Latest developments in Devito

Mathias Louboutin, Michael Lange*, Fabio Luporini*, Navjot Kurjeka*, Jan Hueckelheim*, Gerard Gorman*, Philipp Witte and Felix Herrmann

SINBAD consortium meeting
Wednesday October 4th

* Imperial College London

SLIM
University of British Columbia

Wave-equation based geophysical exploration

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

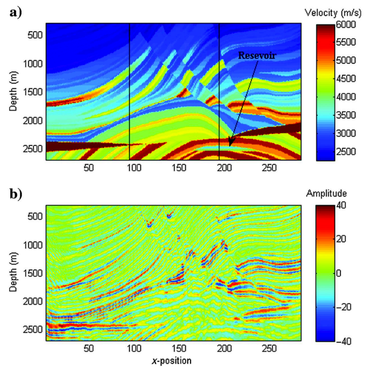
$\mathbf{q}$: source term

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

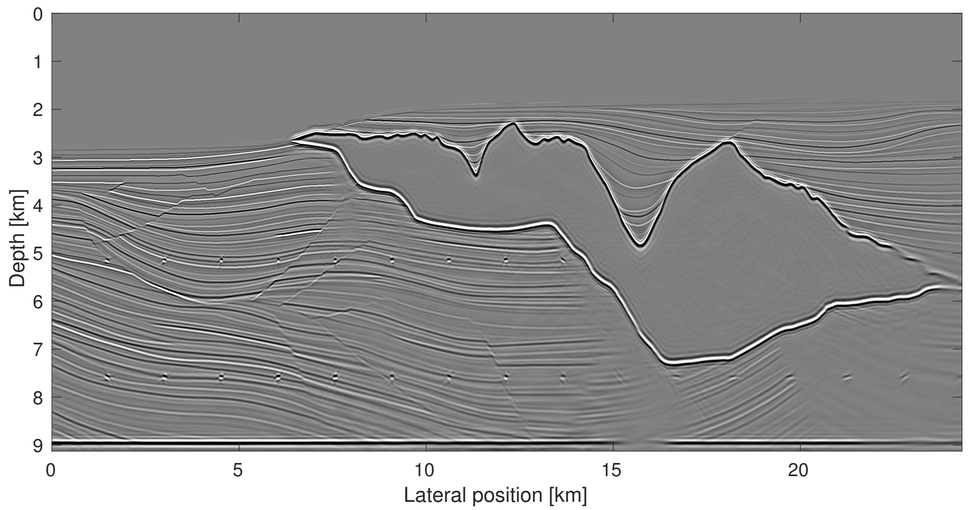
$\mathbf{P}_s$: projection onto the source location

Challenges

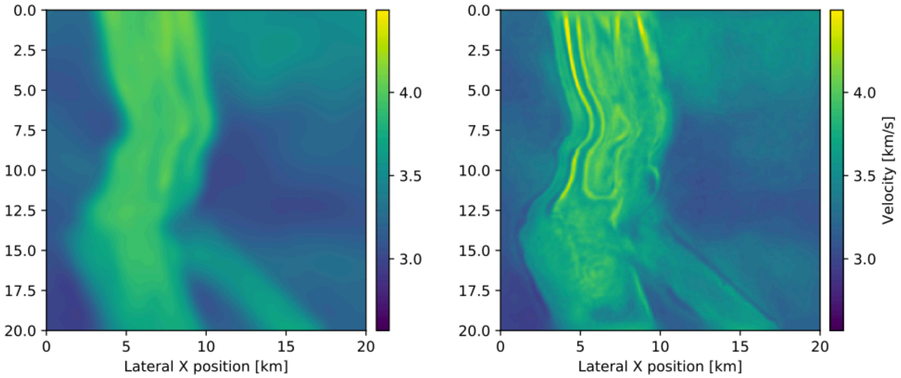
- Problem sizes are huge:
  - Seismic surveys consist of tens of thousands of individual experiments
  - Model wave propagation over thousands of time steps in large domains
  - Typical size of modeling matrix → $\mathbf{A}(\mathbf{m}) \in \mathbb{R}^{n \times n}, n = \mathcal{O}(1e13)$
- Multiple/complex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



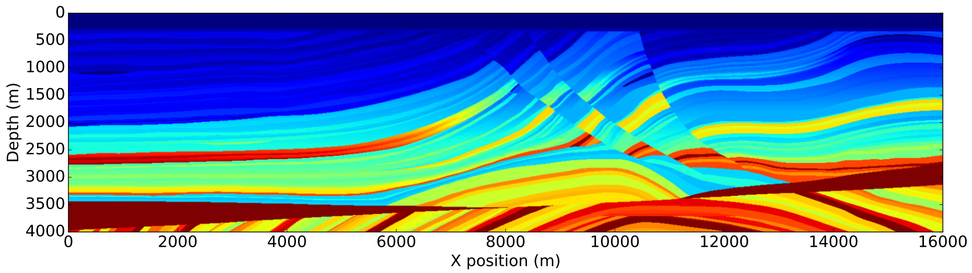
Tiny marmousi
62k gridpoints
1.5MFlop/time-step



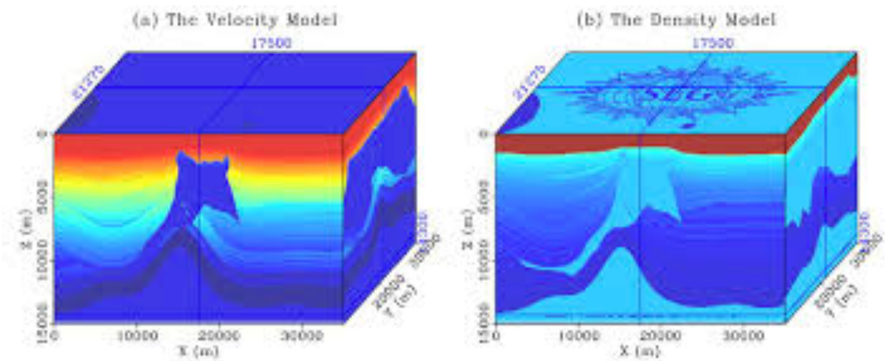
Sigsbee
2.2M gridpoints
56MFlop/time-step



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



Full marmousi
640k gridpoints
15MFlop/time-step



SEAM
2.2G gridpoints
56GFlop/time-step

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

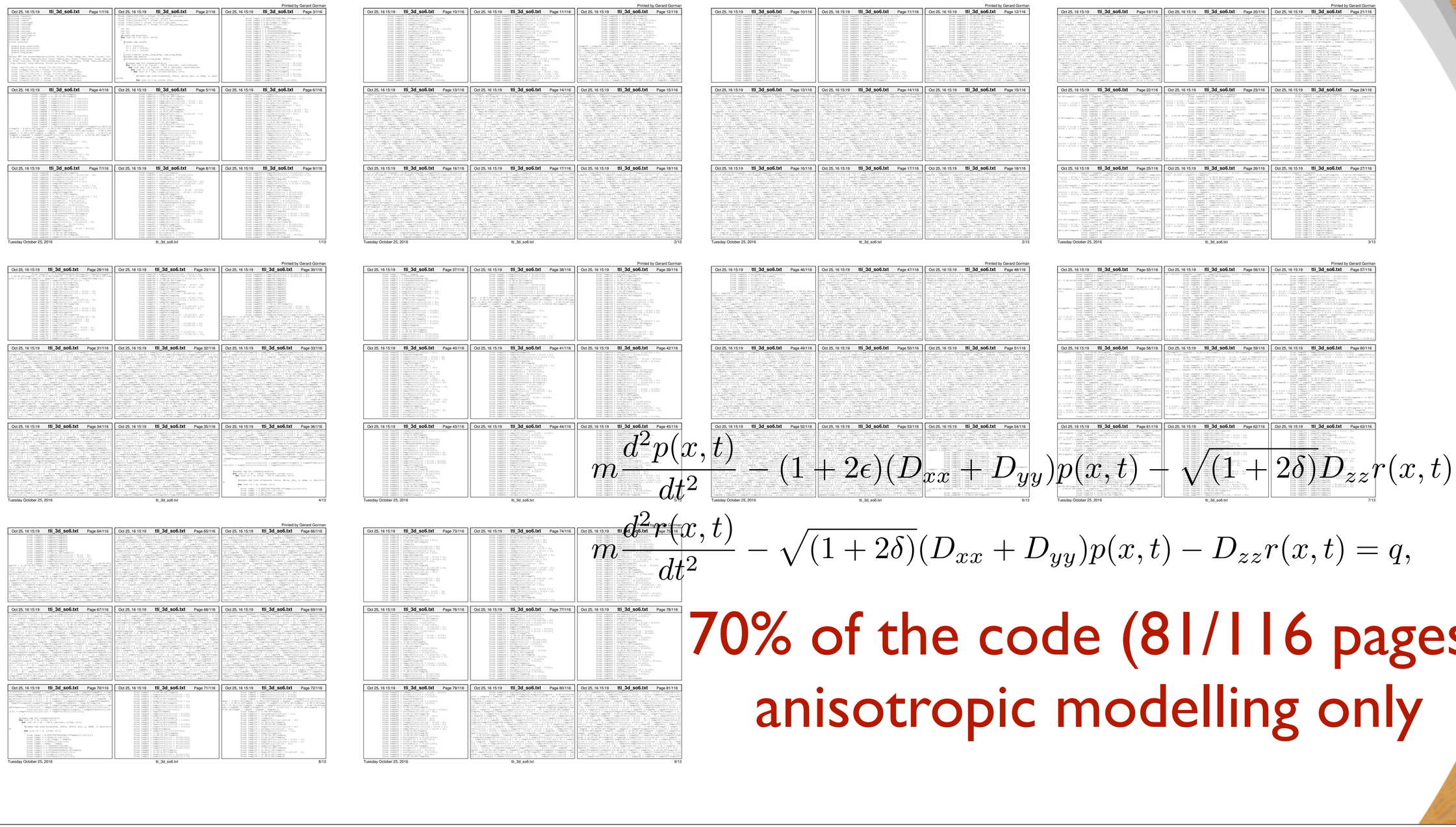
$$m\frac{d^2p(x,t)}{dt^2} - (1+2\epsilon)(G_{\bar{x}\bar{x}} + G_{\bar{y}\bar{y}})p(x,t) - \sqrt{(1+2\delta)}G_{\bar{z}\bar{z}}r(x,t) = q,$$
$$m\frac{d^2r(x,t)}{dt^2} - \sqrt{(1+2\delta)}(G_{\bar{x}\bar{x}} + G_{\bar{y}\bar{y}})p(x,t) - G_{\bar{z}\bar{z}}r(x,t) = q,$$

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)

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

Design motivation

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

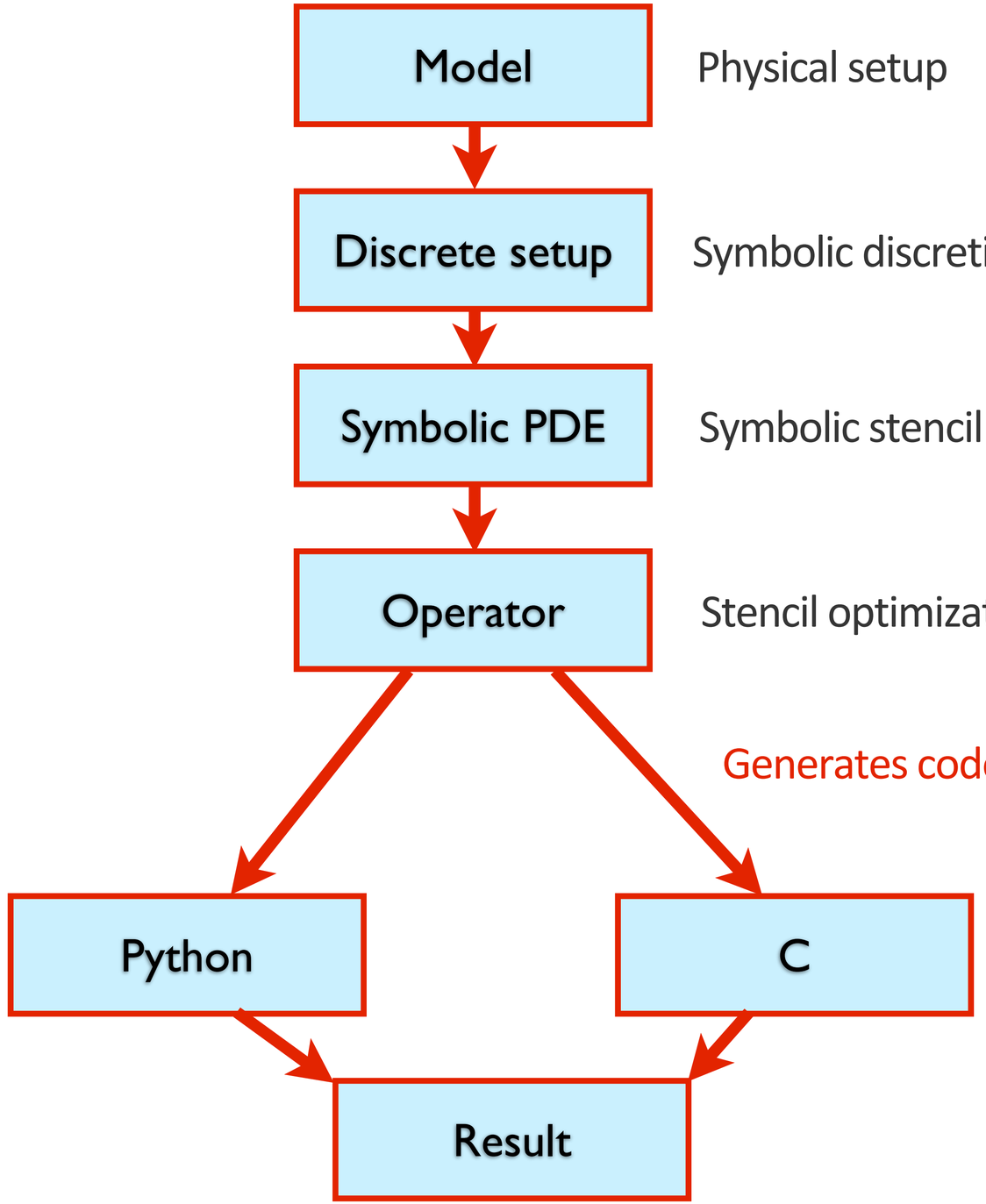
Finite-difference DSL

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

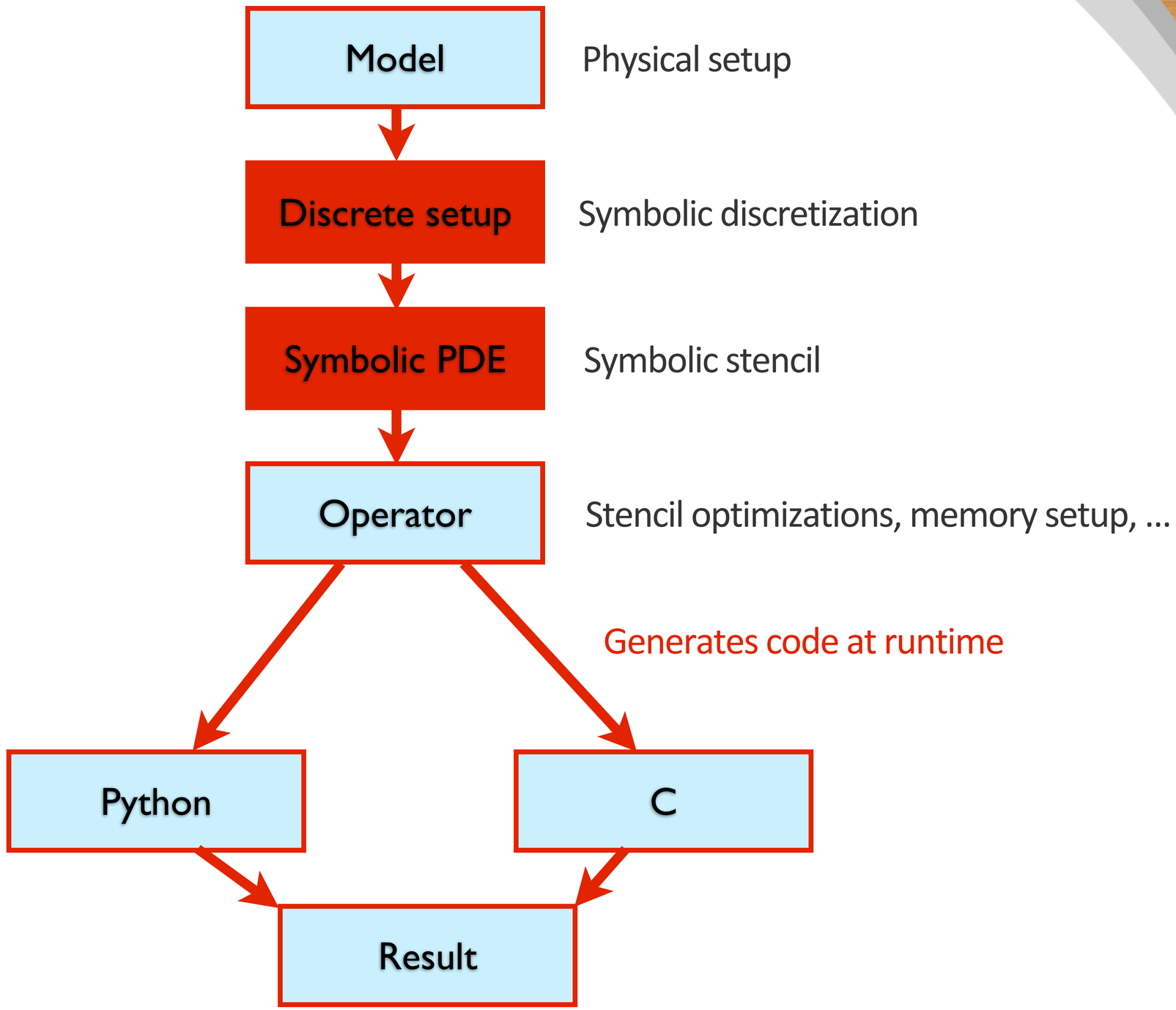
Devito

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

Symbolic discretization

Symbolic object with finite-difference discretization as a property

```
u =TimeData(name="u", shape=(nx, ny, nz),
            time_order=self.t_order,
            space_order=self.s_order)
```

is a symbolic object with derivatives properties

```
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

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

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

Acoustic 4th order in time

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

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

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

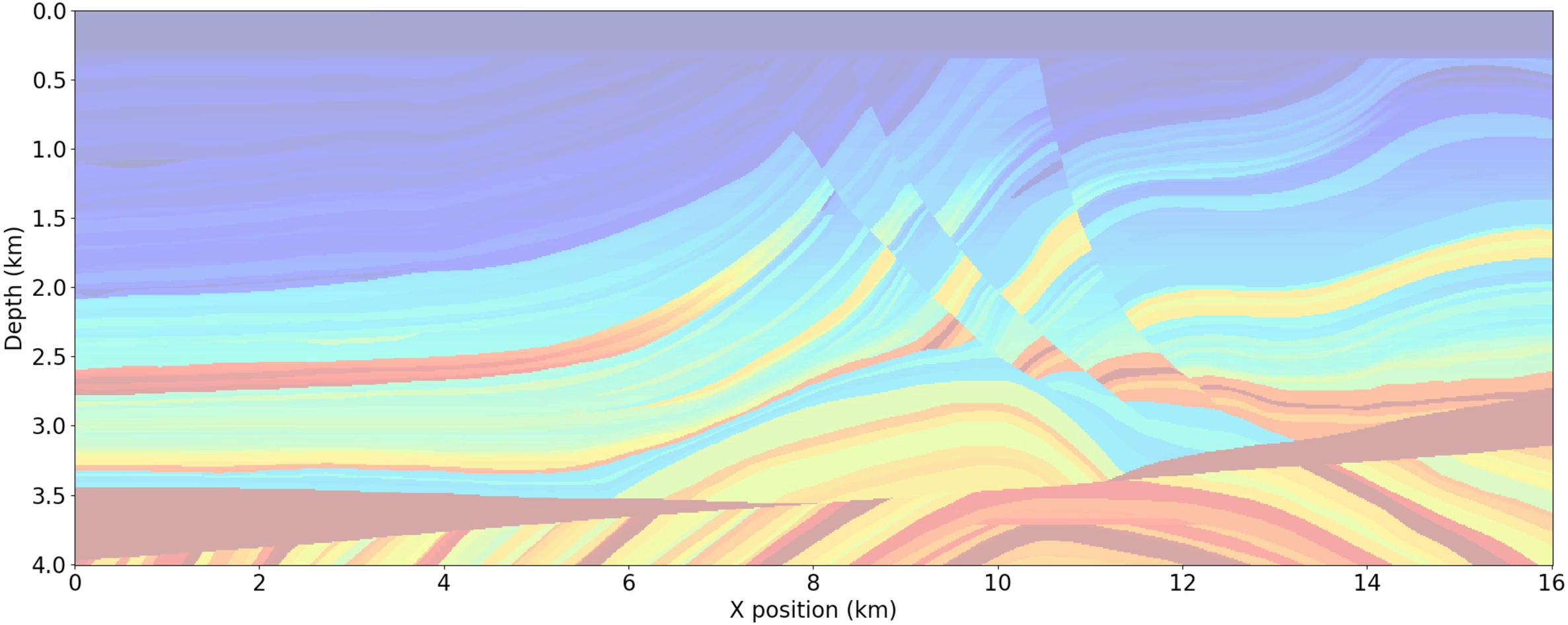
```
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <string>
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
  double loop_body;
  double kernel;
} ;
struct 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
```

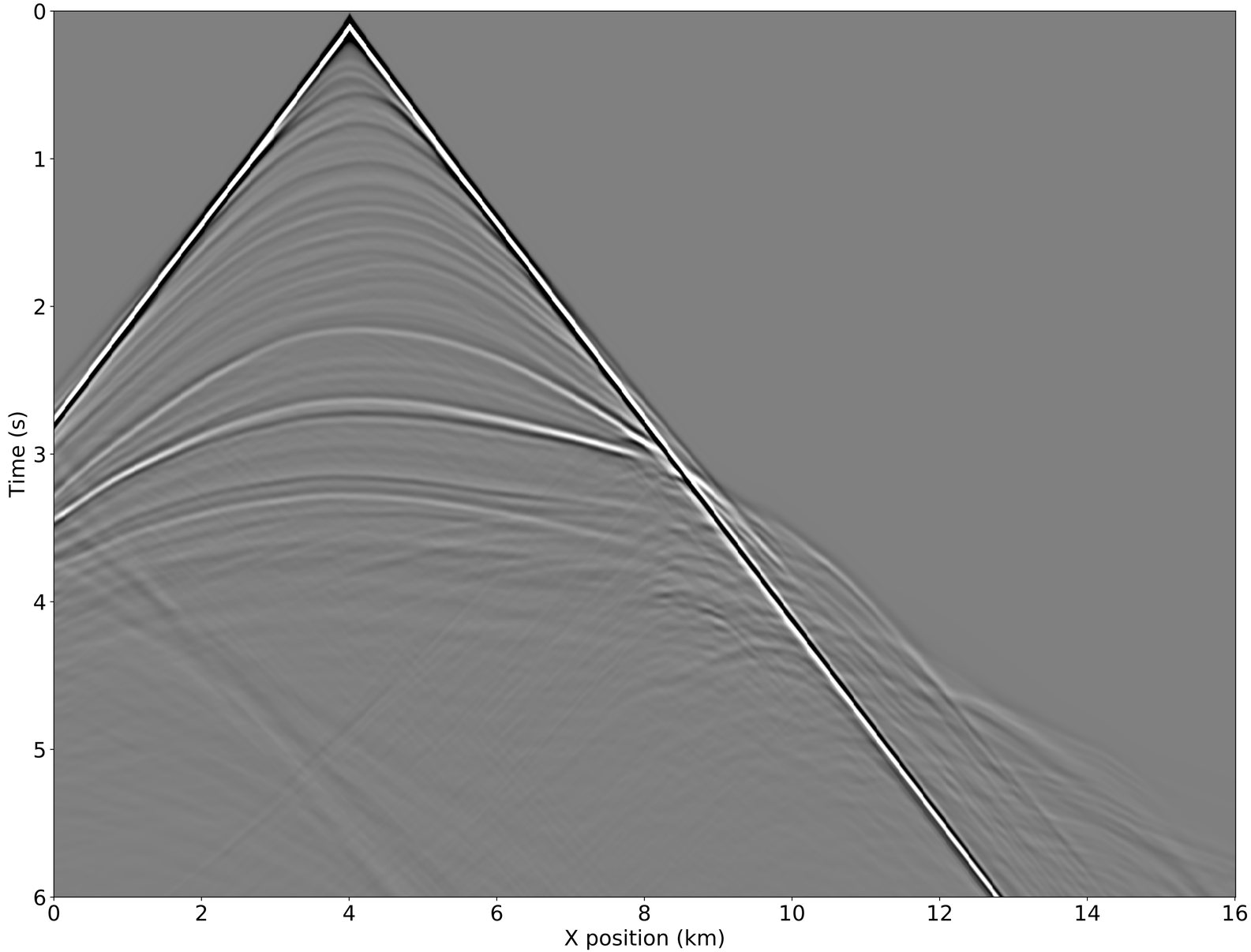
```
[int(floor(5.0e-2F*rec_coords[p][0])) + 40][int(floor(5.0e-2F*rec_coords[p][1])) + 41] +
(2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p]
[0])*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]) -
5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p][0]) -
5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]) + 1)*u[t2]
[int(floor(5.0e-2F*rec_coords[p][0])) + 40][int(floor(5.0e-2F*rec_coords[p][1])) + 40] +
2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p]
[0])*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1])*u[t2][int(floor(5.0e-2F*rec_coords[p]
[0])) + 41][int(flo
          }
          {
            gettimeofday(&end_loop_stencils_a, NULL);
            timings->loop_stencils_a += (double)(end_loop_stencils_a.tv_sec-start_loop_stencils_a.tv_sec)+(double)
(end_loop_stencils_a.tv_usec-start_loop_stencils_a.tv_usec)/1000000;
          }
        }
      }
    }
    {
      gettimeofday(&end_kernel, NULL);
      timings->kernel += (double)(end_kernel.tv_sec-start_kernel.tv_sec)+(double)(end_kernel.tv_usec-
start_kernel.tv_usec)/1000000;
    }
  }
  return 0;
}
```

(rec, u) = Acoustic.Forward()

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

```c
#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
```


Depth (km)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
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)

(srca, v) = Acoustic.Adjoint()

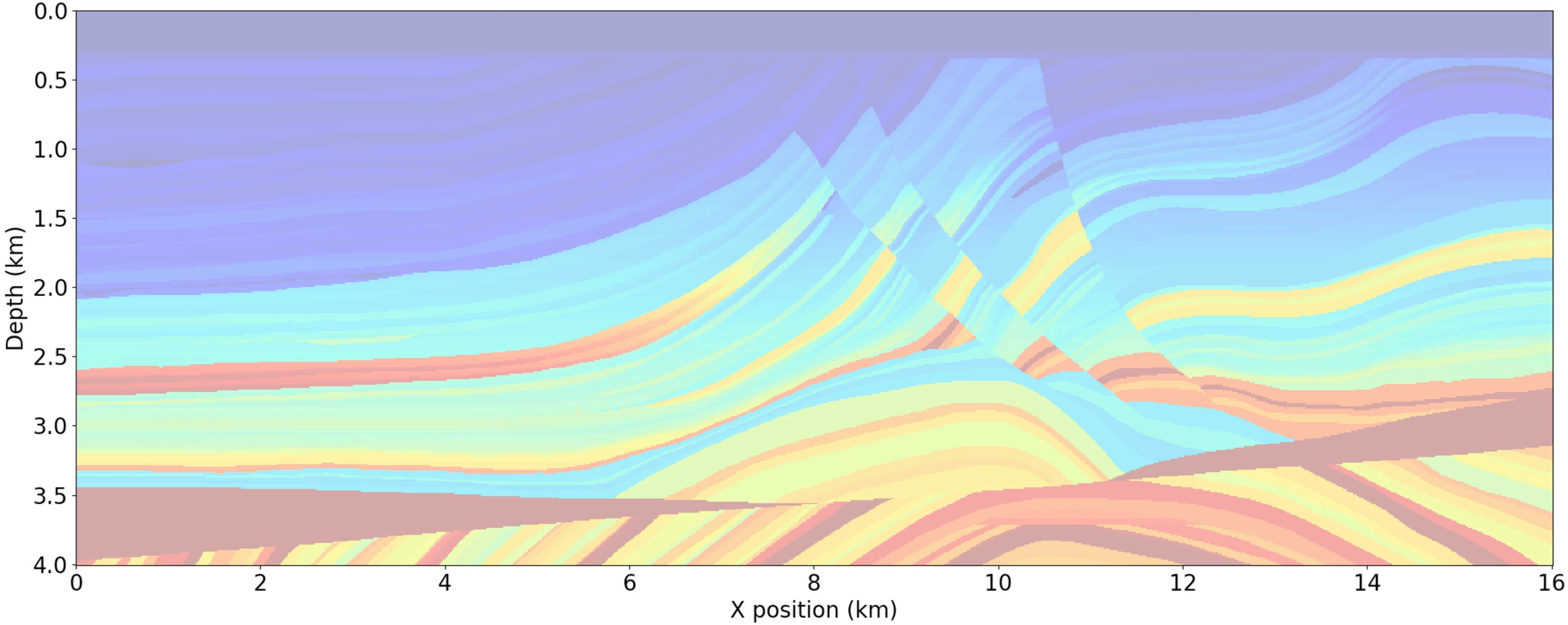
```
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <string>
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
  double loop_body;
  double kernel;
} ;
struct 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++)
            {
```

```
5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]))*u[t2]
[int(floor(5.0e-2F*rec_coords[p][0])) + 40][int(floor(5.0e-2F*rec_coords[p][1])) + 41] +
(2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p]
[0])*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]) -
5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p][0]) -
5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]) + 1)*u[t2]
[int(floor(5.0e-2F*rec_coords[p][0])) + 40][int(floor(5.0e-2F*rec_coords[p][1])) + 40] +
2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p]
[0])*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]) -
5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p][0]) -
5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]) + 1)*rec_coords[p]
[0])) + 41][int(f
          }
          {
            gettimeofday(&end_loop_stencils_a, NULL);
            timings->loop_stencils_a += (double)(end_loop_stencils_a.tv_sec-start_loop_stencils_a.tv_sec)+(double)
(end_loop_stencils_a.tv_usec-start_loop_stencils_a.tv_usec)/1000000;
          }
        }
      }
    }
    {
      gettimeofday(&end_kernel, NULL);
      timings->kernel += (double)(end_kernel.tv_sec-start_kernel.tv_sec)+(double)(end_kernel.tv_usec-
start_kernel.tv_usec)/1000000;
    }
  }
  return 0;
}
```

(srca, v) = Acoustic.Adjoint()

```
(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)

Devito performance

Timings

Rooflines

Devito performance

"Devito's TTI looks "production level" "

Jeremy Tillay
BP summer intern

Timings

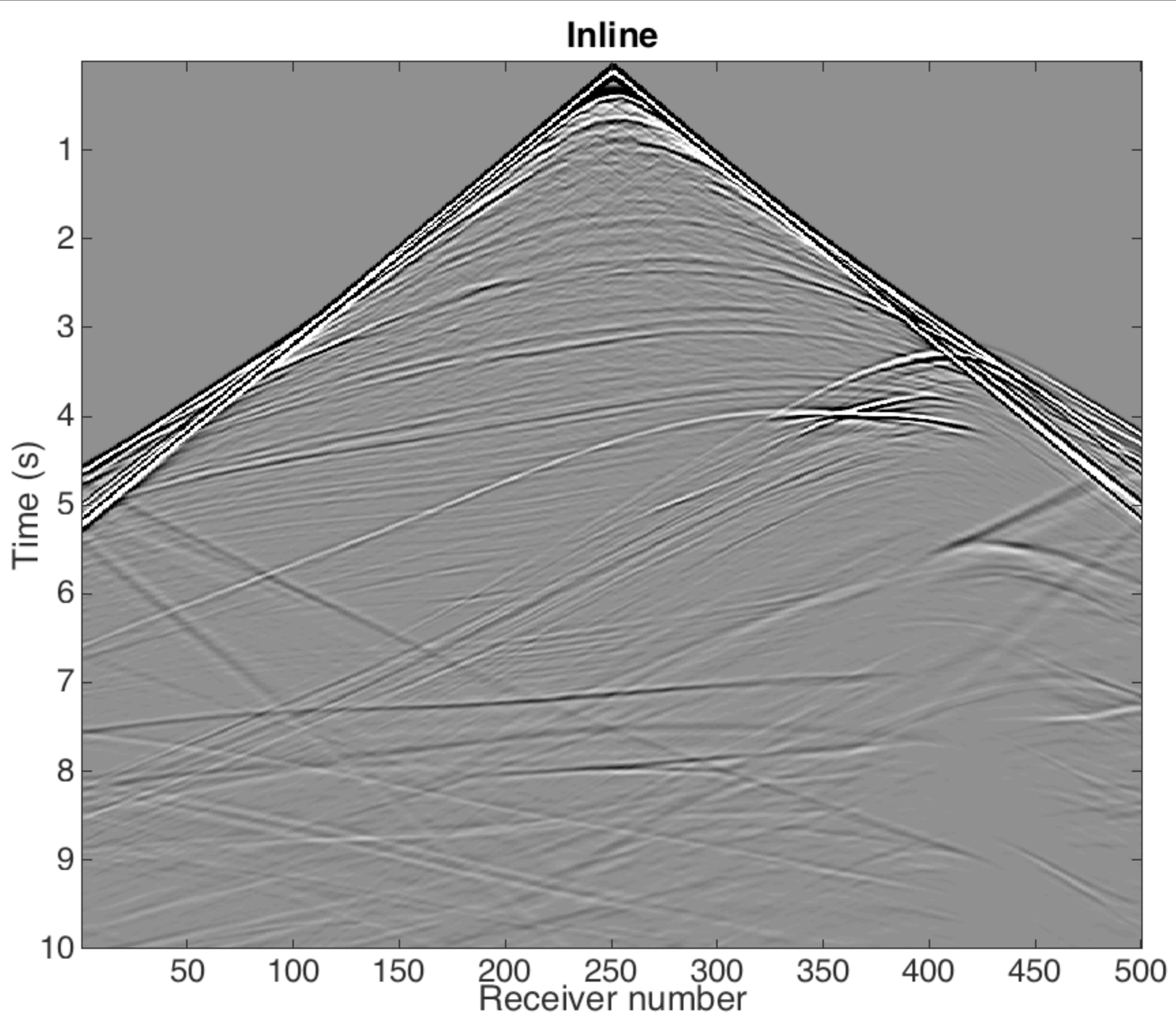
SEAM benchmark

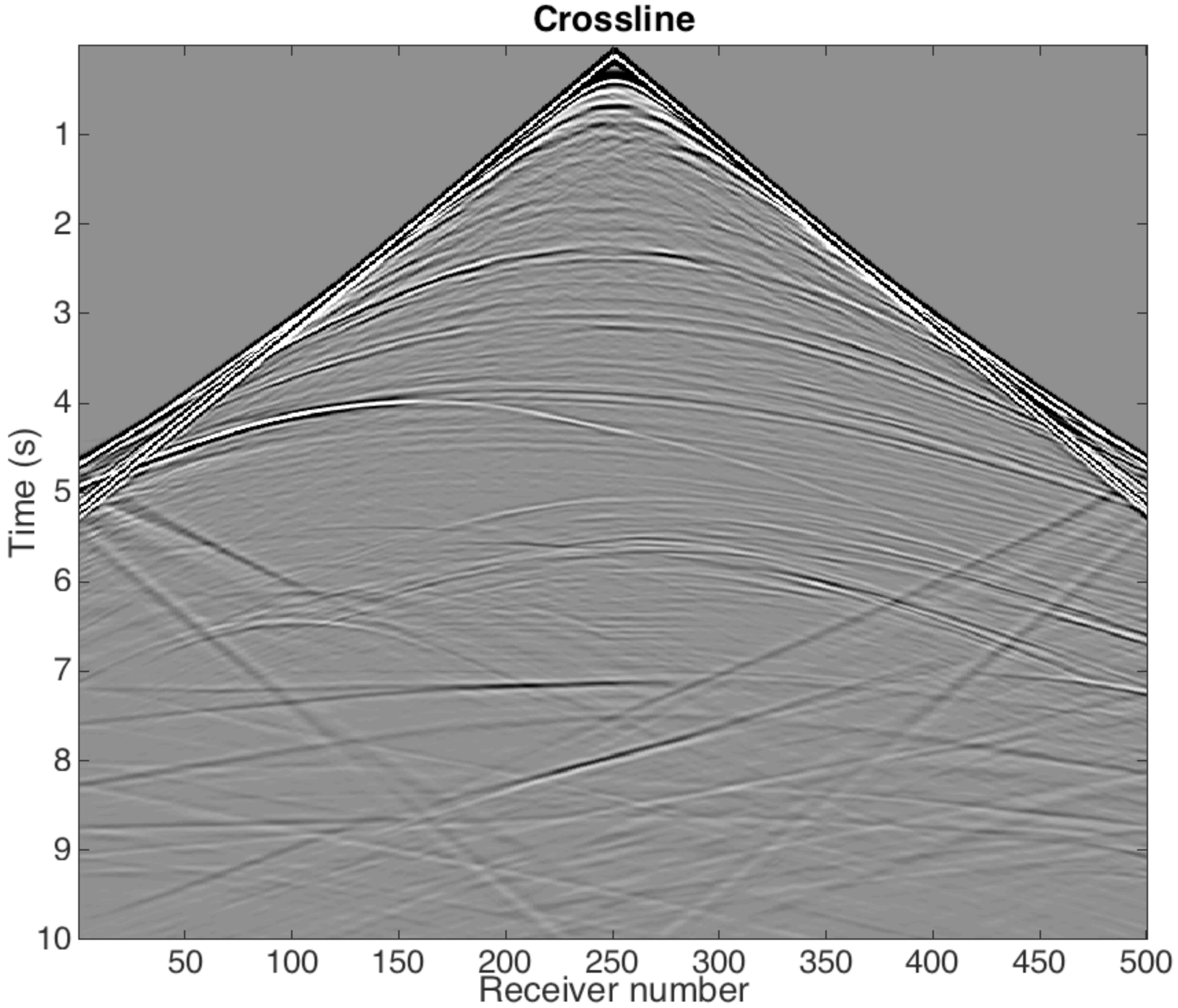
Model

- 1500x1500x1050 model (+80 ABC)
- 1480 to 4500 km/s velocity
- 10th order discretization
- 10Hz Ricker wavelet at (750, 750, 50), 10 sec recording
- 20m Grid

Timings

- **Reference industry code : 8h14**
- **Devito : 4h15 (45% faster)**


Inline
Time (s)
1
2
3
4
5
6
7
8
9
10
50
100
150
200
250
300
350
400
450
500
Receiver number


Crossline
Time (s)
1
2
3
4
5
6
7
8
9
10
50
100
150
200
250
300
350
400
450
500
Receiver number

TTI benchmark

Model

- 512x512x512 model (+80 ABC)
- 1500 km/s velocity
- epsilon 0.3, delta 0.1, theta 40°, phi 20°
- 8th/12th order discretization
- 10Hz Ricker wavelet at (256, 256, 256), 0.25sec recording
- 10m Grid

Timings

- **Reference industry code: 1min 46sec (10sec recording => 33 min)**
- **Devito 8th order: (2min 18sec) (10sec recording => 46 min)**
- **Devito 12th order: (3min 31sec) (10 sec recording => 1h 10min)**

Compiler & rooflines

YASK (Yet Another Stencil Compiler) updates

"A framework to facilitate exploration of the HPC stencil-performance "

- Highly tuned by Intel folks for Intel platforms
- Already in use in some legacy codes
- YASK is pure C++, Devito is pure symbolics. The Devito compiler does the magic: takes the symbolics and writes out "YASK code" (C++).
- Biggest challenge of the integration: difference in data layout (classical row-major in Devito, "vector-folding-based" in YASK for maximum SIMD performance)

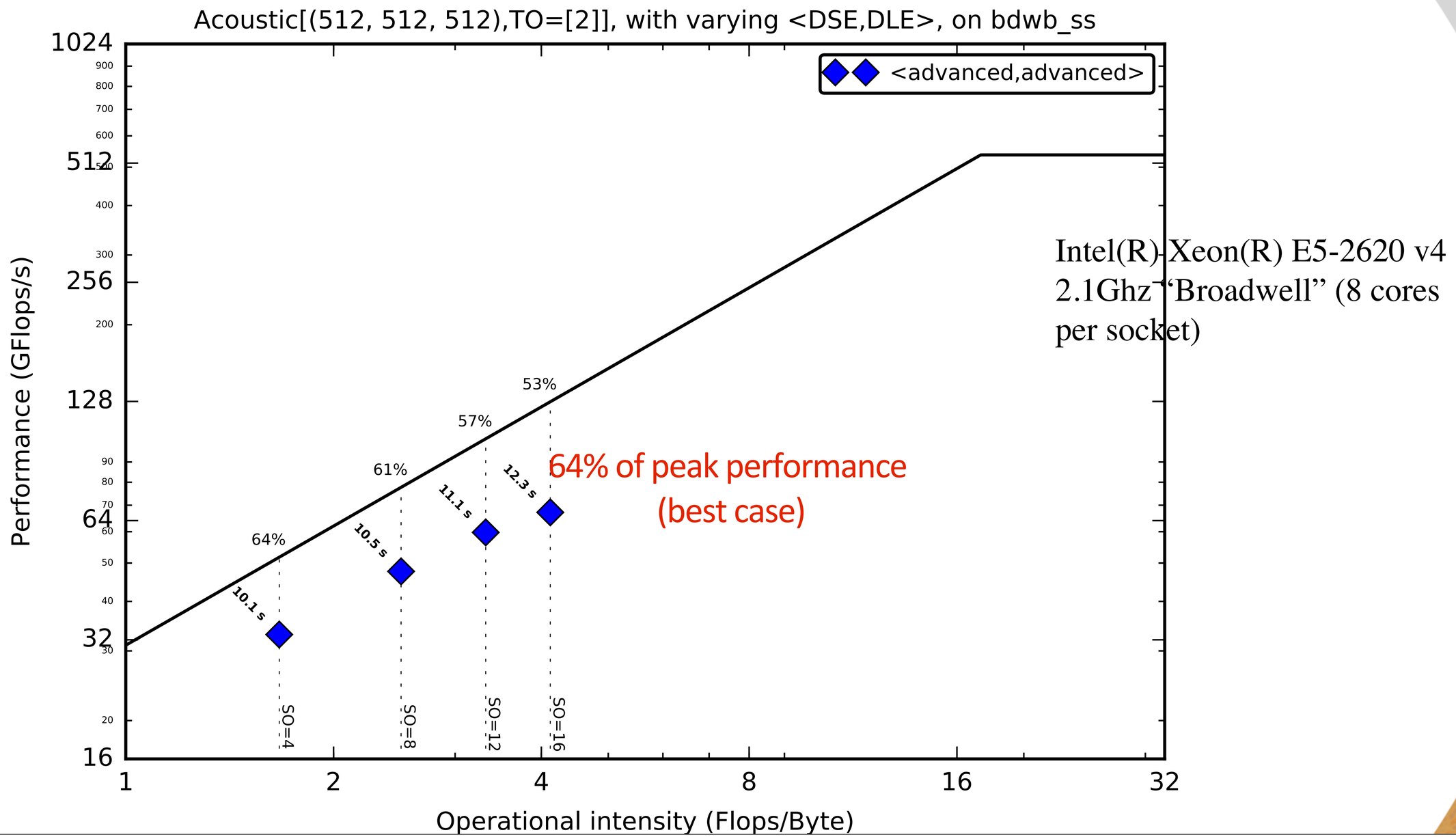
YASK (Yet Another Stencil Compiler) updates

Current status of the integration:

- **[in master]:** Can run simple operators; YASK data layout properly abstracted away (users have the illusion their objects use classic row-major);
- **[almost in master]**: support for source/receiver loops
- **[longer term, (by the end of 2017)]**: full MPI integration (domain decomposition)

No changes to user code required ! When running Devito code, the "YASK backend" is selected on startup through environment variables.

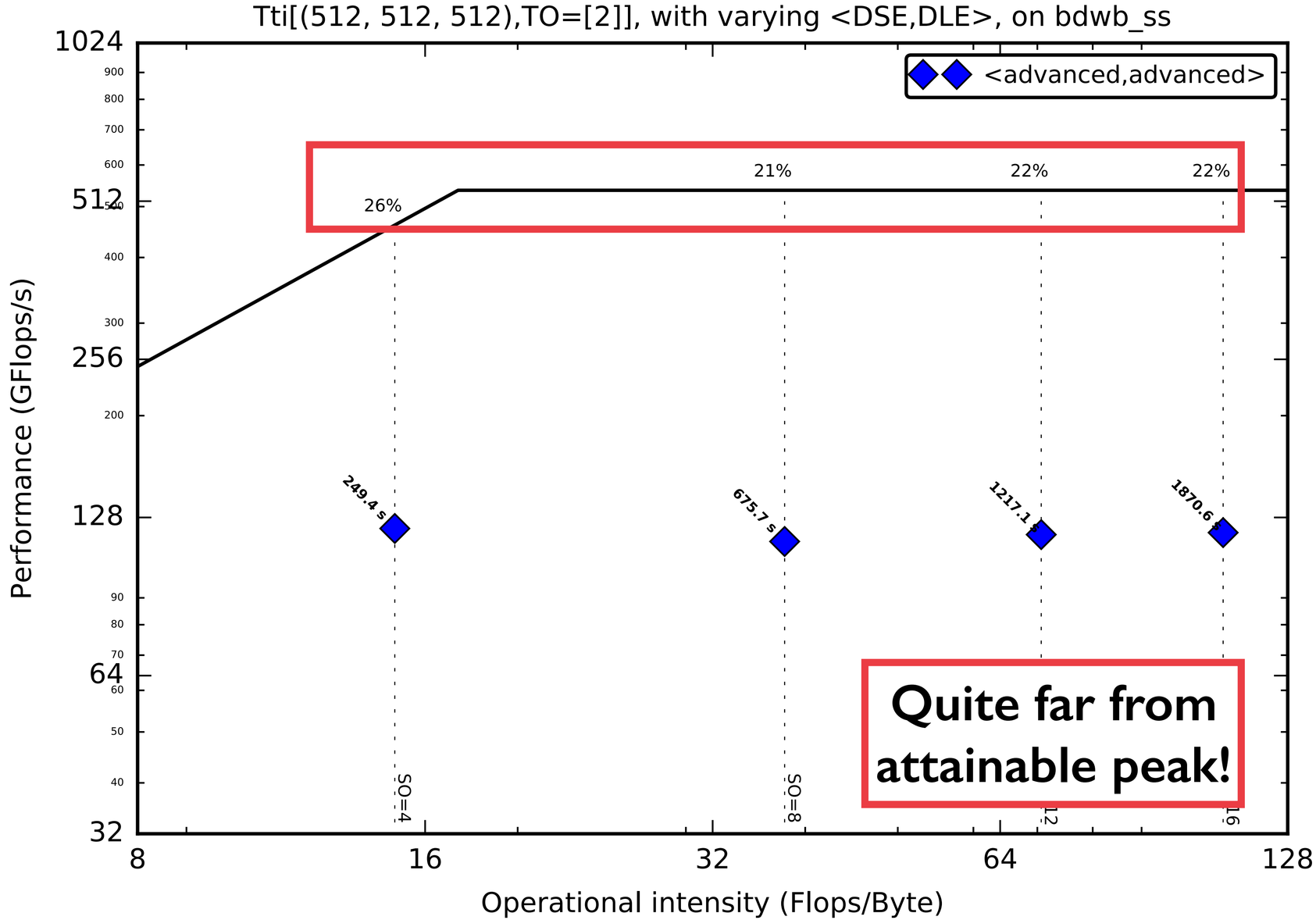
Performance with Devito (with code optimizations)



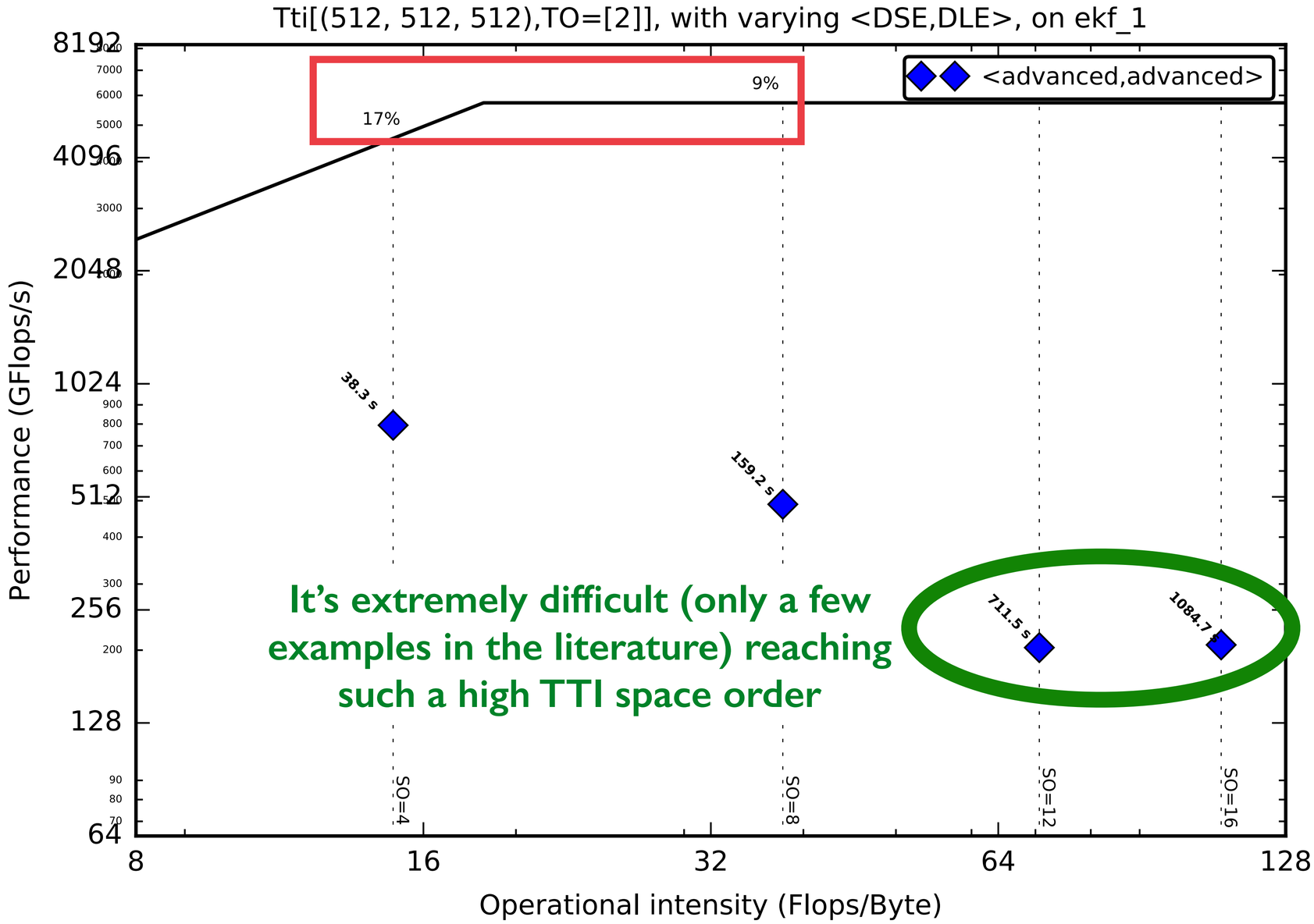
Intel(R) Xeon(R) E5-2620 v4 2.1Ghz "Broadwell" (8 cores per socket)

64% of peak performance (best case)

TTI on Broadwell (8 threads, single socket)



TTI on Xeon Phi (64 threads, cache mode, quadrant)



On the fly Fourier

02:15—02:45 PM

Philipp Witte

A large-scale framework in Julia for fast prototyping of seismic inversion algorithms

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)

Future work

- Apply the PDE instead of solving it
  - How to manage the memory?
  - Implicit instead of explicit scheme?

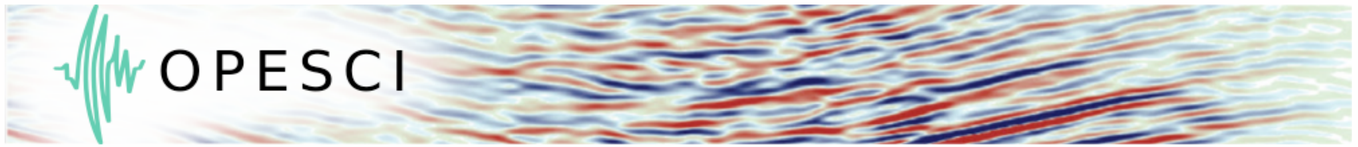
- Grid abstraction and staggered grids

- Abstraction for the boundary conditions
  - How accurate?
  - How expensive?

Shashin Sharan, Rongrong Wang, Tristan van Leeuwen, and Felix J. Herrmann,2016

Acknowledgements

This research was carried out as part of the SINBAD project with the support of the member organizations of the SINBAD Consortium in collaboration with Imperial College London



Thank you for your attention.

Devito publications

- Mathias Louboutin, Michael Lange, Navjot Kukreja, Fabio Luporini, Felix J. Herrmann, and Gerard Gorman, “Raising the abstraction to separate concerns: enabling different physics for geophysical exploration”, in SIAM Conference on Computational Science and Engineering, 2017.
- Navjot Kukreja, Michael Lange, Mathias Louboutin, Fabio Luporini, and Gerard Gorman, “Devito: symbolic math for automated fast finite difference computations”, in SIAM Conference on Computational Science and Engineering, 2017.
- Philipp A. Witte, Mathias Louboutin, and Felix J. Herrmann, “Large-scale workflows for wave-equation based inversion in Julia”, in SIAM Conference on Computational Science and Engineering, 2017.

- Marcos de Aguiar, Gerard Gorman, Felix J. Herrmann, Navjot Kukreja, Michael Lange, Mathias Louboutin, and Felippe Vieira Zacarias, “DeVito: fast finite difference computation”, in Super Computing (SC16), 2016.

- Navjot Kukreja, Mathias Louboutin, Felippe Vieira Zacarias, Fabio Luporini, Michael Lange, and Gerard Gorman, “Devito: automated fast finite difference computation”, in WOLFHPC 2016 Workshop (Super Computing), 2016.

- Navjot Kukreja, Mathias Louboutin, Michael Lange, Fabio Luporini, and Gerard Gorman, “Leveraging symbolic math for rapid development of applications for seismic modeling”. 2016.

- M. Lange, N. Kukreja, Mathias Louboutin, F. Luporini, F. Vieira, V. Pandolfo, P. Velesko, P. Kazakas, and G. Gorman. “Devito: Towards a generic Finite Difference DSL using Symbolic Python”. Accepted for PyHPC2016, to appear in ACM SIGHPC, 2016