Fast in-memory elastic full-waveform inversion using consumer-grade GPUs

Tore S. Bergslid[1], Espen Birger Raknes[2] and Børge Arntsen[1]

1: Norwegian University of Science and Technology (NTNU)
Department of Geoscience and Petroleum

2: Formerly NTNU, presently AkerBP

E-mail: tore.bergslid@ntnu.no

Trondheim
April 25th 2017

NTNU – Trondheim
Norwegian University of Science and Technology

Outline

Introduction

- Full-waveform inversion involves numerical modeling of wave propagation.

Introduction

- Full-waveform inversion involves numerical modeling of wave propagation.
- Computationally demanding.

Introduction

- Full-waveform inversion involves numerical modeling of wave propagation.
- Computationally demanding.
- Produces large amounts of data.

Introduction

- Full-waveform inversion involves numerical modeling of wave propagation.
- Computationally demanding.
- Produces large amounts of data.
- Elastic wave equation adds more computations and data storage requirements over the acoustic approximation.

Introduction

- Graphics processing units can accelerate FWI.

Introduction

- Graphics processing units can accelerate FWI.
- Industrial graphics processing units are expensive.

Introduction

- Graphics processing units can accelerate FWI.
- Industrial graphics processing units are expensive.
- Can cheaper gaming GPUs be used instead of expensive industrial GPUs?

Introduction

- Graphics processing units can accelerate FWI.
- Industrial graphics processing units are expensive.
- Can cheaper gaming GPUs be used instead of expensive industrial GPUs?
- Can we eliminate much of the slow file I/O by keeping wavefields in memory?

Theory

- In FWI we want to find a parameter model $\mathbf{m}$ that can produce modeled data $\mathbf{u}$ which is close to some measured data $\mathbf{d}$.

Theory

- In FWI we want to find a parameter model $\mathbf{m}$ that can produce modeled data $\mathbf{u}$ which is close to some measured data $\mathbf{d}$.
- Apply a numerical wave operator that maps $\mathbf{m}$ from the model domain into the data domain:
$$\mathcal{L}(\mathbf{m}) = \mathbf{u}. \tag{1}$$

Theory

- In FWI we want to find a parameter model $\mathbf{m}$ that can produce modeled data $\mathbf{u}$ which is close to some measured data $\mathbf{d}$.
- Apply a numerical wave operator that maps $\mathbf{m}$ from the model domain into the data domain:

$$\mathcal{L}(\mathbf{m}) = \mathbf{u}. \tag{1}$$

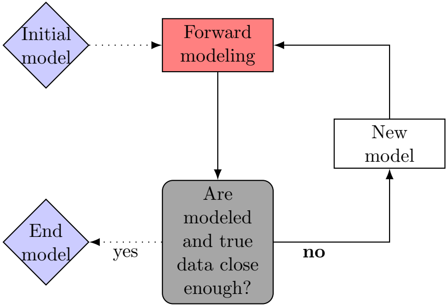
- Ideally, find an inverse operator to map $\mathbf{d}$ from the data domain to the model domain:

$$\mathbf{m} = \mathcal{L}^{-1}(\mathbf{d}). \tag{2}$$

Theory

- Define a misfit functional:

$$\mathcal{F}(\mathbf{m}) = \frac{1}{2}\sum_{j=0}^{n_s}\sum_{i=0}^{n_r} ||\hat{\mathbf{u}}_{i,j}(\mathbf{m}) - \hat{\mathbf{d}}_{i,j}||_2^2. \quad (3)$$



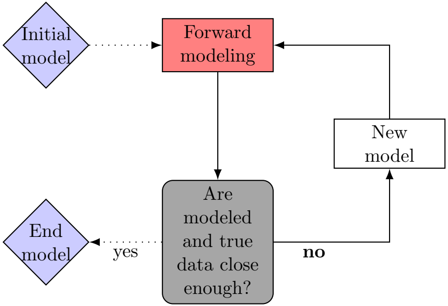
Theory

- Define a misfit functional:

$$\mathcal{F}(\mathbf{m}) = \frac{1}{2}\sum_{j=0}^{n_s}\sum_{i=0}^{n_r} ||\hat{\mathbf{u}}_{i,j}(\mathbf{m}) - \hat{\mathbf{d}}_{i,j}||_2^2. \qquad (3)$$

- The solution is an extreme point of $\mathcal{F}(\mathbf{m})$:

$$\mathbf{m}' = \arg\min_{\mathbf{m}} \mathcal{F}(\mathbf{m}). \qquad (4)$$



Theory

- Update the model iteratively:

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \alpha_k \mathbf{H}_k^{-1} \delta \mathbf{m}_k. \tag{5}$$

Theory

- Update the model iteratively:

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \alpha_k \mathbf{H}_k^{-1} \delta \mathbf{m}_k. \tag{5}$$

- Hessian matrix contains second derivatives of the misfit functional

Theory

- Update the model iteratively:

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \alpha_k \mathbf{H}_k^{-1} \delta \mathbf{m}_k. \tag{5}$$

- Hessian matrix contains second derivatives of the misfit functional
    - Approximated from previous gradients (L-BFGS)

Theory

- Update the model iteratively:

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \alpha_k \mathbf{H}_k^{-1} \delta \mathbf{m}_k. \tag{5}$$

- Hessian matrix contains second derivatives of the misfit functional
    - Approximated from previous gradients (L-BFGS)
- Gradients are found via the adjoint method, Mora (1987).

$$\delta \hat{\mathbf{m}}(\mathbf{x}) = \sum_{n_s} \int \mathrm{d}t \sum_{n_r} \frac{\partial u_i(\mathbf{x_S}, \mathbf{x_R}, t)}{\partial \mathbf{m}(\mathbf{x})} \delta u_i(\mathbf{x_S}, \mathbf{x_R}, t). \tag{6}$$

$$\delta u_i(\mathbf{x_S}, \mathbf{x_R}, t) = \int_V \mathrm{d}V \frac{\partial u_i(\mathbf{x_S}, \mathbf{x_R}, t)}{\partial \mathbf{m}(\mathbf{x})} \delta \mathbf{m}(\mathbf{x}). \tag{7}$$

Gradients, 2D

$$\delta\rho = -\sum_{n_s}\int \mathrm{d}t\,\dot{u}_j\dot{\Psi}_j,$$

$$\delta c_{11} = -\sum_{n_s}\int \mathrm{d}t\,u_{1,1}\Psi_{1,1},$$

$$\delta c_{33} = -\sum_{n_s}\int \mathrm{d}t\,u_{3,3}\Psi_{3,3},$$

$$\delta c_{13} = -\sum_{n_s}\int \mathrm{d}t\Big(\Psi_{3,3}u_{1,1} + \Psi_{1,1}u_{3,3}\Big),$$

$$\delta c_{44} = -\sum_{n_s}\int \mathrm{d}t(\Psi_{3,1} + \Psi_{1,3})(u_{3,1} + u_{1,3}).$$

Hardware

- "Maur" ("Ant")
- 21 nodes
- 2 × Intel Xeon E5-2660 10-core CPUs
- 2 × Nvidia GTX Titan X GPU
- 128 GB RAM

Photo: NTNU HPC

Implementation

- Source-by-source parallelization.

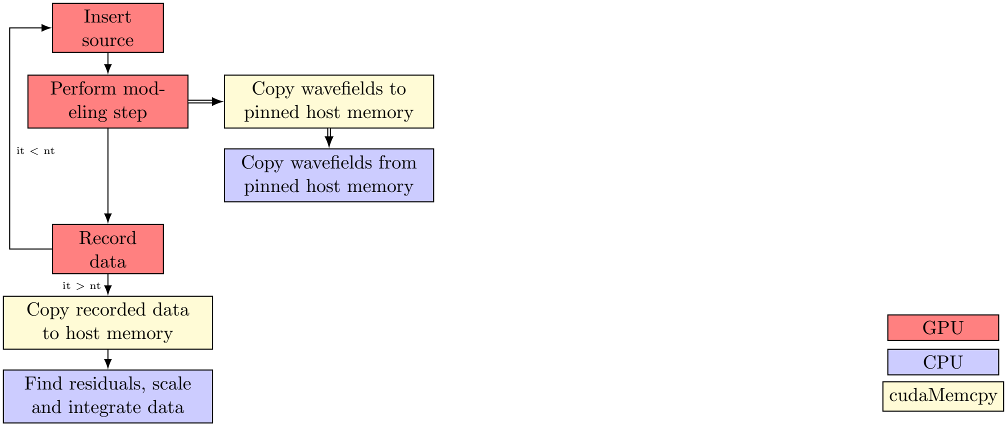
Implementation

- Source-by-source parallelization.
- Less jobs per node leads to more memory available per source modeling, which is a large part of what enables us to do FWI in-memory.

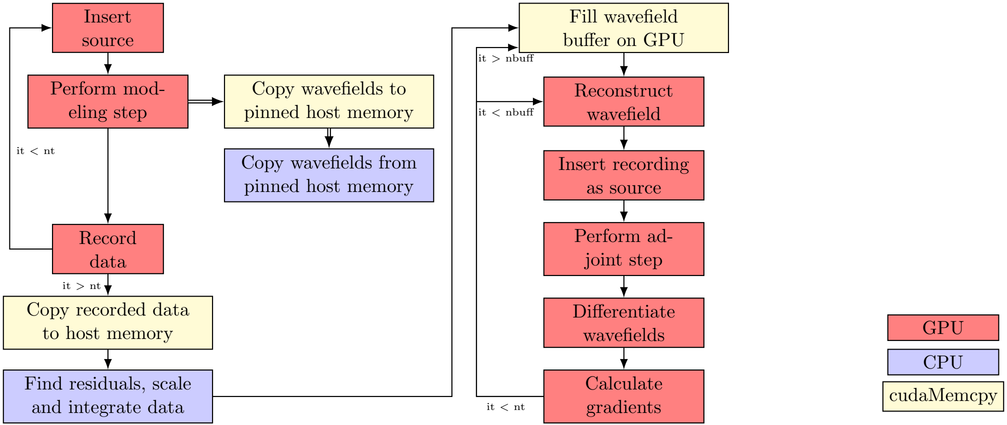
Implementation

- Source-by-source parallelization.
- Less jobs per node leads to more memory available per source modeling, which is a large part of what enables us to do FWI in-memory.
- Wavefield reconstruction by simple interpolation.

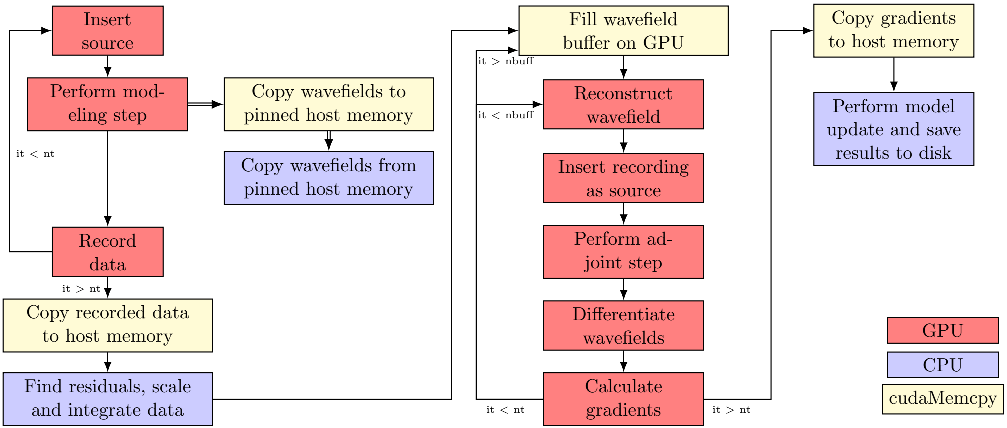
In-memory algorithm



In-memory algorithm

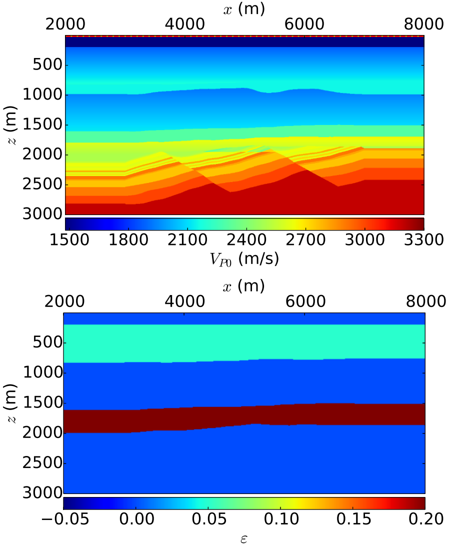


In-memory algorithm

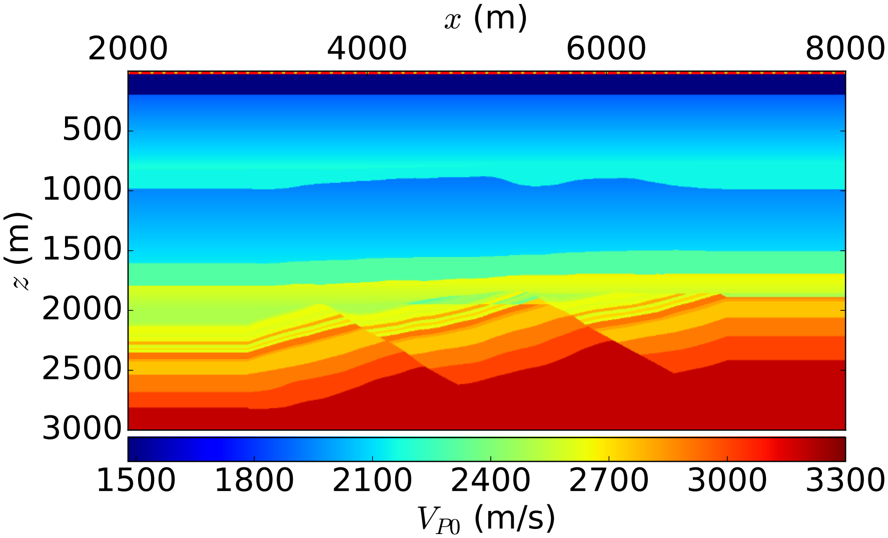


Model

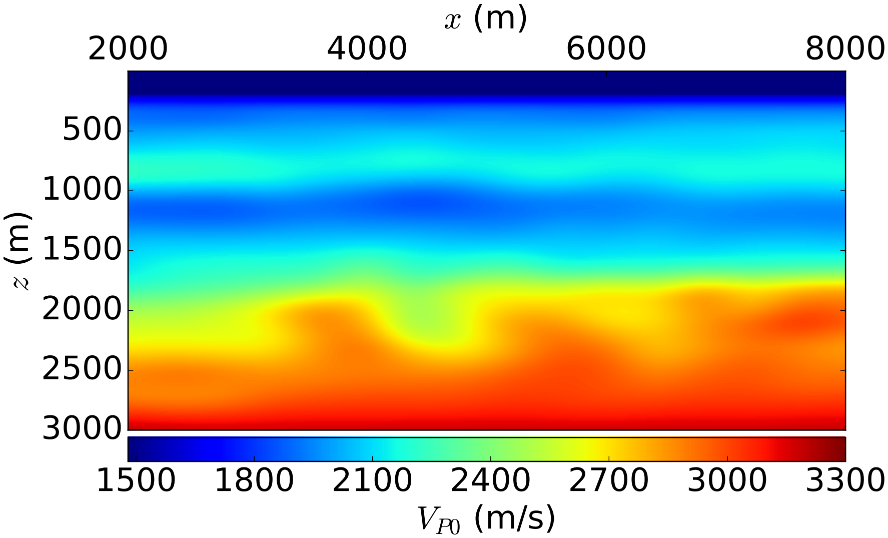
- Synthetic model representative of the Gullfaks field
- 10 km long, 3 km deep
- 2001 $\times$ 600 grid points
- Total of 101 shots and 2001 receivers
- 3.3 second recording, 5500 time steps
- Source: 15 Hz Ricker wavelet bandpass filtered to 0-7 Hz, 0-10 Hz, and unfiltered.
- Receivers: Pressure
- $\sim$ 50 GB RAM per source



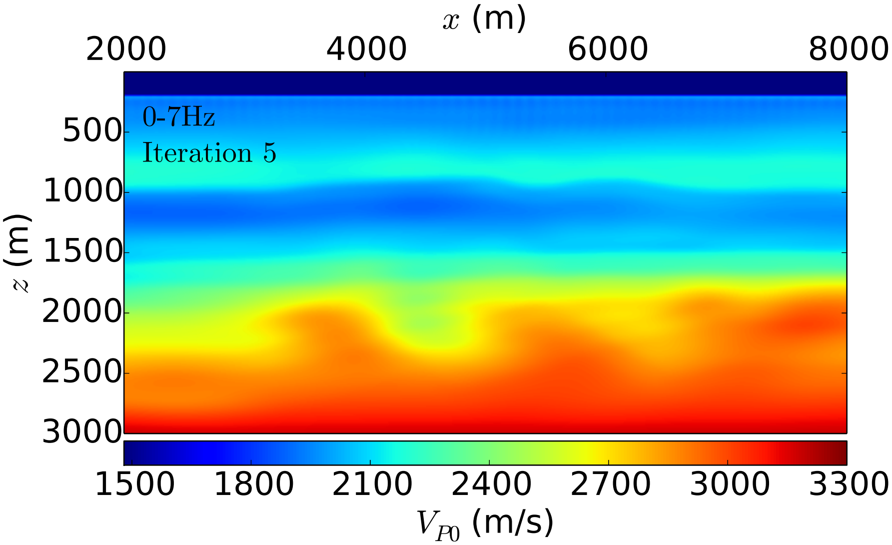
True model


x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

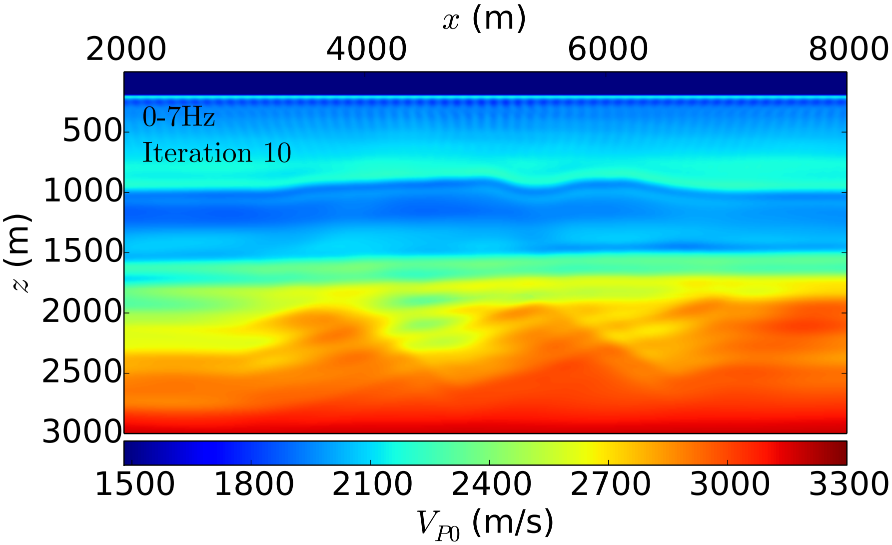
Starting model



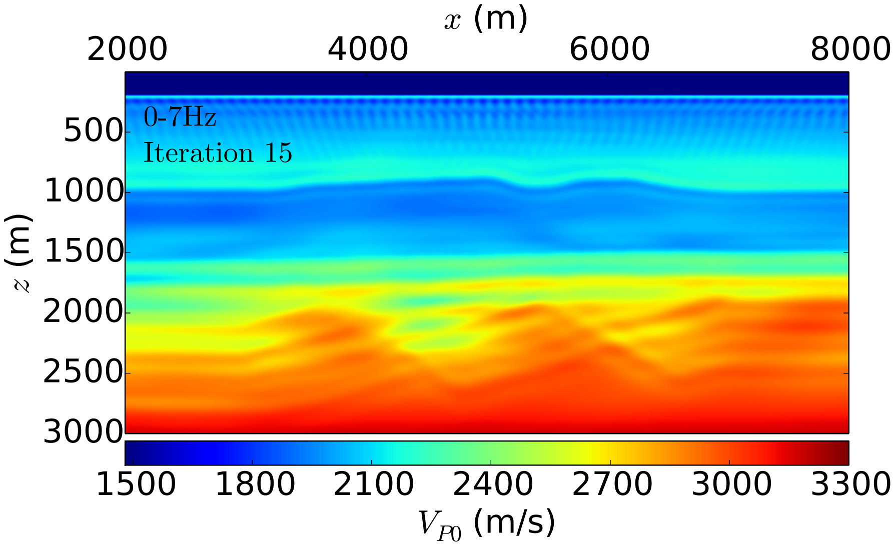
Results


x (m)
2000
4000
6000
8000
500
1000
1500
2000
2500
3000
z (m)
0-7Hz
Iteration 5
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

Results

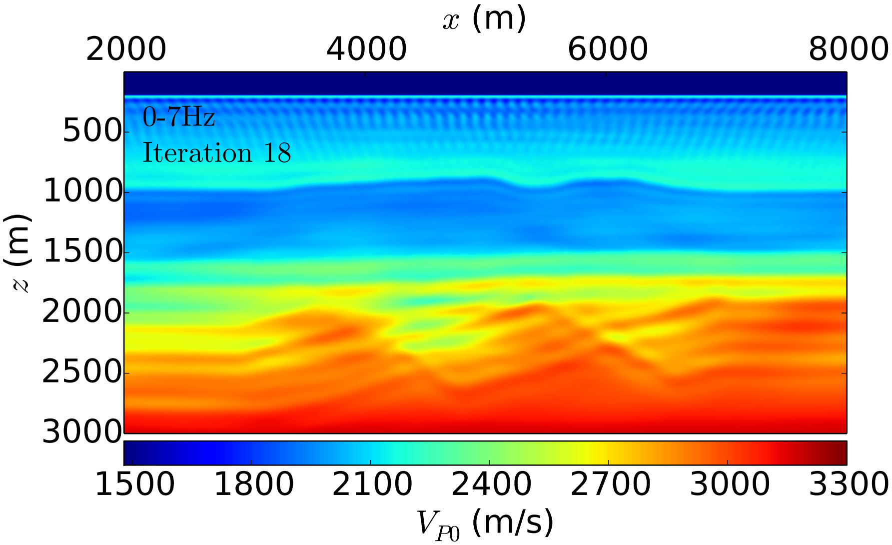

x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
0-7Hz
Iteration 10
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

Results



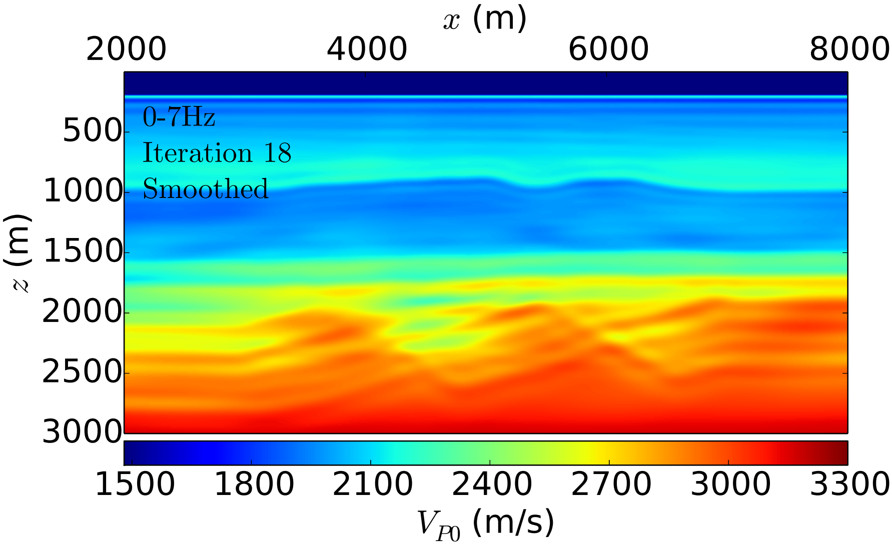
x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
0-7Hz
Iteration 15
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

Results

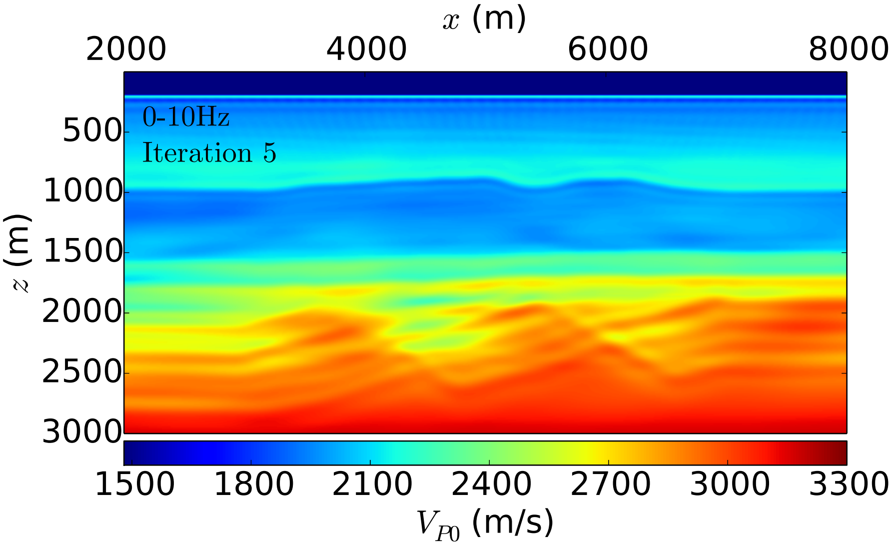


x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
0-7Hz
Iteration 18
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

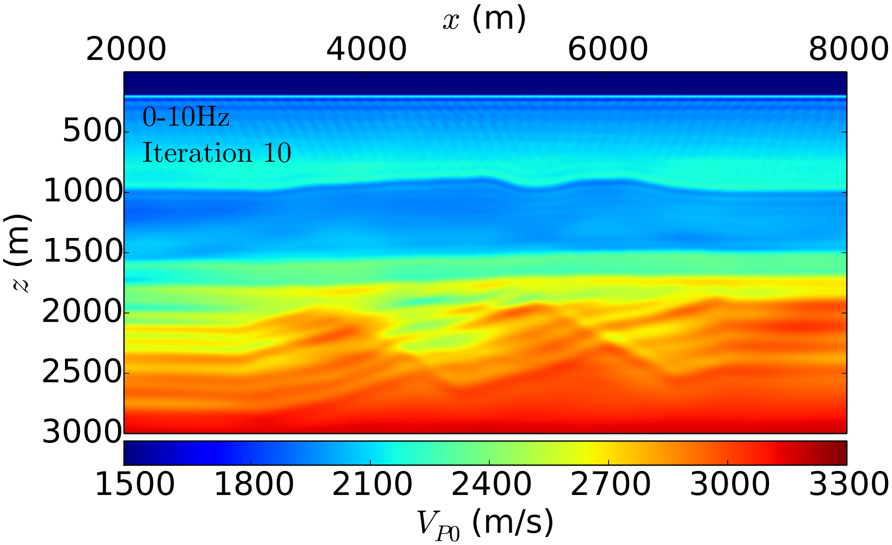
Results


x (m)
2000
4000
6000
8000
0-7Hz
Iteration 18
Smoothed
z (m)
500
1000
1500
2000
2500
3000
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

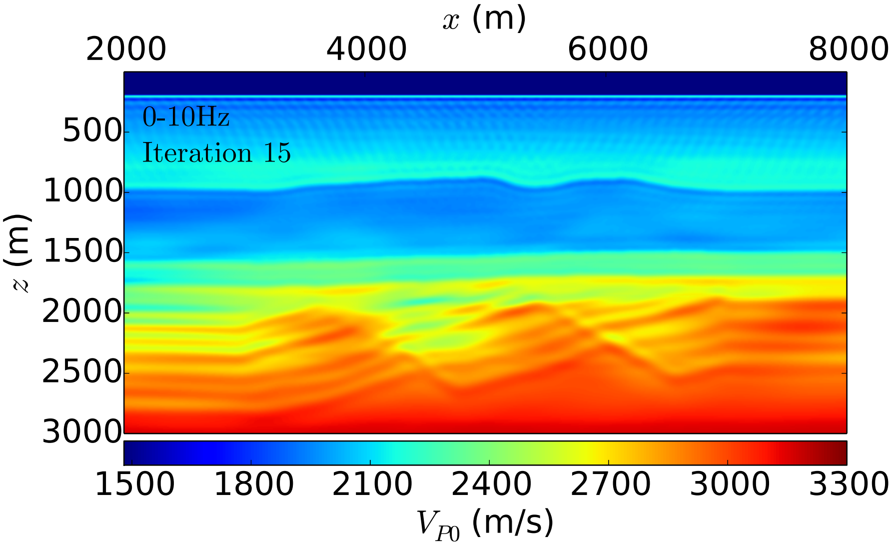
Results


x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
0-10Hz
Iteration 5
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

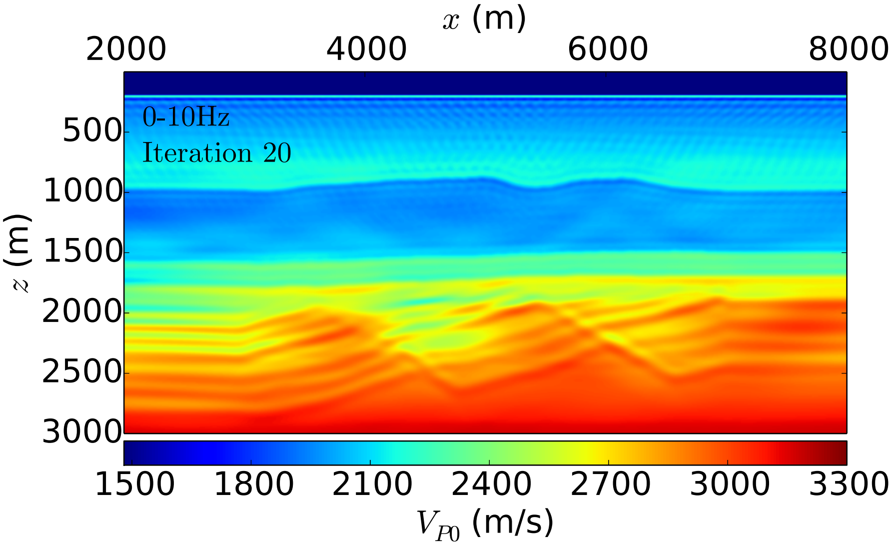
Results


x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
0-10Hz
Iteration 10
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

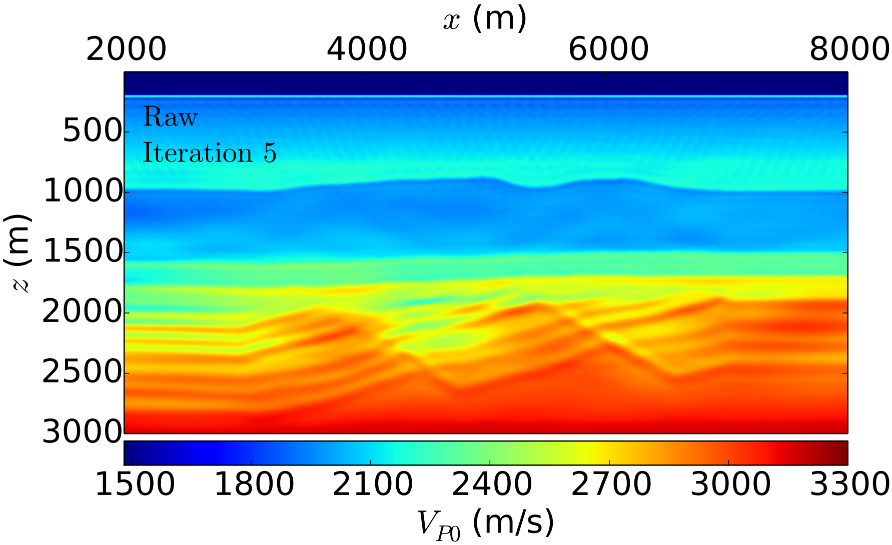
Results


x (m)
2000
4000
6000
8000
0-10Hz
Iteration 15
500
1000
1500
2000
2500
3000
z (m)
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

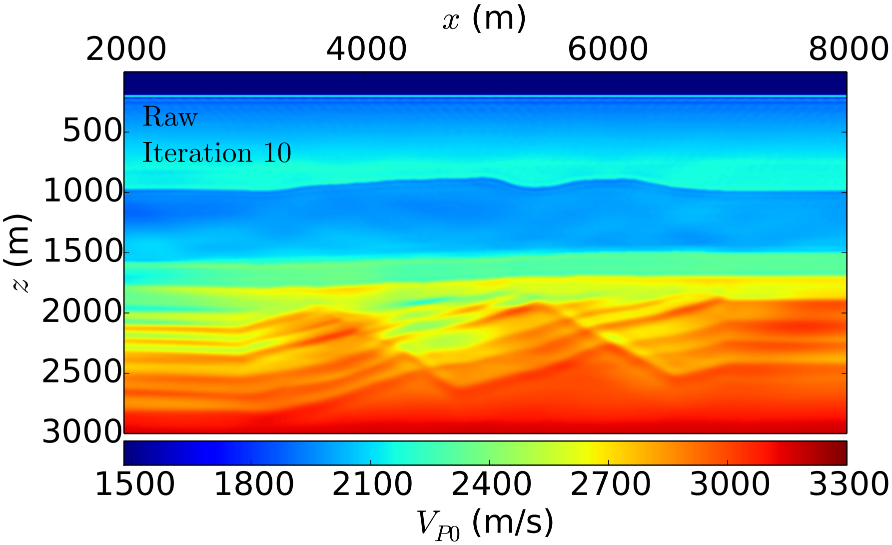
Results


x (m)
2000
4000
6000
8000
0-10Hz
Iteration 20
z (m)
500
1000
1500
2000
2500
3000
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

Results

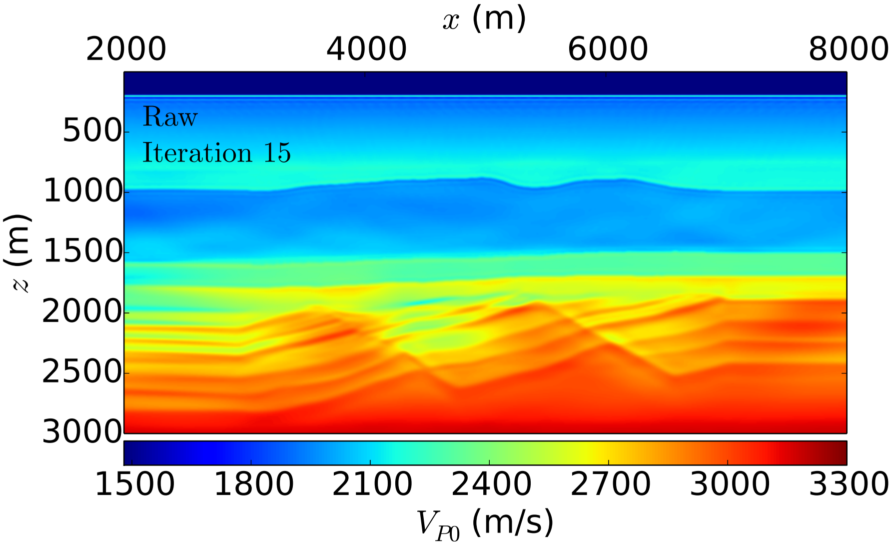

x (m)
2000
4000
6000
8000
Raw
Iteration 5
500
1000
1500
2000
2500
3000
z (m)
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

Results

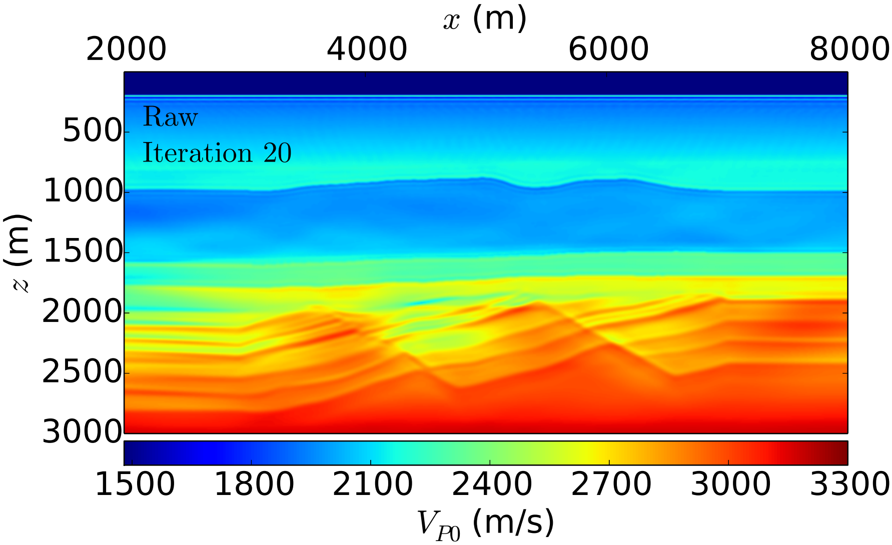


x (m)
2000
4000
6000
8000
500
1000
1500
2000
2500
3000
z (m)
Raw
Iteration 10
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

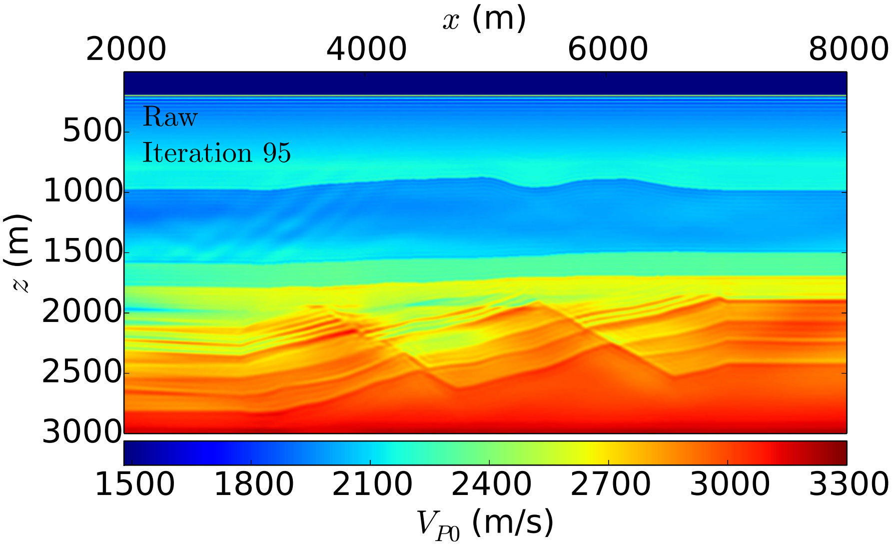
Results


x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
Raw
Iteration 15
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

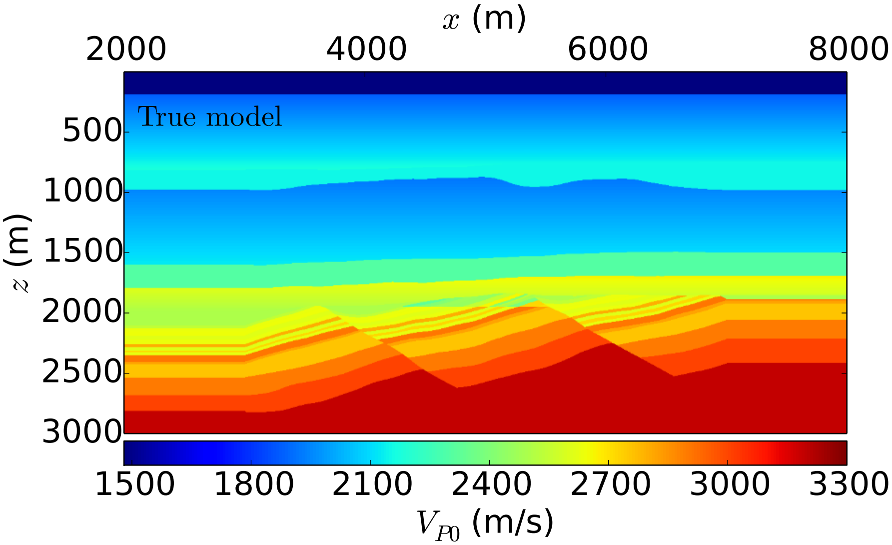
Results


x (m)
2000
4000
6000
8000
500
1000
1500
2000
2500
3000
z (m)
Raw
Iteration 20
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

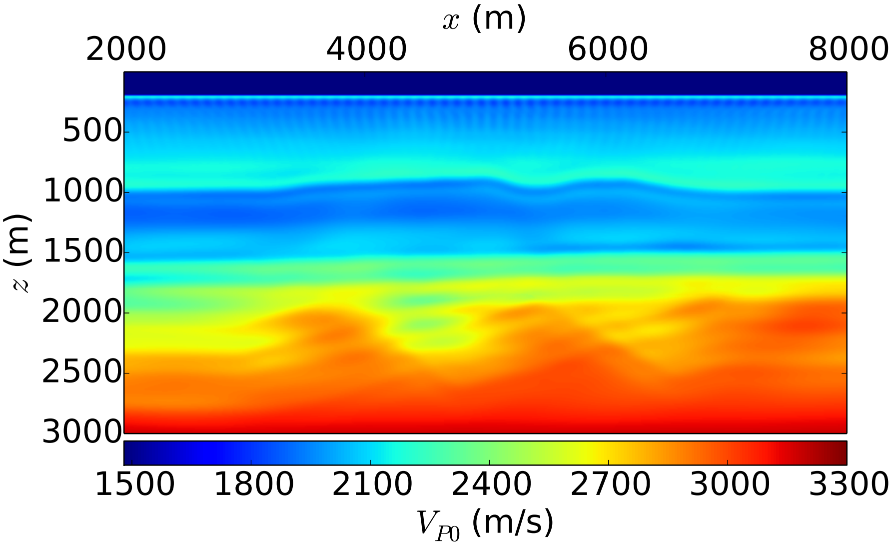
Results


x (m)
2000
4000
6000
8000
Raw
Iteration 95
z (m)
500
1000
1500
2000
2500
3000
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

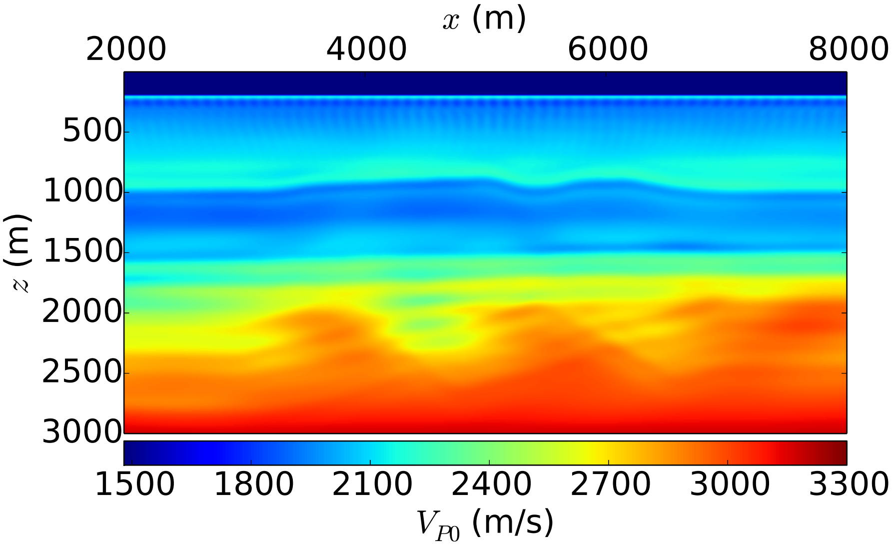
Results


x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
True model
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

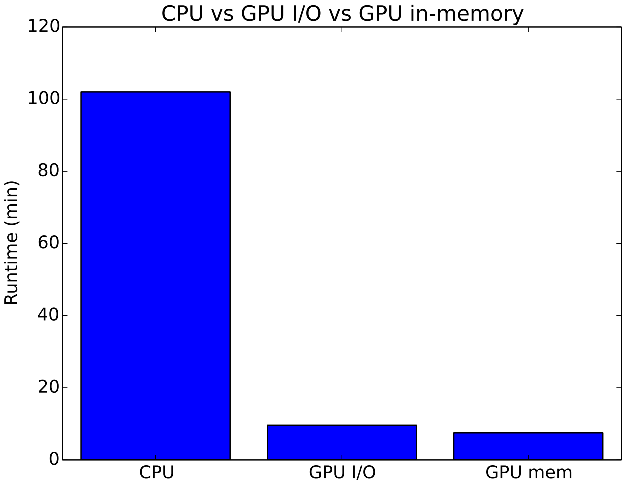
GPU


x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

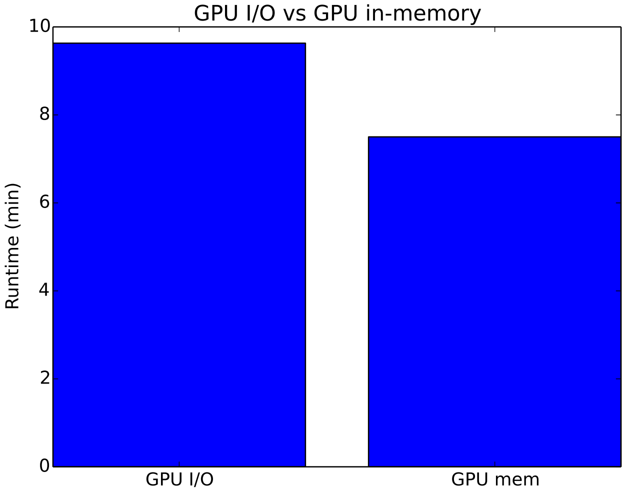
CPU


x (m)
2000
4000
6000
8000
z (m)
500
1000
1500
2000
2500
3000
1500
1800
2100
2400
2700
3000
3300
V_{P0} (m/s)

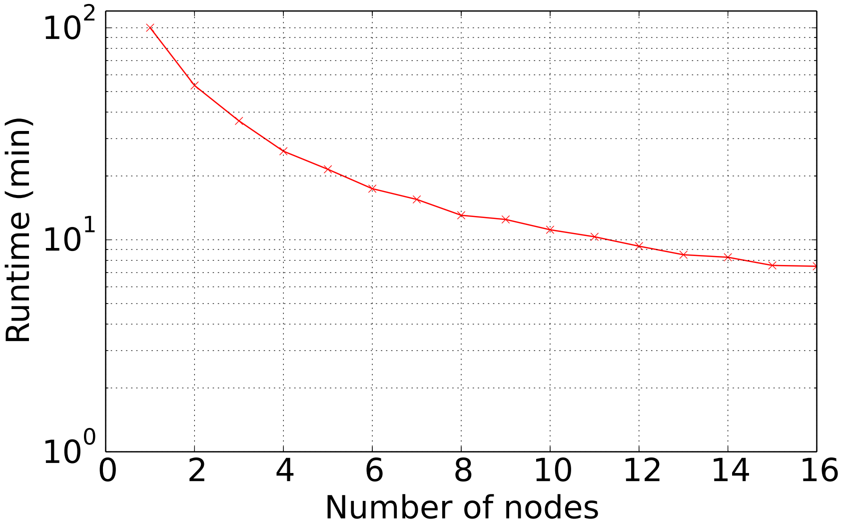
FWI runtimes



FWI runtimes



FWI runtimes



Conclusions

- Achieved approximately 75 times speedup of single source modeling.
- Achieved approximately 12 times speedup of FWI.
- Eliminated all temporary writing to disk.
- In-memory GPU code is fastest, but not by a huge margin.
- Going from CPU to GPU is by far the biggest improvement.

Acknowledgments

We thank the ROSE consortium and their sponsors for support.