Serverless seismic imaging in the cloud

Philipp A. Witte[†], Mathias Louboutin[†], Charles Jones[‡]
and Felix J. Herrmann[†]

SLIM

† Georgia Tech

‡ osokey

Acknowledgments

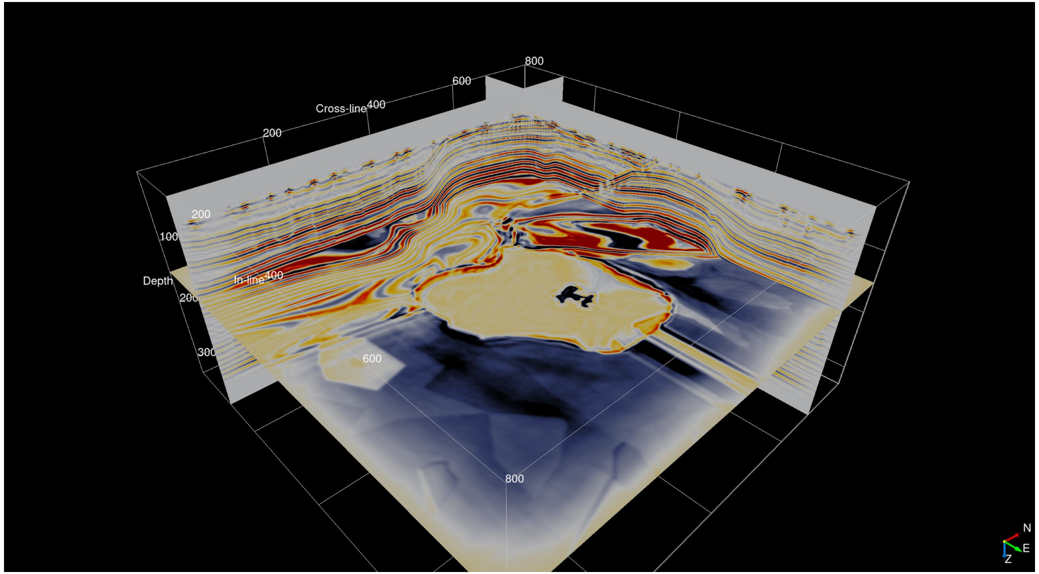
Joint project with Microsoft Azure and Osokey*:

- Sverre Brandsberg-Dahl
- Kadri Umay
- Hussein Shel
- Steven Roach
- Evan Burness
- Alexander Morris
- George Iordanescu
- Qie Zhang
- James Selvage*

Seismic imaging

Seismic imaging: linear least squares optimization problem:

$$\underset{\delta \mathbf{m}}{\text{minimize}} \quad \Phi(\delta \mathbf{m}) = \sum_{i=1}^{n_s} \frac{1}{2} ||\mathbf{J}(\mathbf{m}, \mathbf{q}_i)\delta \mathbf{m} - \mathbf{d}_i||_2^2$$



Deploy software to various cloud providers

amazon web services™

Azure

Seismic inversion on HPC clusters

Conventional compute environment: **HPC clusters**

✓ **Pros**

- Best achievable performance
- 40+ years of experience and existing software
- Low mean-time-between failures (MTBF)
- Very fast inter-node connections possible (Infiniband)

✗ **Cons**

- Very high upfront + maintenance costs
- Only available to few companies + academic institutions
- Compromises regarding hardware (architecture/CPUs/GPUs/RAM)

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
- Latest hardware and architectures available (GPUs, ARM)

✗ Cons

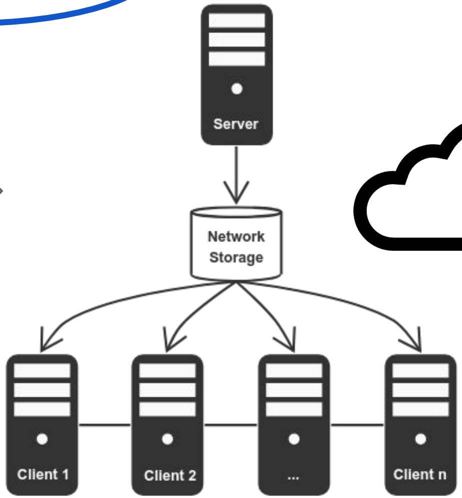
- Slower inter-node connections (depending on platform)
- Oftentimes larger MTBF
- High costs if not used properly
- Need to transition software
- Steep learning curve

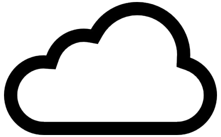
Moving to the cloud

```
#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 and shift



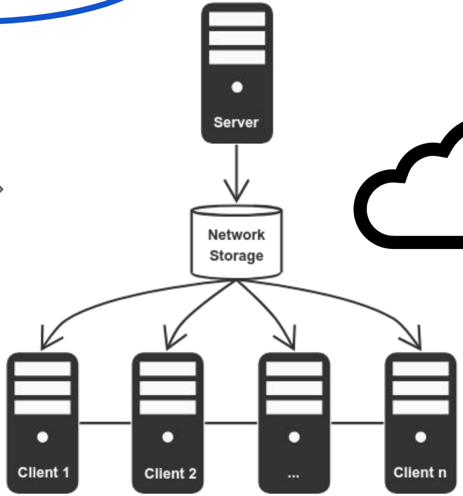


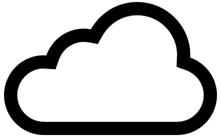
Moving to the cloud

```
#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 and shift

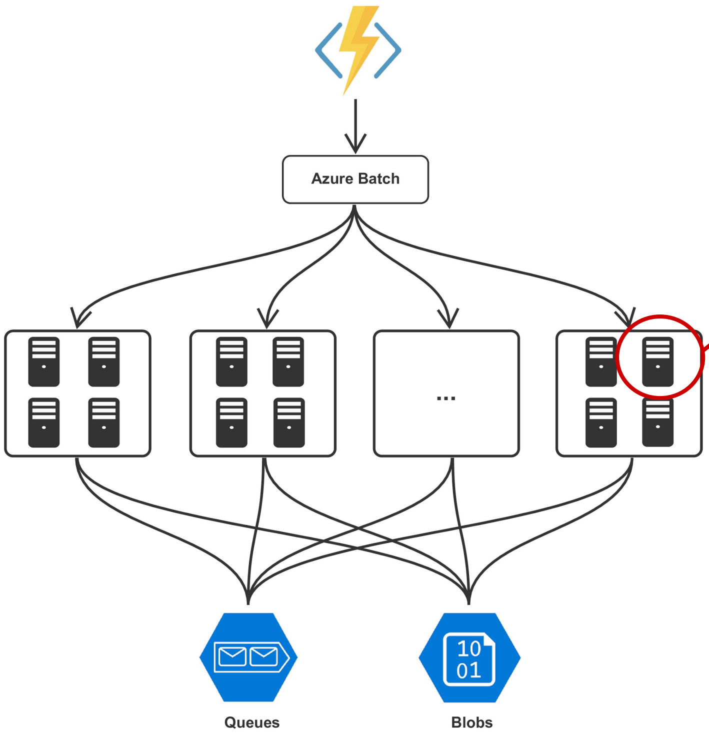




- Requires little to no work
- Long cluster start-up time and cost
- Idle instances/resilience/bandwidth/etc.
- Technically infeasible for industry scale

Moving to the cloud

Go serverless
(and re-engineer)



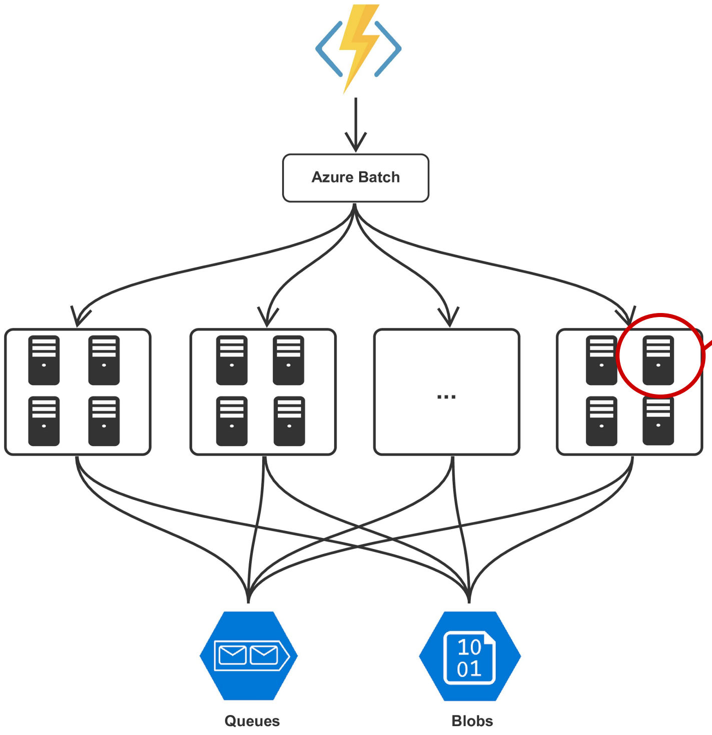
pde = model.m*u.dt2 - u.laplace + model.damp*u.dt

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

* e.g. Valenciano, 2008; Dong et al., 2012; Zeng et al. 2014

SLIM

Serverless LS-RTM in the cloud

Typical components of LS-RTM*:

$$\underset{\delta \mathbf{m}}{\text{minimize}} \quad \sum_{i=1}^{n_s} \frac{1}{2} \left\| \mathbf{J}(\mathbf{m}, \mathbf{q}_i)\ \delta \mathbf{m} - \mathbf{d}_i^{\text{obs}} \right\|_2^2$$

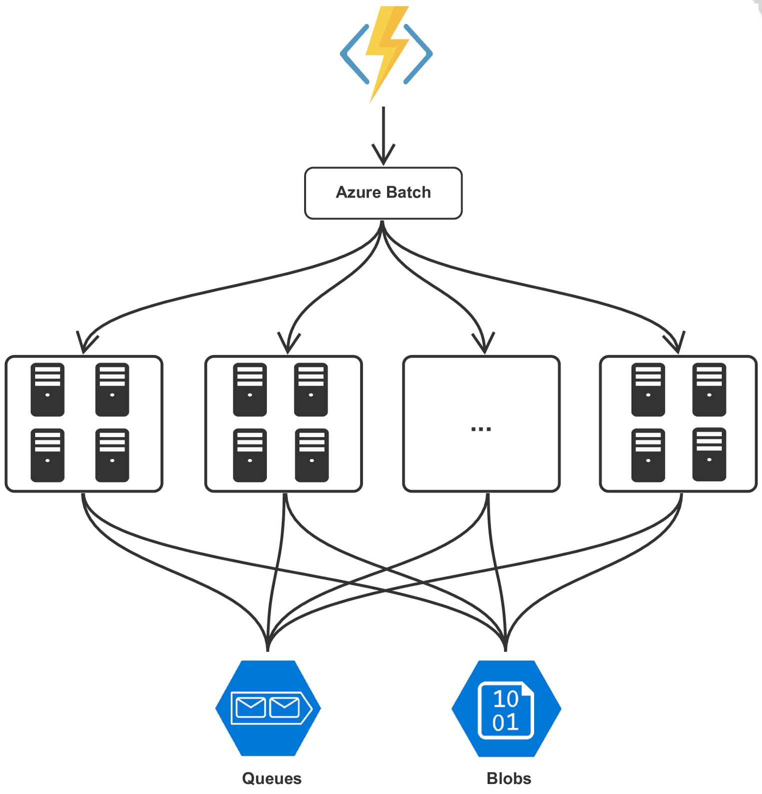
- 1. Compute gradient for all/subset of source locations: $\mathbf{g}_i = \mathbf{J}^\top \left(\mathbf{J}\ \delta \mathbf{m} - \mathbf{d}_i^{\text{obs}} \right)$
- 2. Sum gradients: $\mathbf{g} = \sum_{i=1}^{n_b} \mathbf{g}_i$
- 3. Update image based on optimization algorithm (SGD, CG, etc.):

$$\delta \mathbf{m} = \delta \mathbf{m} - \alpha \mathbf{g}$$

Gradient computations

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)

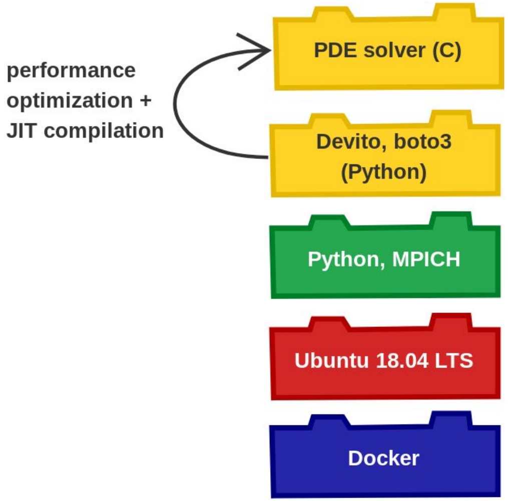


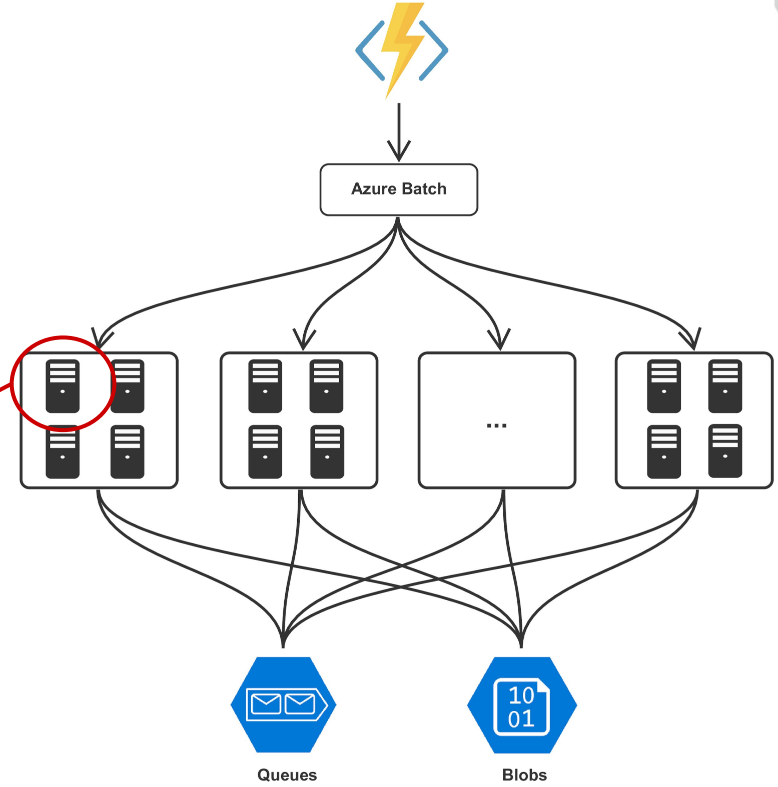
Gradient computations

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

Software to compute gradients:

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

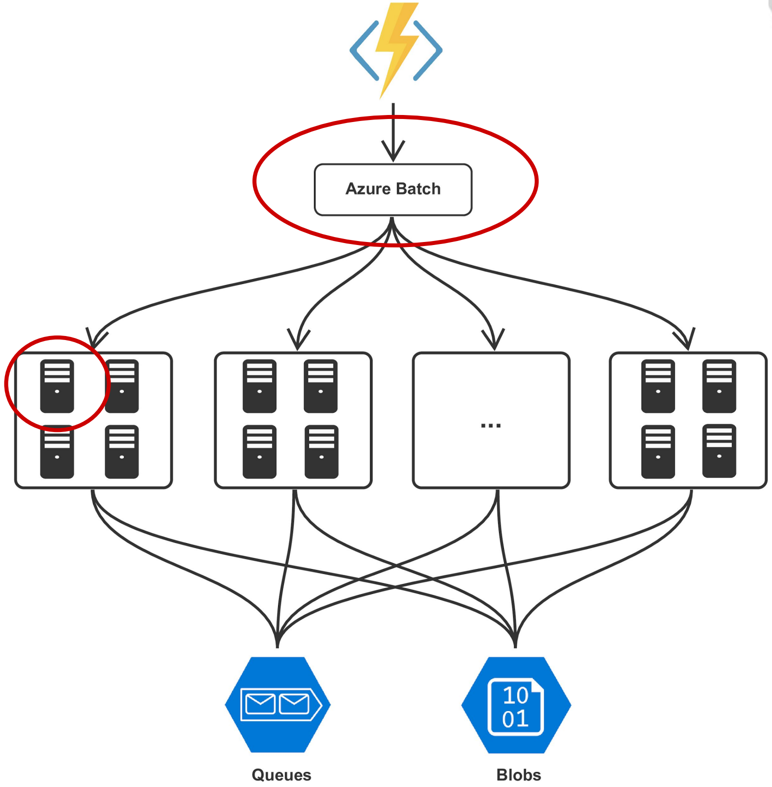




Gradient computations

Azure - Batch Shipyard:

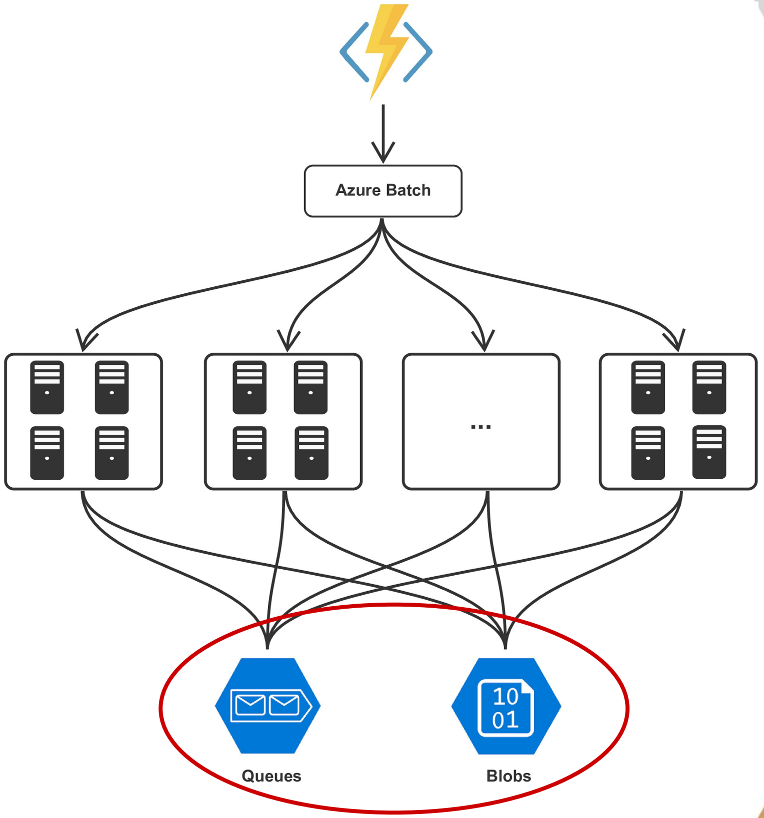
- Tool for executing + monitoring batch jobs
- Many templates for docker + singularity containers
- Pre-exisiting containers for MPI, Infiniband, ML, various compilers, etc.
- Configure pools + jobs using high-level yaml files
- Developed by Microsoft + open source: https://github.com/Azure/batch-shipyard



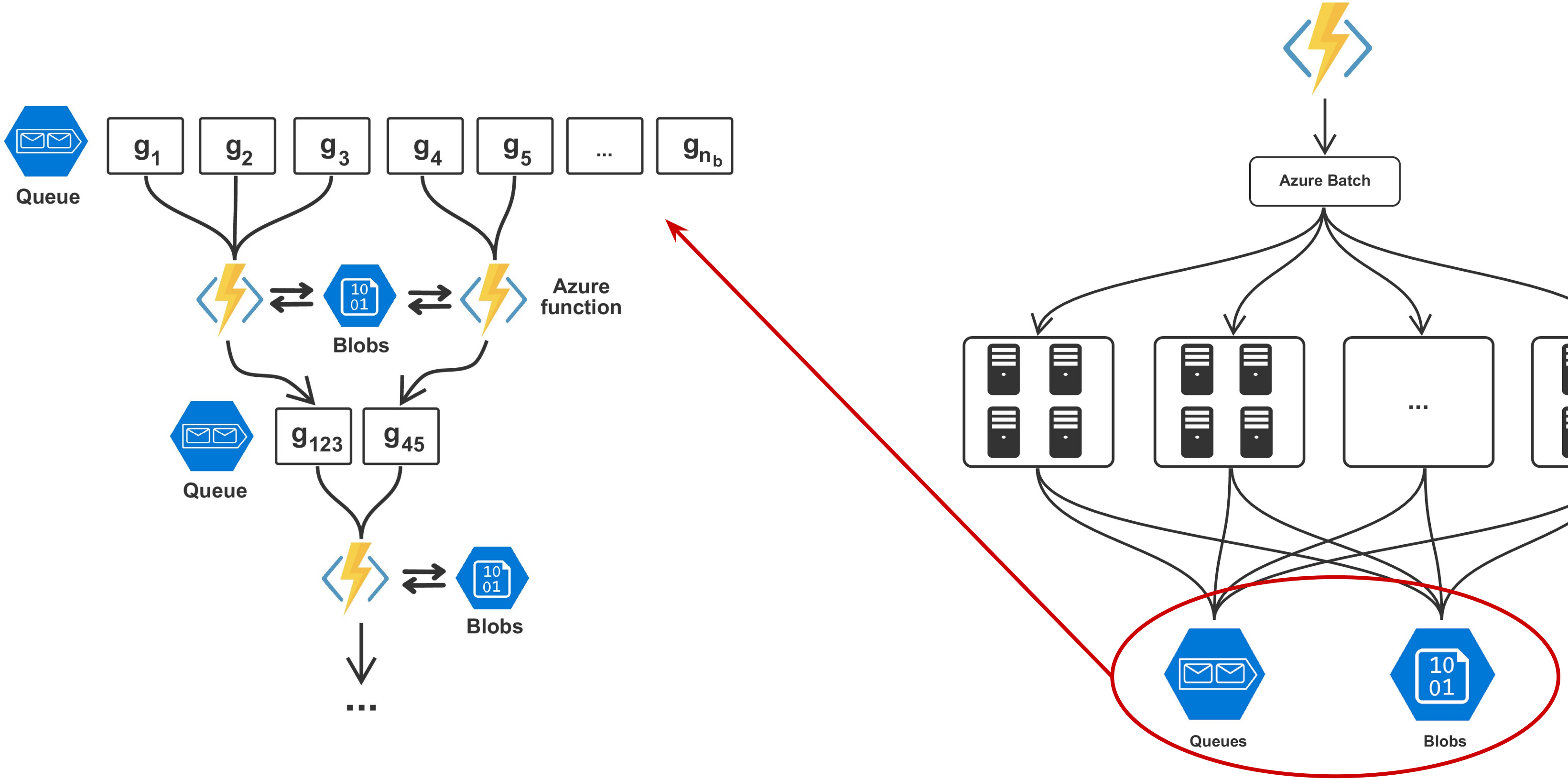
Gradient computations

Summation of gradients

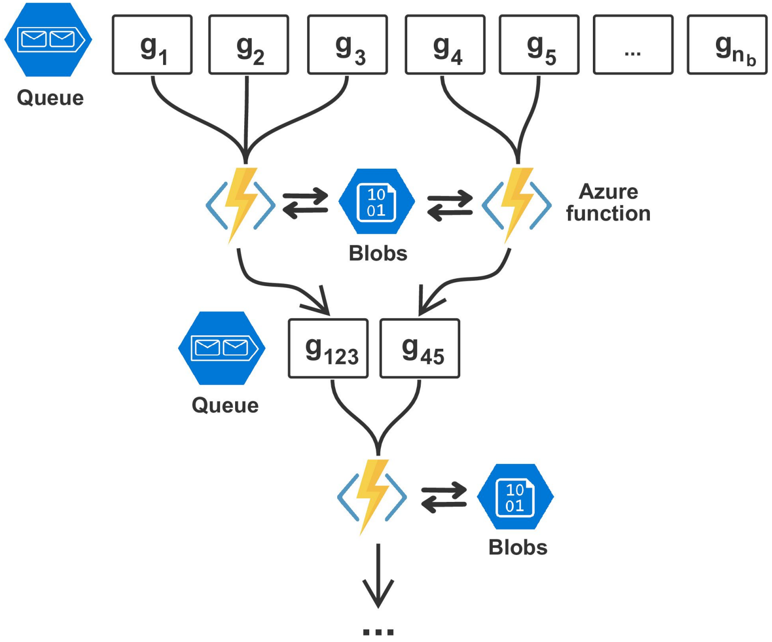
- Gradients stored in object storage (blob)
- Virtually unlimited I/O scalability
- Send object IDs to message queue
- Event-driven gradient summation using Azure functions



Gradient computations



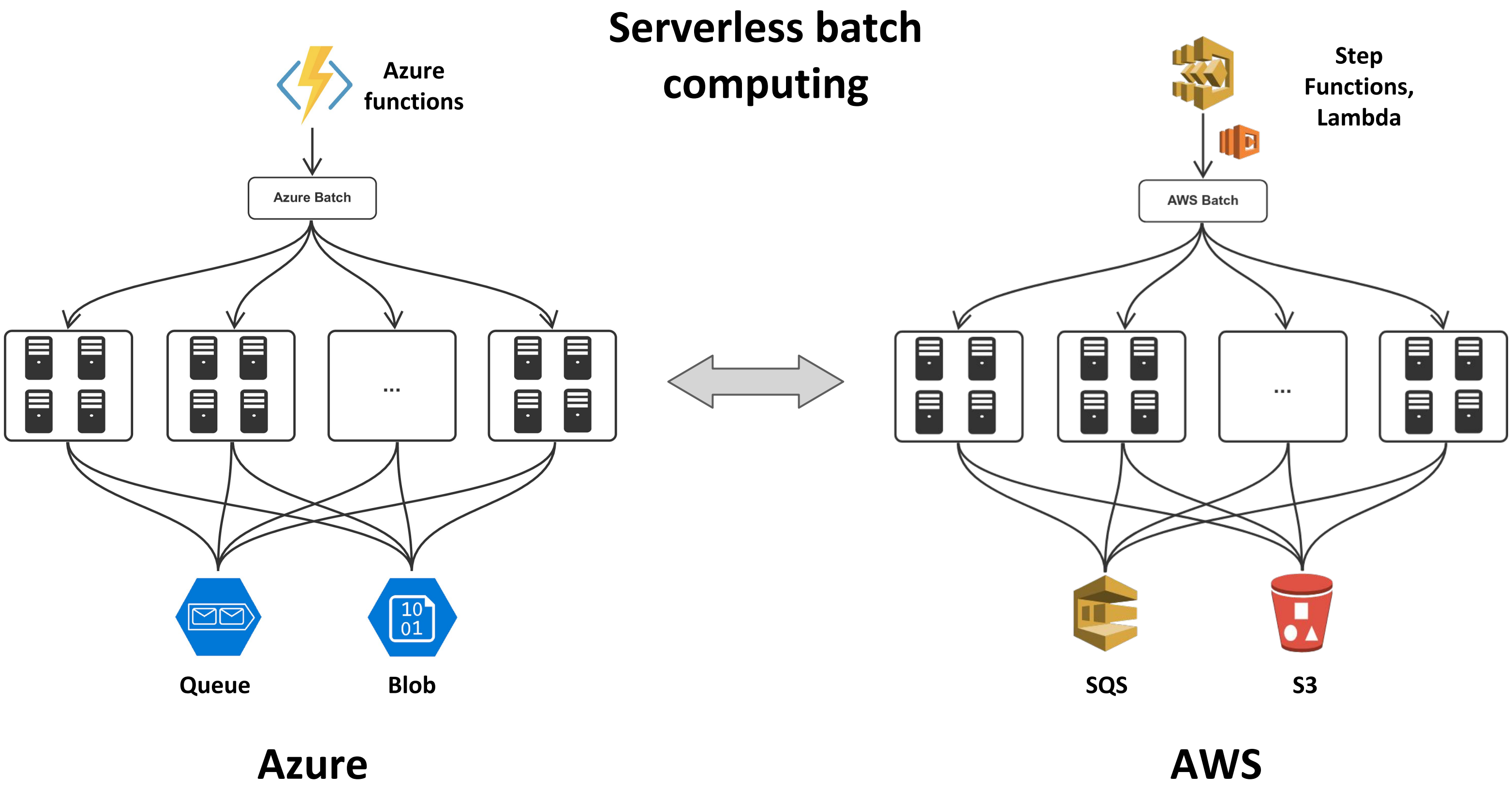
Gradient computations



Event-driven gradient reduction

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

Multi platform approach



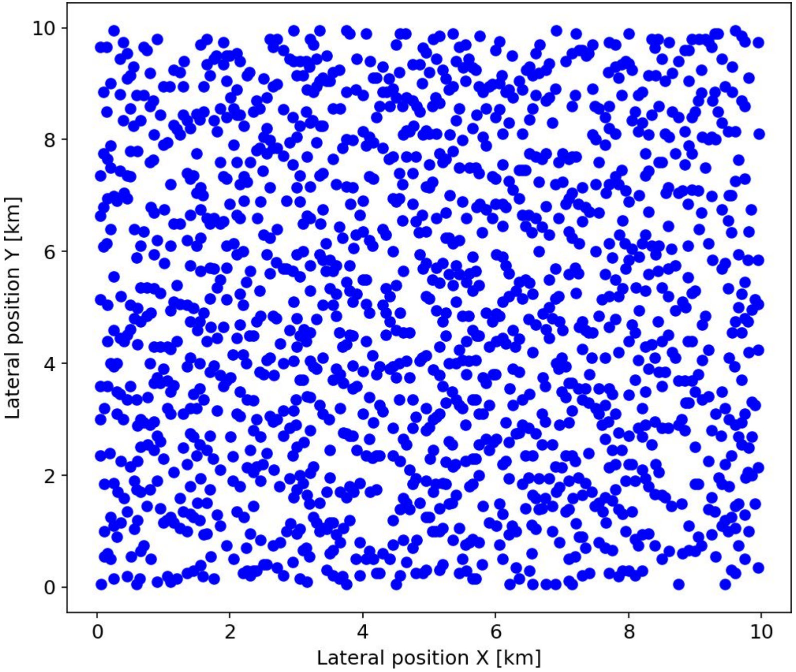
3D TTI RTM on Azure

Synthetic model based on SEG Overthrust + Salt models:

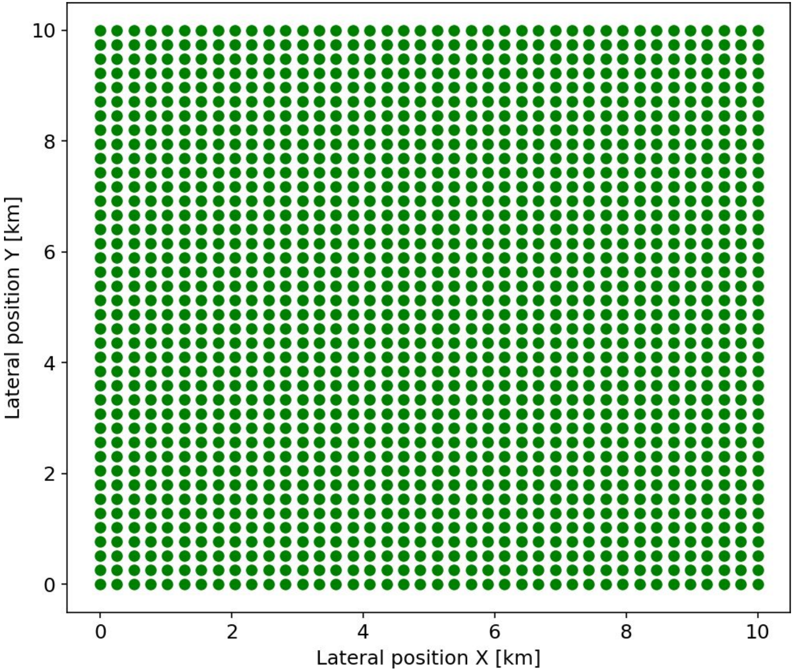
- Domain: 10 x 10 x 3.325 km
- Grid: 881 x 881 x 347 (12.5 m grid + ABCs)
- 1,500 randomly distributed OBNs
- 799 x 799 dense source grid (12.5 m)
- Anisotropic TTI models + density

3D TTI RTM

Acquisition geometry:



OBN receiver grid



Source vessel grid

3D TTI RTM on Azure

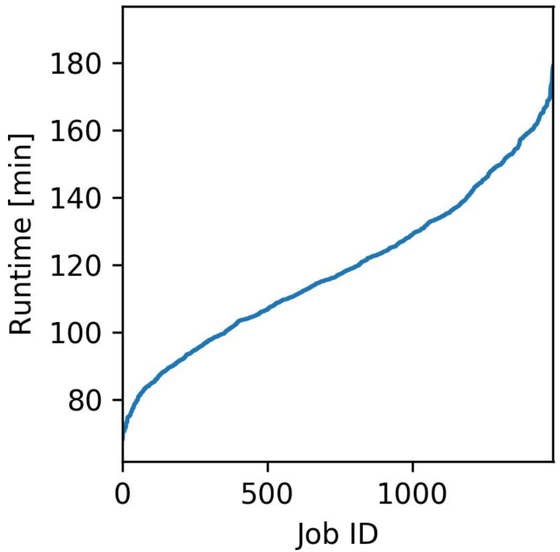
Azure setup:

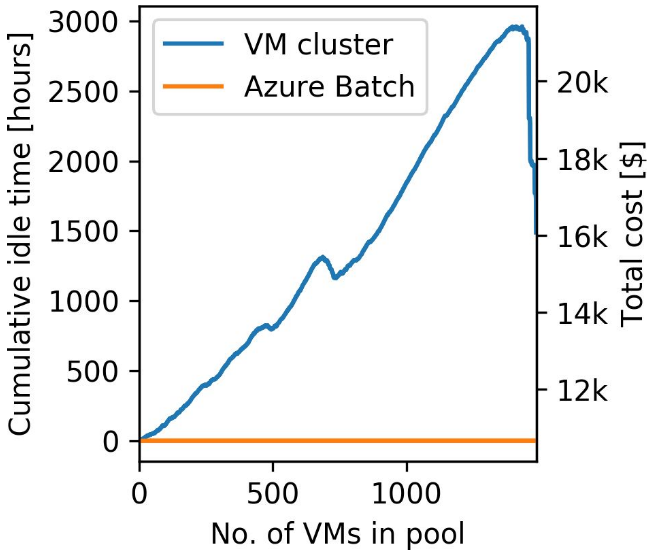
- E64 and ES64 VMs
- 2.3 GHz Intel Xeon ® E5-2673 v4 (Broadwell)
- 432 GB RAM, 64 vCPUs per VM
- 100 VMs -> 6,400 vCPUs
- 2 VMs per gradient (1 MPI rank per socket)
- 600 GB per wavefield
- Peak performance: 140 GFLOPS per VM (14 TFLOPS total)
- Total cost for RTM: ~11,000$ (dedicated/on-demand)

3D TTI RTM on Azure

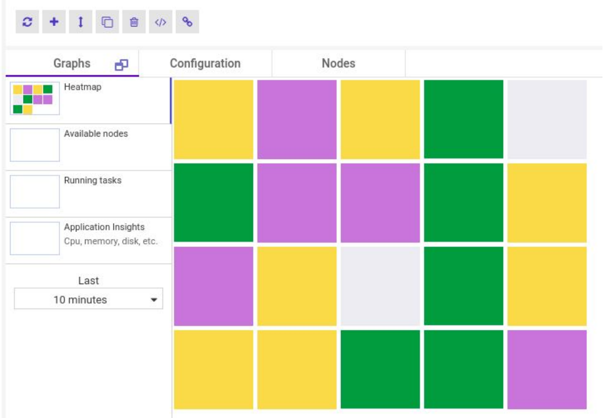
Timings:

- 100 nodes (2 per gradient)
- 1500 source positions
- Average runtime: 110 minutes per gradient (6.7$)
- Peak performance: 140 GFLOPS per VM (14 TFLOPS total)





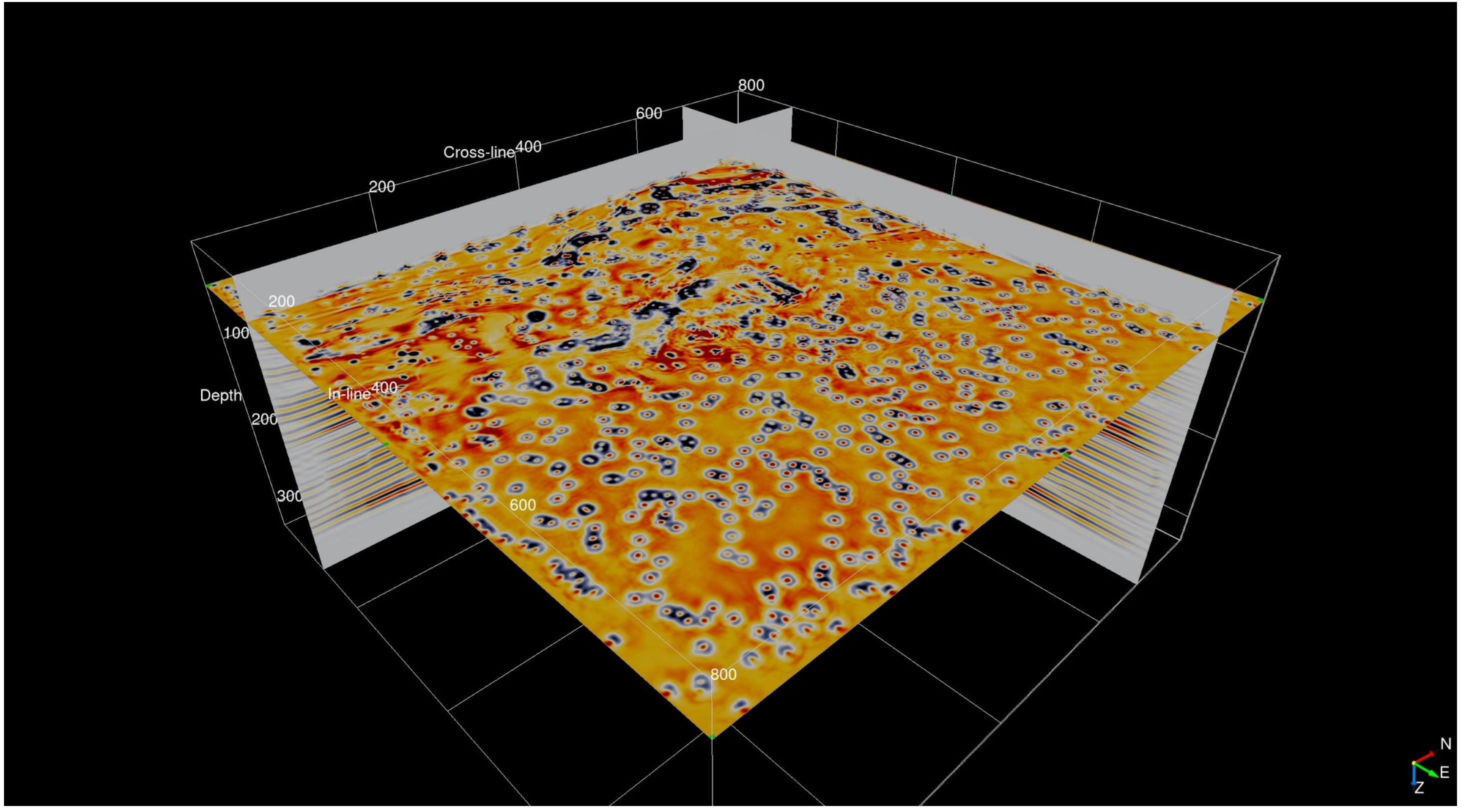
3D TTI RTM



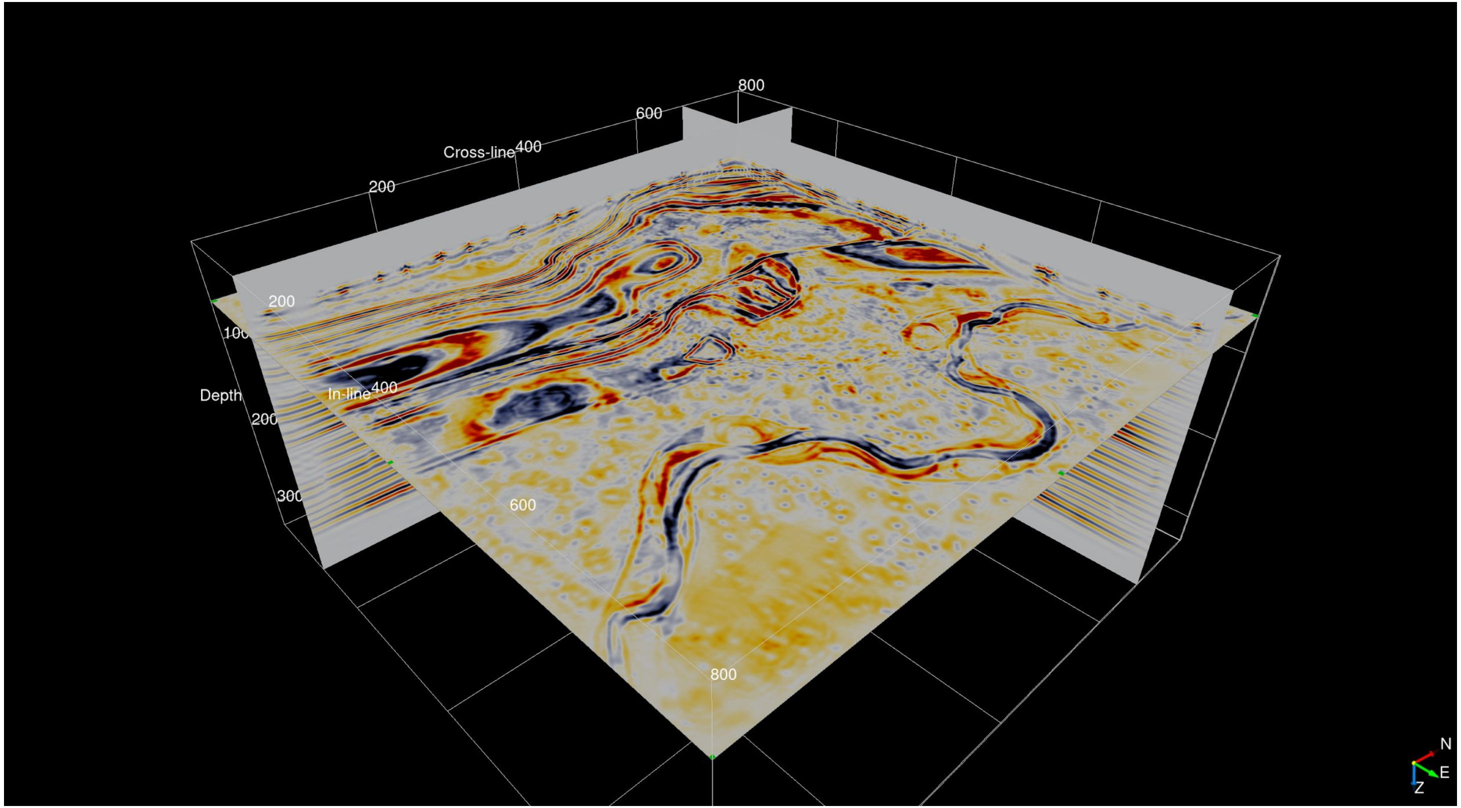
Azure Batch:

- Jobs start as VMs are added to pool
- Do not need to wait for full pool
- No long idle times

3D TTI RTM

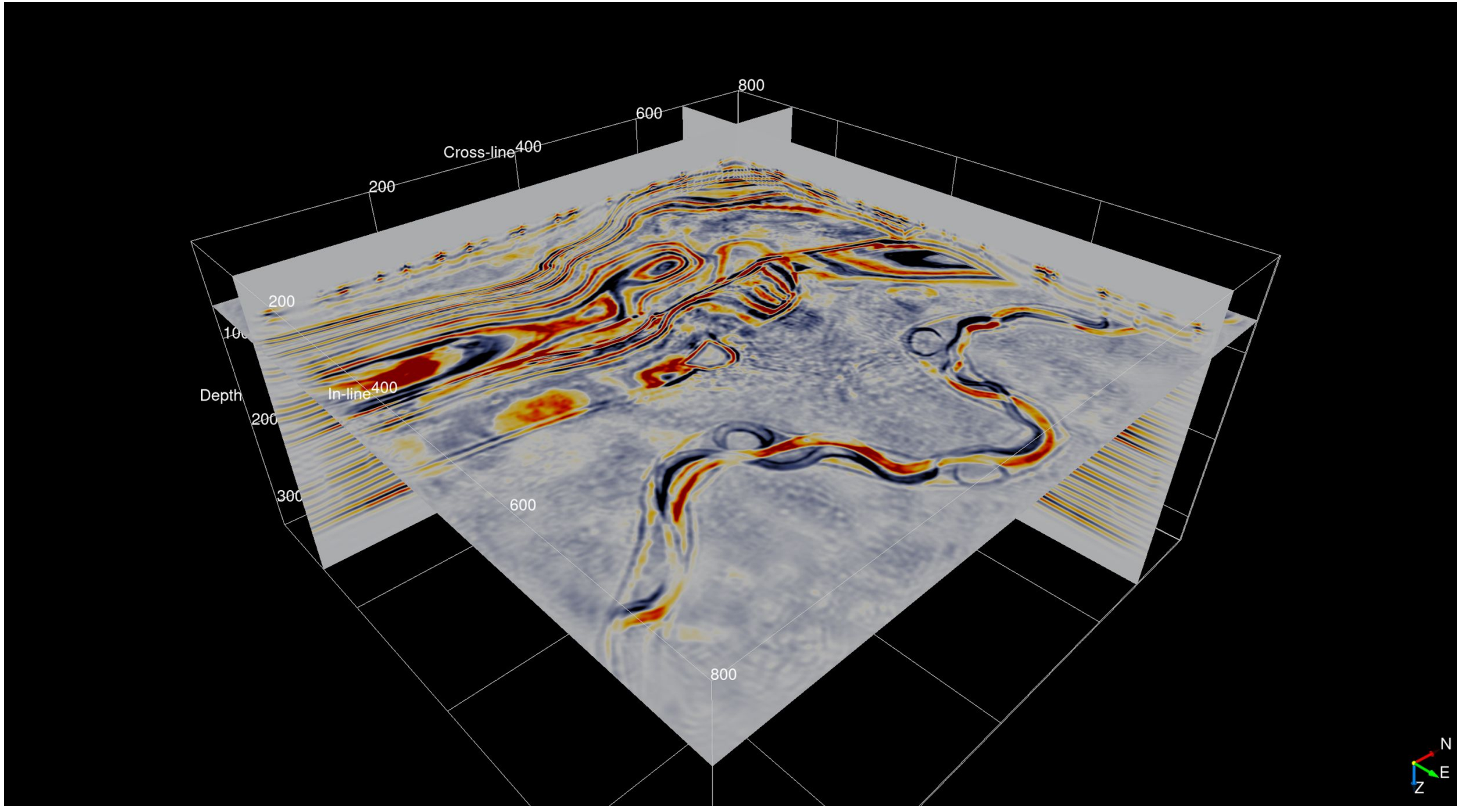

Cross-line
200
400
600
800
Depth
100
200
300
In-line
200
400
600
800
N
E
Z

3D TTI RTM



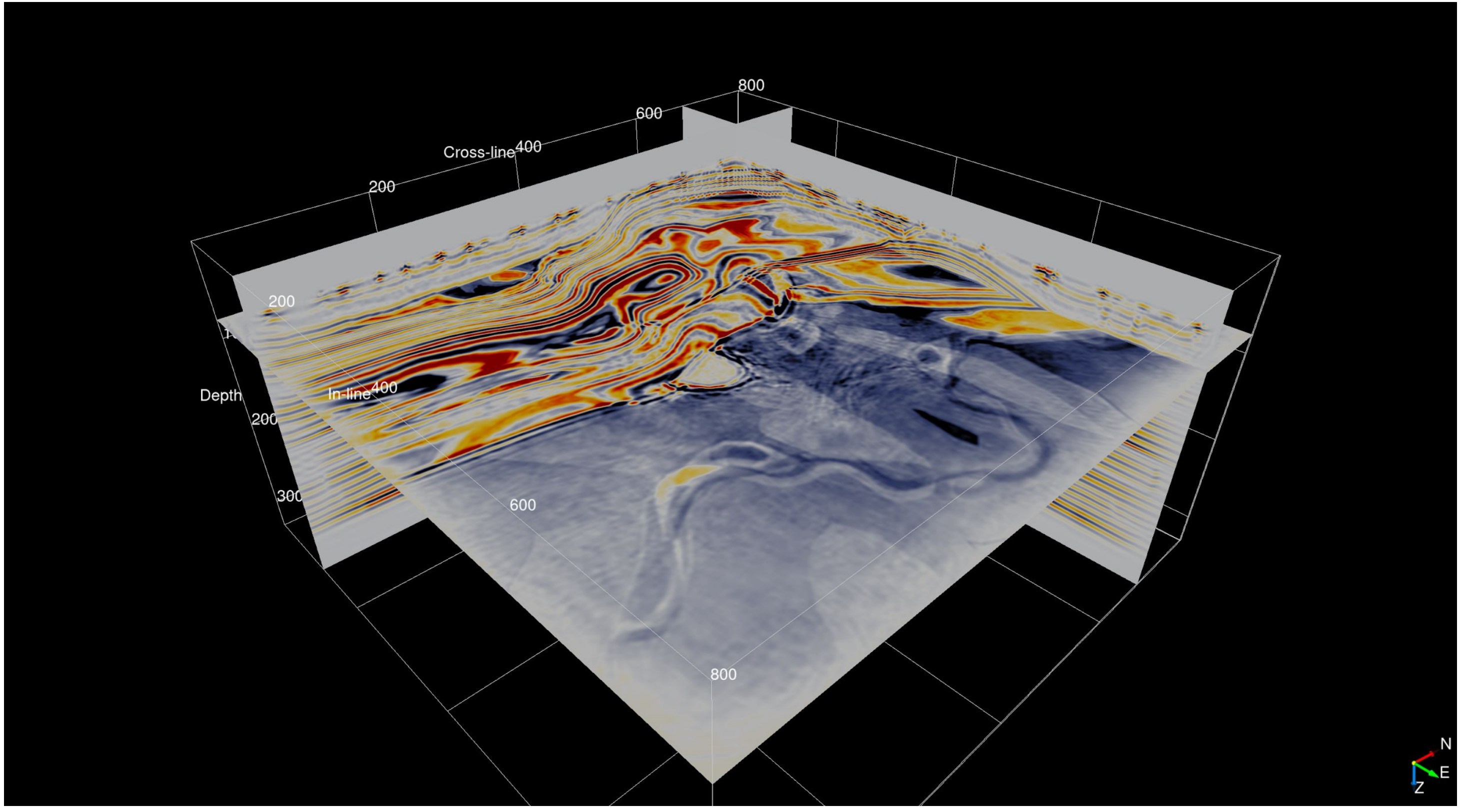
800
600
Cross-line
400
200
200
100
Depth
In-line
400
200
300
600
800
N
E
Z

3D TTI RTM



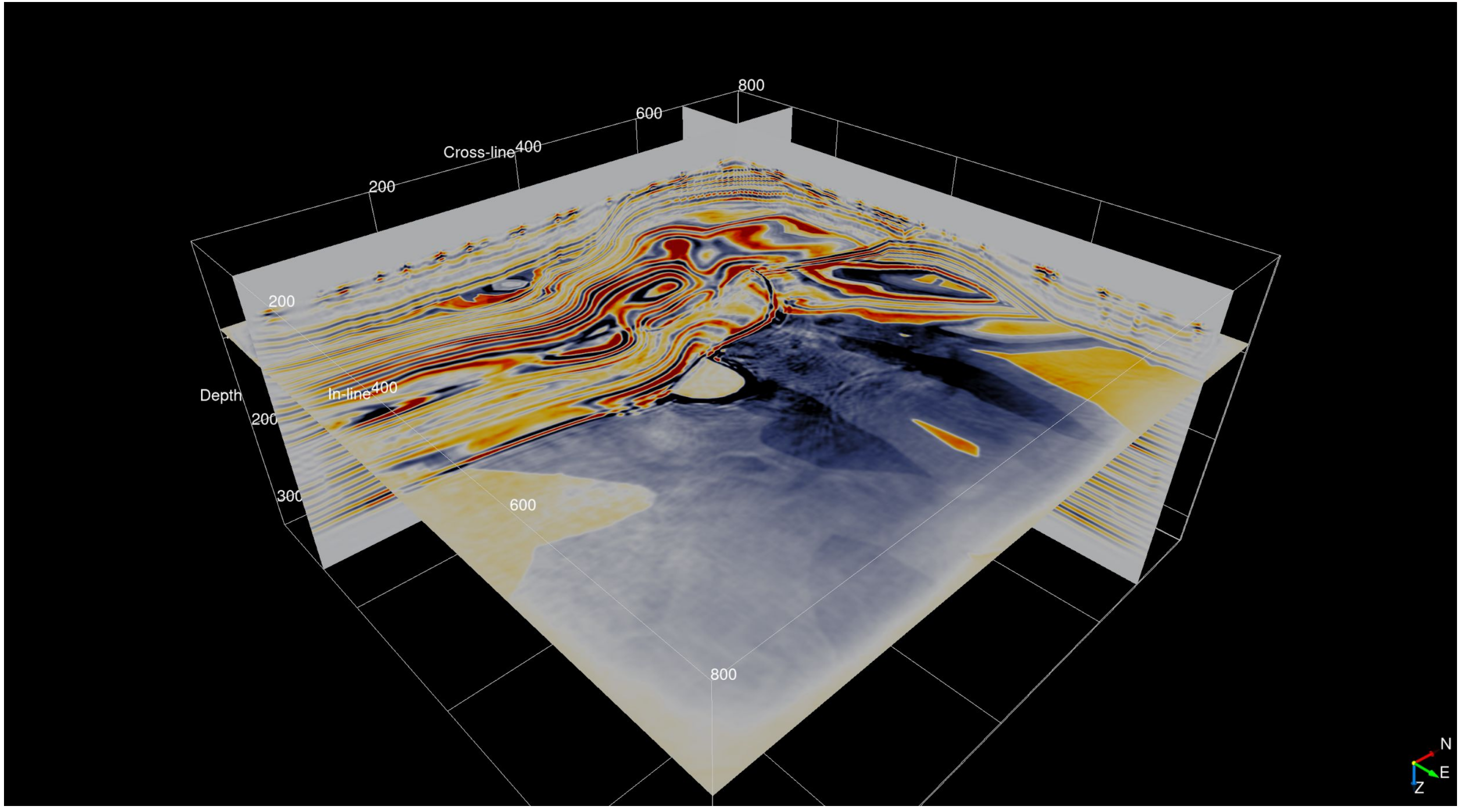
800
600
Cross-line 400
200
200
100
Depth
In-line 400
200
300
600
800
N
E
Z

3D TTI RTM



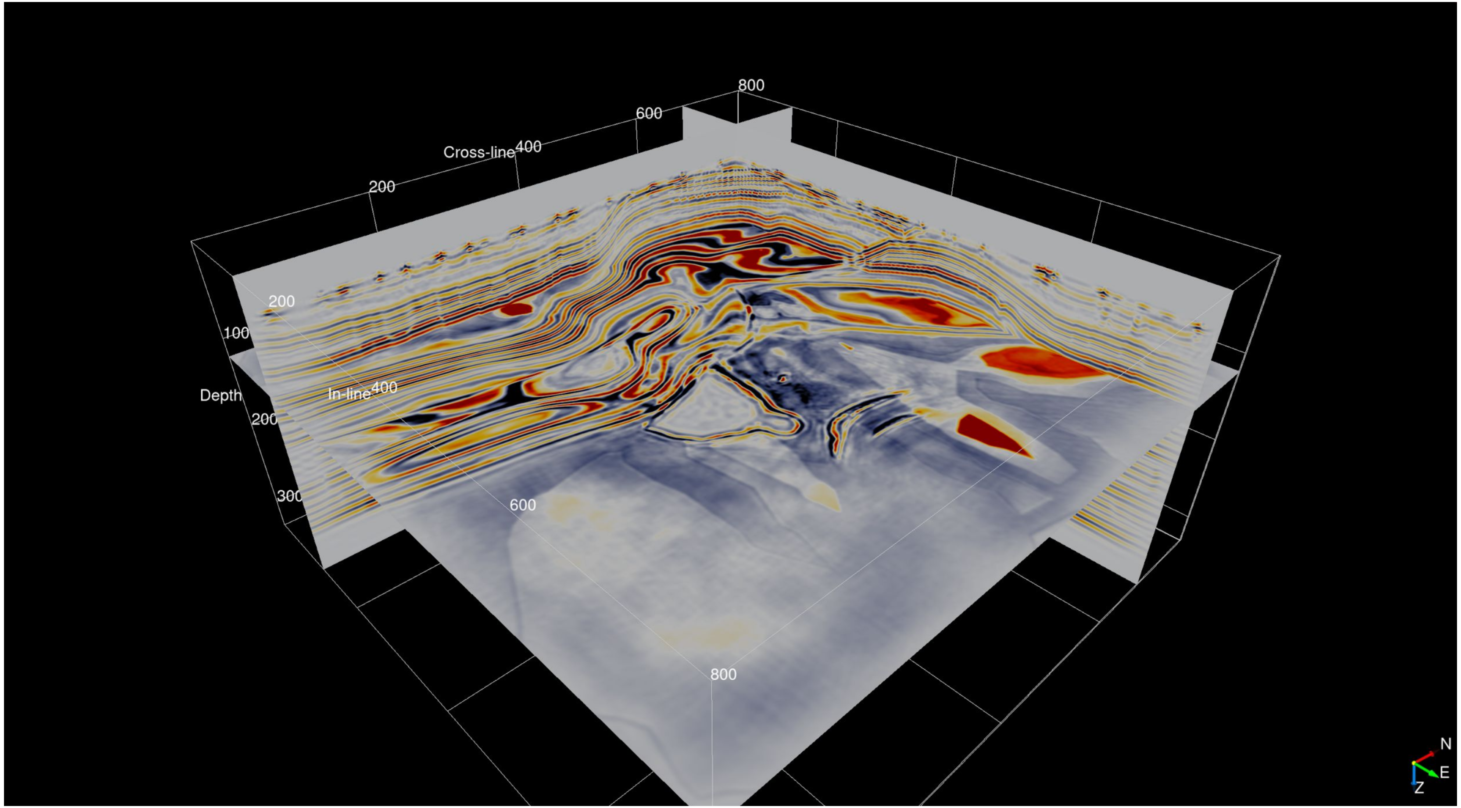
800
600
Cross-line 400
200
200
Depth
In-line 400
200
300
600
800
N
E
Z

3D TTI RTM



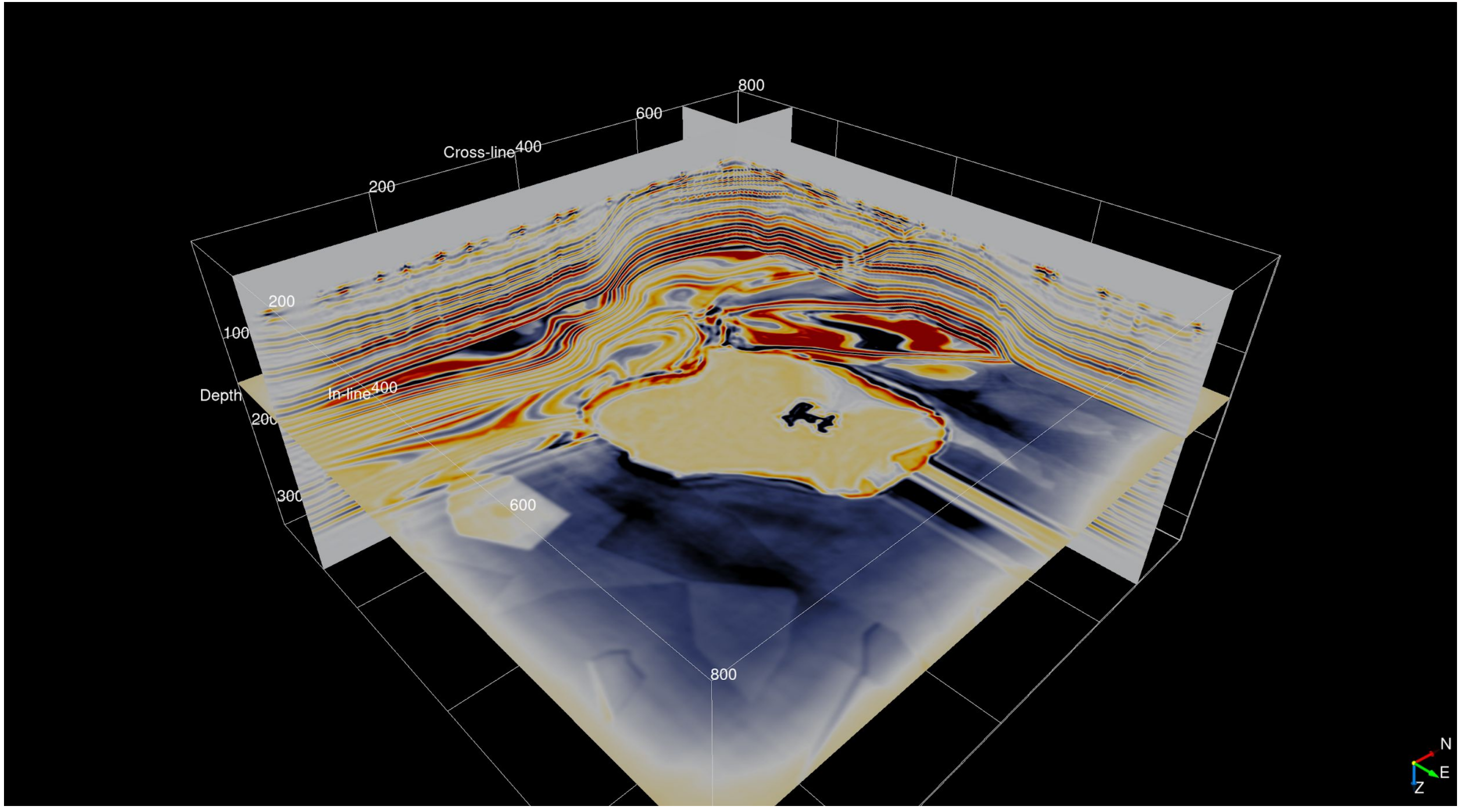
800
600
Cross-line 400
200
200
Depth
In-line 400
200
300
600
800
N
E
Z

3D TTI RTM



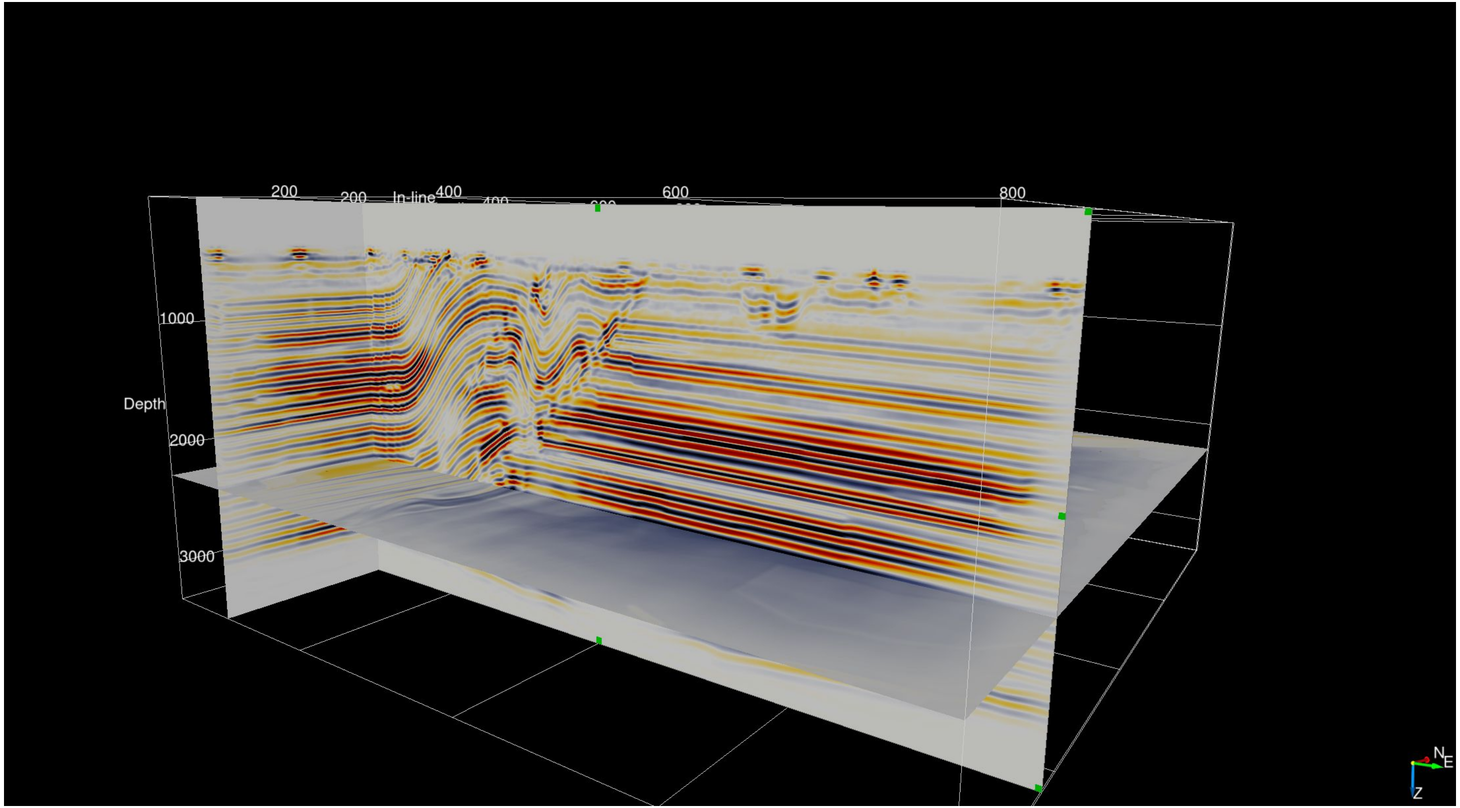
800
600
Cross-line
400
200
200
100
Depth
In-line
400
200
300
600
800
N
E
Z

3D TTI RTM

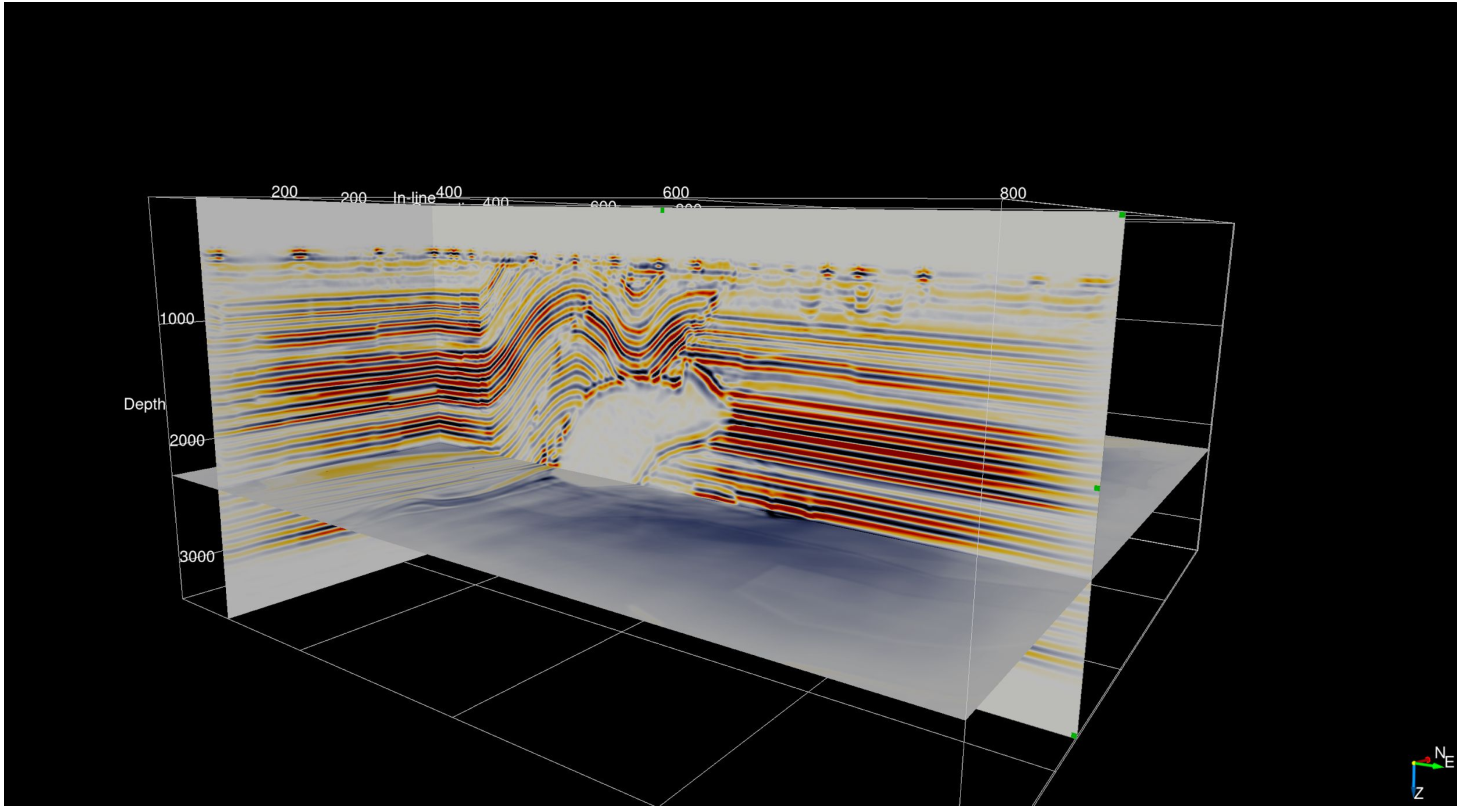


800
600
Cross-line
400
200
200
100
Depth
In-line
400
200
300
600
800
N
E
Z

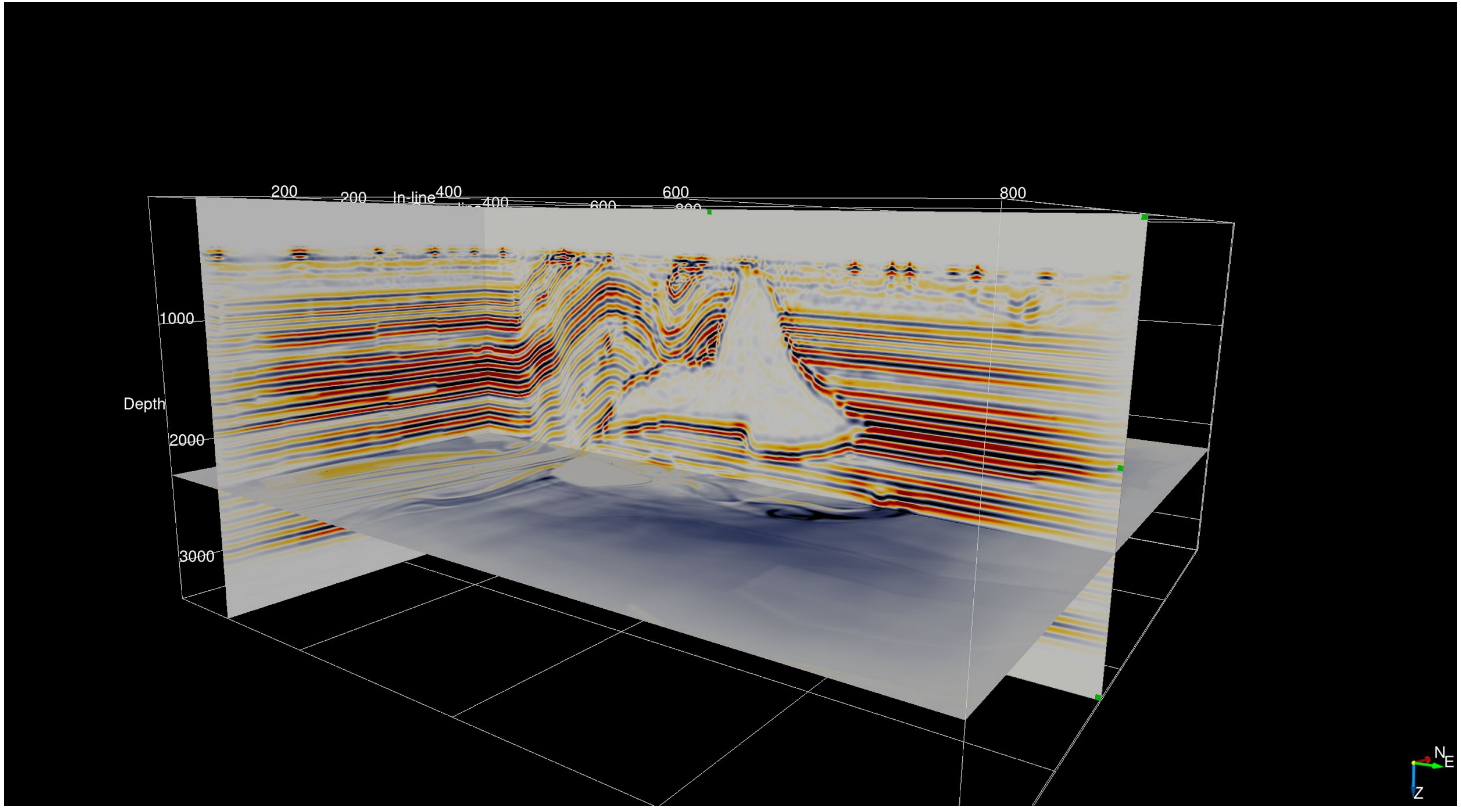
3D TTI RTM



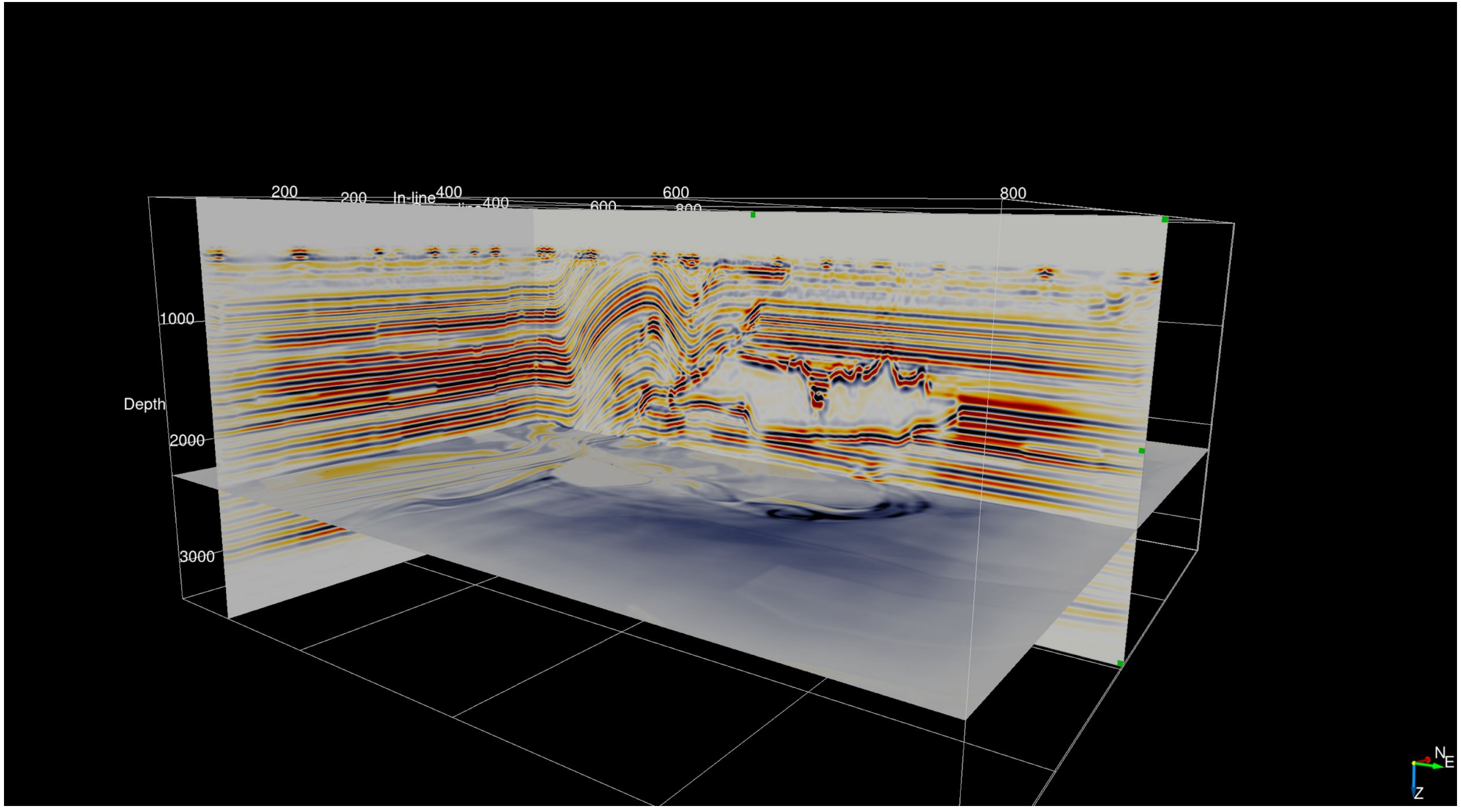
3D TTI RTM



3D TTI RTM

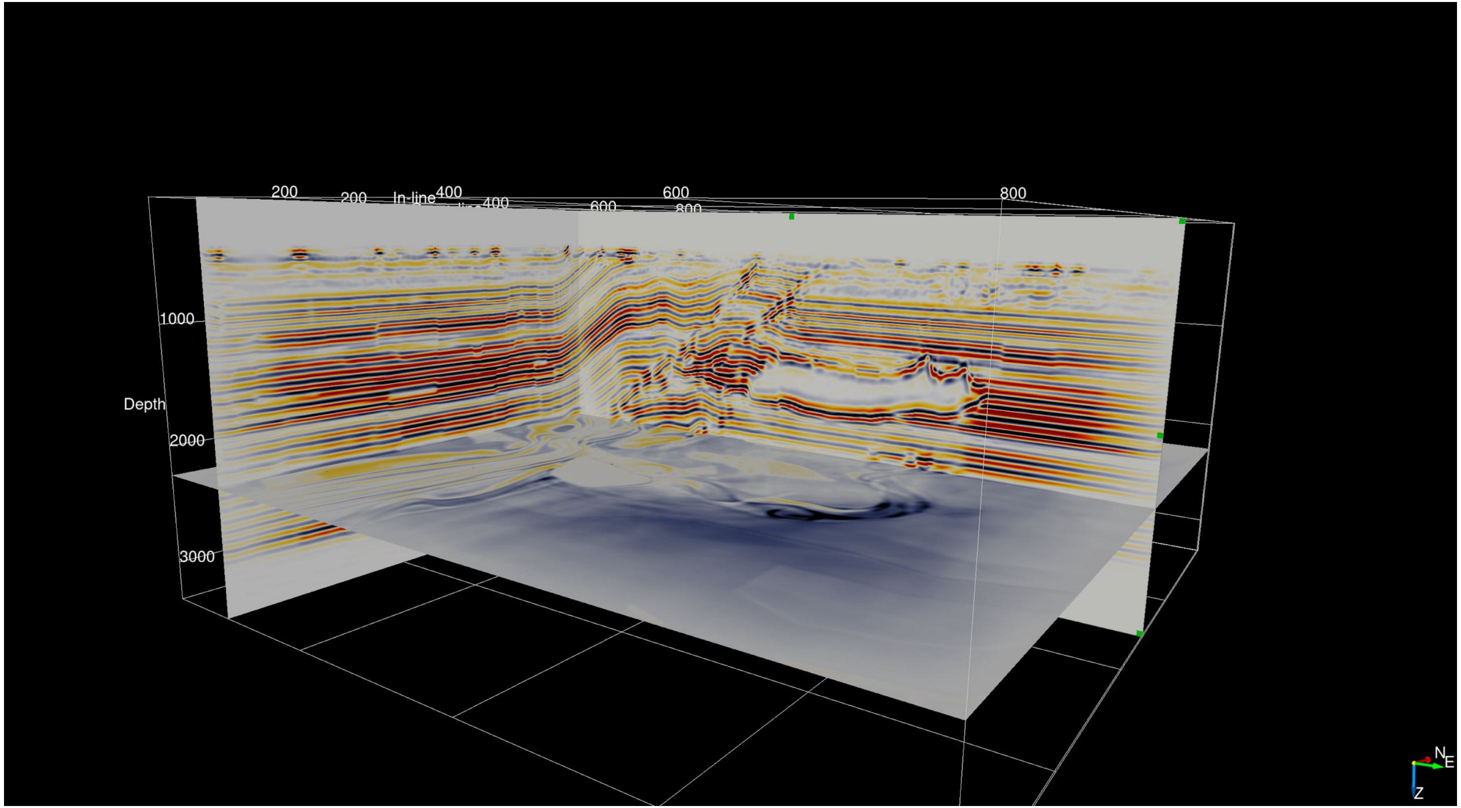


3D TTI RTM



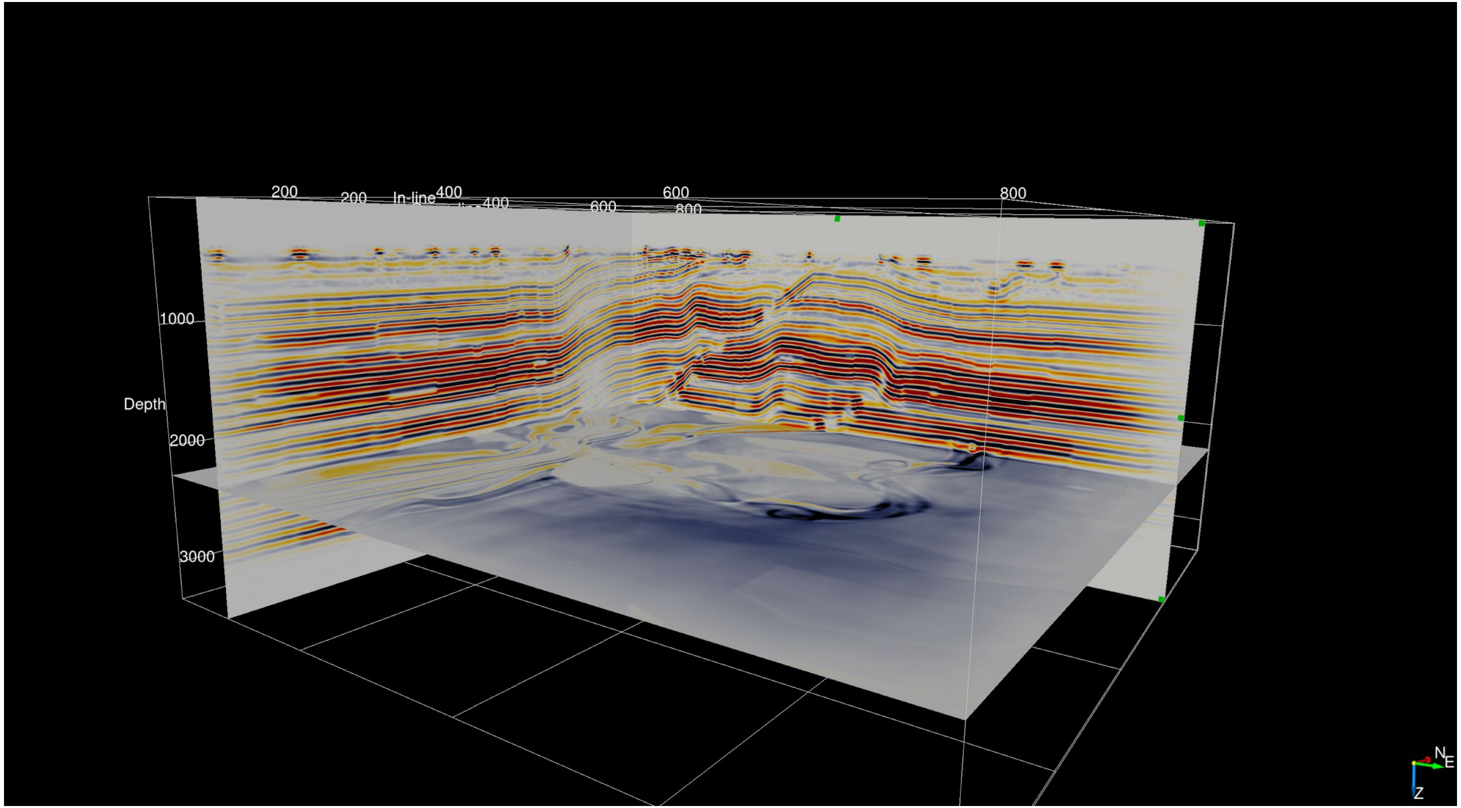
200
200
In-line
400
400
600
600
800
800
1000
Depth
2000
3000
N
E
Z

3D TTI RTM



200
200
In-line
400
400
600
600
800
800
1000
Depth
2000
3000
N
E
Z

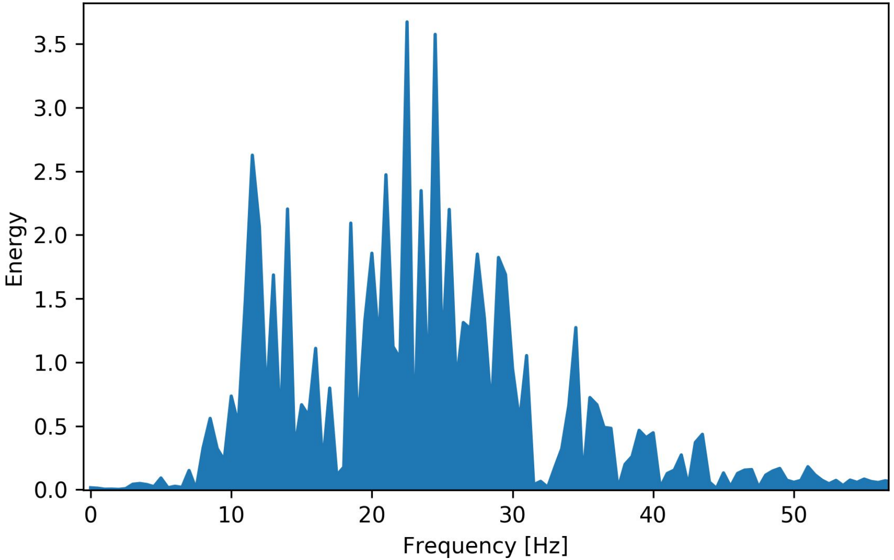
3D TTI RTM



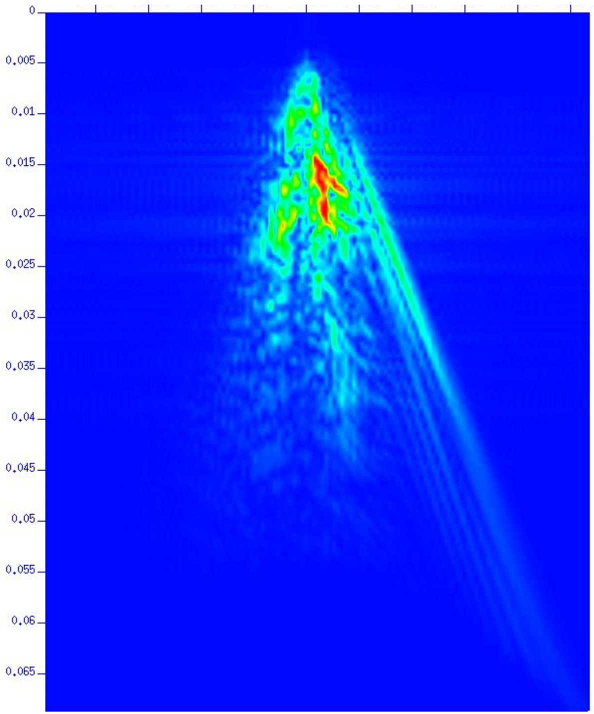
200
200
In-line
400
400
600
600
800
800
1000
Depth
2000
3000
N
E
Z

3D TTI RTM on Azure

Observed data: 1,500 shots



Trace spectrum



FK spectrum of single shot

Conclusions

Seismic imaging in the cloud:

- Rethink how to bring software to the cloud
- Lift and shift approach not ideal
- Take advantage of new cloud technologies
- Batch computing, serverless/event-driven computations, object storage, spot instances
- **Software based on separation of concerns + abstractions is prerequisite to go serverless**

Acknowledgments

Thank you for your attention!

Journal preprint w/ performance & cost analysis:

- *https://arxiv.org/abs/1909.01279*