTeX

M. Louboutin, Lange, M., Herrmann, F. J., Kukreja, N., and Gorman, G., "**Performance prediction of finite-difference solvers for different computer architectures**", Computers & Geosciences, vol. 105, pp. 148–157, 2017.DOI BibTeX

M. Lange, Kukreja, N., Luporini, F., Louboutin, M., Yount, C., Hückelheim, J., and Gorman, G., "**Optimised finite difference computation from symbolic equations**", in Python in Science Conference Proceedings, 2017, pp. 89–96.BibTeX

M. Louboutin and Herrmann, F. J., "**Extending the search space of time-domain adjoint-state FWI with randomized implicit time shifts**", in EAGE Annual Conference Proceedings, 2017.DOI BibTeX

M. Louboutin, Guasch, L., and Herrmann, F. J., "**Data normalization strategies for full-waveform inversion**", in EAGE Annual Conference Proceedings, 2017.DOI BibTeX

N. Kukreja, Louboutin, M., Lange, M., Luporini, F., and Gorman, G., "**Leveraging symbolic math for rapid development of applications for seismic modeling**", in OGHPC, 2017.BibTeX