Accelerating ideation & innovation cheaply in the Cloud

the power of abstraction, collaboration & reproducibility

Felix J. Herrmann

4th EAGE Workshop on High-performance Computing

Dubai, October 8, 2019

SLIM

Georgia Institute of Technology

Georgia Tech

Accelerating ideation & innovation cheaply in the Cloud

the power of abstraction, collaboration & reproducibility

Charles Jones[⚑], Gerard Gorman[†], Jan Hückelheim[†], Keegan Lensink[★], Paul Kelly[†], Navjot Kukreja[†], Henryk Modzelewski[★], Michael Lange[†], Mathias Louboutin[♛], Fabio Luporini[†], James Selvages[⚑], Phillipp Witte[♛]

SLIM
Georgia Institute of Technology

[★] THE UNIVERSITY OF BRITISH COLUMBIA

[⚑] osokey

[†] Imperial College London

[♛] Georgia Tech

Accelerating ideation & innovation cheaply in the Cloud

the power of abstraction, collaboration & reproducibility

Charles Jones[⚑], Gerard Gorman[†], Jan Hückelheim[†], Keegan Lensink[★], Paul Kelly[†], Navjot Kukreja[†], Henryk Modzelewski[★], Michael Lange[†], Mathias Louboutin[♛], Fabio Luporini[†], James Selvages[⚑], Phillipp Witte[♛]

SLIM
Georgia Institute of Technology

Microsoft Azure

amazon web services™

Google Cloud Platform

★ THE UNIVERSITY OF BRITISH COLUMBIA

⚑ osokey

† Imperial College London

♛ Georgia Tech

Disclaimer

We worked w/ Google Cloud Computing Services, Amazon Web Services (AWS) and Microsoft Azure. We therefore refer to services & product names related to these platforms.

Technology presented is not tied to one specific Cloud provider and has been replicated on all major cloud platforms (AWS, Azure, Google Cloud)

I am not trying to sell anything… I am talking from the perspective of an entrepreneurial academician who wants to

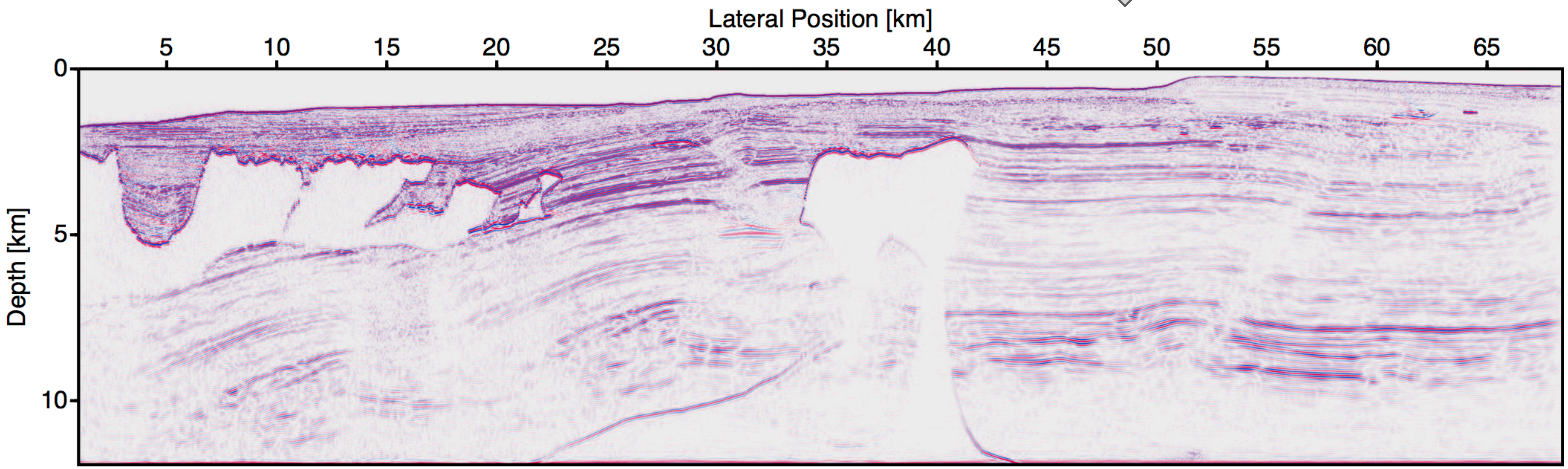
- drive innovations more rapidly
- bring codes close to at scale technology validation
- deal w/ intermittent workloads

Early attempt – 1 y ago

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

Julia code

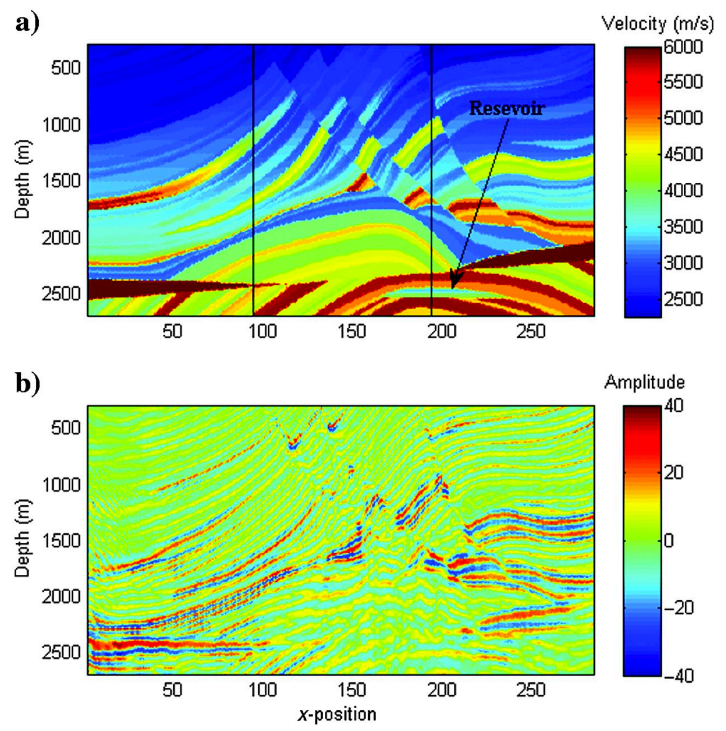
Google Cloud Platform



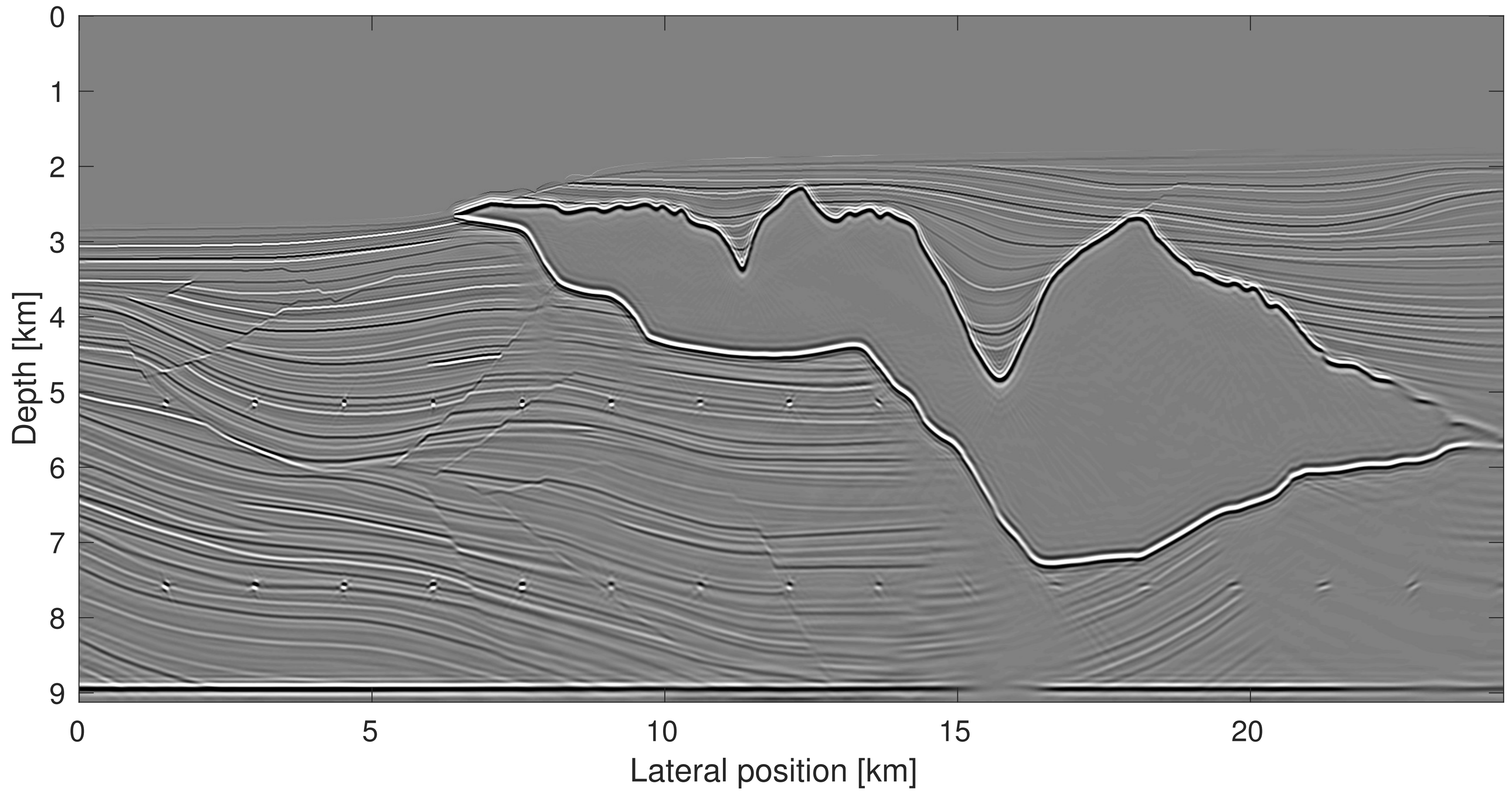
Early attempt

Seismic imaging on GCP: Fall 2018

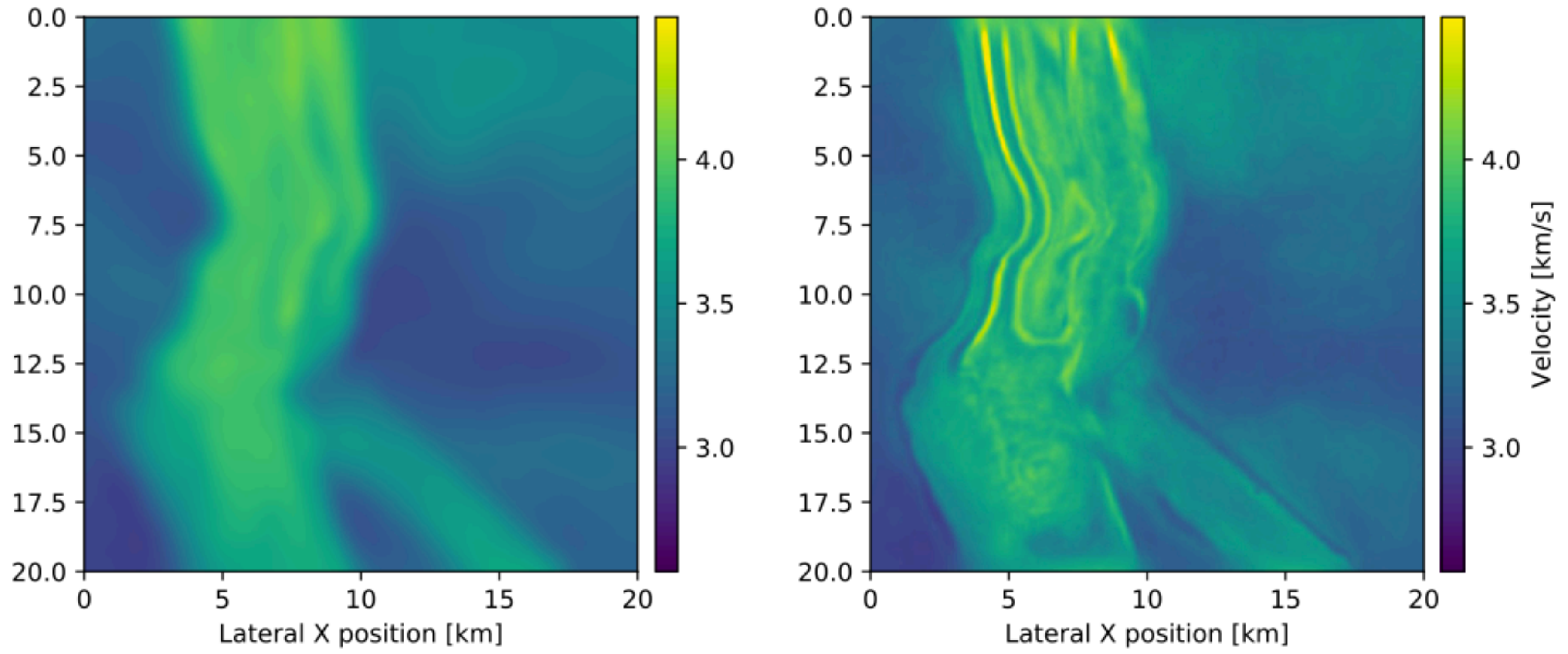
- Lift & shift approach
- 2D LS-RTM on BP Synthetic 2004 model
- **32,000** cores on 1000 nodes
- Parallel pool using Ethernet
- SLURM + parallel Julia session
- 2 hours to launch pool
- Frequent interruptions and restarts of pool
- **Total cost in 10 days: 170,000$**
- **But we were able to hack something in a matter of weeks…**



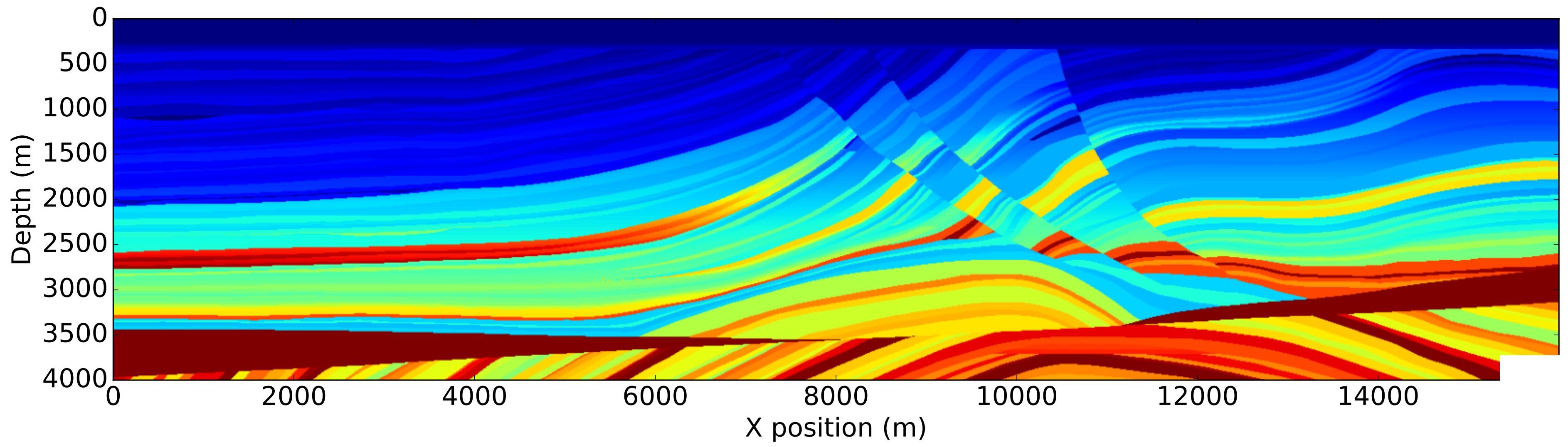
Tiny marmousi
62k gridpoints
1.5MFlop/time-step
$\$10^{-8}$/time-step $\Rightarrow$ $< \$10$



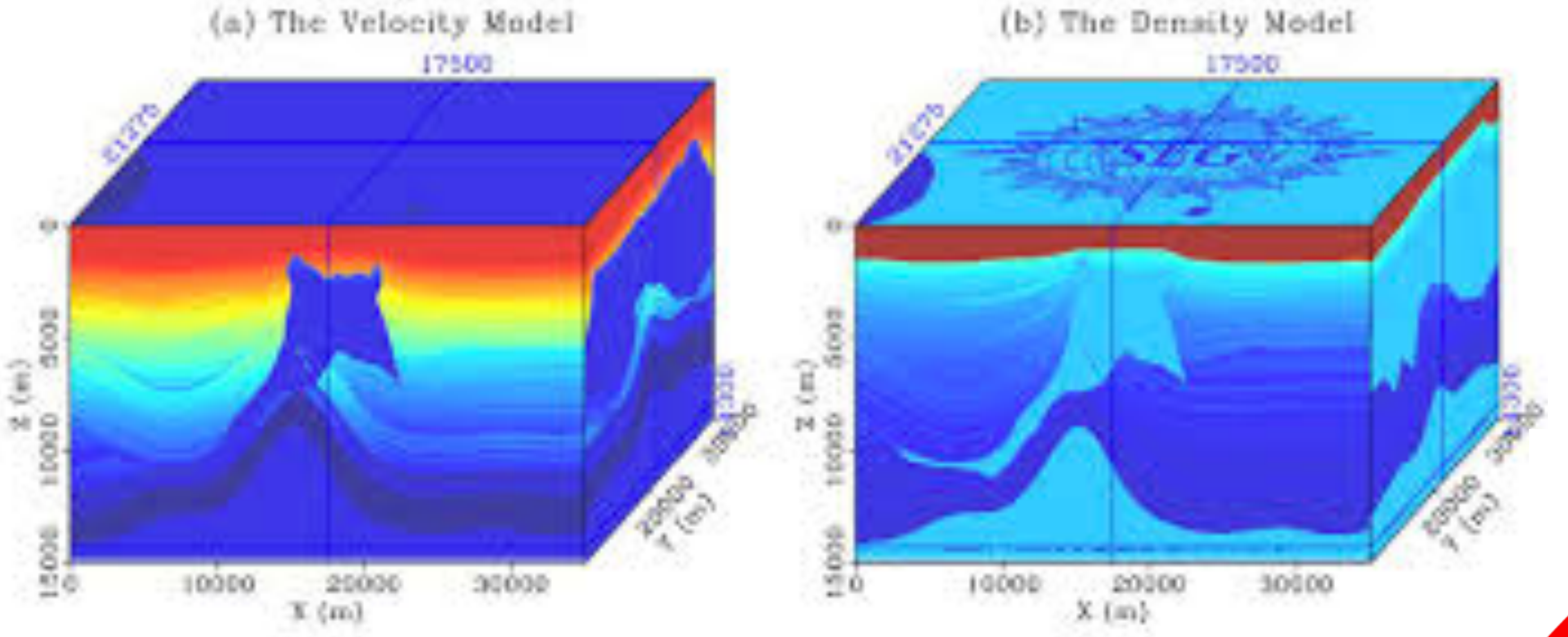
Sigsbee
2.2M gridpoints
56MFlop/time-step
$\$4 \times 10^{-7}$/time-step $\Rightarrow$ $< \$100$



3D overthrust
222M gridpoints
6GFlop/time-step
$\$4 \times 10^{-5}$/time-step
$\Rightarrow$ $\$3000$



Full marmousi
640k gridpoints
15MFlop/time-step
$\$10^{-7}$/time-step $\Rightarrow$ $< \$20$



SEAM elastic
5.3G gridpoints
2.8TFlop/time-step
$\$0.02$/time-step $\Rightarrow$ $\$14M$
full azimuth 35k shots

Cloud

Recent success

ML & AI have been responsible for major breakthroughs

- rapid rate of innovation & radical performance improvements
- sharing of ideas & code
- modern abstracted code bases & tools

HPC developments in Oil & Gas

- relatively slow
- proprietary attitudes
- too small a community

We are lagging behind & operating at too high costs!

Rapid developments

Training Resnet-50 on Imagenet

Facebook Caffe2	UC Berkeley, TACC, UC Davis Tensorflow	Preferred Network ChainerMN	Tencent TensorFlow	Sony Neural Network Library (NNL)	Fujitsu MXNet
1 hour	31 mins	15 mins	6.6 mins	2.0 mins	1.2 mins
Tesla P100 x 256	1,600 CPUs	Tesla P100 x 1,024	Tesla P40 x 2,048	Tesla V100 x 3,456	Tesla V100 x 2,048
Apr	Sept	Nov	July	Nov	Apr
2017	2017	2017	2017	2018	2019

Azure

- short development cycle
- **almost exclusively 2D**

Thanks to Azure

So far

Our successes in FWI & RTM relied on hand code for

- FD stencils on CPUs/GPUs
- sensitivities & “adjoints”
- memory & IO handling

Remarkable achievement RTM/FWI=DCNN w/ 10k layers on $1k^3$ grids

Unfortunately, this approach

- does not scale very well to different wave physics
- is error prone, and
- impedes rapid innovation

Research questions

"How can we exploit ML & JIT compiler technology in the Cloud?"

- manage complexities of often monolithic code bases
- be more agile, reduce development time & (running) costs
- use serverless technology that removes need to touch all data all the time

Today's agenda:

- abstractions for FD-based FWI & RTM w/ Devito* + Judi*
- serverless implementations* in the Cloud
- case study & road ahead

Not a lift & shift solution!

***open source under MIT license**

Our approach

Create performant open source platform in the Cloud

	modeling	Q	gradient
3D acoustic	✔	✘	✔
3D acoustic TTI	✔	✘	✔
3D elastic	✔	✔	✘
3D elastic TTI	✘	✘	✘

✘ **Will be finalized next months**

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.
Mathias Louboutin, Michael Lange, Fabio Luporini, Navjot Kukreja, Philipp A. Witte, Felix J. Herrmann, Paulius Velesko and Gerard J. Gorman Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration. Geoscientific Model Development, Volume 12, p 1165-1187, 2019

Solution

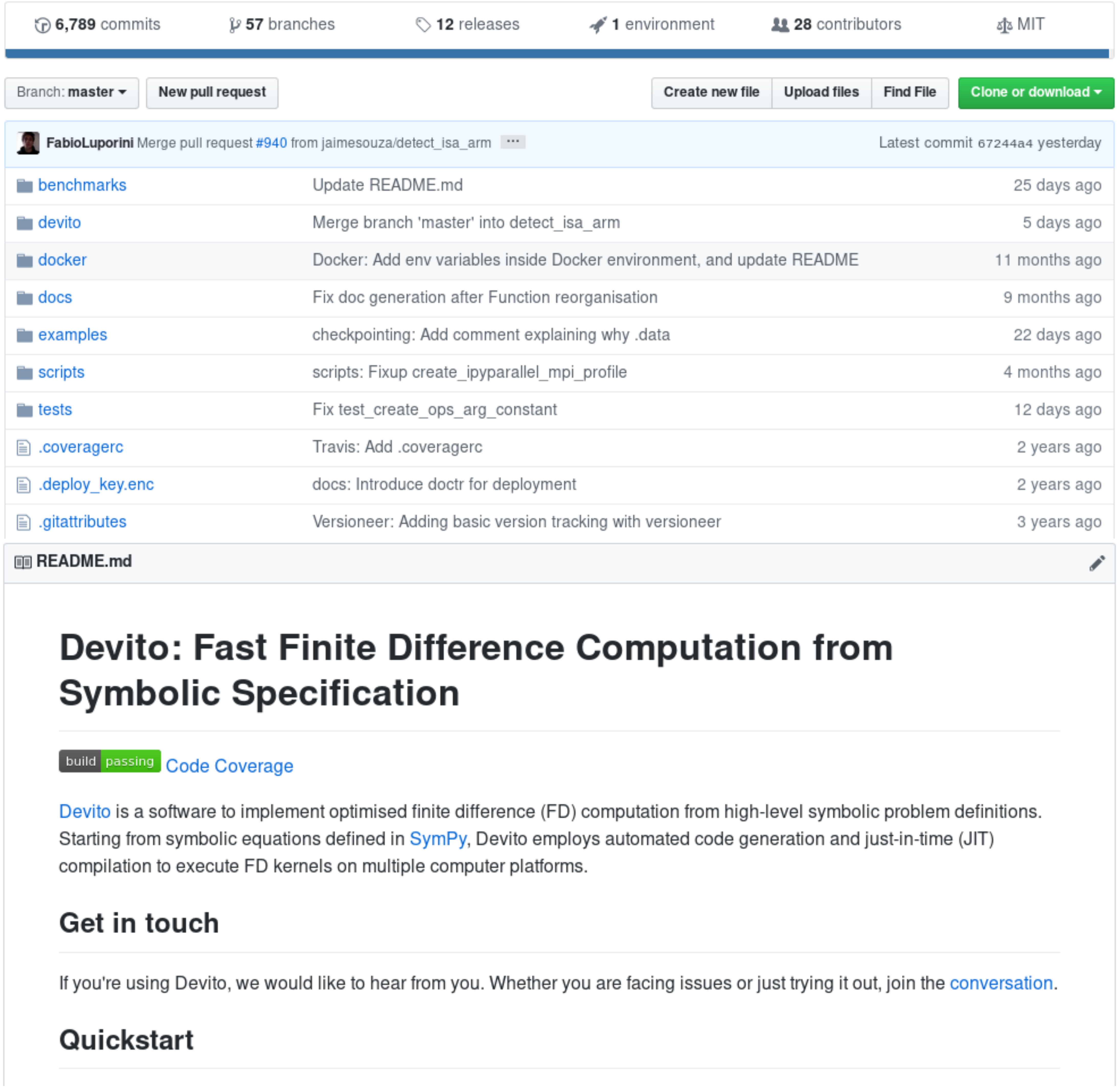
DEVITO – Domain specific language for stencil-based finite difference code generation for PDEs w/ explicit time stepping in Python using SymPy.

https://www.devitoproject.org

Open-source software

Devito:

- Open-source MIT license
- High-level Python interface for discretization of ODEs + PDEs using finite differences
- Automatic performance optimization and JIT code generation
- https://github.com/opesci/devito



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>
  …
}
```

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>
  …
}
```

Raising the level of abstraction

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

Raising the level of abstraction

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

```
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

Raising the level of abstraction

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

```
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

```
void kernel(…) { … }
```

Raising the level of abstraction

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

Devito

```
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

```
void kernel(…) { … }
```

Flexibility in space/time discretization

```
u = TimeFunction(…, space_order=so)
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

so=4

```
  for (int time = time_m, t0 = (time)%(3), t1 = (time + 1)%(3), t2 = (time +
2)%(3); time <= time_M; time += 1, t0 = (time)%(3), t1 = (time + 1)%(3), t2 =
(time + 2)%(3)) {
    for (int x = x_m; x <= x_M; x += 1) {
      for (int y = y_m; y <= y_M; y += 1) {
        for (int z = z_m; z <= z_M; z += 1) {
          u[t1][x + 4][y + 4][z + 4] = 2*pow(dt,
3)*(-2.08333333333333e-4F*u[t0][x + 2][y + 4][z + 4] +
3.33333333333333e-3F*u[t0][x + 3][y + 4][z + 4] - 2.08333333333333e-4F*u[t0]
[x + 4][y + 2][z + 4] + 3.33333333333333e-3F*u[t0][x + 4][y + 3][z + 4] -
2.08333333333333e-4F*u[t0][x + 4][y + 4][z + 2] + 3.33333333333333e-3F*u[t0]
[x + 4][y + 4][z + 3] - 1.875e-2F*u[t0][x + 4][y + 4][z + 4] +
3.33333333333333e-3F*u[t0][x + 4][y + 4][z + 5] - 2.08333333333333e-4F*u[t0]
[x + 4][y + 4][z + 6] + 3.33333333333333e-3F*u[t0][x + 4][y + 5][z + 4] -
2.08333333333333e-4F*u[t0][x + 4][y + 6][z + 4] + 3.33333333333333e-3F*u[t0]
[x + 5][y + 4][z + 4] - 2.08333333333333e-4F*u[t0][x + 6][y + 4][z + 4])/
(pow(dt, 2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 4][y + 4][z + 4]) +
pow(dt, 2)*damp[x + 1][y + 1][z + 1]*u[t2][x + 4][y + 4][z + 4]/(pow(dt,
2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 4][y + 4][z + 4]) + 4*dt*m[x + 4][y
+ 4][z + 4]*u[t0][x + 4][y + 4][z + 4]/(pow(dt, 2)*damp[x + 1][y + 1][z + 1]
+ 2*dt*m[x + 4][y + 4][z + 4]) - 2*dt*m[x + 4][y + 4][z + 4]*u[t2][x + 4][y +
4][z + 4]/(pow(dt, 2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 4][y + 4][z +
4]);
        }
      }
    }
  }
```

so=12

```
  for (int time = time_m, t0 = (time)%(3), t1 = (time + 1)%(3), t2 = (time +
2)%(3); time <= time_M; time += 1, t0 = (time)%(3), t1 = (time + 1)%(3), t2 =
(time + 2)%(3)) {
    for (int x = x_m; x <= x_M; x += 1) {
      for (int y = y_m; y <= y_M; y += 1) {
        for (int z = z_m; z <= z_M; z += 1) {
          u[t1][x + 12][y + 12][z + 12] = 2*pow(dt,
3)*(-1.5031265031265e-7F*u[t0][x + 6][y + 12][z + 12] +
2.5974025974026e-6F*u[t0][x + 7][y + 12][z + 12] - 2.23214285714286e-5F*u[t0][x
+ 8][y + 12][z + 12] + 1.32275132275132e-4F*u[t0][x + 9][y + 12][z + 12] -
6.69642857142857e-4F*u[t0][x + 10][y + 12][z + 12] + 4.28571428571429e-3F*u[t0]
[x + 11][y + 12][z + 12] - 1.5031265031265e-7F*u[t0][x + 12][y + 6][z + 12] +
2.5974025974026e-6F*u[t0][x + 12][y + 7][z + 12] - 2.23214285714286e-5F*u[t0][x
+ 12][y + 8][z + 12] + 1.32275132275132e-4F*u[t0][x + 12][y + 9][z + 12] -
6.69642857142857e-4F*u[t0][x + 12][y + 10][z + 12] + 4.28571428571429e-3F*u[t0]
[x + 12][y + 11][z + 12] - 1.5031265031265e-7F*u[t0][x + 12][y + 12][z + 6] +
2.5974025974026e-6F*u[t0][x + 12][y + 12][z + 7] - 2.23214285714286e-5F*u[t0][x
+ 12][y + 12][z + 8] + 1.32275132275132e-4F*u[t0][x + 12][y + 12][z + 9] -
6.69642857142857e-4F*u[t0][x + 12][y + 12][z + 10] + 4.28571428571429e-3F*u[t0]
[x + 12][y + 12][z + 11] - 2.23708333333333e-2F*u[t0][x + 12][y + 12][z + 12] +
4.28571428571429e-3F*u[t0][x + 12][y + 12][z + 13] - 6.69642857142857e-4F*u[t0]
[x + 12][y + 12][z + 14] + 1.32275132275132e-4F*u[t0][x + 12][y + 12][z + 15] -
2.23214285714286e-5F*u[t0][x + 12][y + 12][z + 16] + 2.5974025974026e-6F*u[t0]
[x + 12][y + 12][z + 17] - 1.5031265031265e-7F*u[t0][x + 12][y + 12][z + 18] +
4.28571428571429e-3F*u[t0][x + 12][y + 13][z + 12] - 6.69642857142857e-4F*u[t0]
[x + 12][y + 14][z + 12] + 1.32275132275132e-4F*u[t0][x + 12][y + 15][z + 12] -
2.23214285714286e-5F*u[t0][x + 12][y + 16][z + 12] + 2.5974025974026e-6F*u[t0]
[x + 12][y + 17][z + 12] - 1.5031265031265e-7F*u[t0][x + 12][y + 18][z + 12] +
4.28571428571429e-3F*u[t0][x + 13][y + 12][z + 12] - 6.69642857142857e-4F*u[t0]
[x + 14][y + 12][z + 12] + 1.32275132275132e-4F*u[t0][x + 15][y + 12][z + 12] -
2.23214285714286e-5F*u[t0][x + 16][y + 12][z + 12] + 2.5974025974026e-6F*u[t0]
[x + 17][y + 12][z + 12] - 1.5031265031265e-7F*u[t0][x + 18][y + 12][z + 12])/
(pow(dt, 2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 12][y + 12][z + 12]) +
```

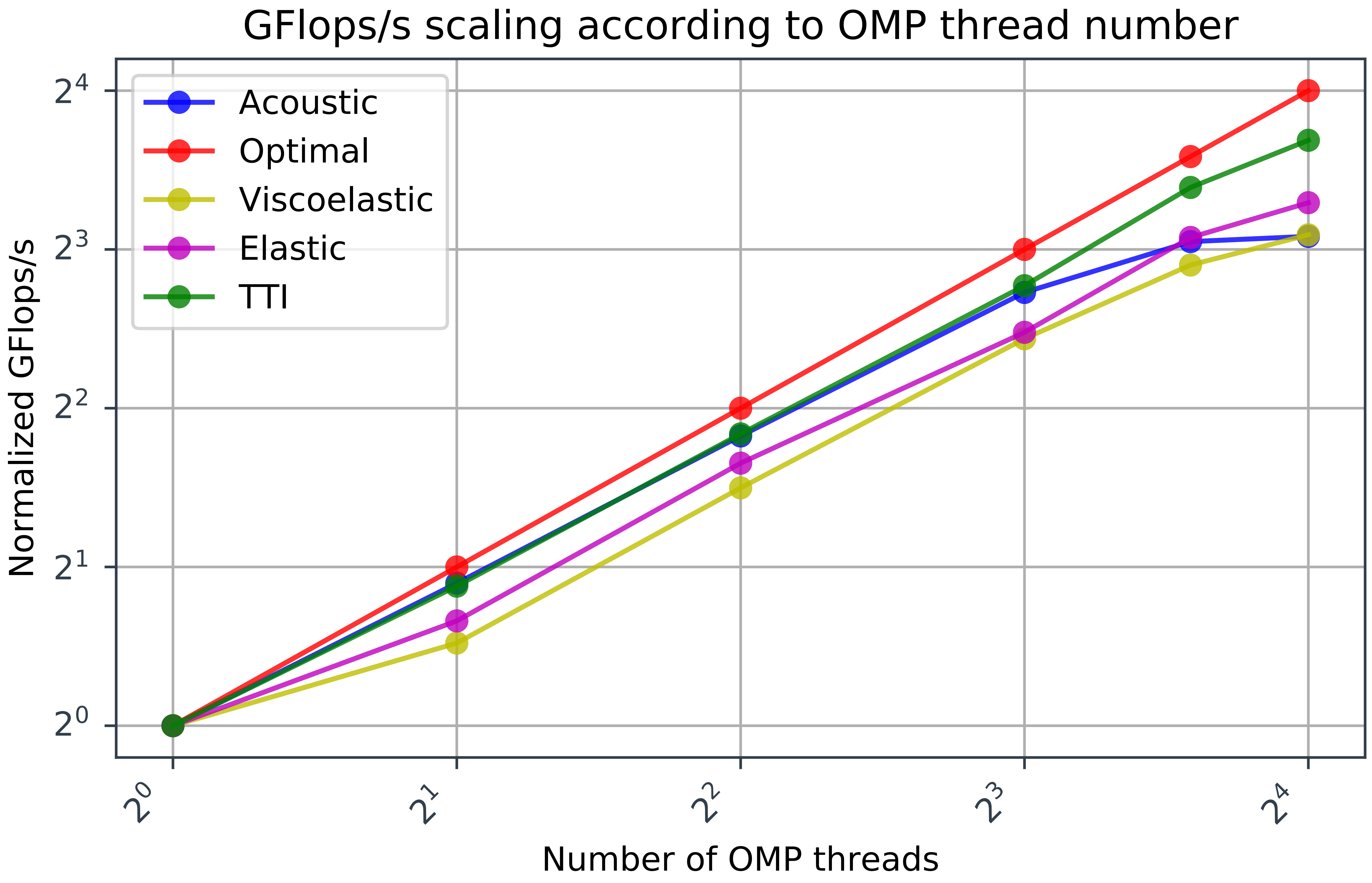
Mathias Louboutin, Michael Lange, Fabio Luporini, Navjot Kukreja, Philipp A. Witte, Felix J. Herrmann, Paulius Velesko, and Gerard J. Gorman, "**Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration**", Geoscientific Model Development, 2019
Fabio Luporini, Michael Lange, Mathias Louboutin, Navjot Kukreja, Jan Hückelheim, Charles Yount, Philipp A. Witte, PAUL H. J. KELLY, Gerard J. Gorman, and Felix J. Herrmann, "**Architecture and performance of Devito, a system for automated stencil computation**". 2018.
F. Luporini, R. Nelson, M. Louboutin, N. Kukreja, G. Bisbas, P. Witte, Amik St-Cyr, C. Yount, T. Burgess, F. Herrmann, G. Gorman
"Automatic Generation of Production-Grade Hybrid MPI-OpenMP Parallel Wave Propagators using Devito"
Presented at Platform for Advanced Scientific Computing (PASC 2019) Conference.

OMP/MPI scaling

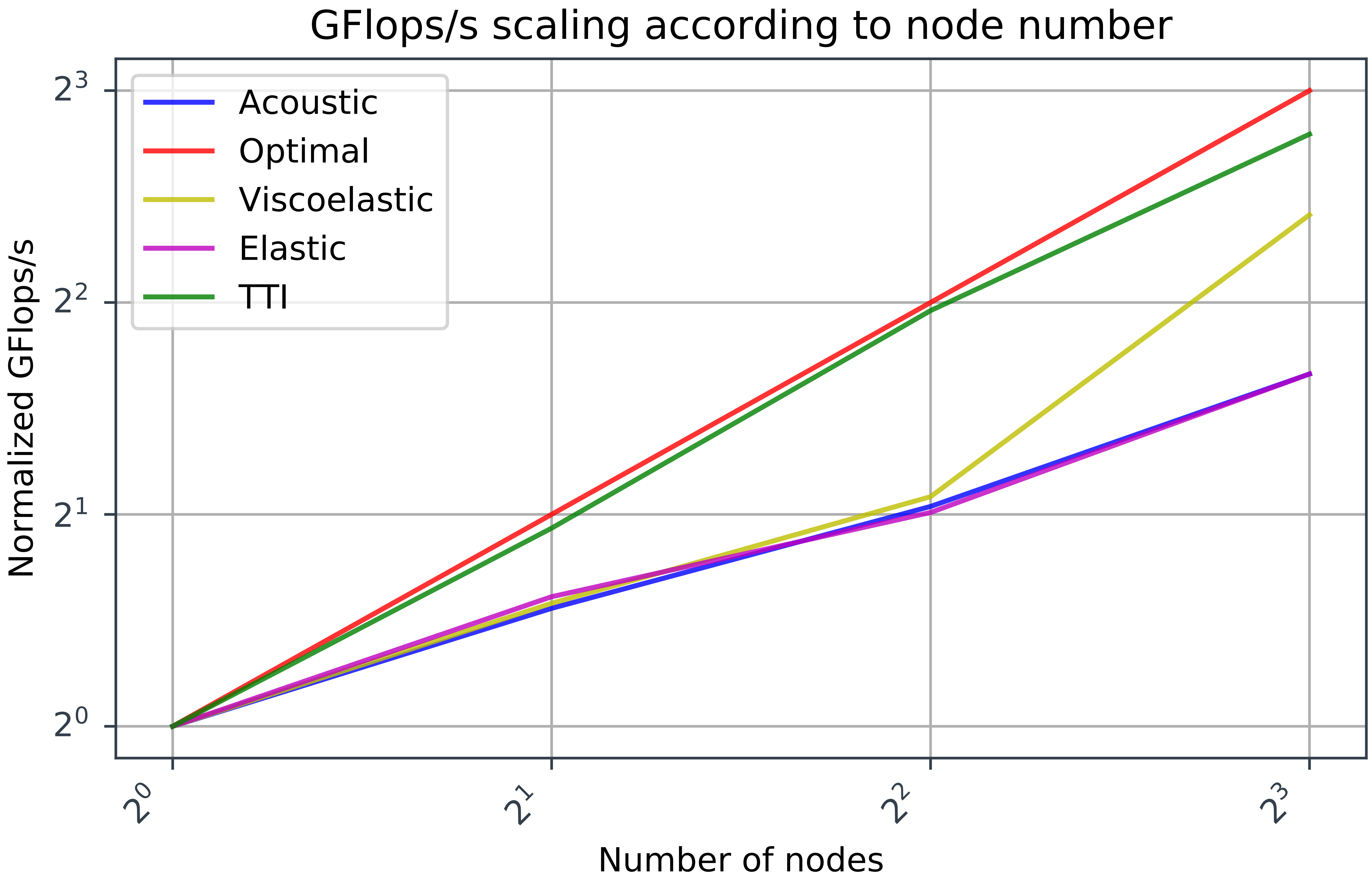
Setup

- 300 x 300 x 300 grid
- 16th order FD
- 100 Time-steps
- Xeon E5-2670 8C
- Single socket for OMP scaling
- one MPI rank per socket per node for MPI scaling

Strong scaling OMP threads – near optimal



Strong scaling MPI – TTI compute bound



Setup

- 512 x 512 x 512 grid
- varying FD order
- 1000ms modeling
- Intel Skylake 8180
- Single socket, OMP only

20 X flop reduction

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

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

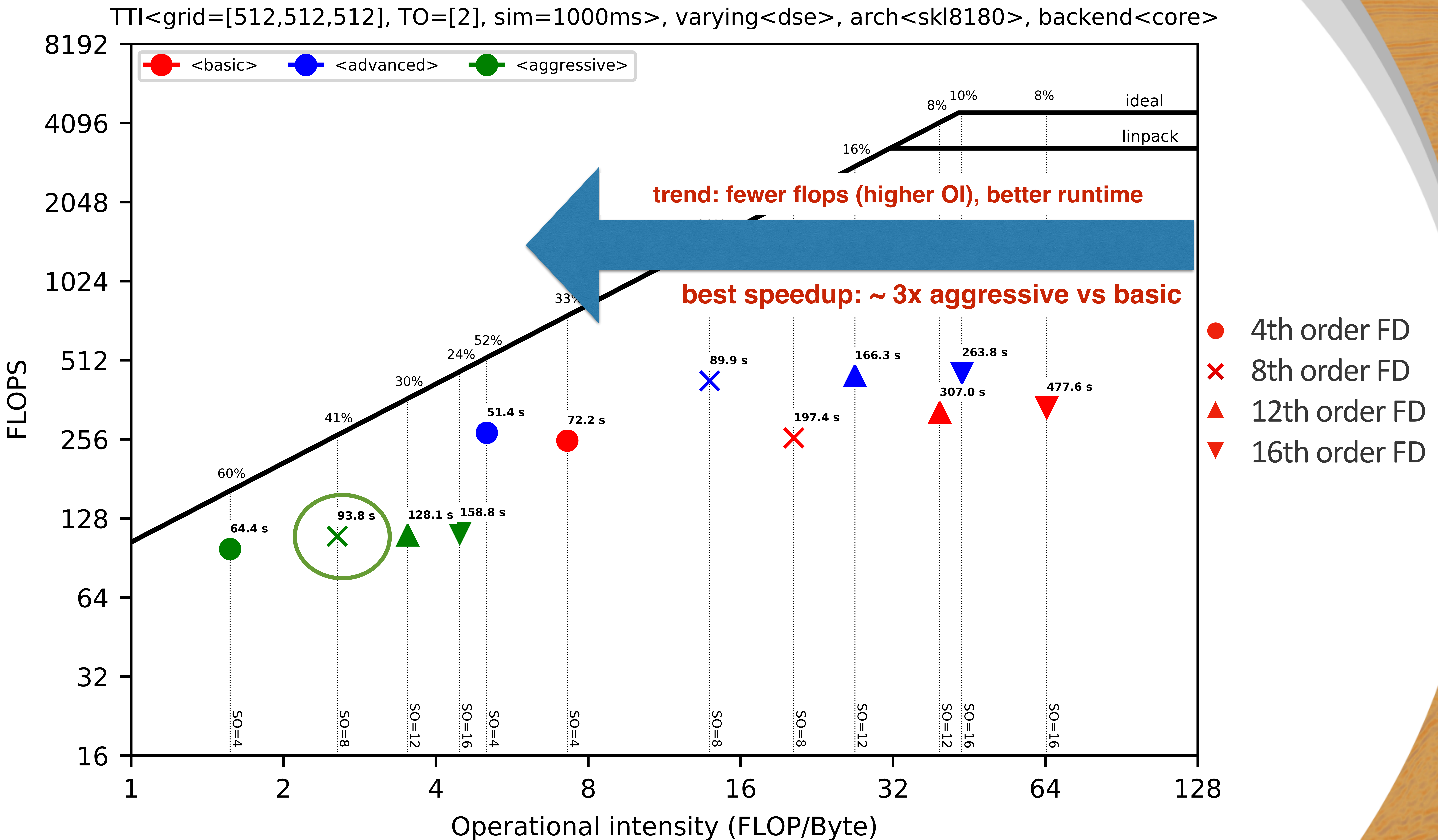
Type and variable declarations, instrumentation for profiling, header files, globals, macros, …

CORE specialization | OPS specialization | YASK specialization

produces C code

compiled w/ -O3 + standard flags

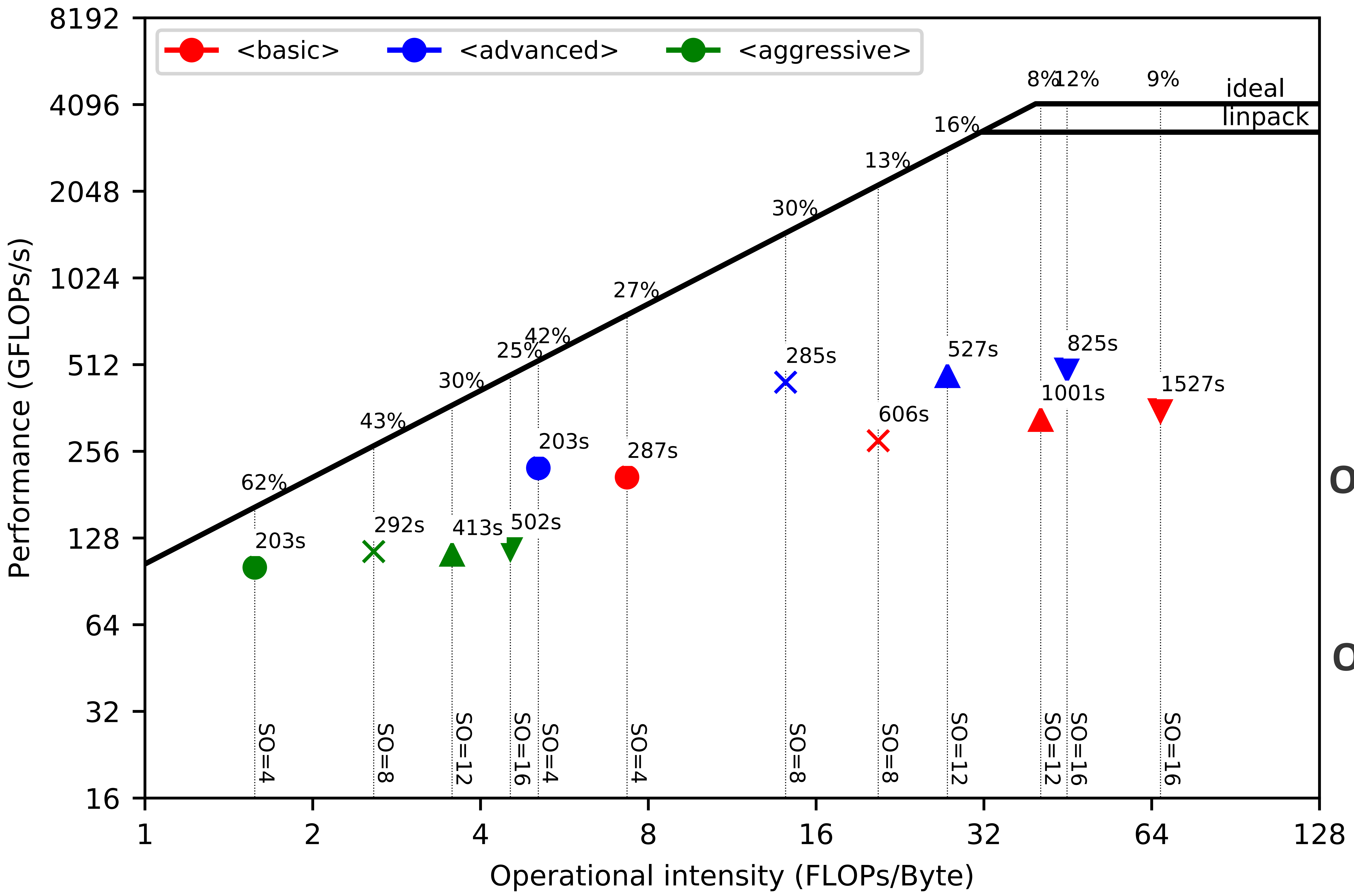
Single-socket — TTI on Skylake 8180



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

SLIM

Solution

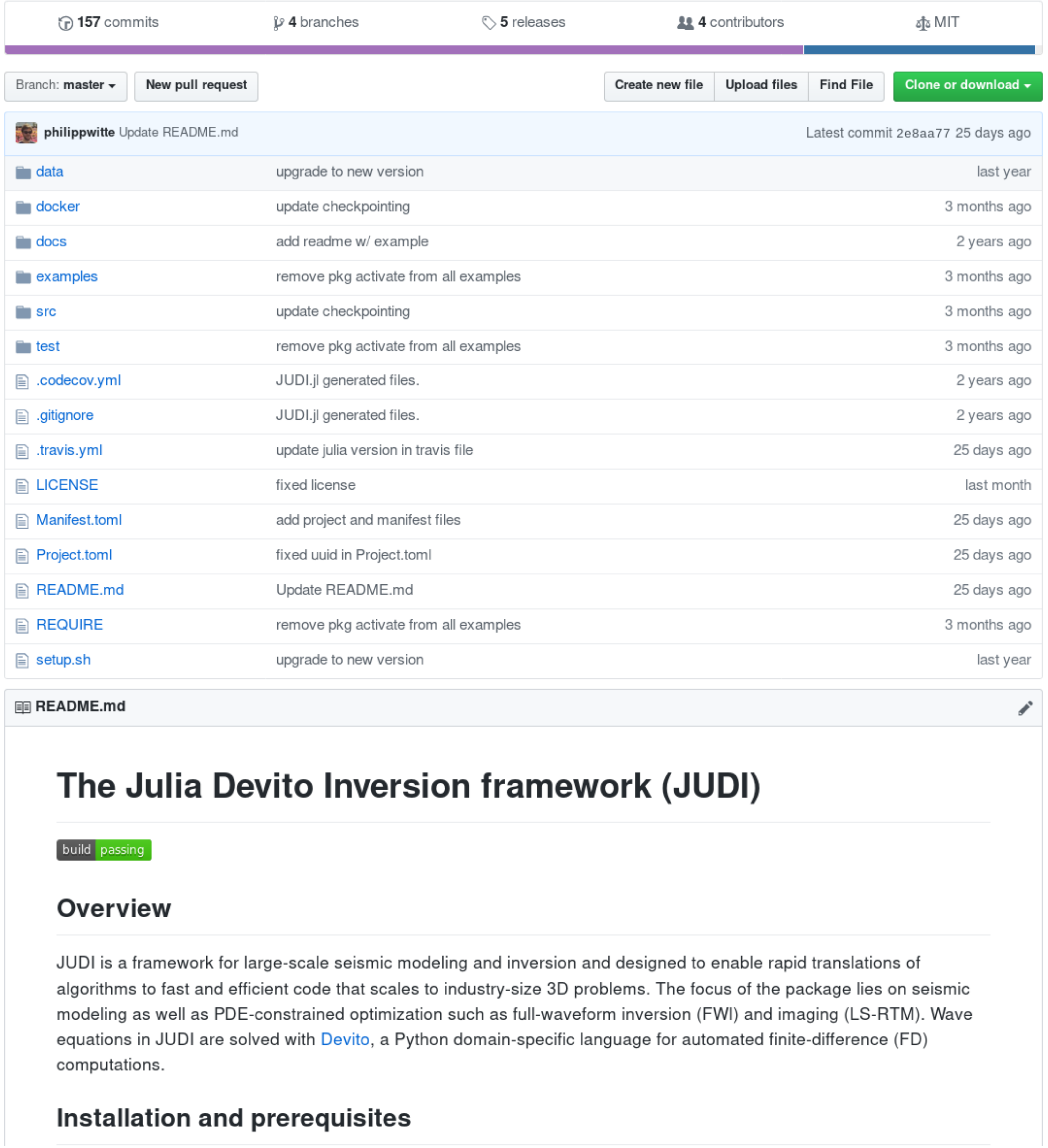
JUDI – Domain specific language for linear algebra abstractions, data parallelism & meta data in Julia

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

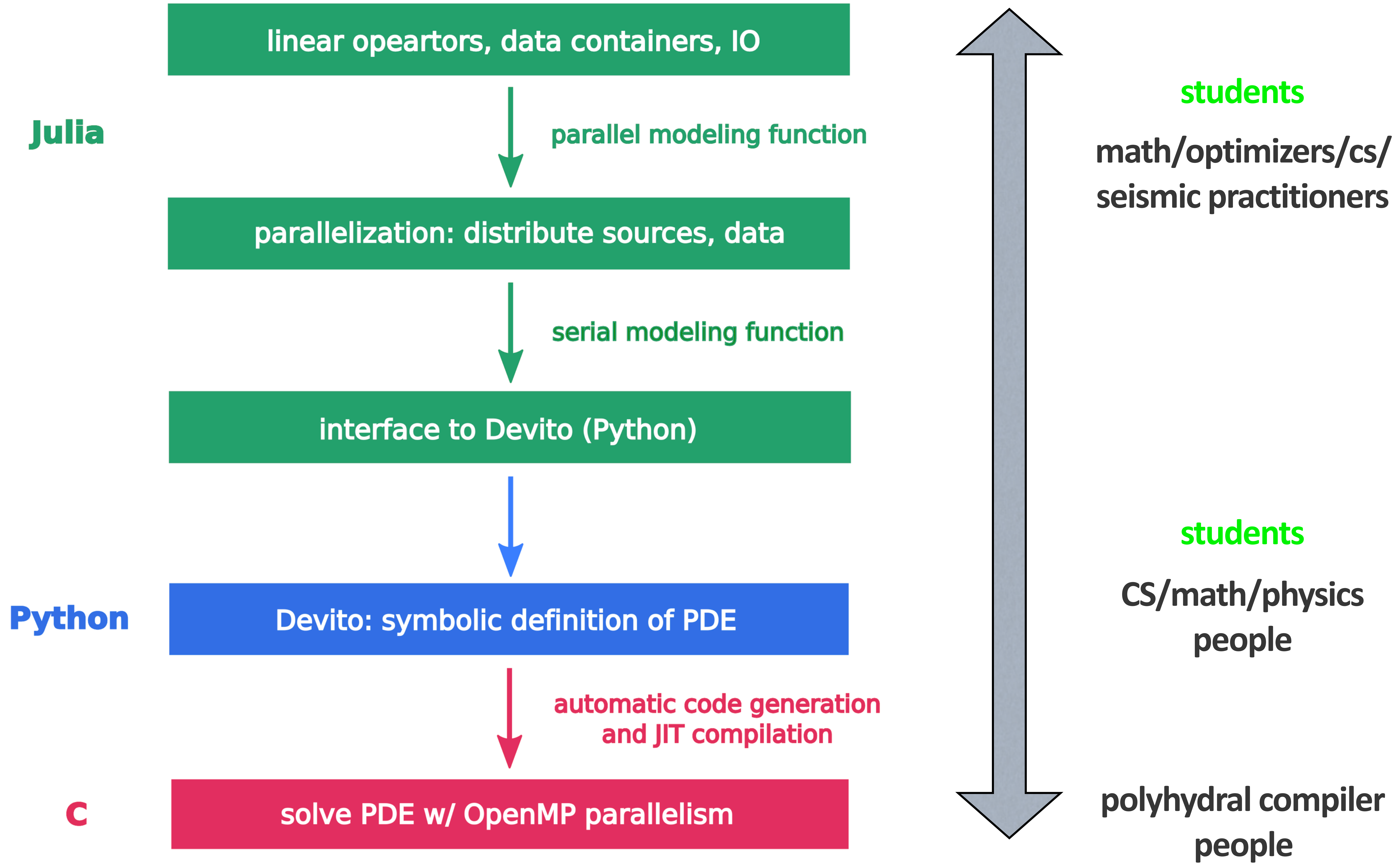
Open-source software

Julia Devito Inversion framework:

- JUDI.jl - MIT license
- Abstract linear operators and objective functions for FWI + LS-RTM
- Parallel out-of-core SEG-Y reader interface
- Interface to ML library Flux.jl
- URL: https://github.com/slimgroup/JUDI.jl



JUDI – true vertical integration



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
```

Example – LS-RTM w/ serial & parallel SGD

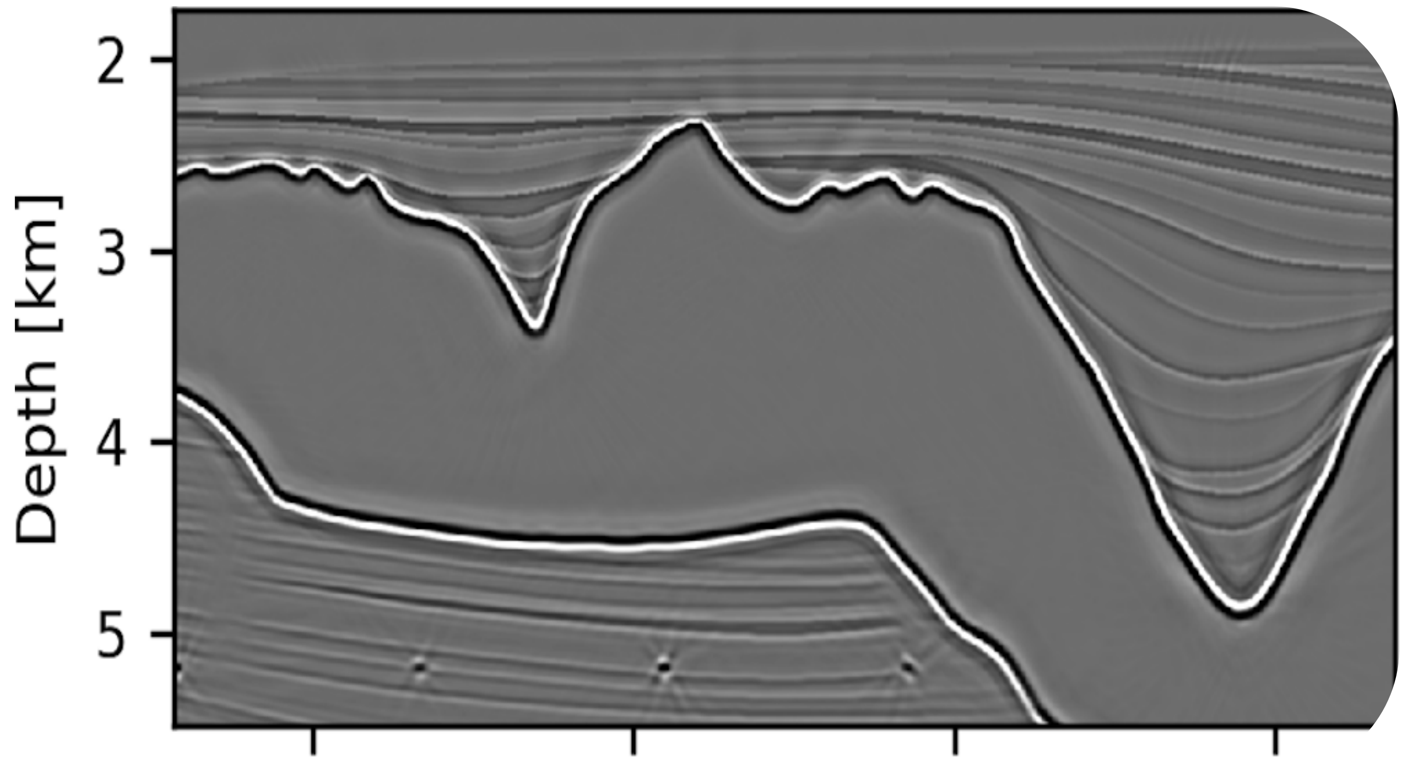
Algorithm 2 Preconditioned LS-RTM with elastic average SGD

for $j = 1$ to n **do**
  for $k = 1$ to p **do**
    $\mathbf{r}_j = \mathbf{M}_l^{-1}\mathbf{J}_{jk}\mathbf{M}_r^{-1}\mathbf{x}_j^k - \mathbf{M}_l^{-1}\delta\mathbf{d}_{jk}$
    $\mathbf{g}_j = \mathbf{M}_r^{-\top}\mathbf{J}_{jk}^{\top}\mathbf{M}_l^{-\top}\mathbf{r}_j$ and
    $\mathbf{x}_{j+1}^k = \mathbf{x}_j^k - \eta\mathbf{g}_j^k(\mathbf{x}_j^k) - \alpha(\mathbf{x}_j^k - \tilde{\mathbf{x}}_j)$
  end for
  $\tilde{\mathbf{x}}_{j+1} = (1-\beta)\tilde{\mathbf{x}}_j + \beta(\frac{1}{p}\sum_{i=1}^{p}\mathbf{x}_j^i)$
end for

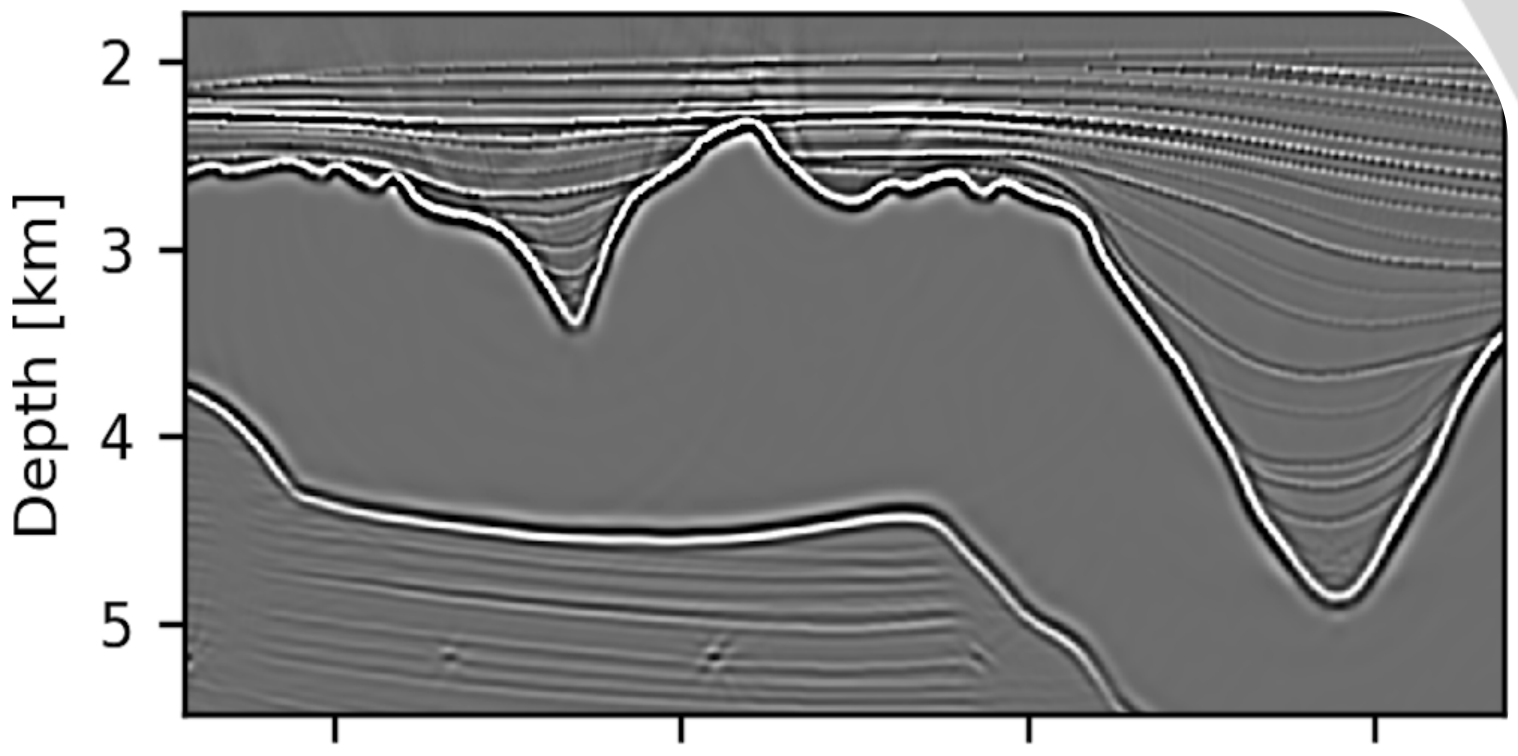
```
# Gradient function
@everywhere function update_x(Ml,J,Mr,x,d,eta,alpha,xav)
  r = Ml*J*Mr*x - Ml*d
  g = Mr'*J'*Ml'*r
  return x - eta*g - alpha*(x - xav)
end
update_x_par = remote(update_x) # Parallel function wrapper

for j=1:niter
  @sync begin
    for k=1:p

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

      # Calculate x update in parallel
      xnew[:,k] = update_x_par(Ml,Jsub,Mr,x[:,k],
                               dsub,eta,alpha,xav)
    end
  end

  # Update average variable
  xav = (1 - beta)*xav + beta*(1/p *sum(x,2))
  x = copy(xnew)
end
```

Least-squares migration

- Making Broadband Least-squares Reverse-Time Migration affordable
- Iteratively refining the output toward true reflectivity
- Suppressing migration artefacts, wavelet sidelobes, incorrect amplitudes, poor illumination
- Move LS-RTM from 10 x RTM compute cost to 2 x RTM compute cost
- Applicable to all existing datasets

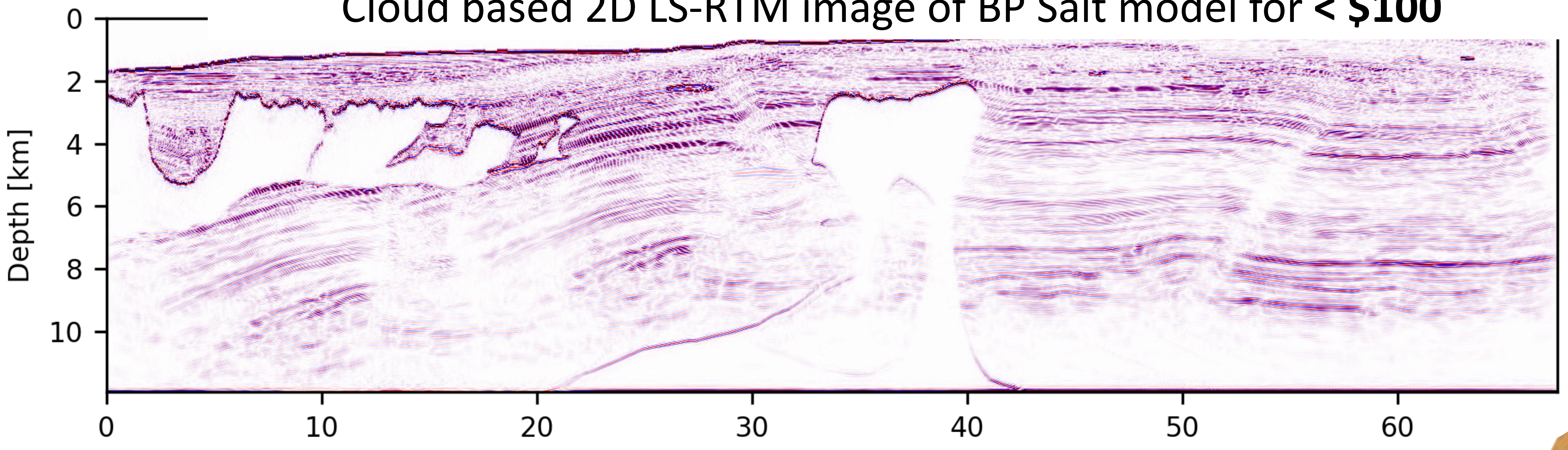


Sparsity-promoting LS-RTM: 2 data passes



RTM: 1 data pass

Cloud based 2D LS-RTM image of BP Salt model for **< $100**



Observations

Demonstrated power of abstractions towards data & compute intensive tasks

- flexibility w.r.t hardware (GPU, different CPUs etc.)
- exploits data-space parallelism
- exploits model space parallelism w/ multithreading & domain decompositions
- allows for reproducibility
- **ready to scale technology to 3D TTI in Cloud?**

"Not" ready for

- elastic

Philipp A. Witte, Mathias Louboutin, Henryk Modzelewski, Charles Jones, James Selvage, and Felix J. Herrmann, "**Event-driven workflows for large-scale seismic imaging in the cloud**", 2019
Philipp A. Witte, Mathias Louboutin, Henryk Modzelewski, Charles Jones, James Selvage, and Felix J. Herrmann, "**An Event-Driven Approach to Serverless Seismic Imaging in the Cloud**". 2019

Solution

Serverless Cloud – Large-scale event-driven seismic imaging w/ automatic resource allocations, resilient nested levels of parallelization, and containerization

https://www.devitoproject.org

Seismic inversion in the cloud

Cloud computing:

Microsoft Azure

amazon web services

Google Cloud Platform

✓ Pros

- Theoretically unlimited scalability
- High flexibility (hardware, jobs)
- No upfront + maintenance costs: pay-as-you-go
- Available to anyone
- **No compromise – latest hardware & architectures available (GPUs, ARM)**

✗ Cons

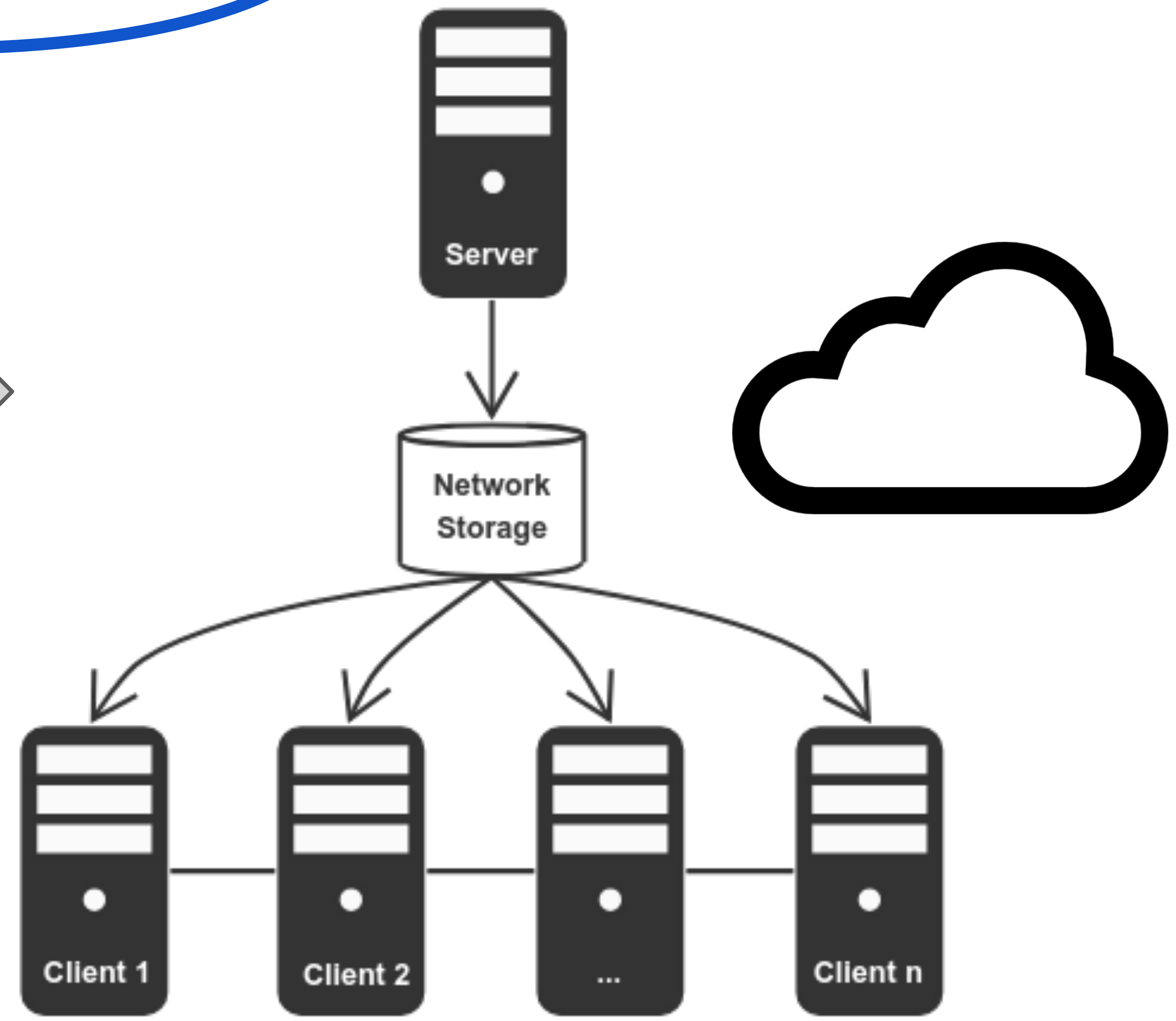
- Slower inter-node connections (depending on platform)
- Oftentimes larger MTBF
- High costs if not used properly
- Need to transition software
- Steep learning curve

Moving to the cloud

```c
#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);
```

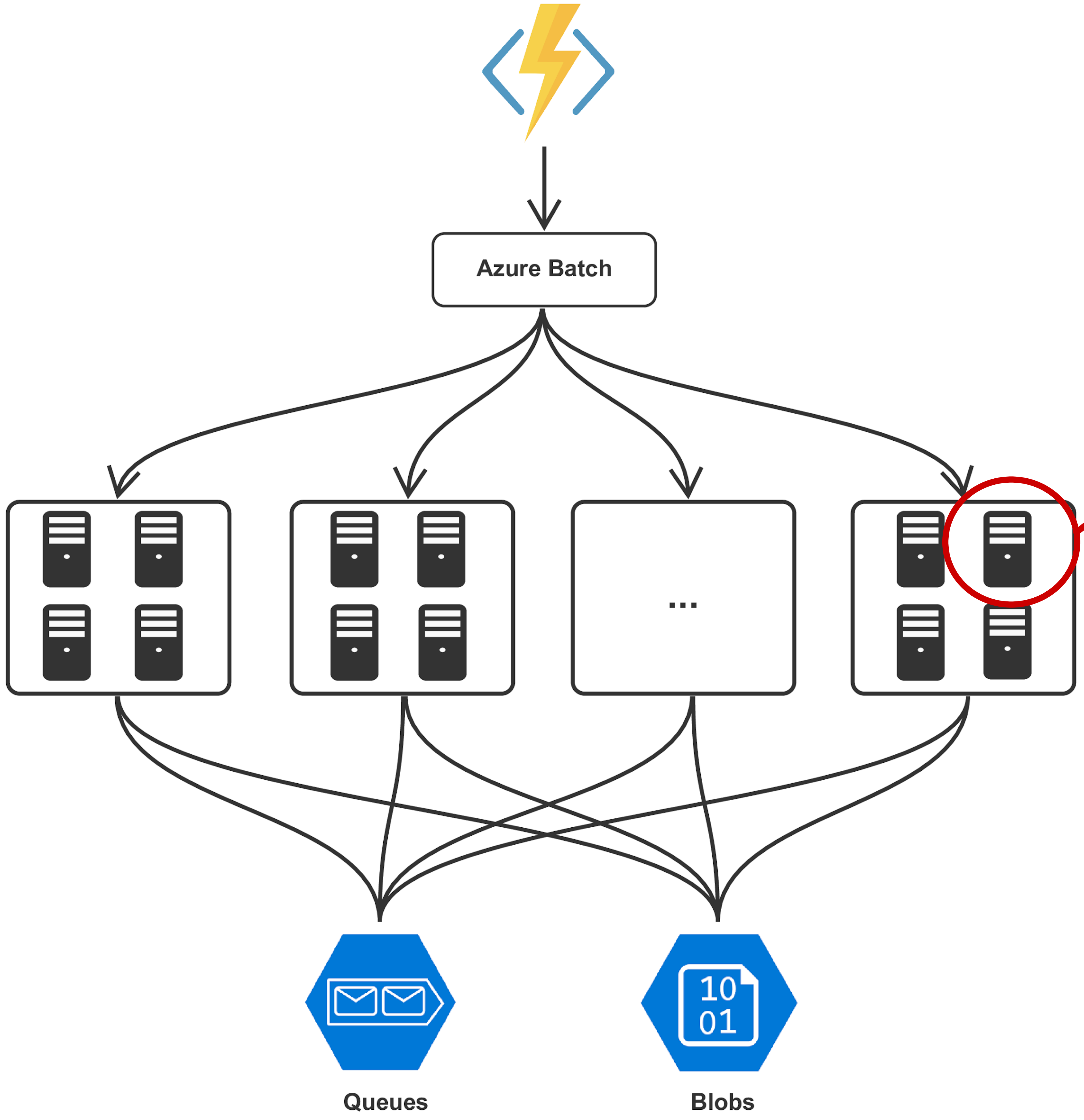
Legacy Fortran or C code

Lift & shift



Moving to the cloud

Go serverless
(and re-engineer)

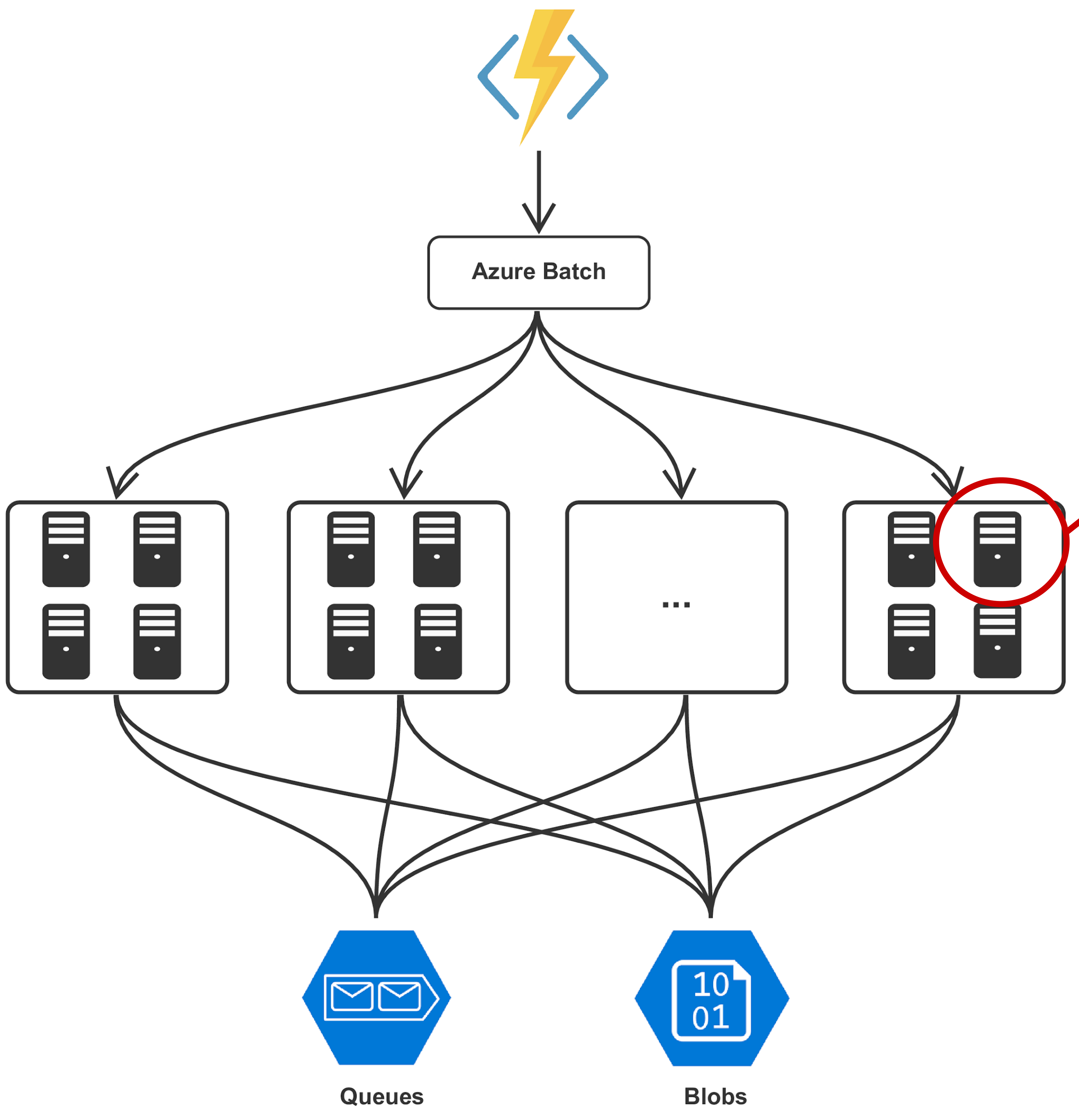


Automatic code generation

```c
#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);
```

Moving to the cloud

Go serverless (and re-engineer)



Devito

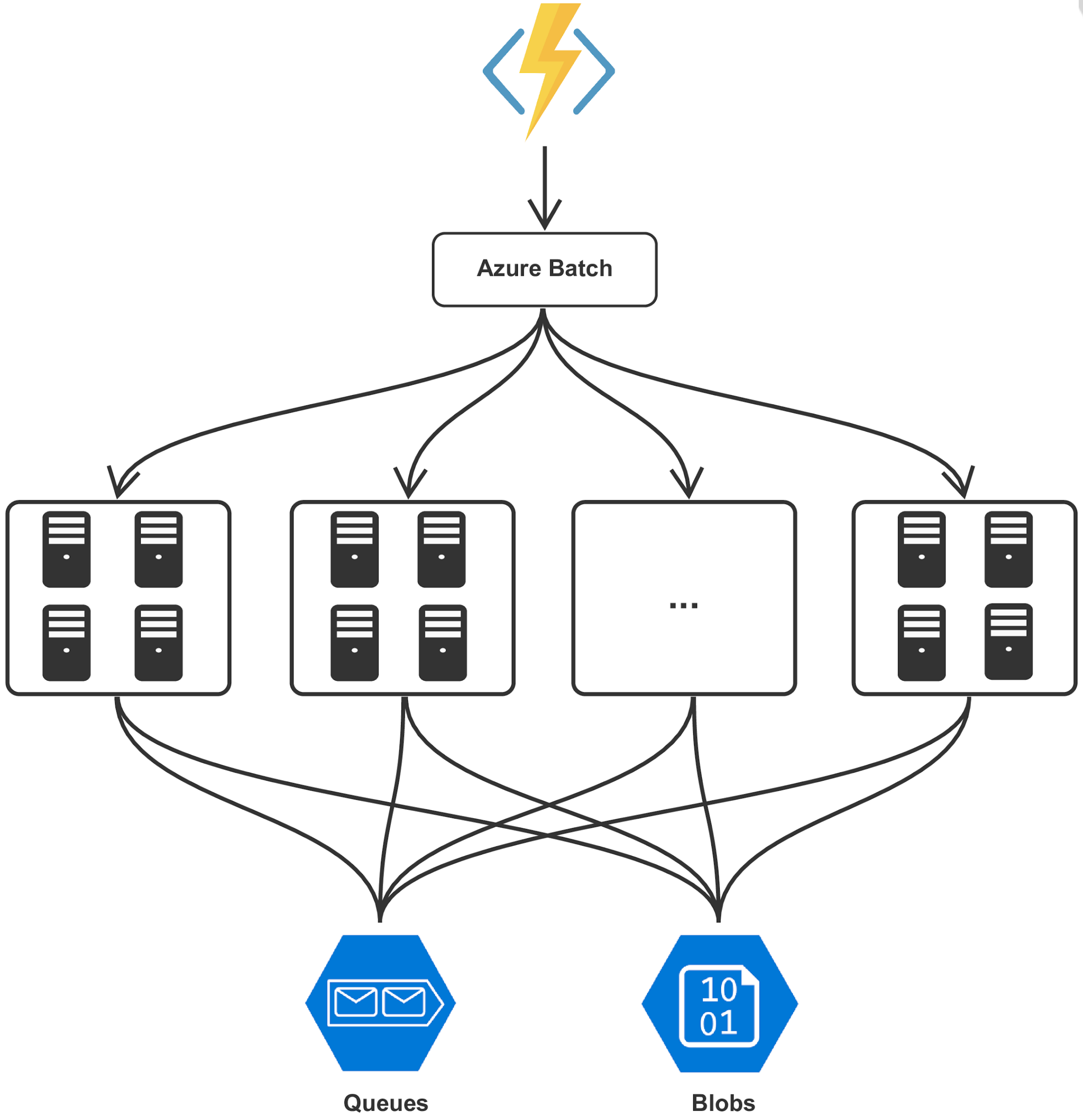
```
pde = model.m*u.dt2 - u.laplace + model.damp*u.dt
```

- **Save cost (up to 10x): no idle instances, lower start-up time**
- Resilience managed by cloud platform
- **Requires re-engineering of software**

RTM/FWI gradients

Nested levels of parallelization:

- Parallelize shot records (Azure Batch)
- Domain decomposition (MPI)
- Multithreading (OpenMP)
- Each gradient computed on individual instance **or** cluster of instances (cluster of clusters)

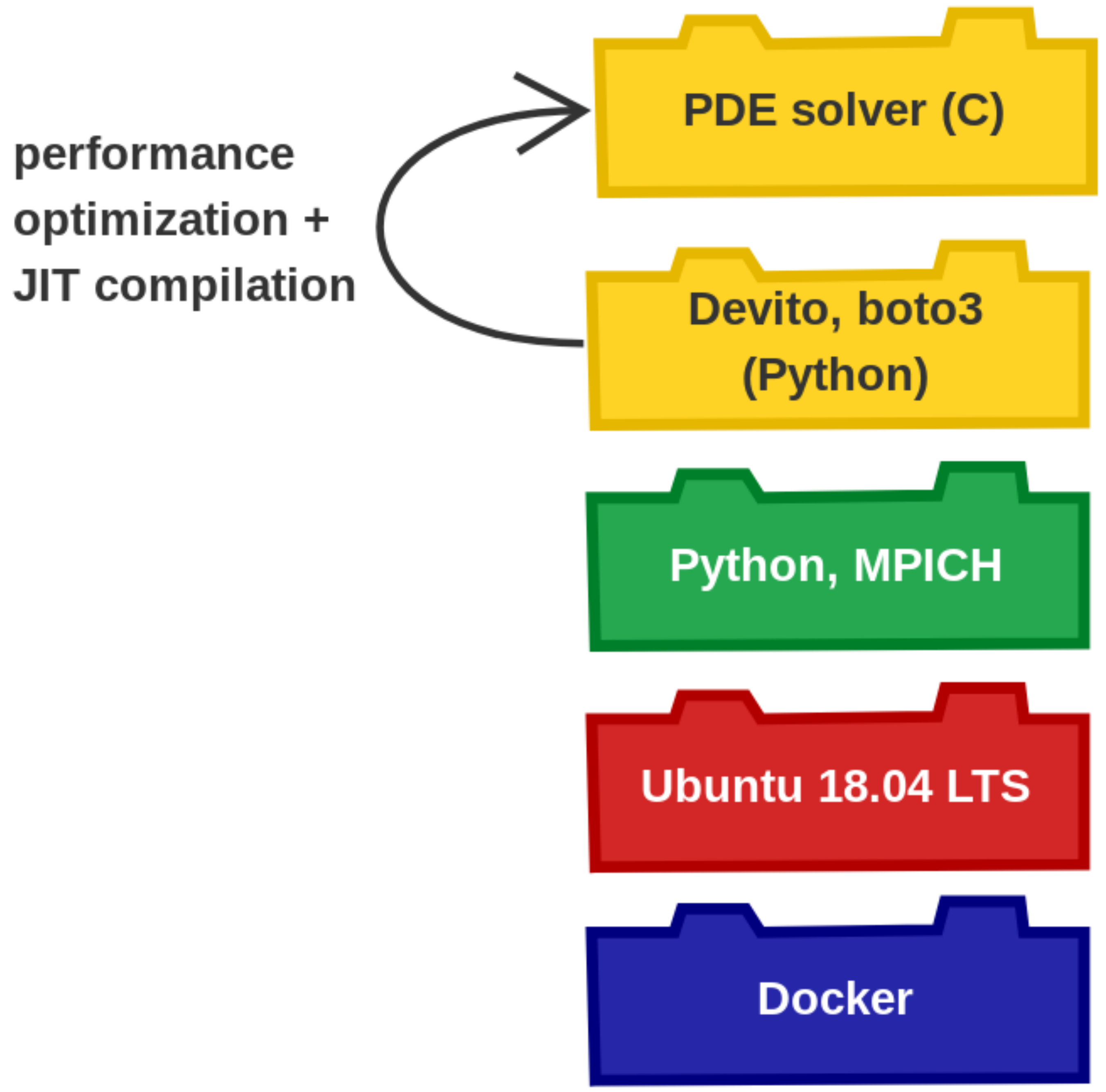


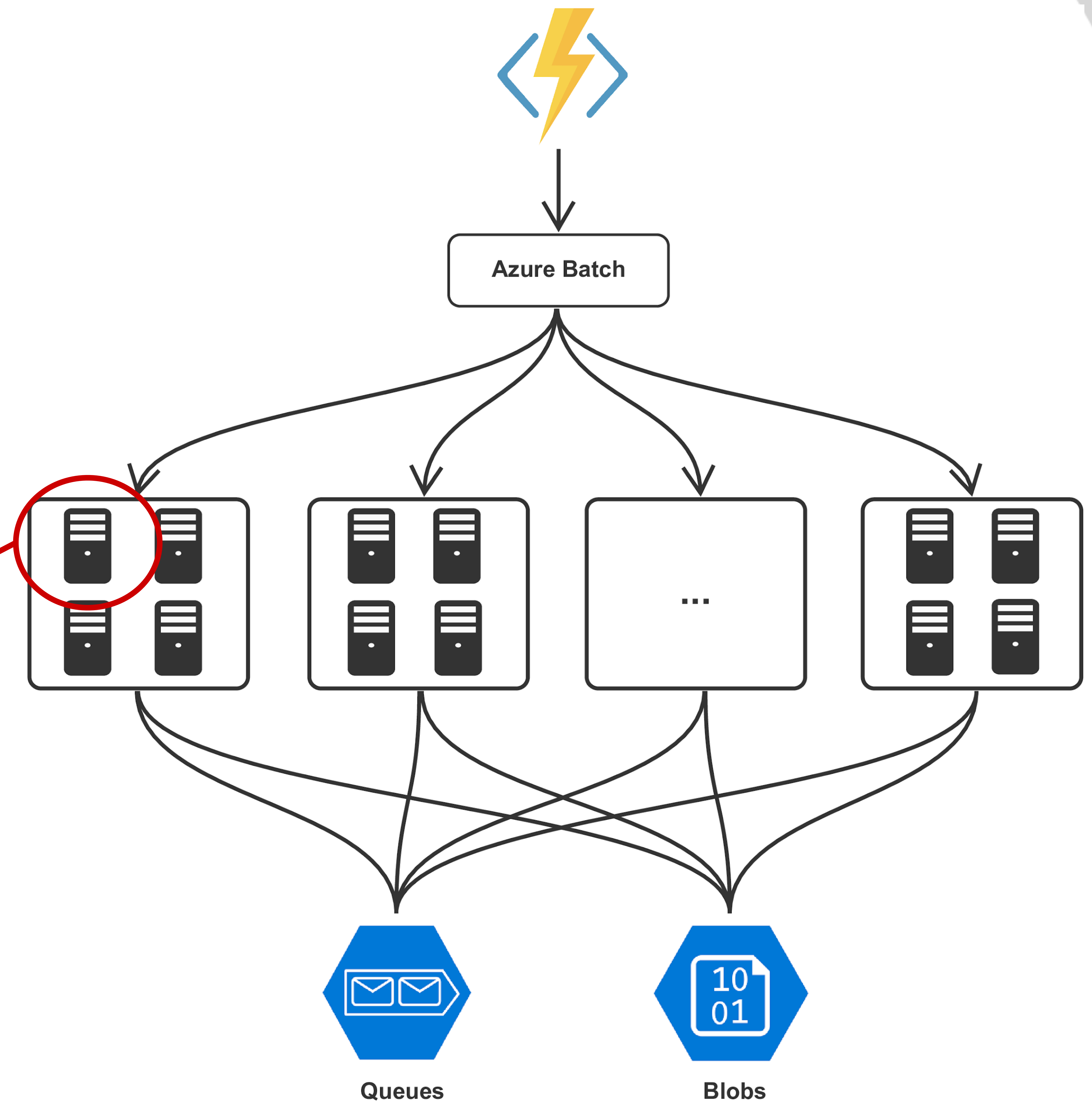
* Luporini et al., 2018; Louboutin et al., 2019

RTM/FWI gradients

Software stack:

- Batch runs docker containers
- Solve wave equations using Devito*
- Automated performance optimizations (loop blocking, vectorization, refactoring, OMP, MPI, etc.)

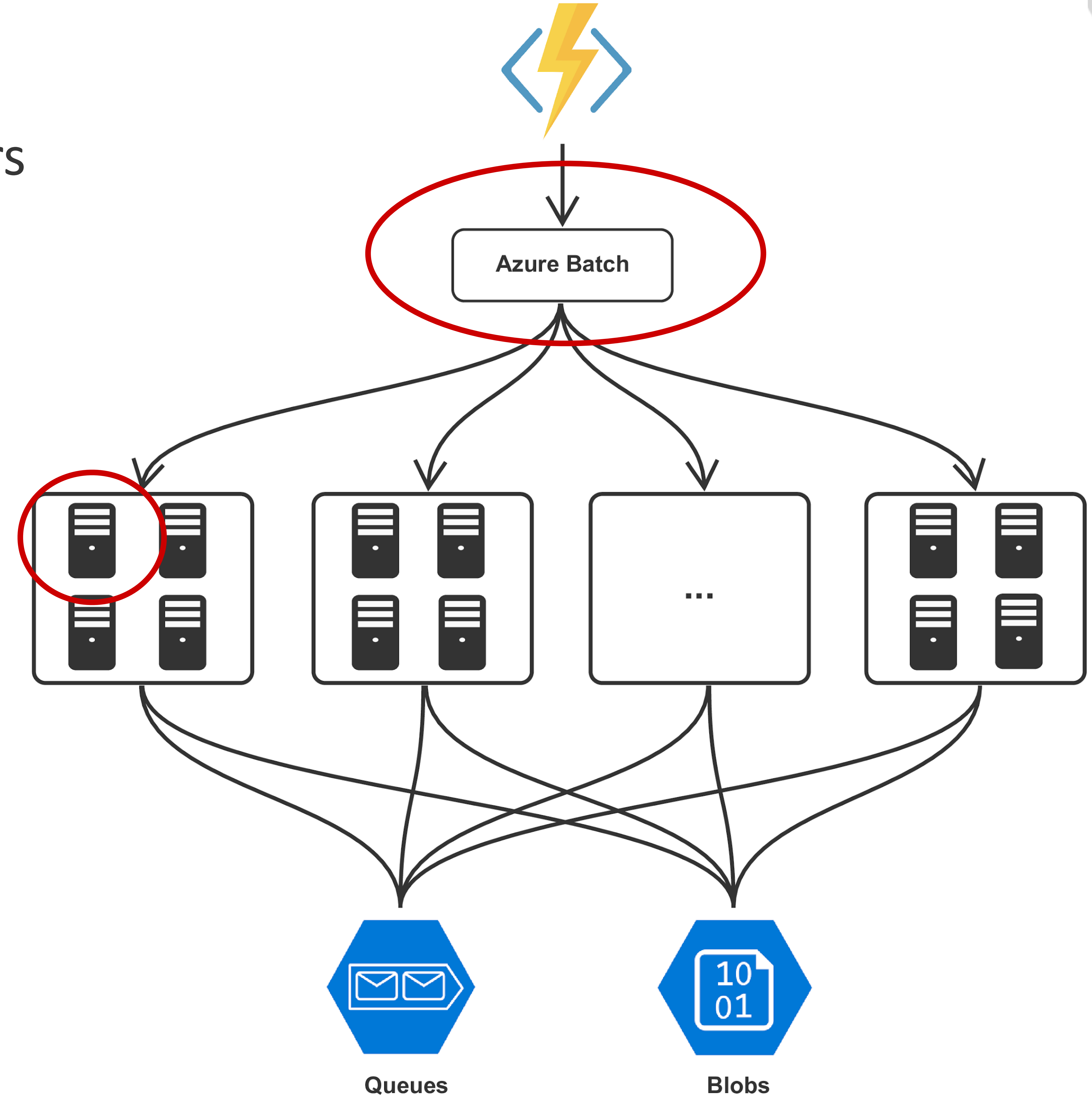




RTM/FWI gradients

Azure - Batch Shipyard:

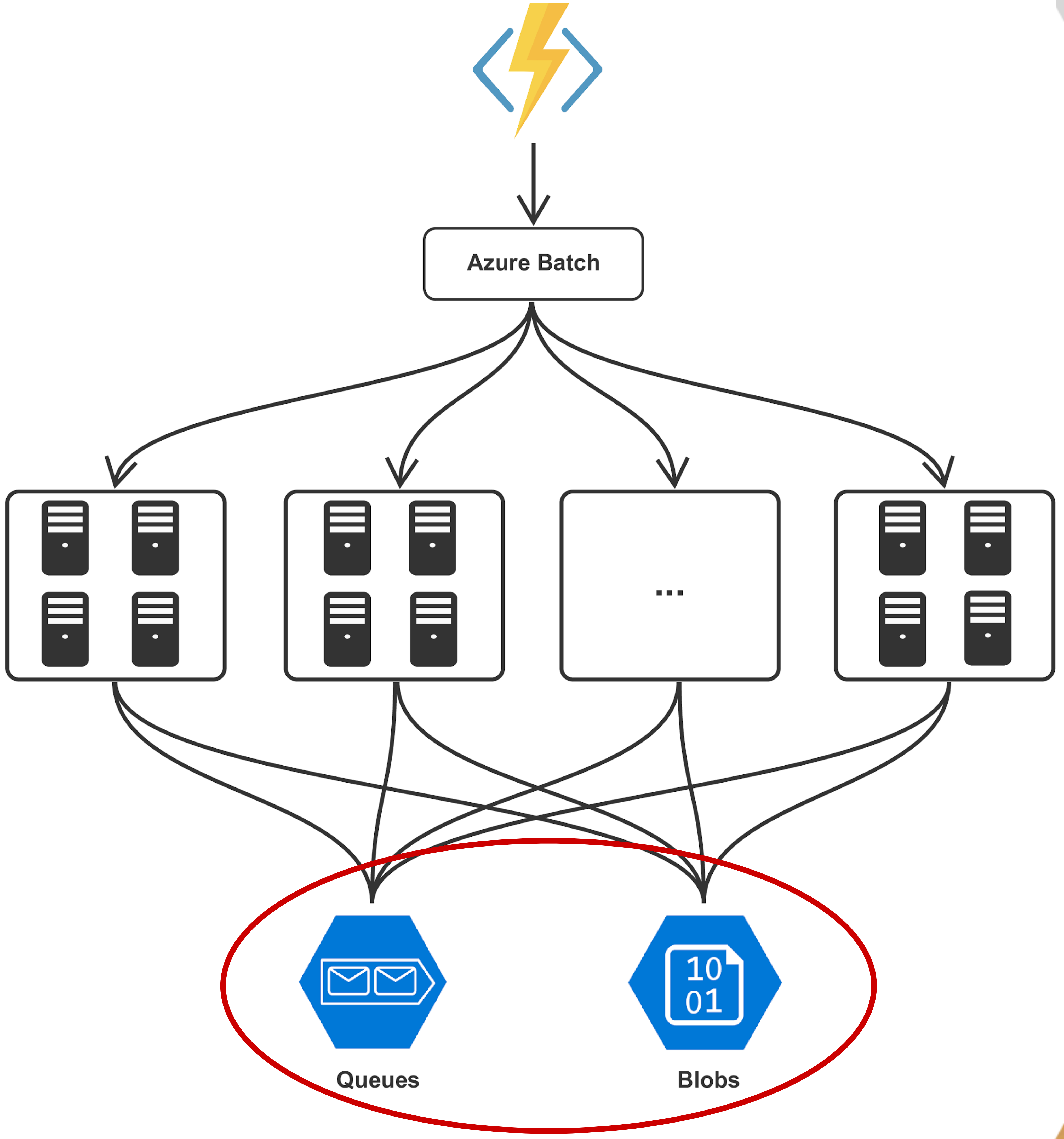
- Tool for executing + monitoring batch jobs
- Many templates for docker + **singularity** containers
- Pre-exisiting containers for MPI, **Infiniband**, ML, various compilers, etc.
- Configure pools + jobs using high-level yaml files
- Developed by Microsoft + **open source:** https://github.com/Azure/batch-shipyard



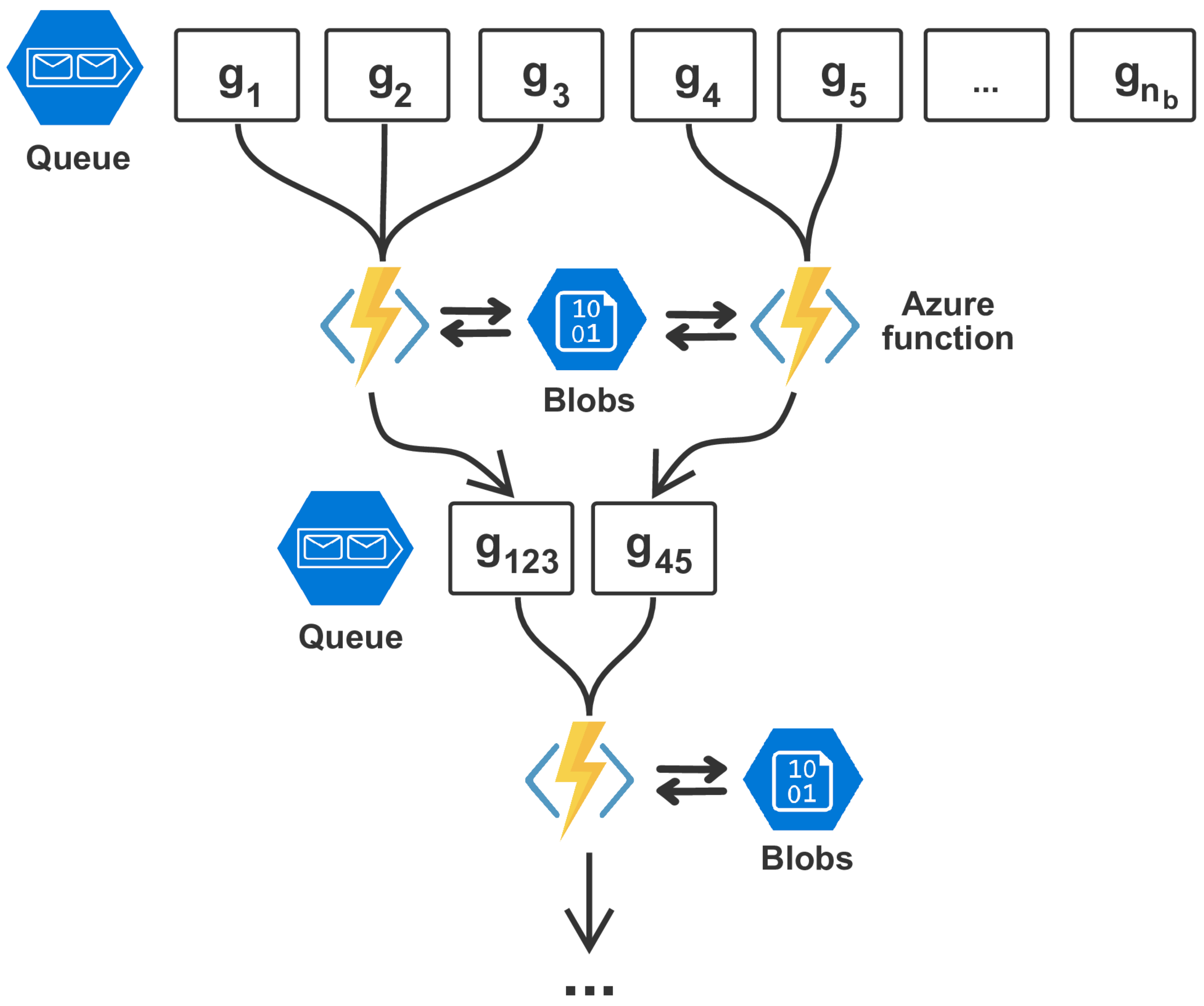
RTM/FWI gradients – map reduce

Summation:

- Gradients stored in object storage (blob)
- Virtually unlimited I/O scalability
- Send object IDs to message queue
- Event-driven gradient summation using Azure functions



RTM/FWI gradient computations

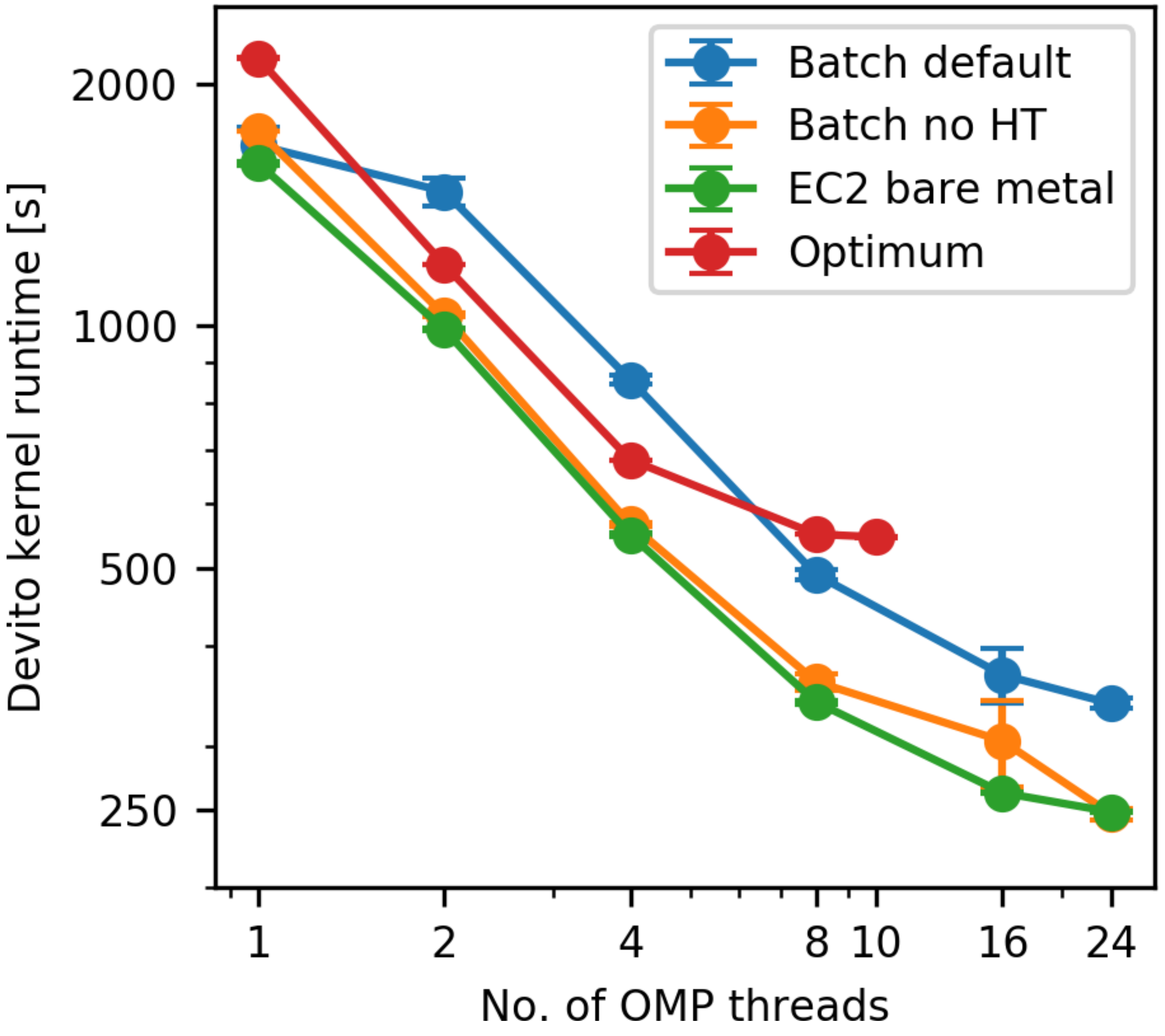


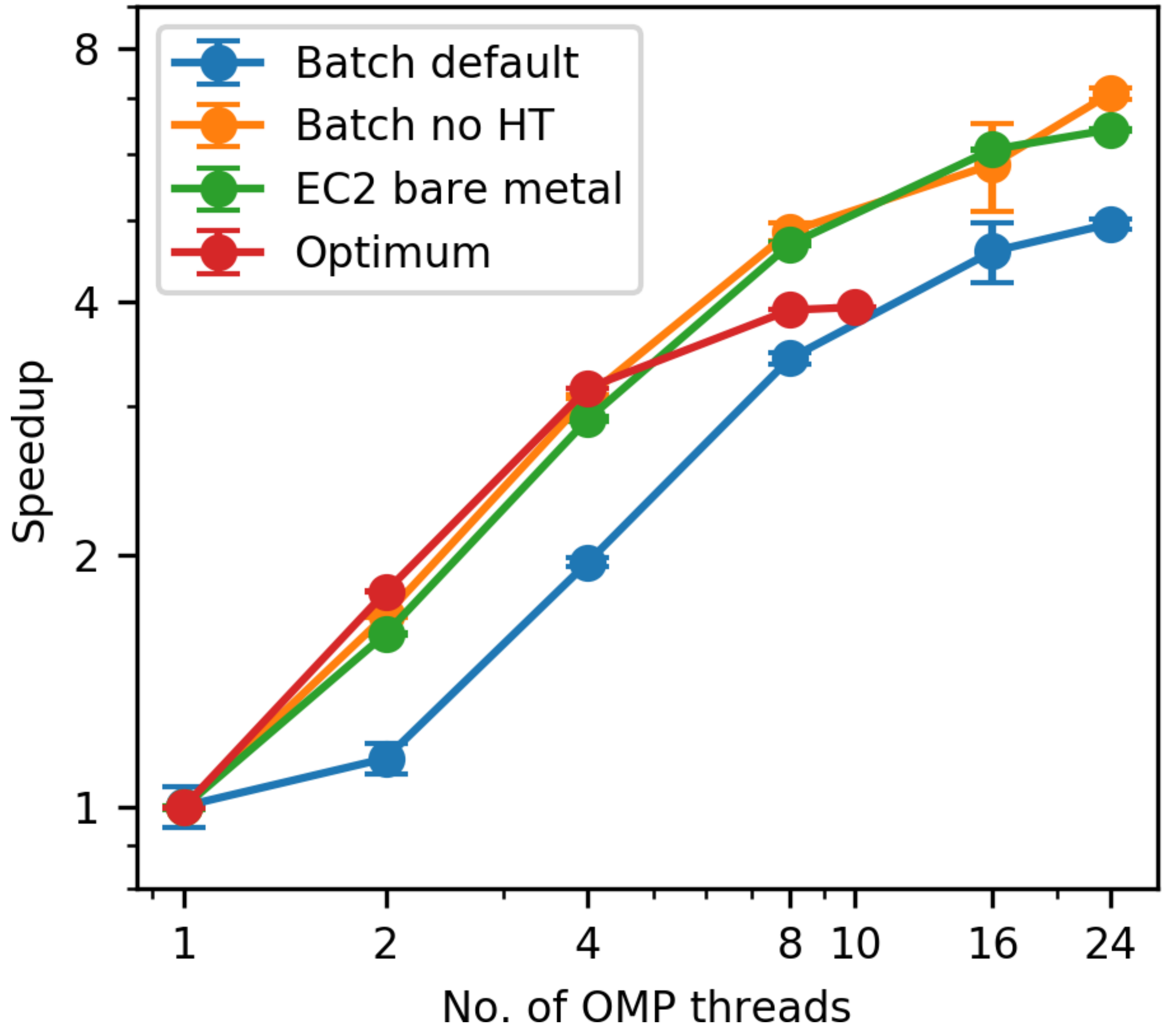
Event-driven gradient reduction:

- Azure functions
- Cheaper than pay-as-you-go nodes
- Asynchronous & parallel
- Invoked as soon as at least 2 gradients are available
- Stream gradients from blob → sum → write back
- Update image after final summation

Strong scaling - OpenMP

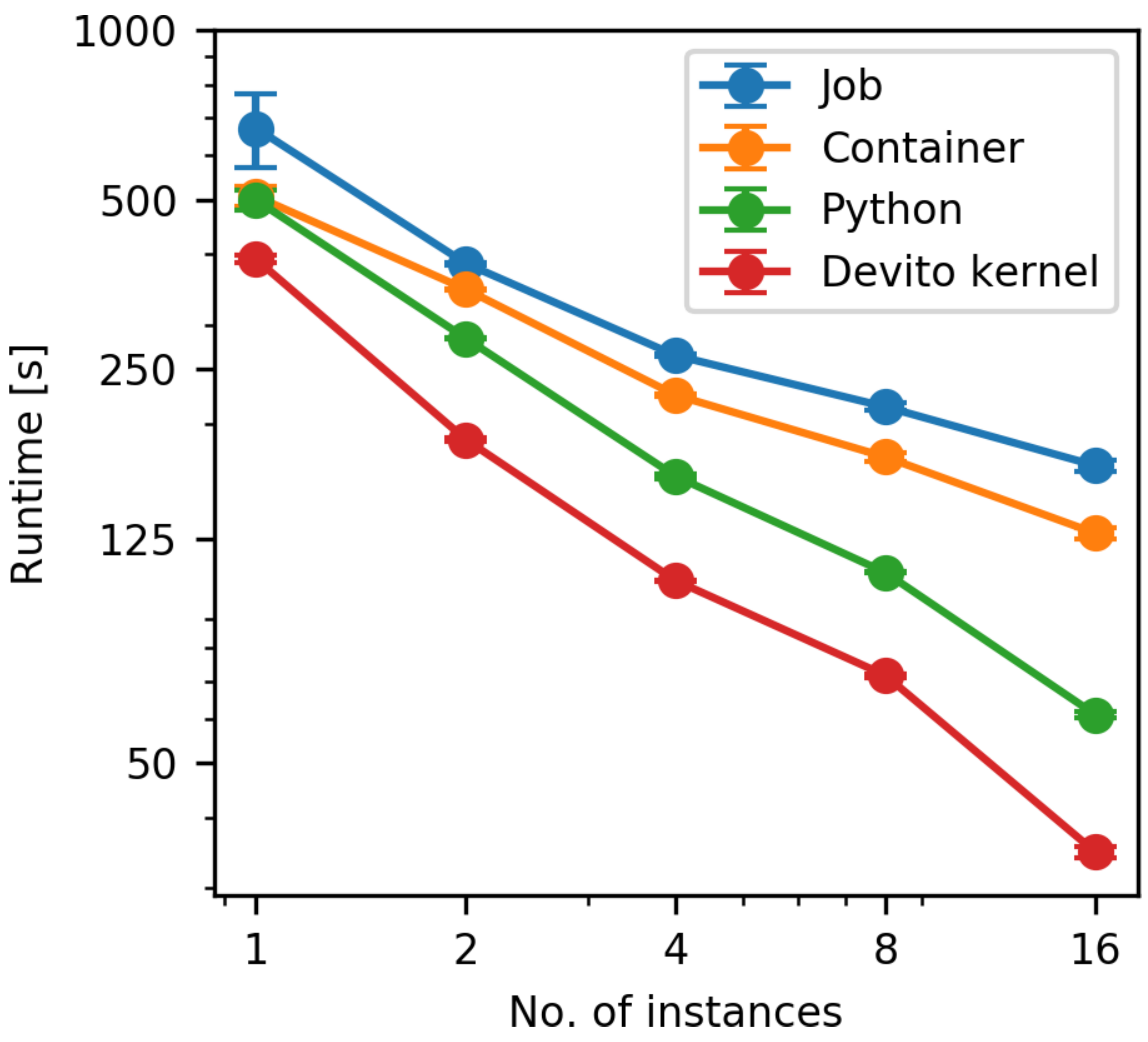
- Fixed workload: 1 gradient
- Runtime as function of no. of threads
- Performance on bare metal vs. container similar (w/o hyperthreading)

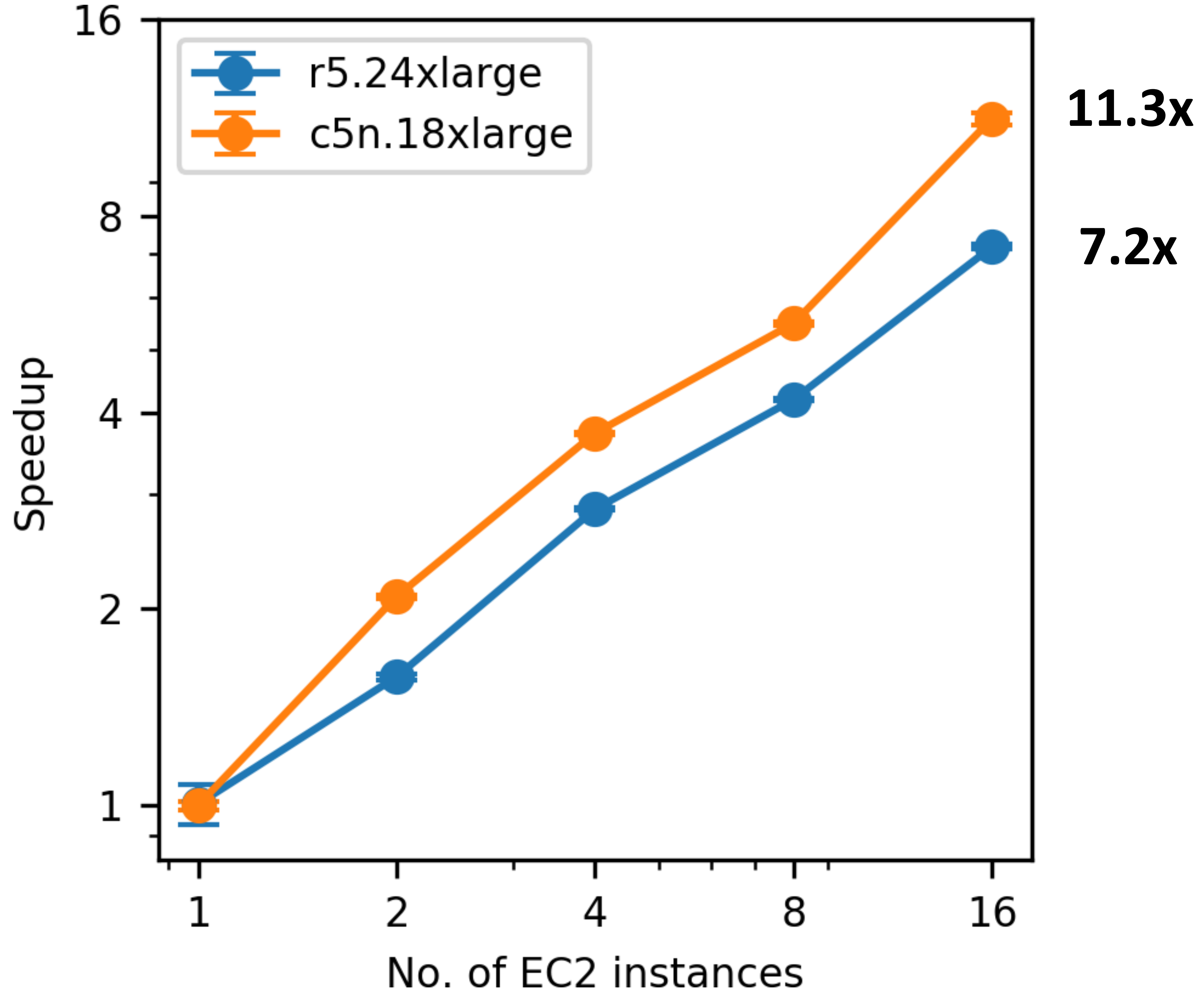




Strong scaling - MPI

- Fixed workload: 1 gradient
- Runtime as function of no. of instances (per gradient)
- Good speed-up **but** significant cost increase





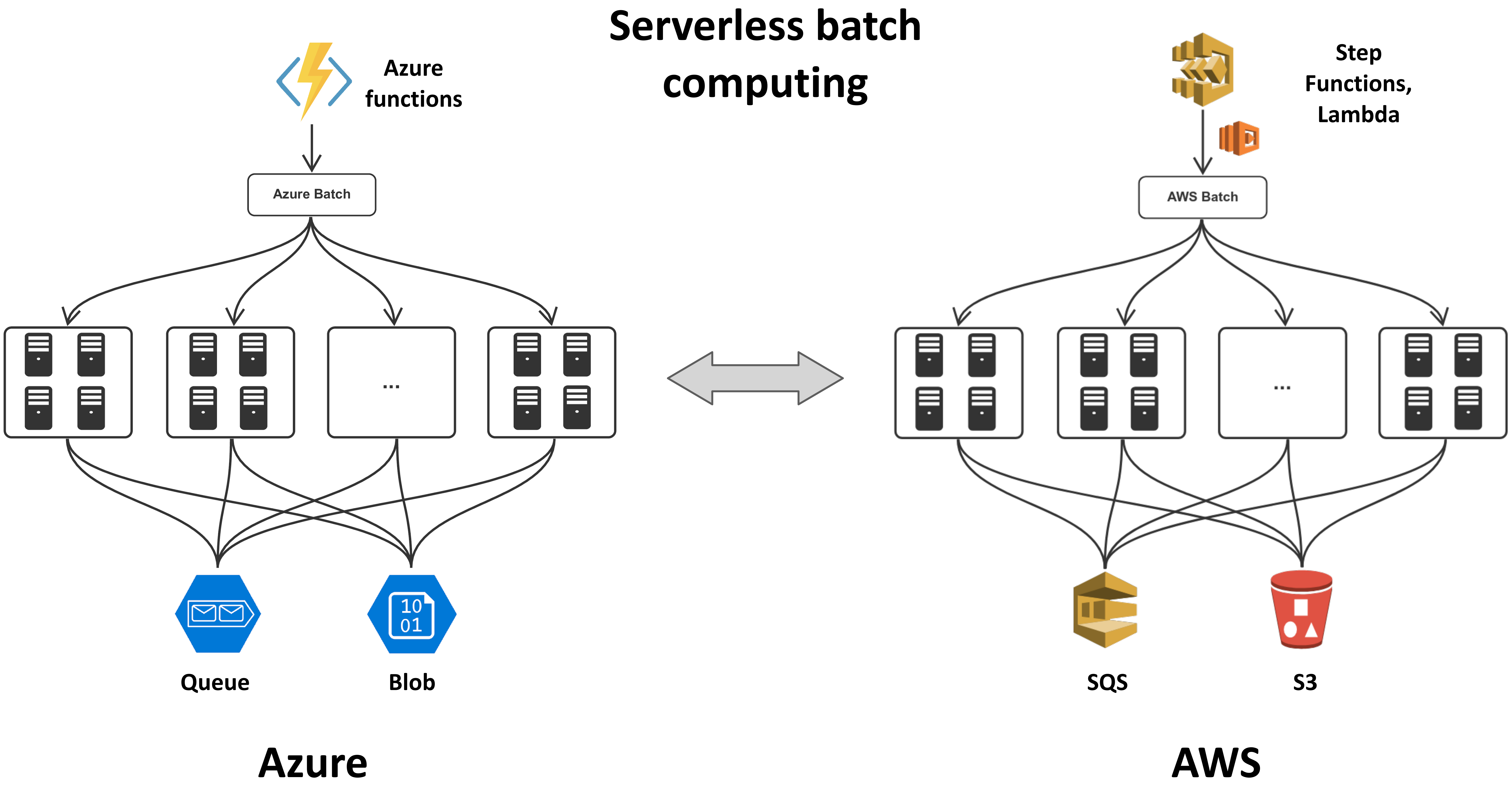
Multi platform approach

	Azure	AWS	GCP
Compute instances	Virtual machines	EC2	Compute engine
Object storage	Blob	S3	Cloud storage
Batch computing	Azure Batch	AWS Batch	Pipelines
Serverless functions	Azure functions	Lambda functions	Cloud functions
Message queues	Queue storage	SQS	Cloud Pub/Sub
Distributed file system	Azure files	EFS	Cloud filestore

https://docs.microsoft.com/en-us/azure/architecture/aws-professional/services

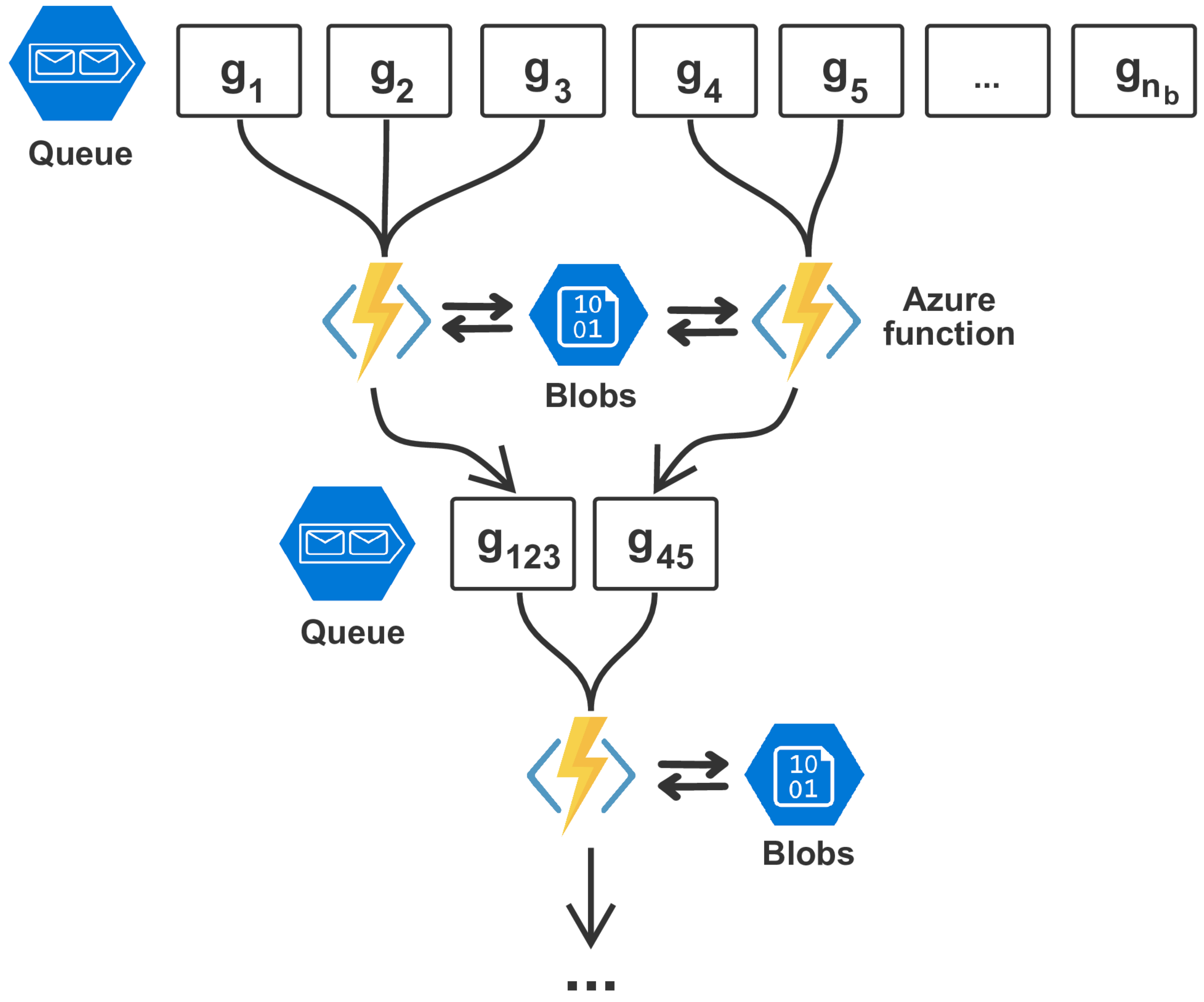
https://cloud.google.com/docs/compare/aws/

Multi platform approach

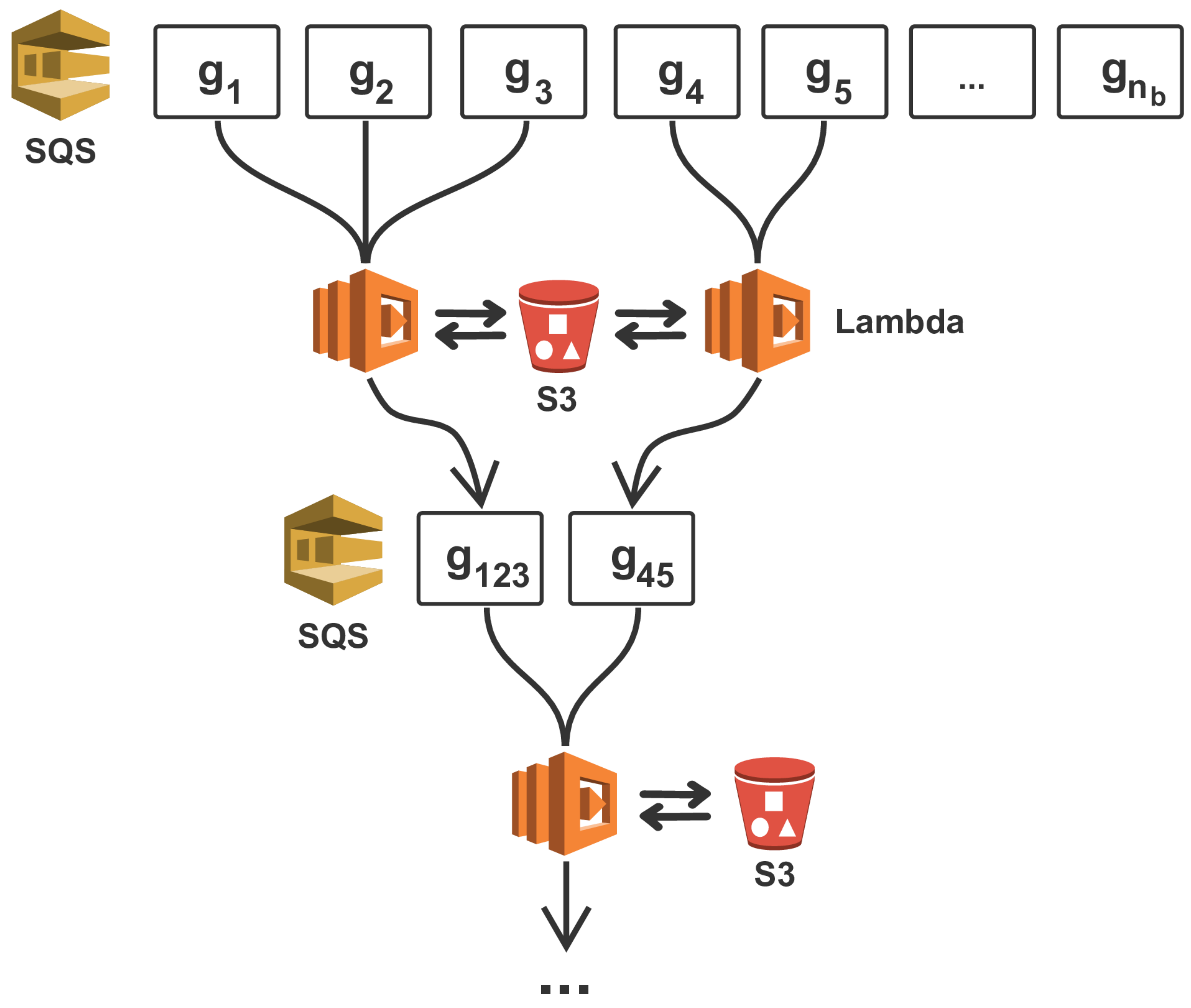


Multi platform approach

Event-driven gradient summation



Azure



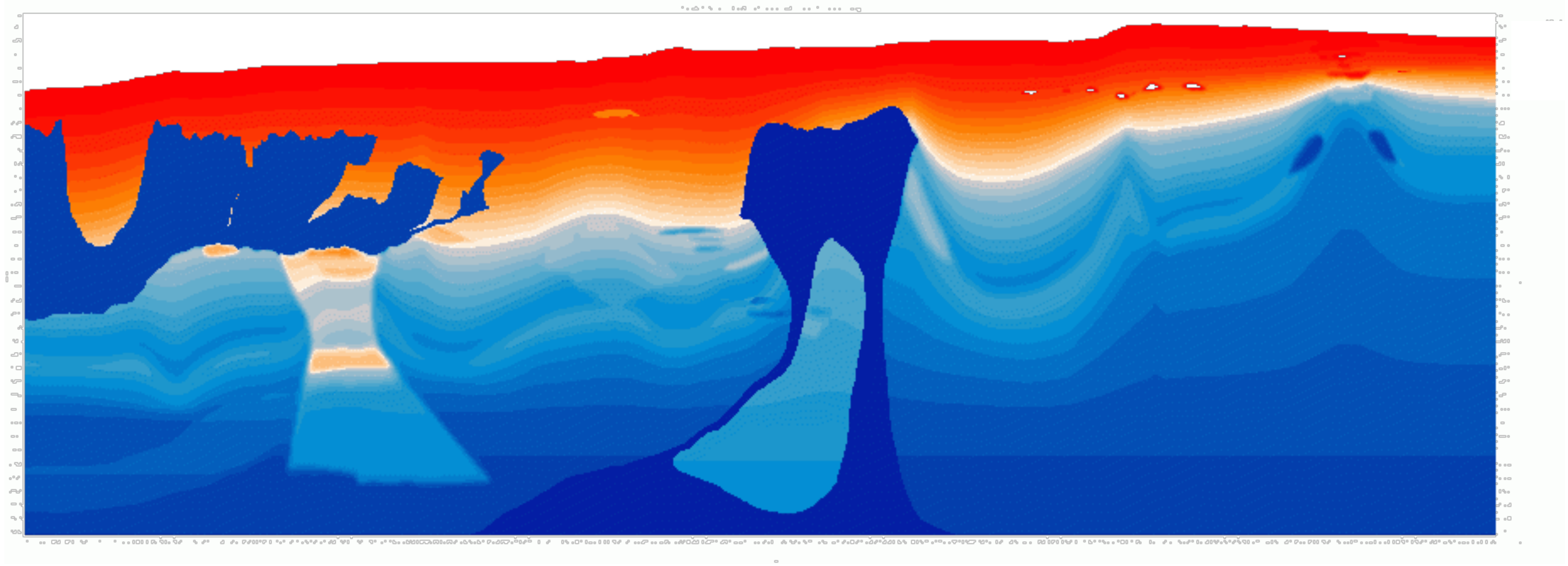
AWS

Billette and Brandsberg-Dahl, 2004

Numerical examples

Sparsity-promoting LS-RTM of the BP Synthetic 2004 model:

- 1348 shot records
- Velocity model: 67.4 x 11.9 km (10,789 x 1,911 grid points)
- 20 iterations of linearized Bregman method
- Batchsize of 200 shot records per iteration
- Curvelet-based sparsity promotion



BP Synthetic 2004

Numerical examples

Sparsity-promoting LS-RTM on the BP Synthetic 2004 model

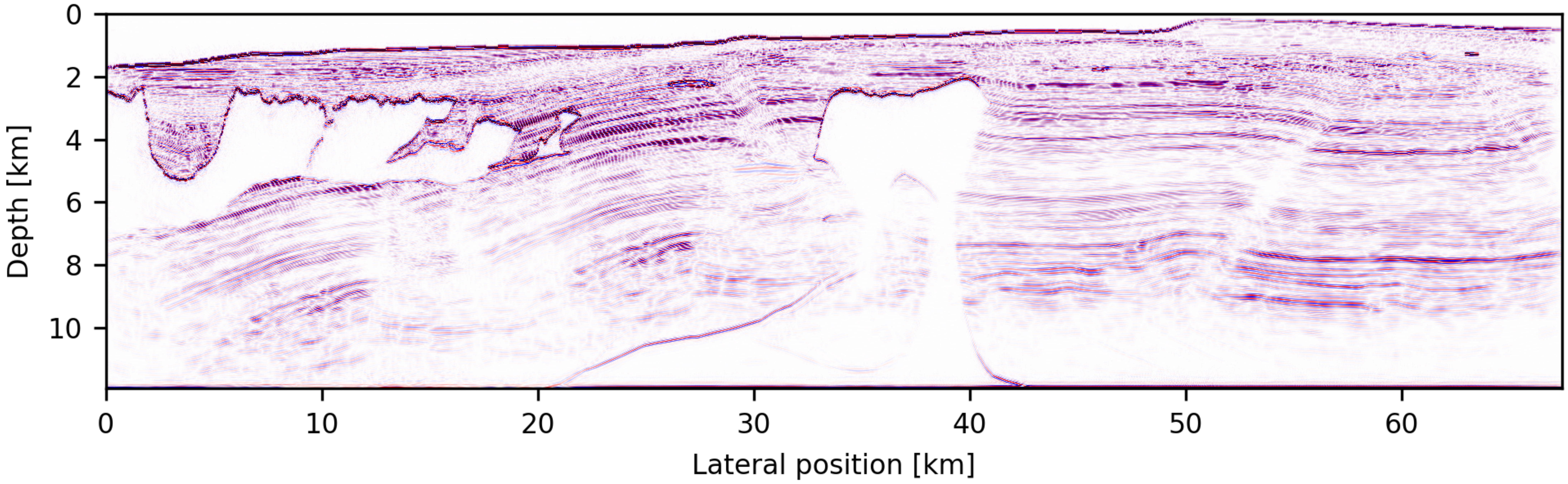
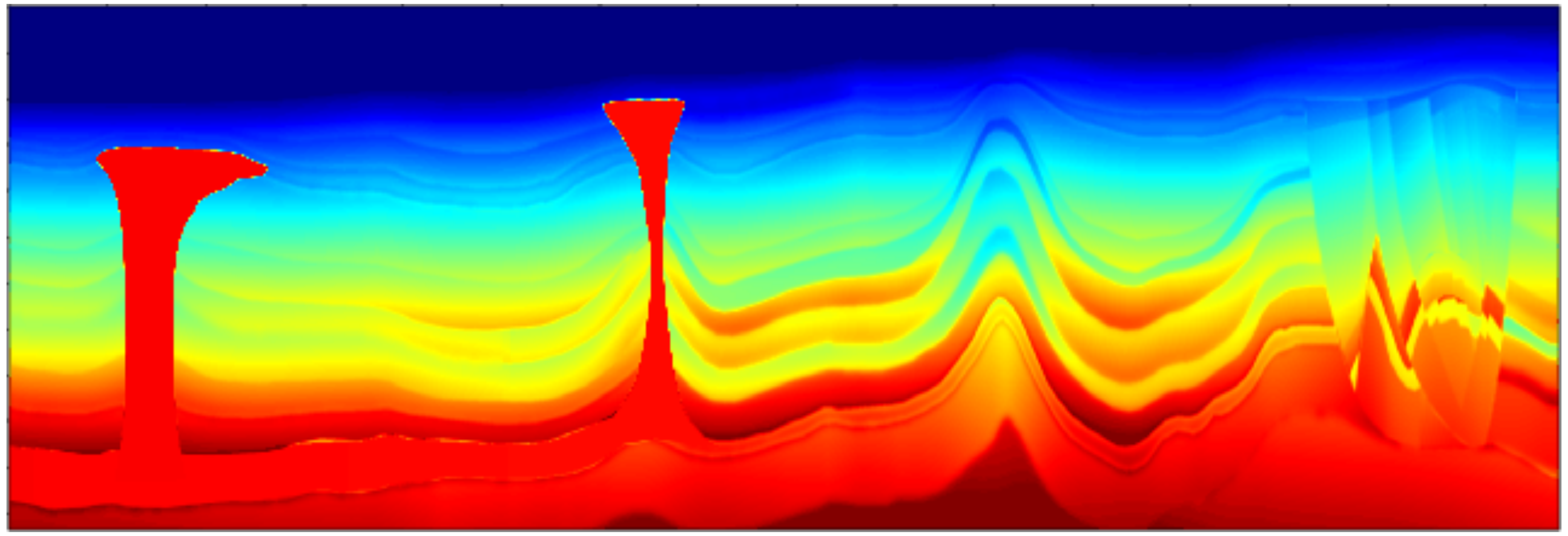


Image after ≈3 data passes (total cost of < 120 $)

Numerical examples

Reverse-time migration of the BP TTI model:

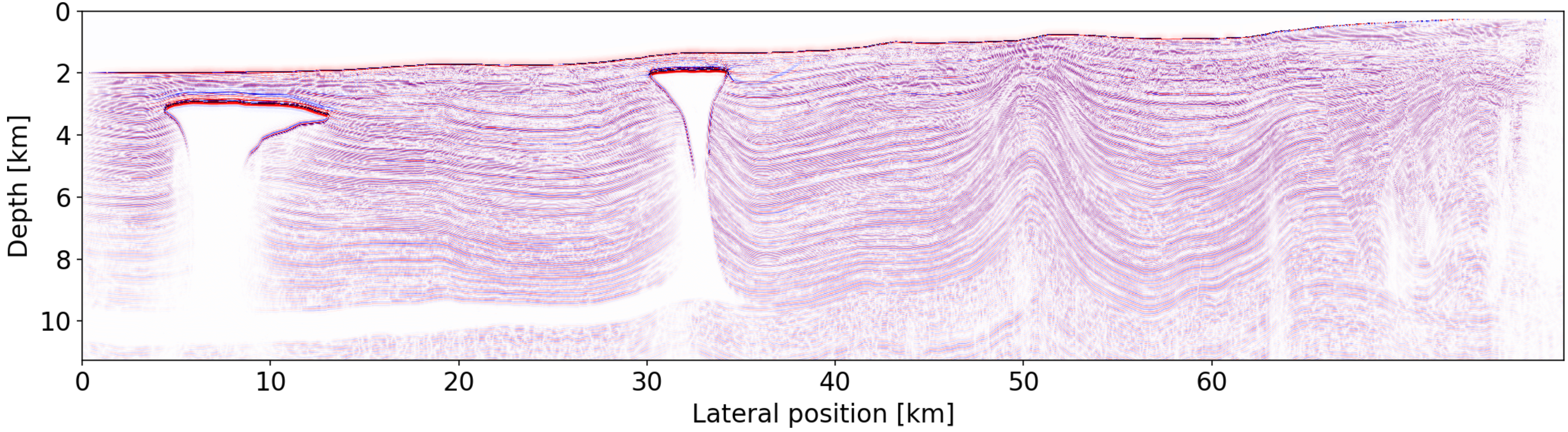
- 1641 shot records
- Velocity model: 78.7 x 11.3 km (12,596 x 1,801 grid points)
- Anisotropic modeling using pseudo-acoustic TTI equations*
- True adjoints of linearized Born scattering operator
- Domain-decomposition to compute gradients
- Each gradient computed on MPI cluster of 6 instances (no spot instances)



BP TTI 2007

Numerical examples

Reverse-time migration of the BP TTI model:



RTM image (total cost of $\approx$420 $)

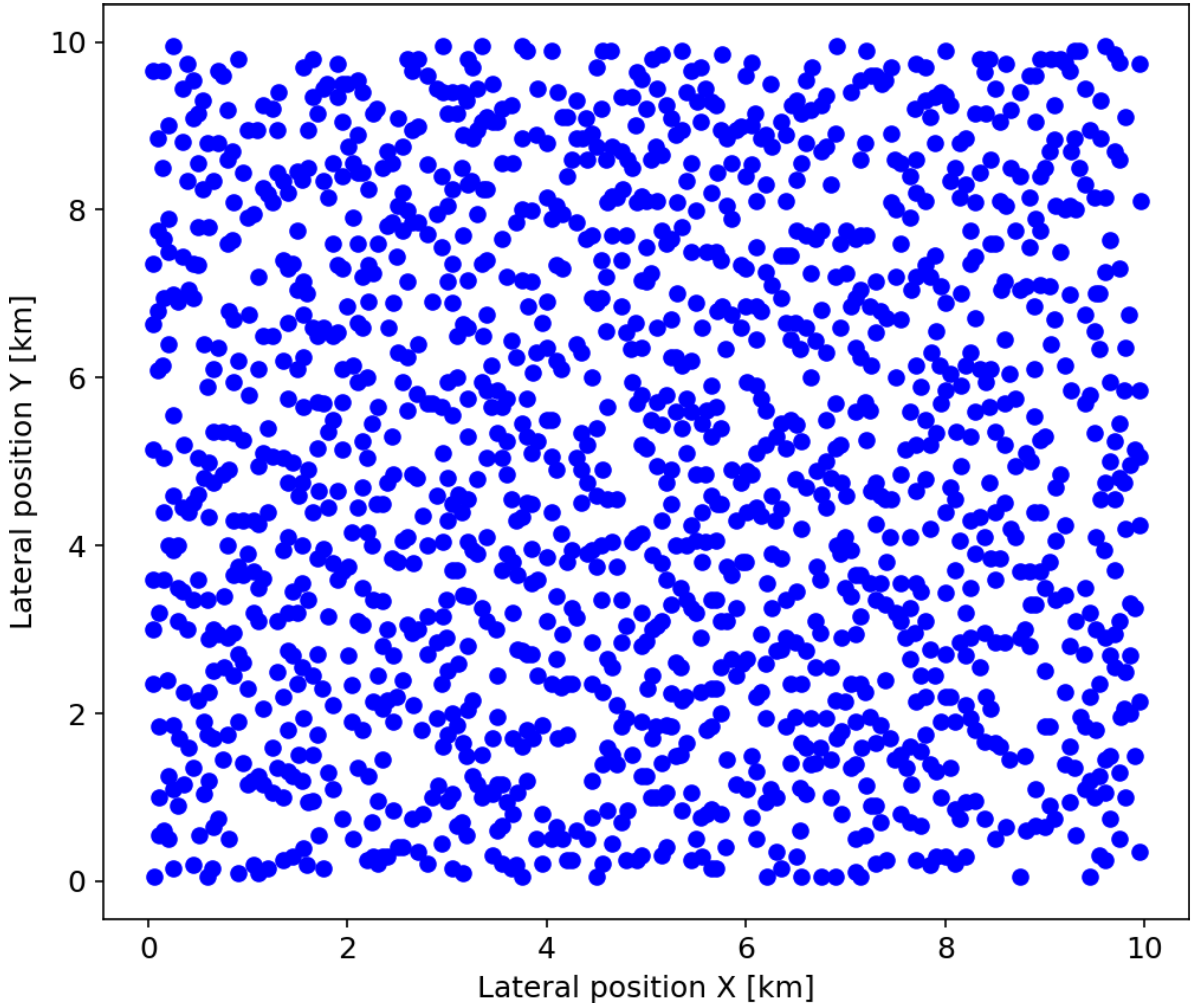
3D TTI RTM on Azure

Synthetic model based on SEG Overthrust + Salt models:

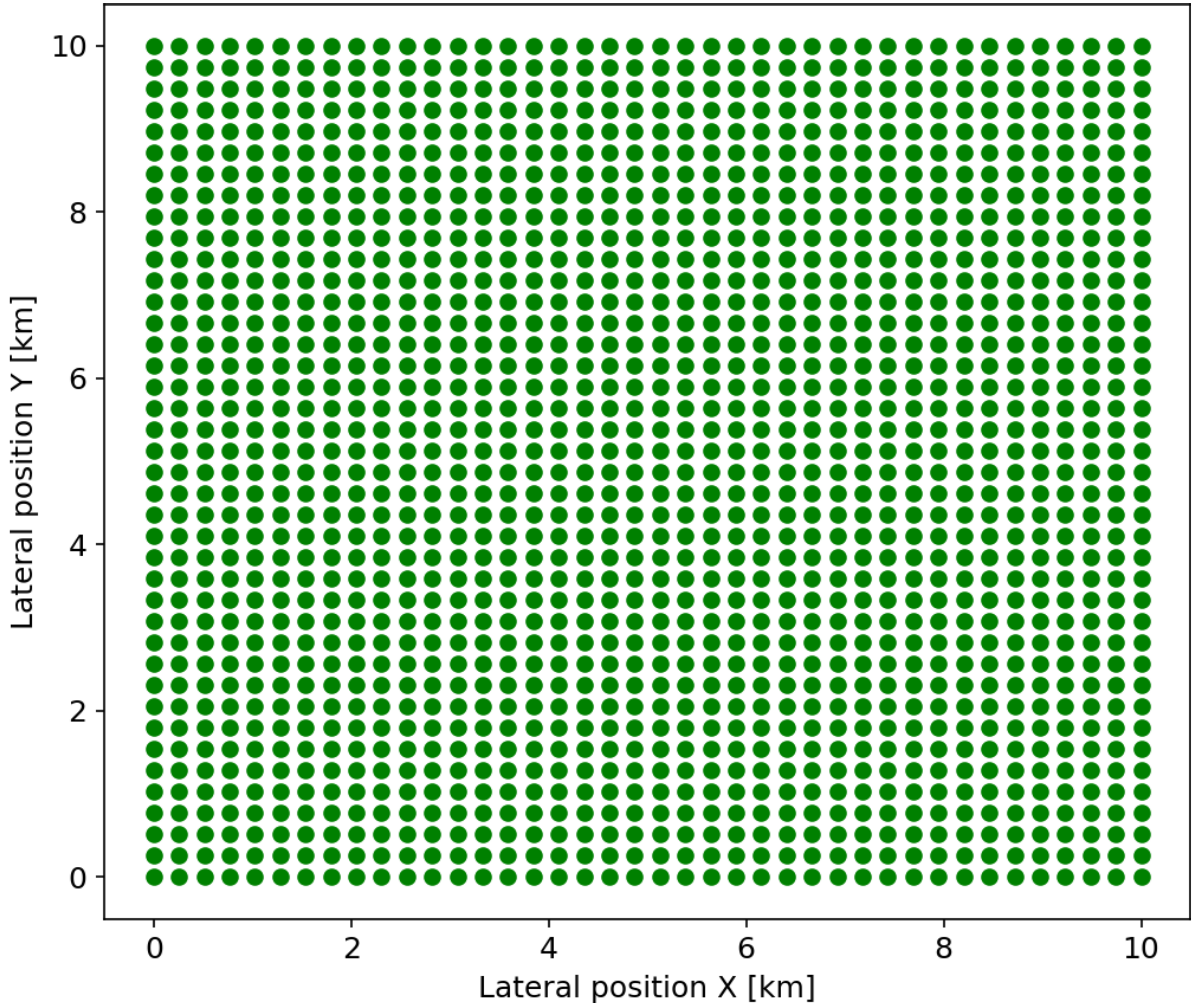
- Domain: 10 x 10 x 3.325 km
- Grid: 881 x 881 x 347 (12.5 m grid + ABCs)
- **Wide-azimuth acquisition w/ 1,500 randomly distributed OBNs**
- 799 x 799 dense source grid (12.5 m)
- Anisotropic TTI models + density
- Used source-receiver reciprocity

3D TTI RTM on Azure

Acquisition geometry:



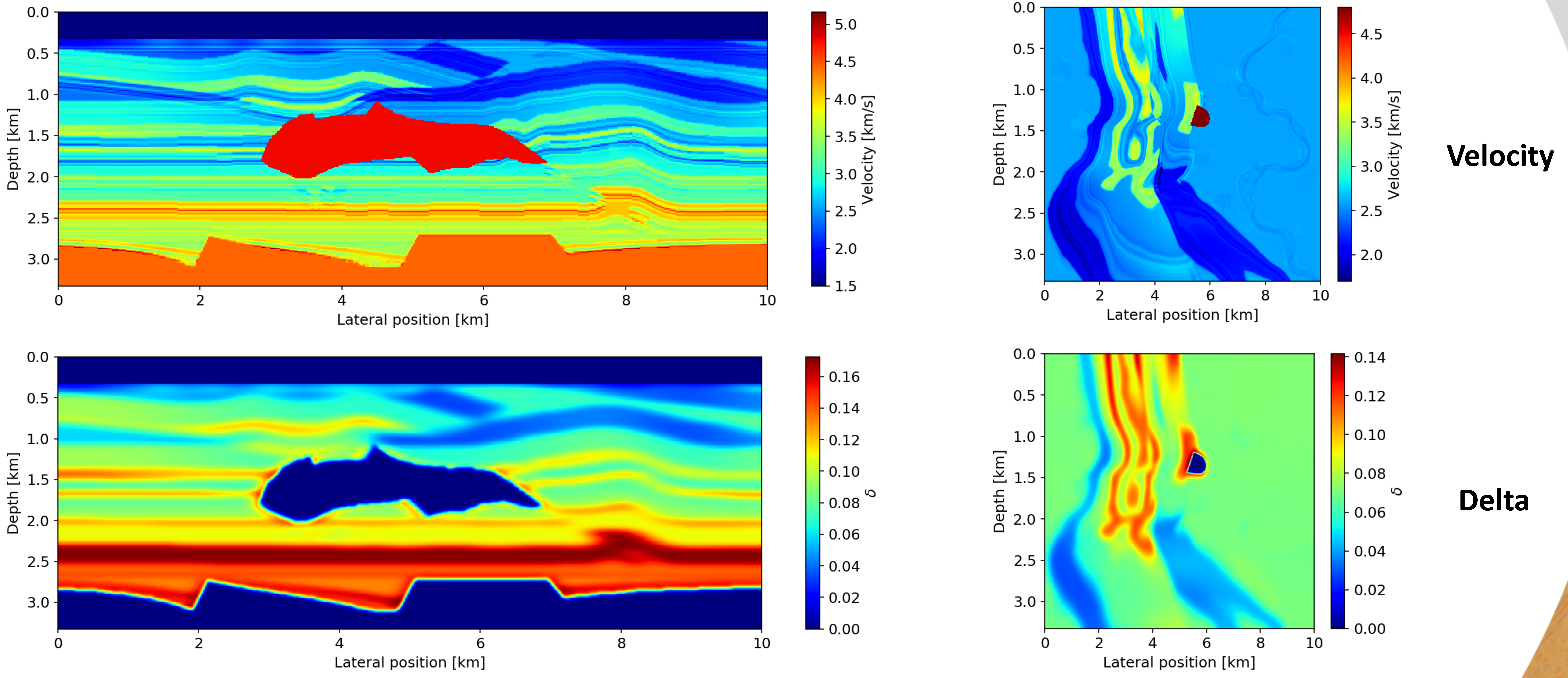
OBN receiver grid
50 X 50 m



Source vessel grid
12.5 X 12.5 m

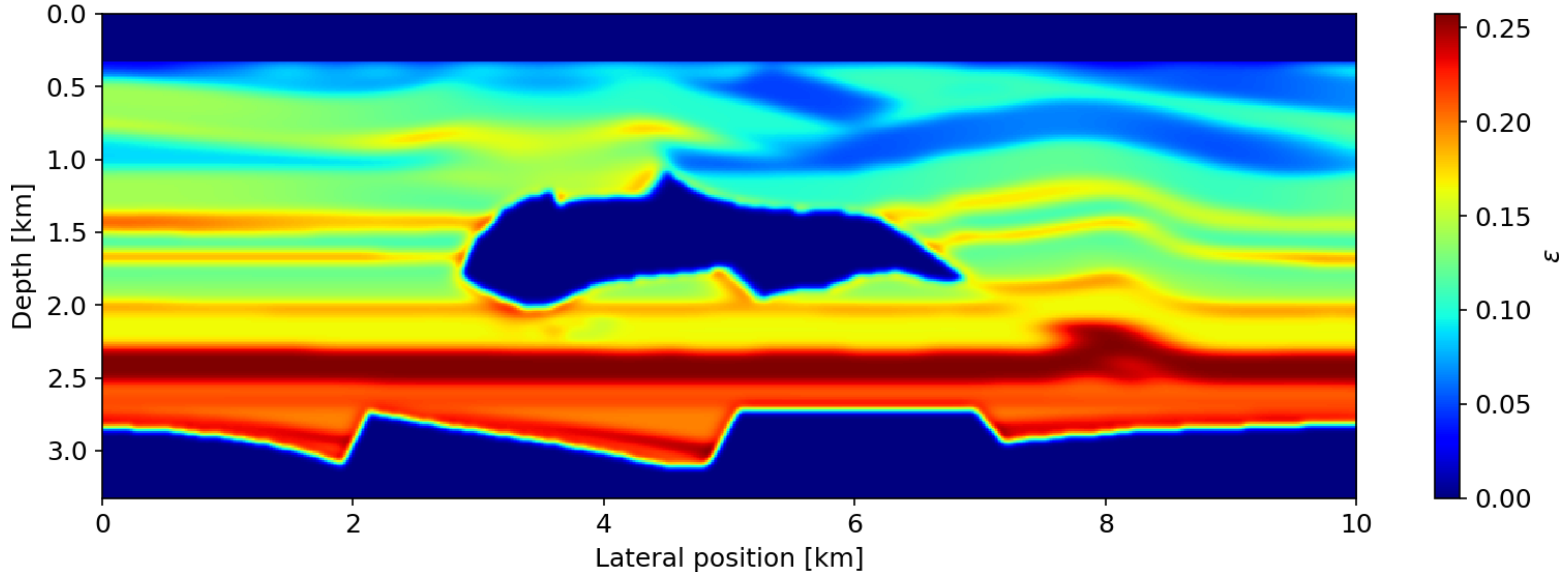
3D TTI RTM on Azure

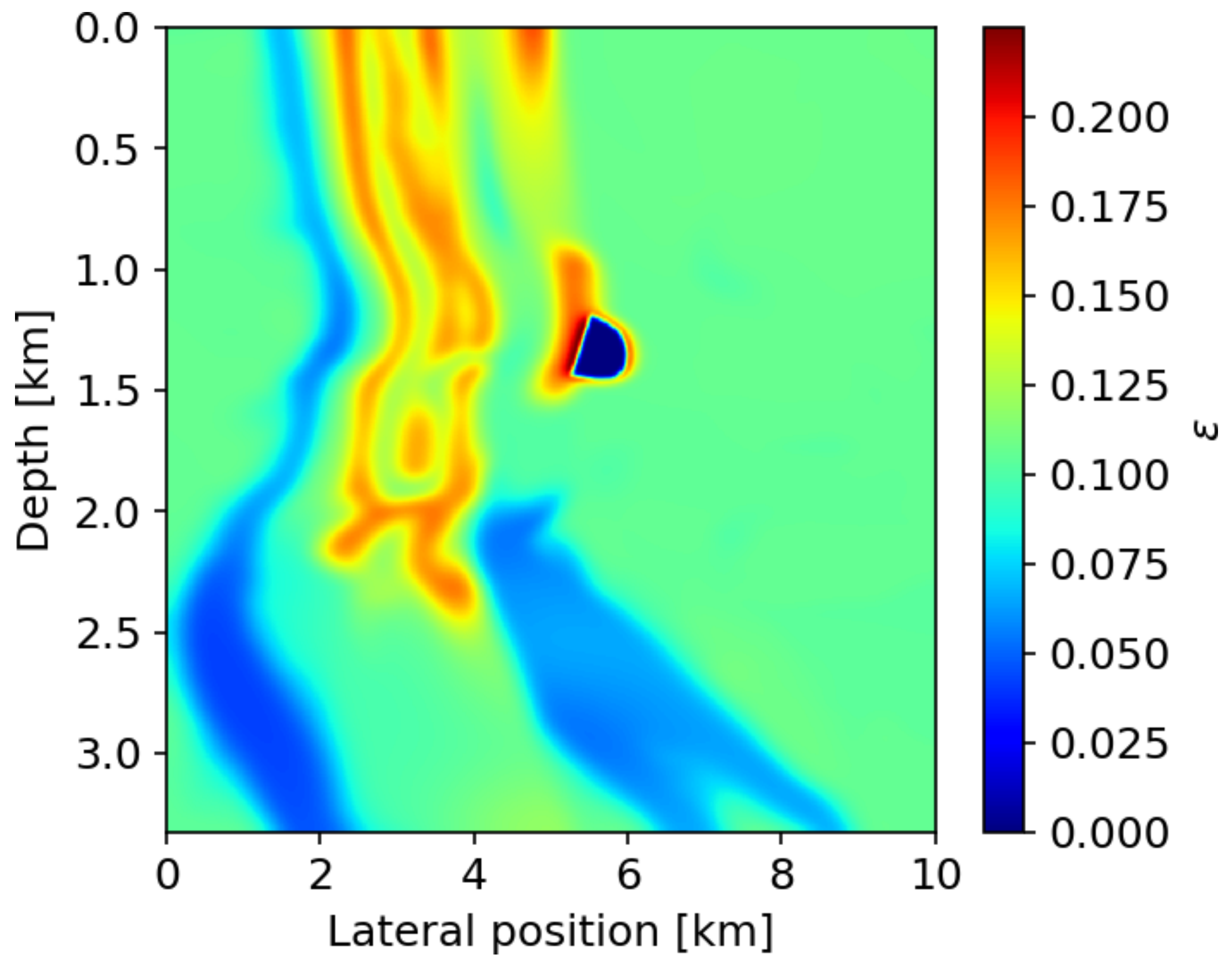
3D Overthrust + Salt model:



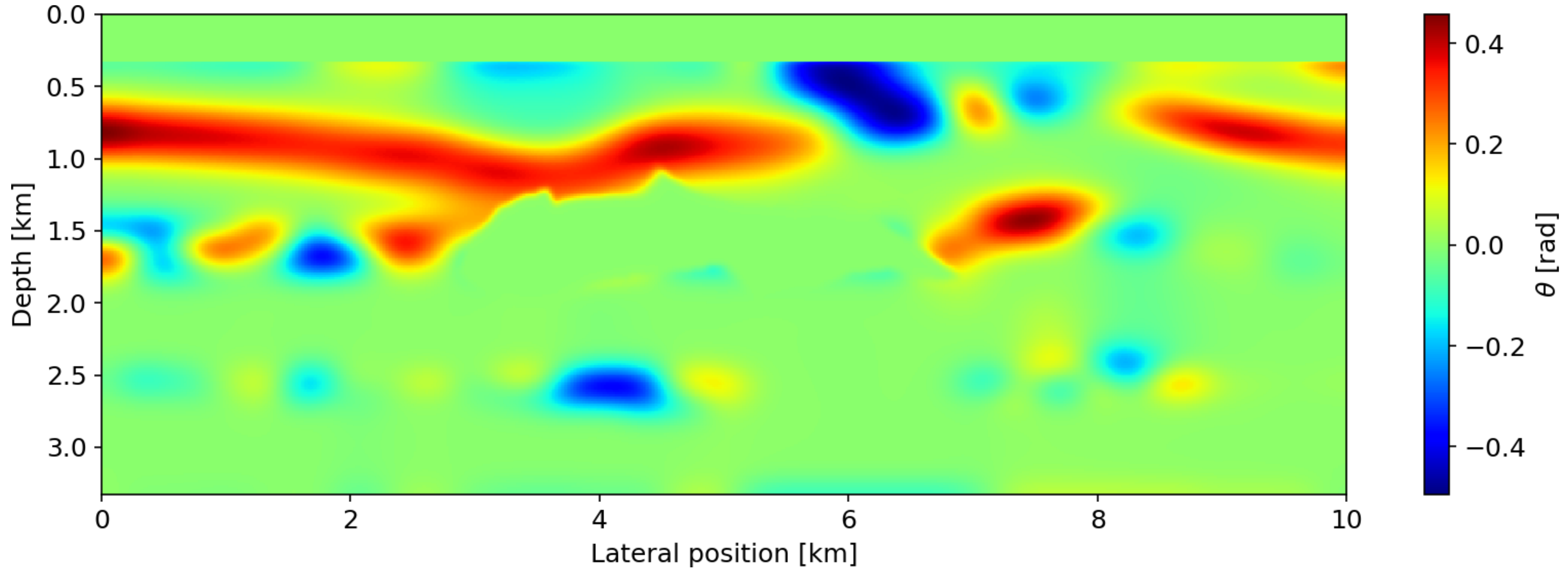
3D TTI RTM on Azure

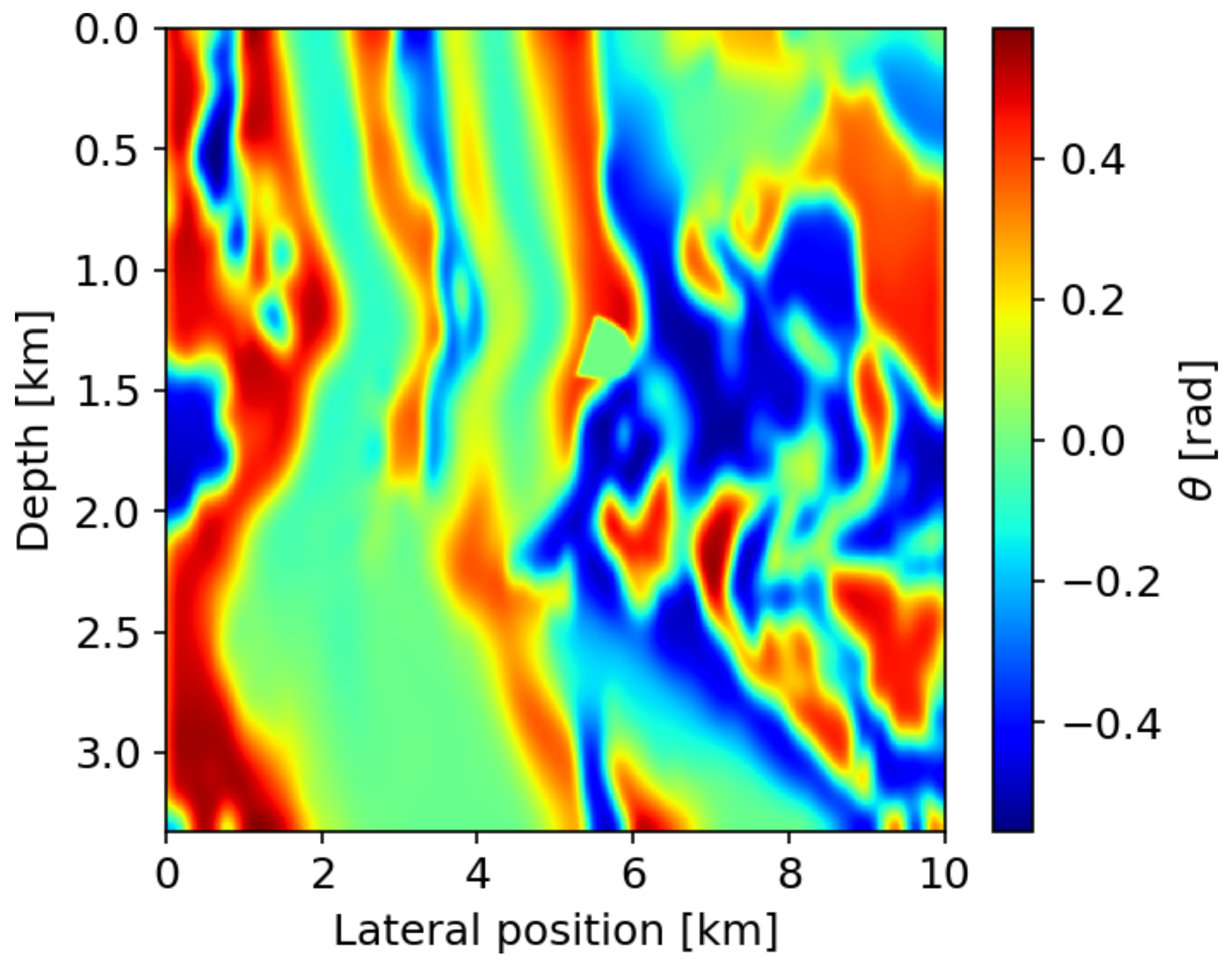
3D Overthrust + Salt model:





Epsilon

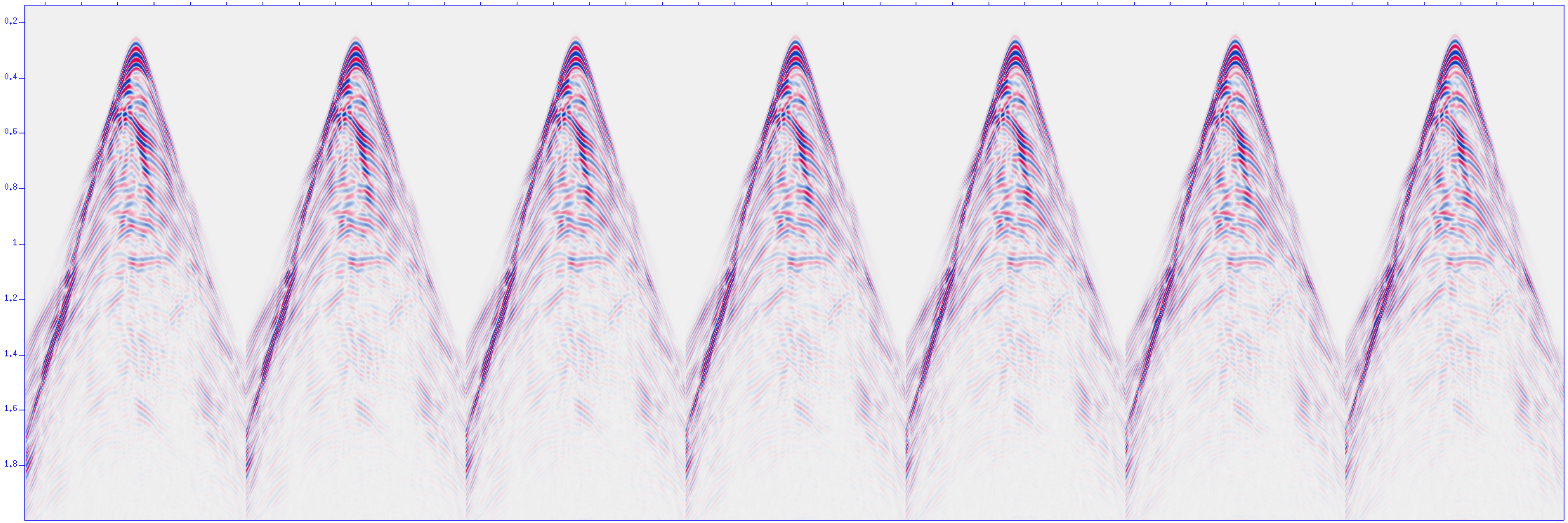




Azimuth

3D TTI RTM on Azure

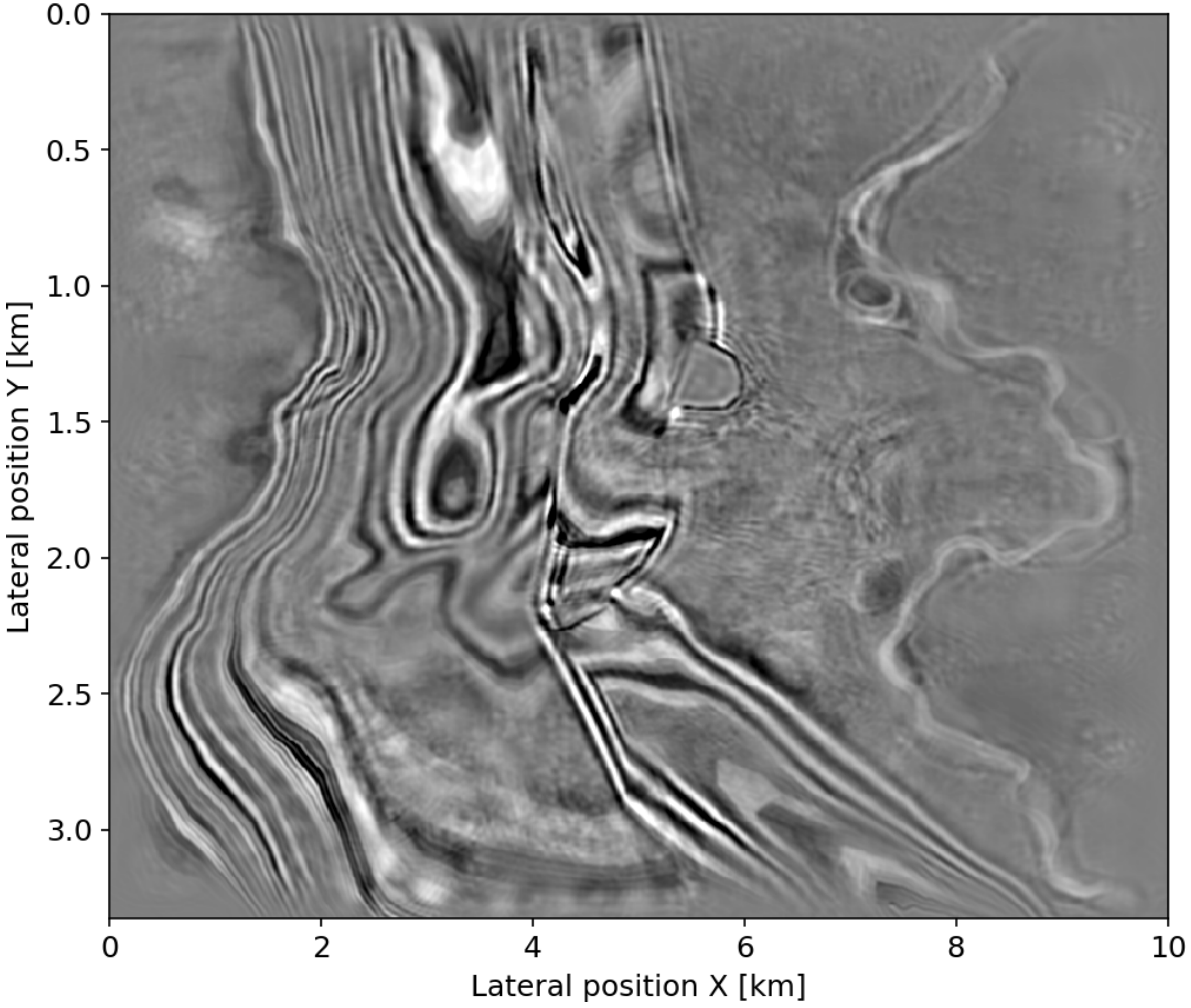
Observed data: 1,500 shots



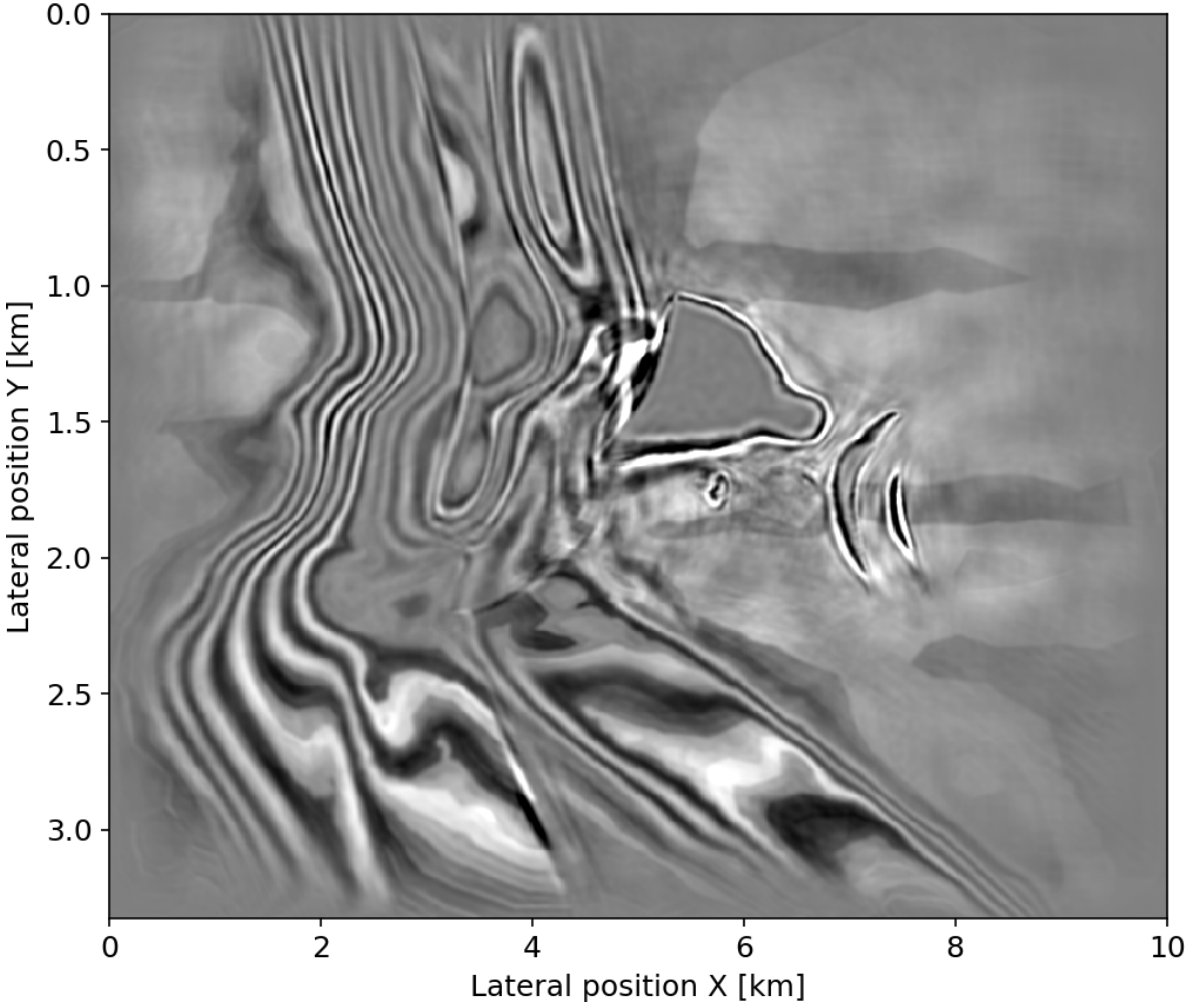
Shot records in xline

3D TTI RTM on Azure

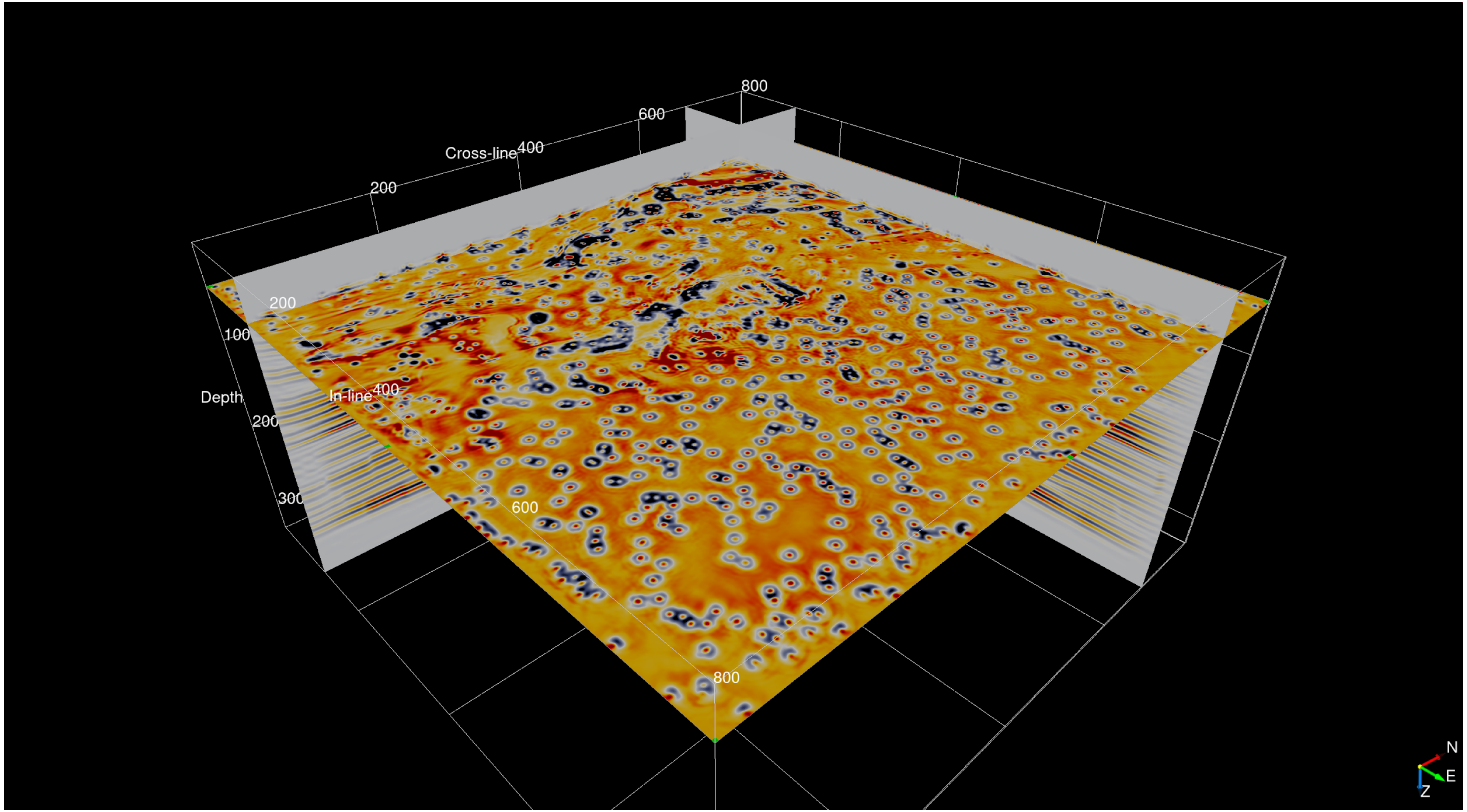
Depth slice 725 m



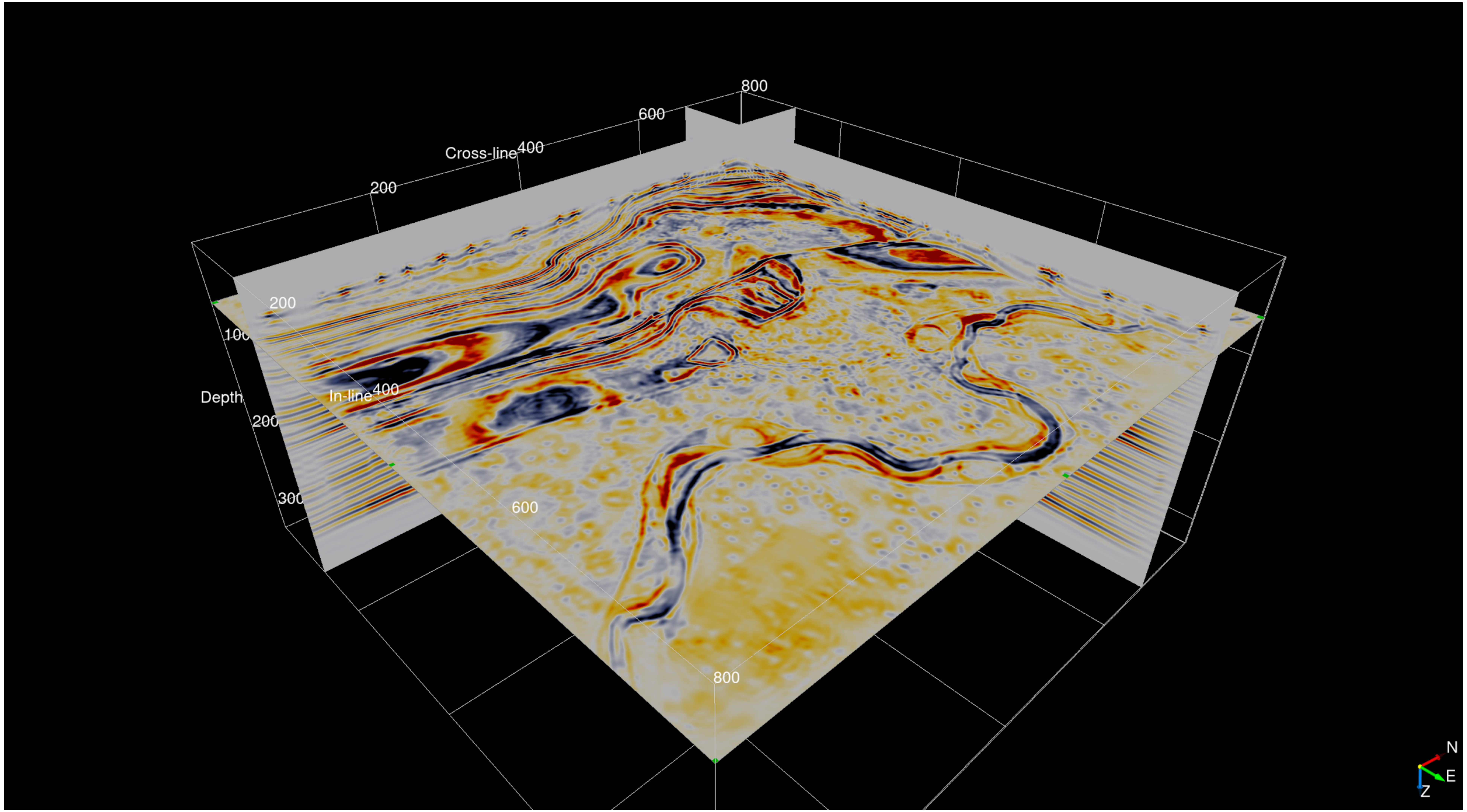
Depth slice 1250 m



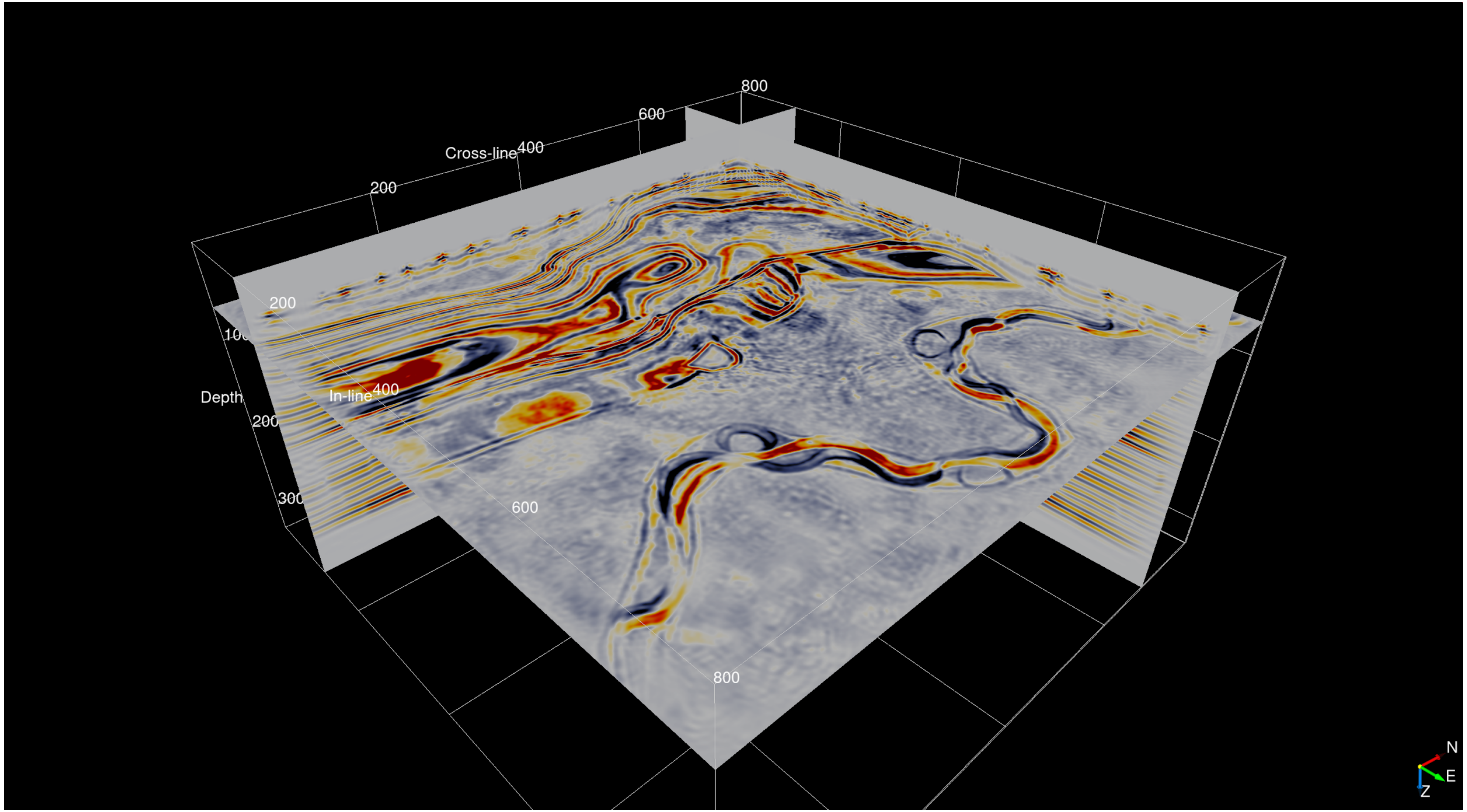
3D TTI RTM on Azure



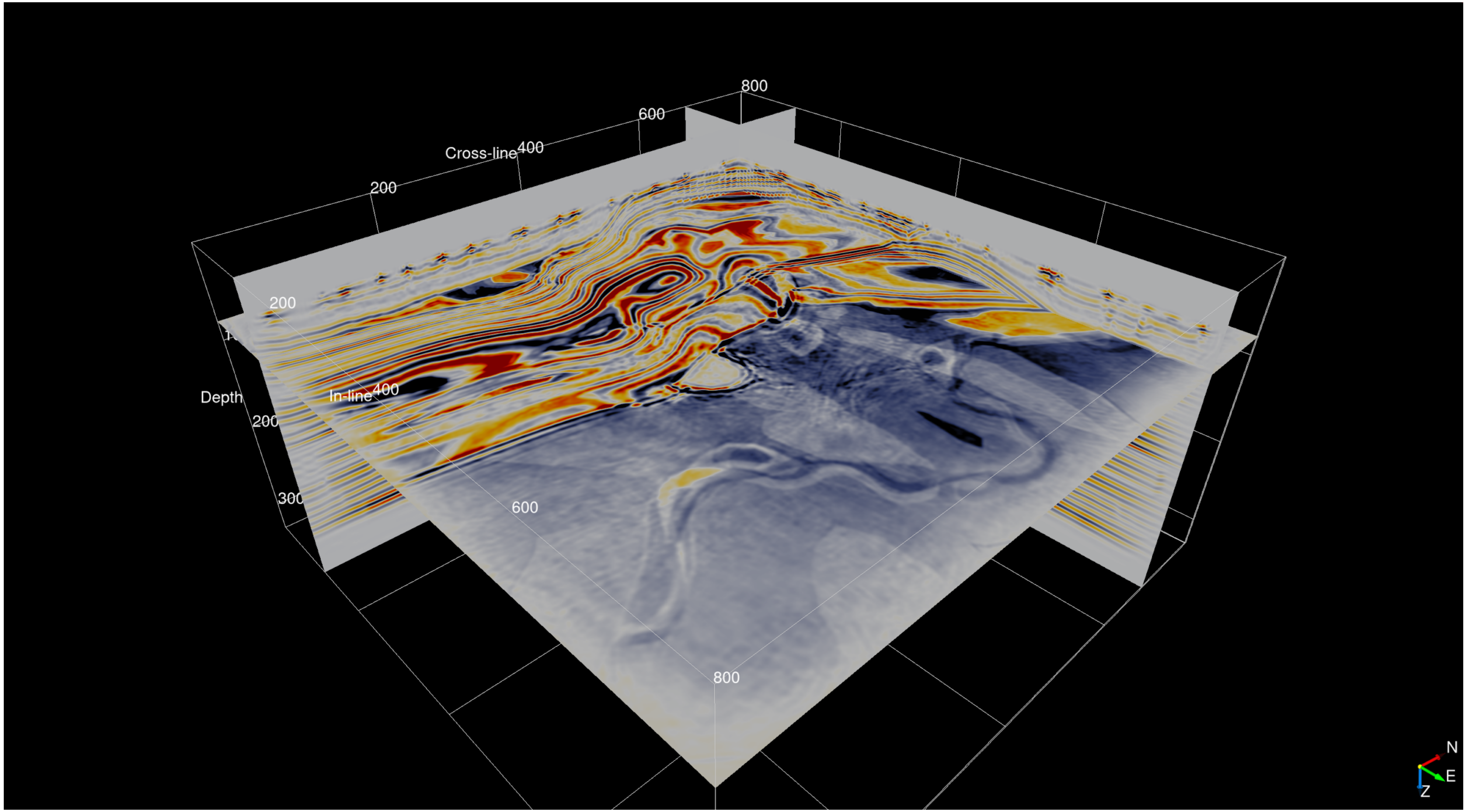
3D TTI RTM on Azure



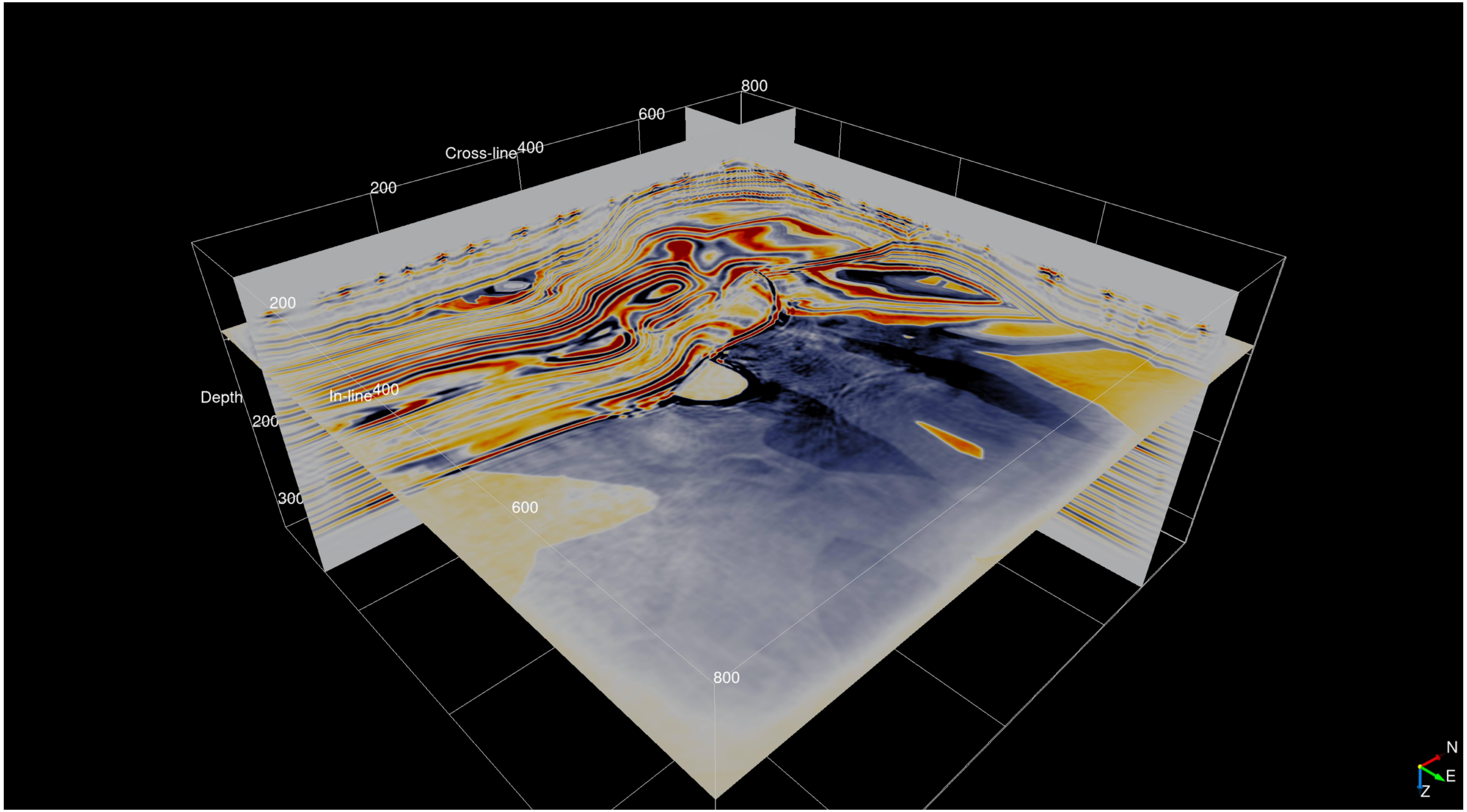
3D TTI RTM on Azure



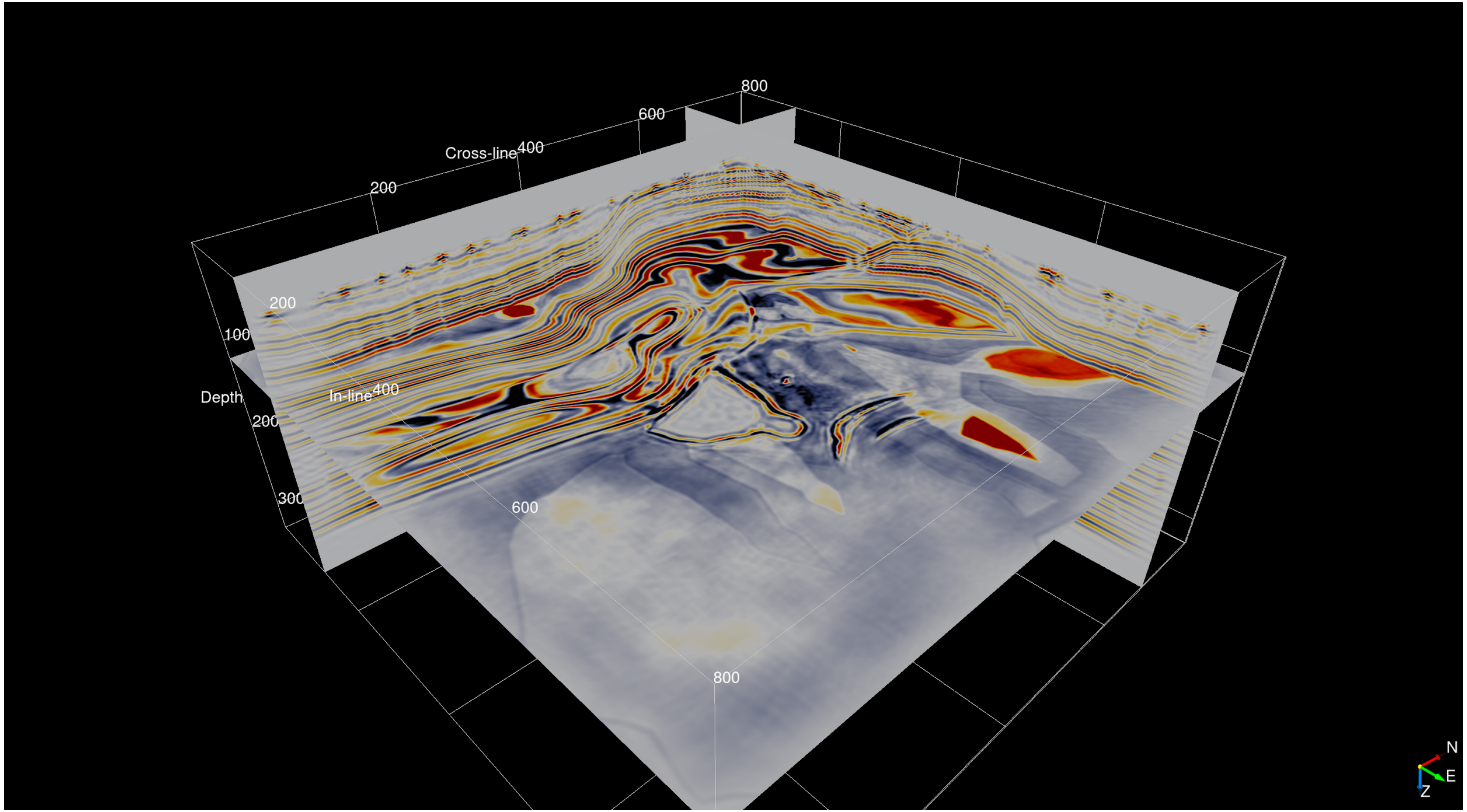
3D TTI RTM on Azure



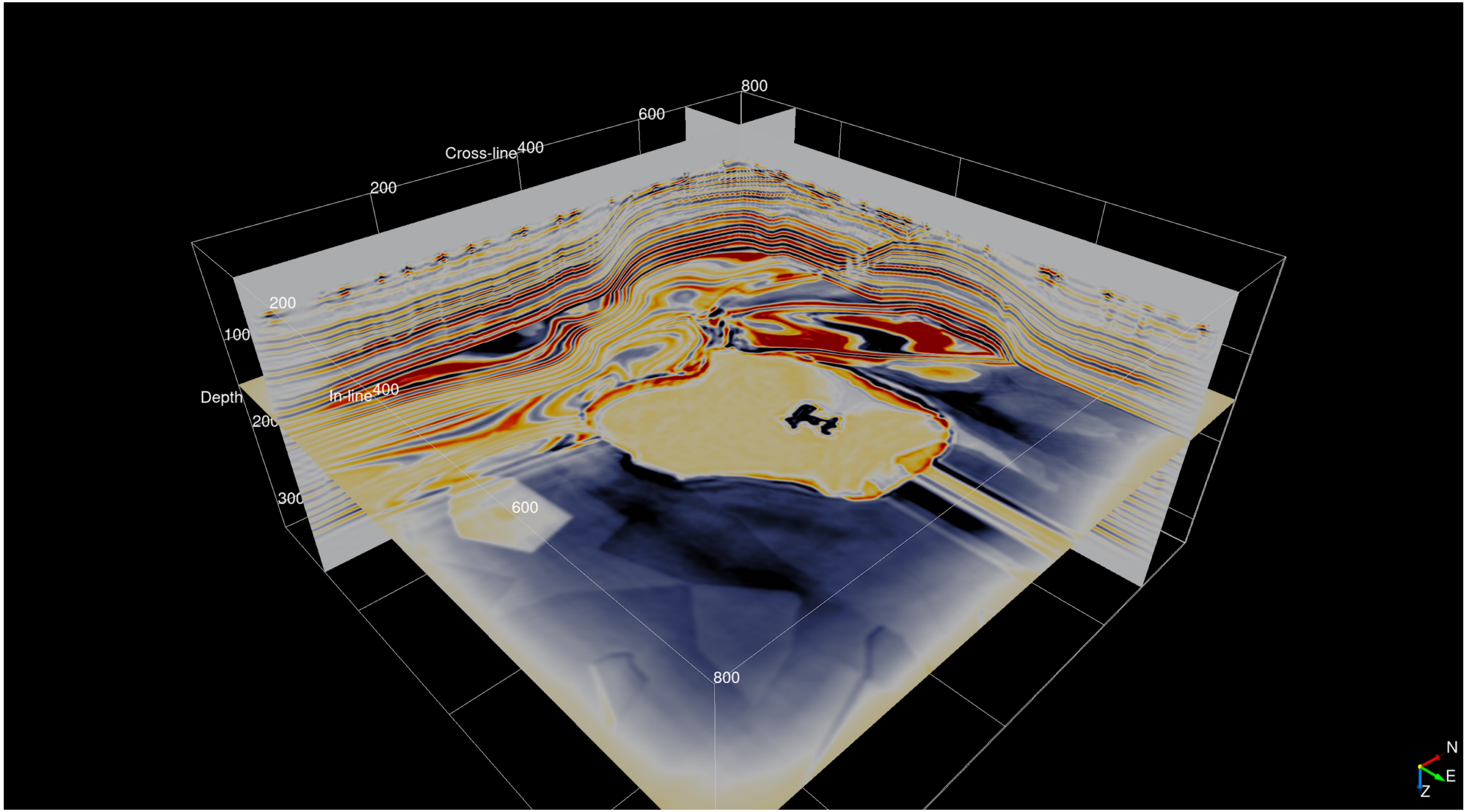
3D TTI RTM on Azure



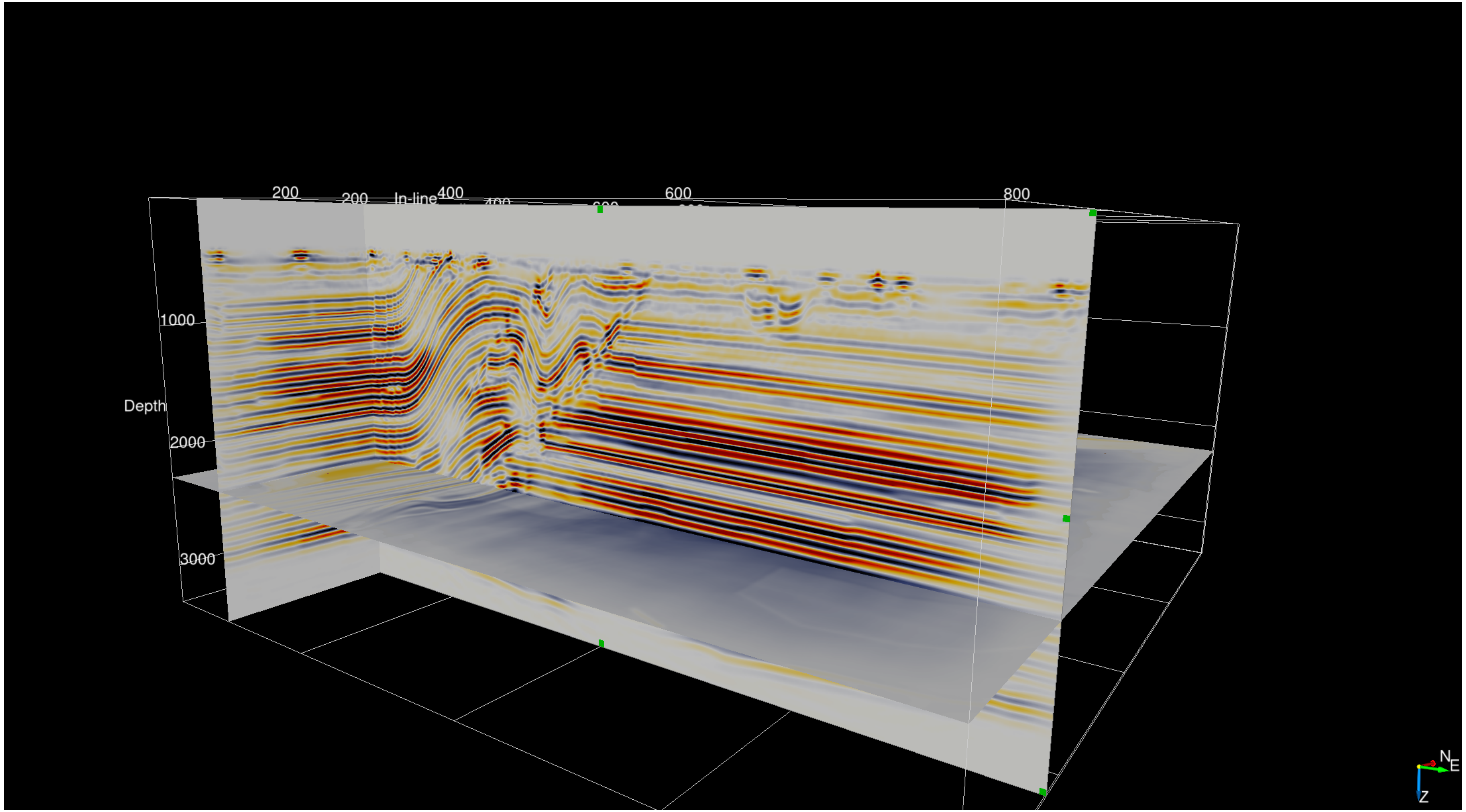
3D TTI RTM on Azure



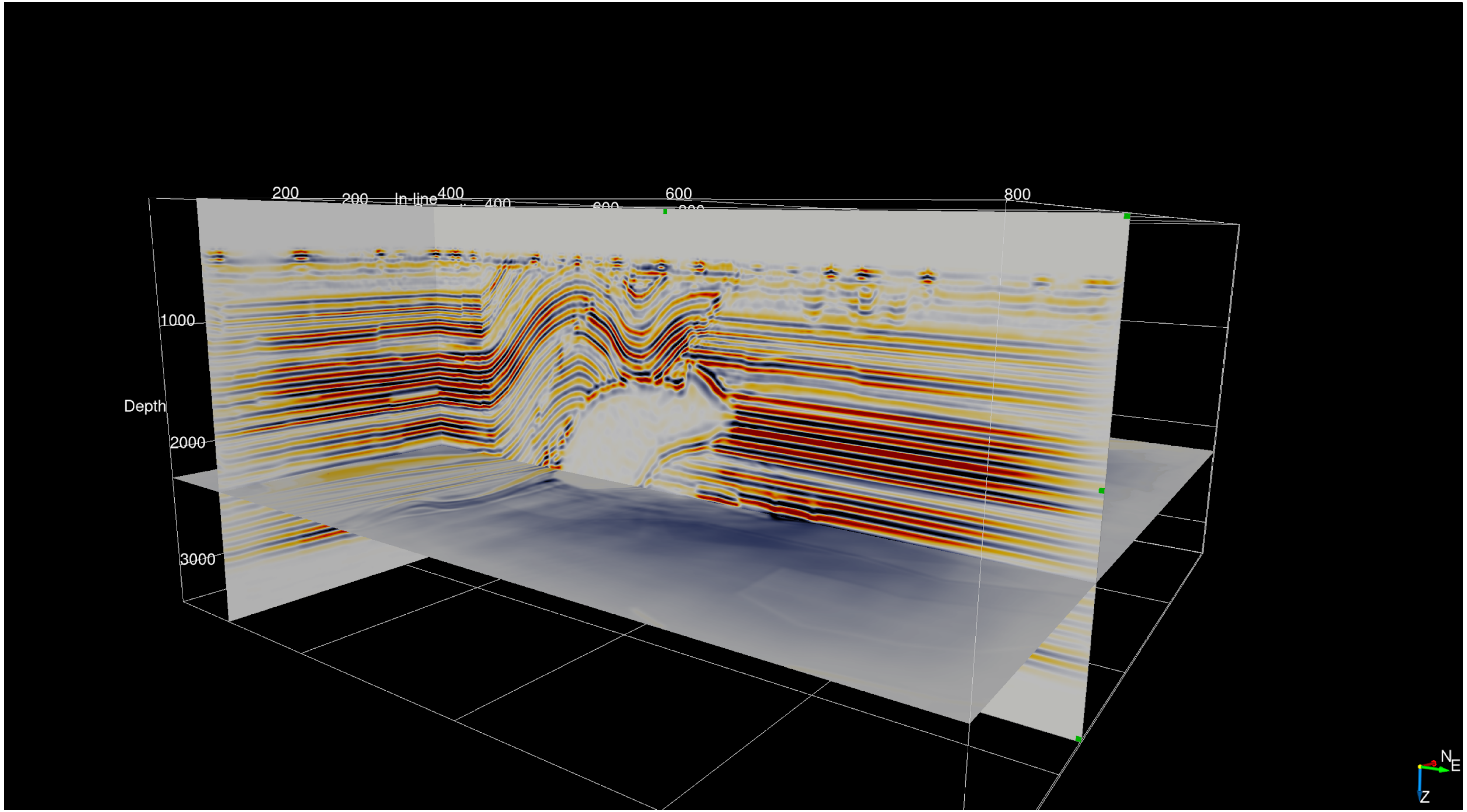
3D TTI RTM on Azure



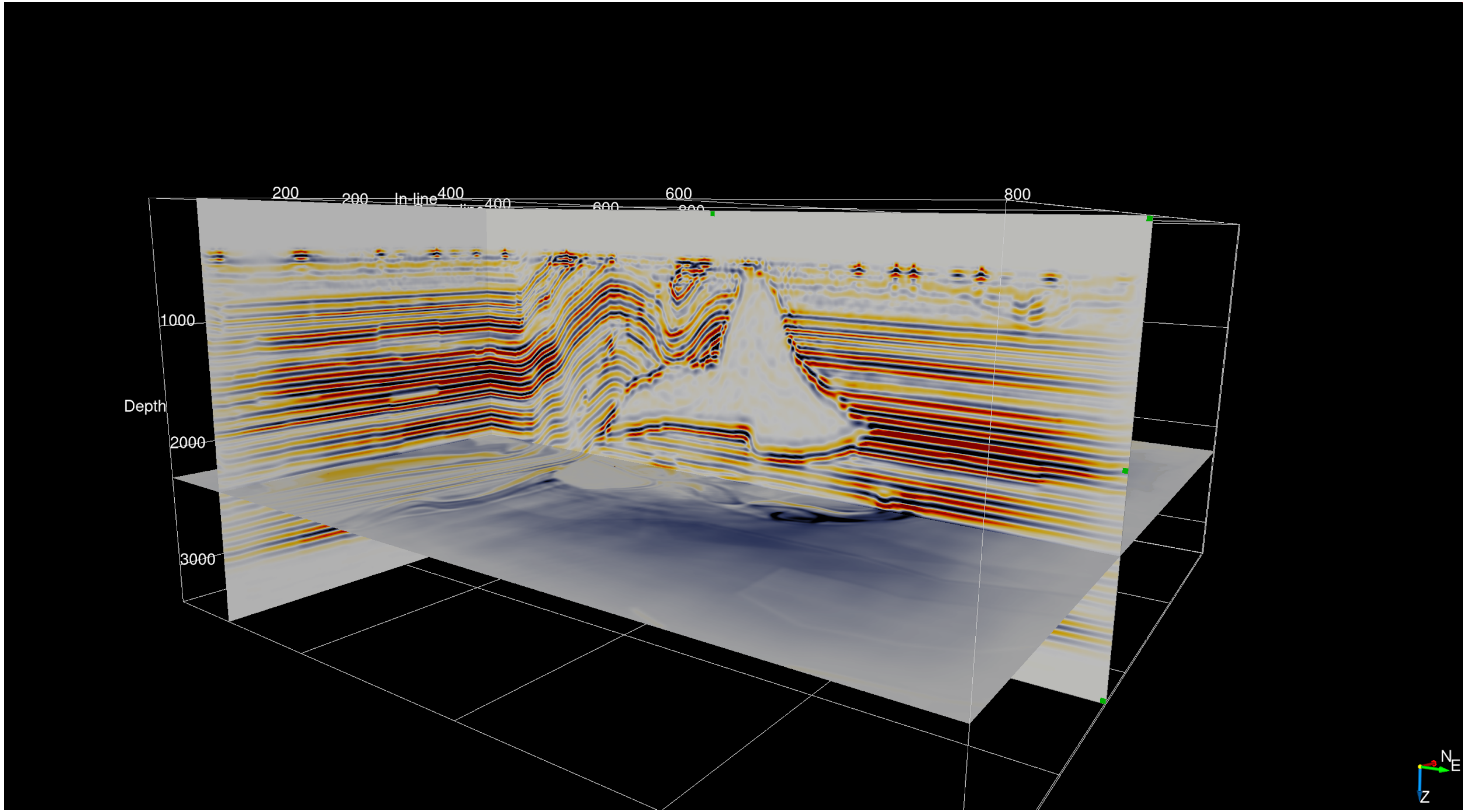
3D TTI RTM on Azure



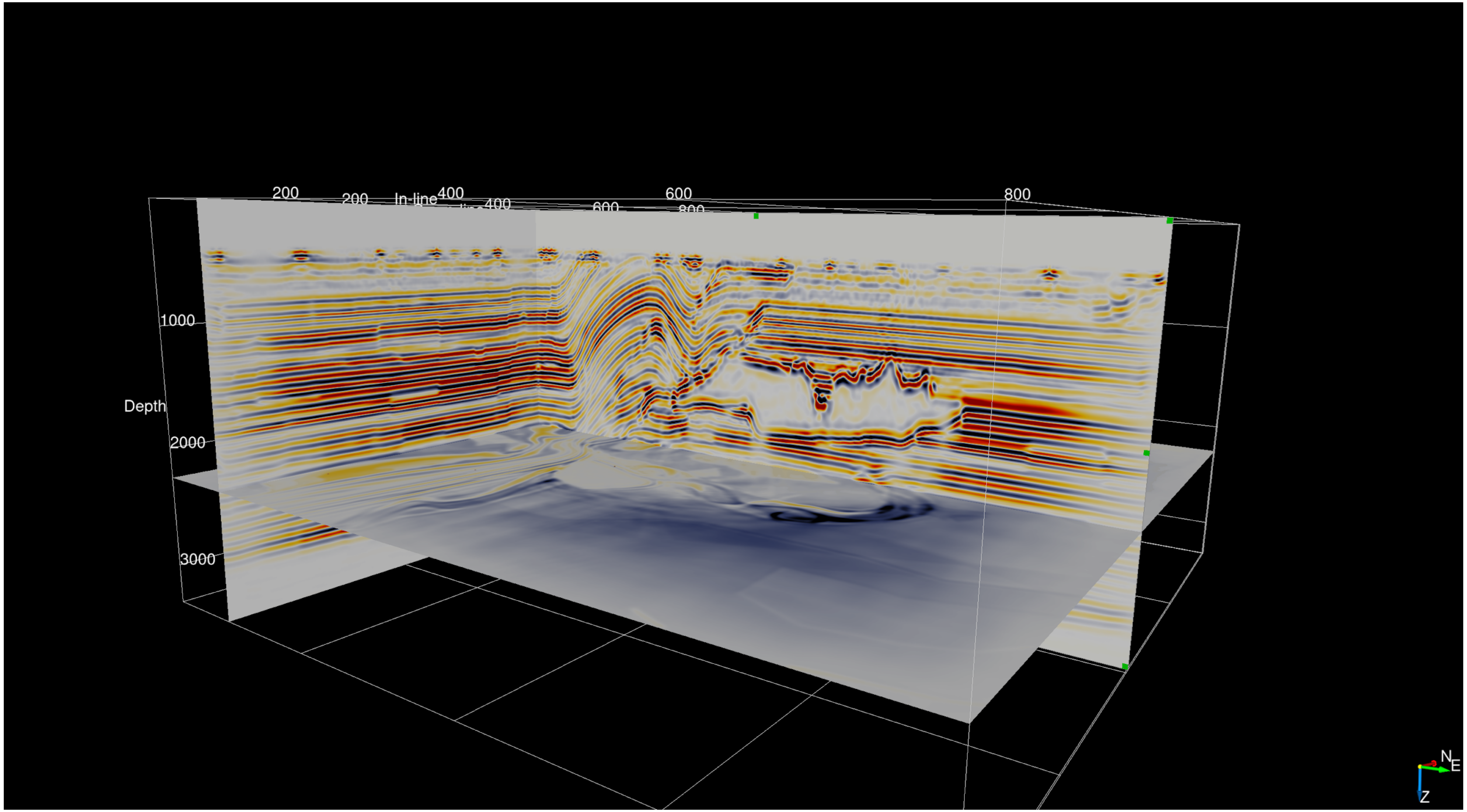
3D TTI RTM on Azure



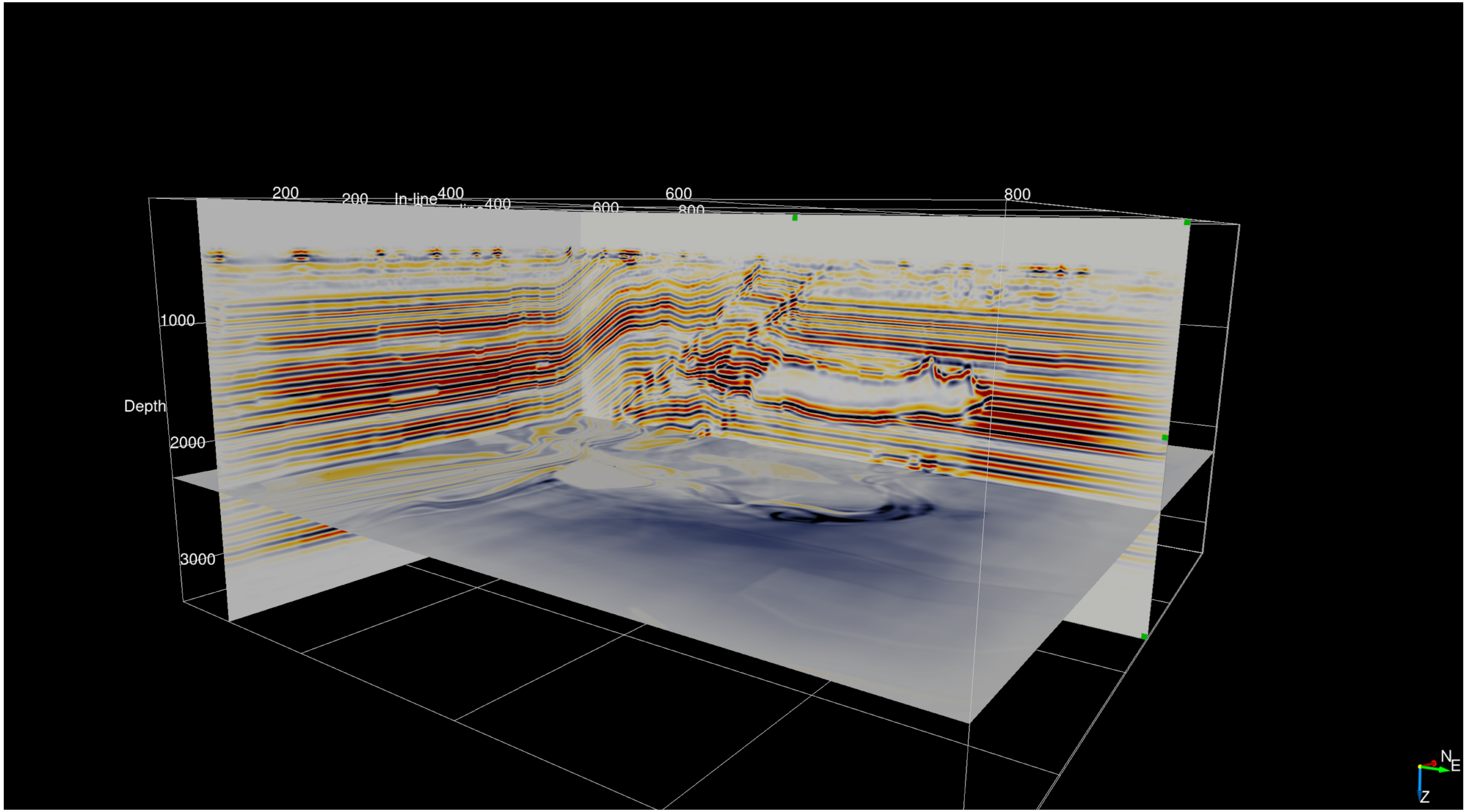
3D TTI RTM on Azure



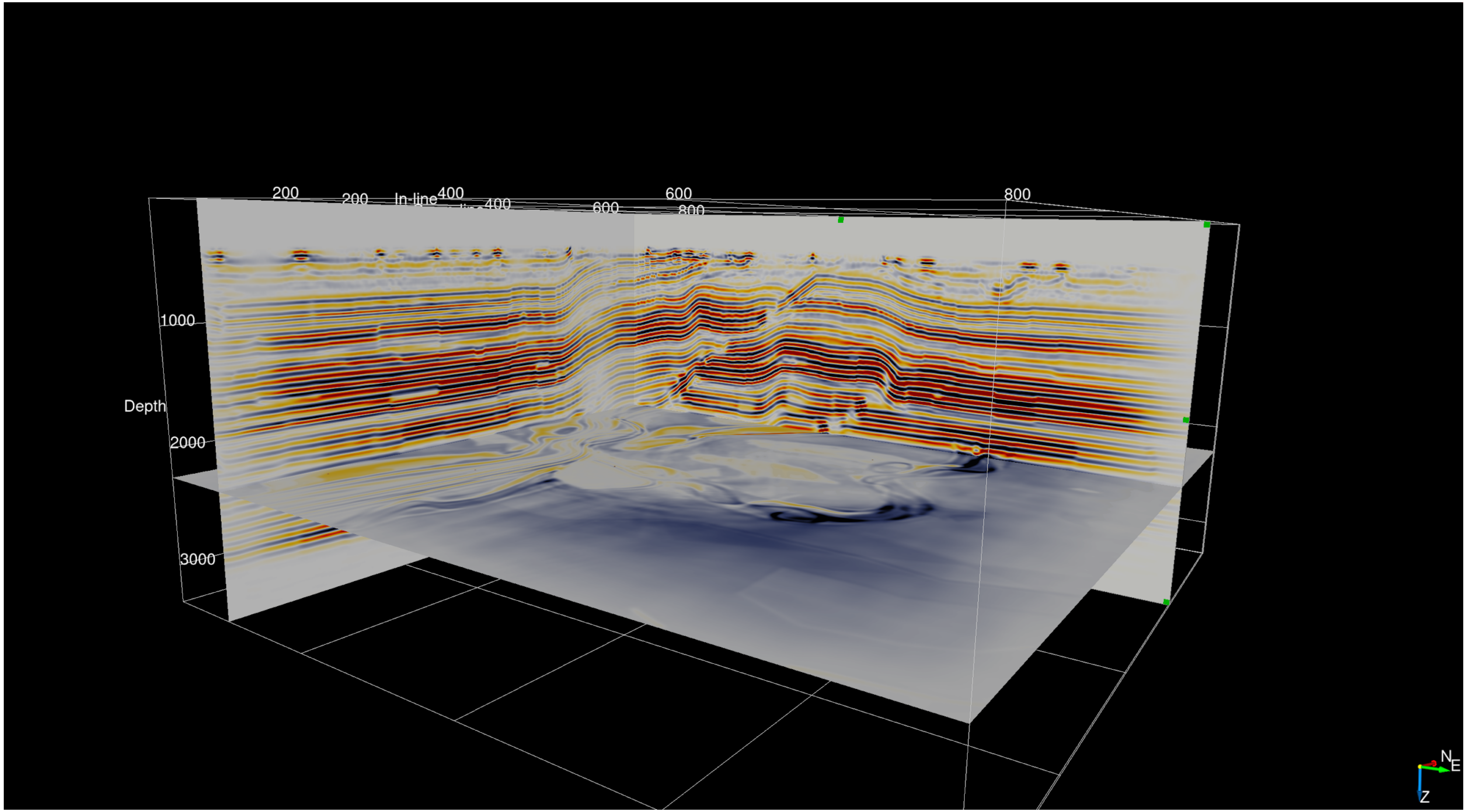
3D TTI RTM on Azure



3D TTI RTM on Azure



3D TTI RTM on Azure



3D TTI RTM on Azure

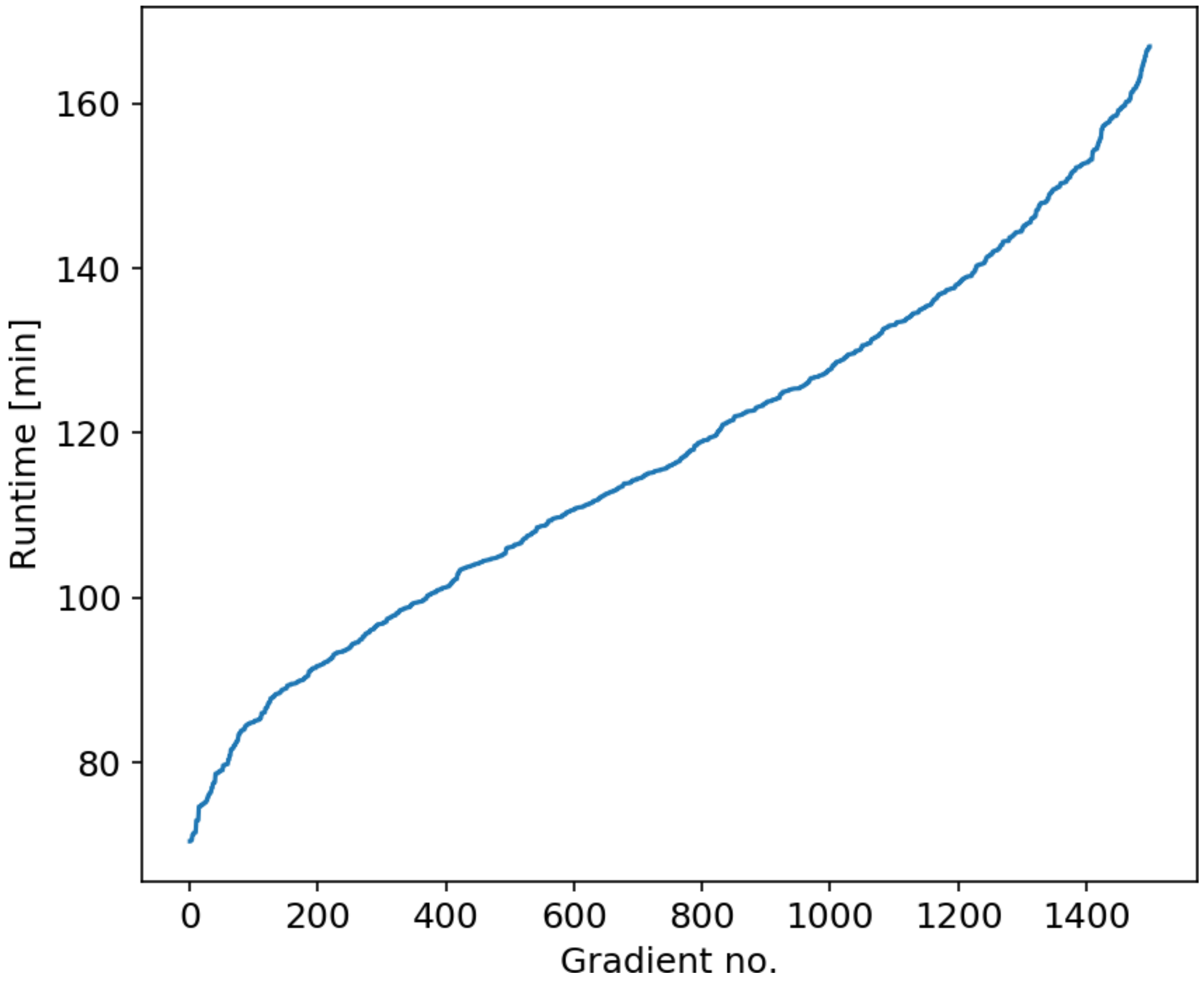
Azure setup:

- E64 and ES64 VMs
- 2.3 GHz Intel Xeon ® E5-2673 v4 (Broadwell)
- 432 GB RAM, 64 vCPUs per VM
- 100 VMs → 6,400 vCPUs
- 2 VMs per gradient (1 MPI rank per socket)
- 600 GB per wavefield
- Peak performance: 140 GFLOPS per VM (14 TFLOPS total)
- **Total cost for RTM: ≈17,000$ (dedicated/on-demand)**

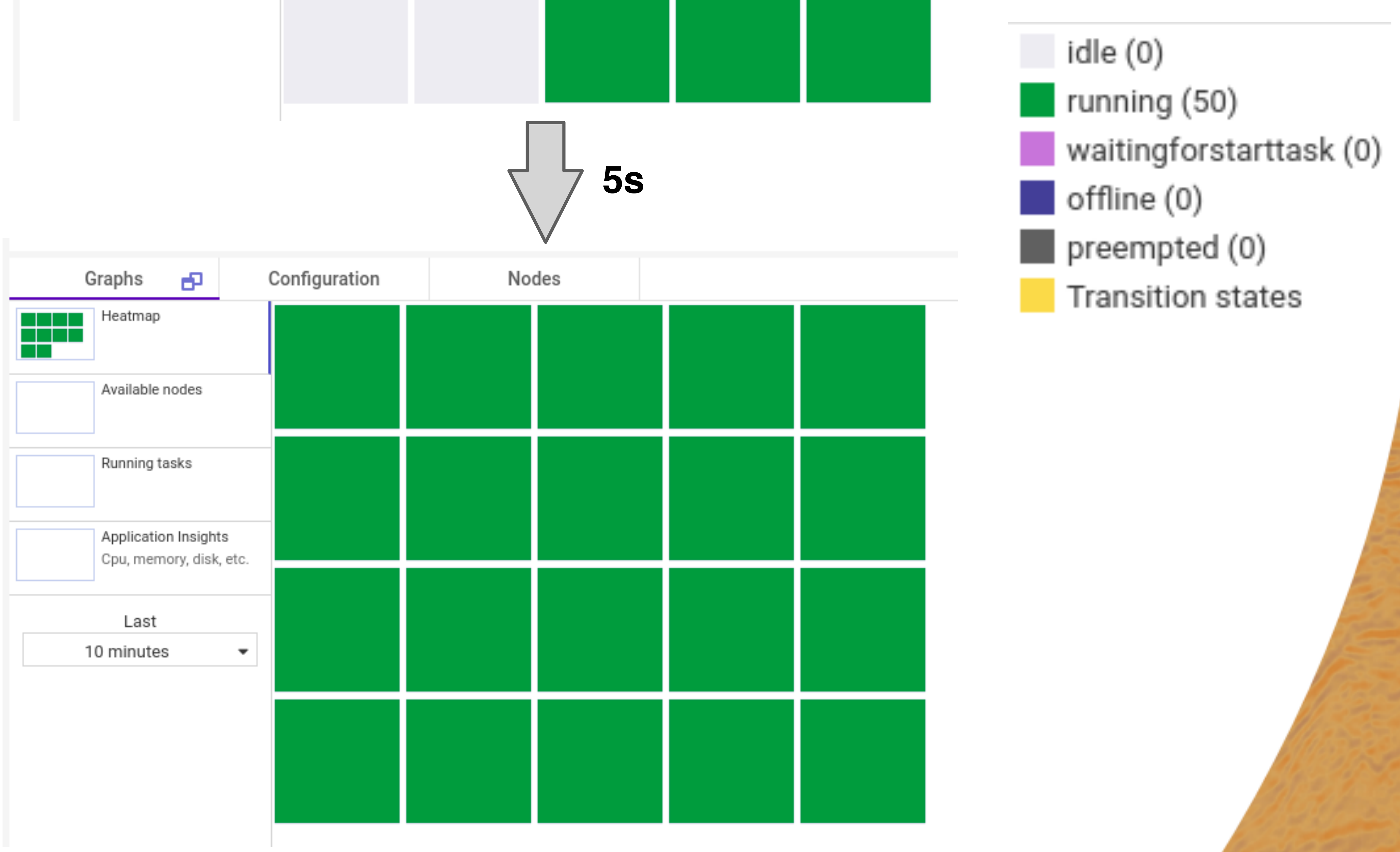
3D TTI RTM on Azure

Timings:

- 100 nodes
- 2 nodes per gradient
- 1500 source positions
- Average runtime: 110 minutes per gradient
- **Average cost per gradient: 11$ (dedicated)**
- **Peak performance: 140 GFLOPS per VM (14 TFLOPS total)**



3D TTI RTM on Azure



Azure Batch:

- Jobs start as VMs are added to pool
- Do not need to wait for full pool
- No long idle times
- At least **6X cost reduction** when using low priority…

Future directions

Go large:

- Ongoing collaboration Azure to run at industry-scale
- Iterative LS-RTM on large-scale 3D TTI
- SEAM model: long offset data acquisition w/ 3D elastic modeling

Check for updates on our website:

https://slim.gatech.edu/

Elastic

3D SEAM elastic, full offset, full azymuth

- **35km x 40km x 15 km**
- 20m x 20m x 10m grid
- 12th order FD
- Velocity-stress formulation (9 coupled PDEs)
- Full offset & full azimuth
- **16s recording**
- Modeling only (elastic imaging requires more than just compute)

.5Tb RAM
5.3 Gpoints
2.8TFlop/time-step

10 lines in Devito

```
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))))
```

Setup

- Out of the box Devito
- Source at the centre (17.5km, 20km)
- 16s recording of Vx, Vy, Vz at ocean bottom
- 32 compute nodes (small node on azure, no InfiniBand)
- **3s per time-step**
- 7 TFlop/s

Crossline at x = 5km
Crossline at x = 17.5km
Crossline at x = 22.5km
OBN position along y (km)
Inline at y = 10km
Inline at y = 20km
Inline at y = 30km
OBN position along x (km)
0
2
4
6
8
10
12
14
16
5
15
20
25
30
35

Observations

Built a scalable reproducible imaging solution in the Cloud in 1y timeframe

- leveraging abstractions, open source, and collaboration
- using serverless Cloud native tools
- levels the play field

Proved that focussed

- industry-supported Consortia & public-private partnerships deliver
- needs to be sustainably funded

Contrast w/ industry-wide initiatives

- integration of different systems often fail
- suffer from scope creep

Observations

Created a industry-strength low-cost tensorflow/Pytorch-like environment

- makes research findings directly available & reproducible
- changes how we spend our research budgets & interact w/ Consortia & Startups
- that drives innovations more rapidly by giving everybody a chance

Conclusions

Seismic imaging in the Cloud:

- feasible in Cloud but requires rethinking algorithms & implementations
- take advantage of high-throughput batch computing, serverless/event-driven computations, object storage, spot instances
- access to hardware w/o compromise w/ potential of hyperscaling
- only pay what you use: up to **10x** cost reductions
- **software based on separation of concerns + abstractions is prerequisite to go serverless**

SLIM

Acknowledgments

This project was made possible through the help of:

- Microsoft Azure
- Sverre Brandsberg-Dahl
- Evan Burness
- Kadri Umay
- Alexander Morris
- Steve Roach
- Hussein Shel
- Georgia Research Alliance & Georgia Institute of Technology

Georgia Tech

UBC THE UNIVERSITY OF BRITISH COLUMBIA

osokey