Acoustic Full-Waveform Inversion in Julia on Multi-xPUs

Giacomo Aloisi, Andrea Zunino, Christian Boehm, Andreas Fichtner

PASC 23, June 28th 2023

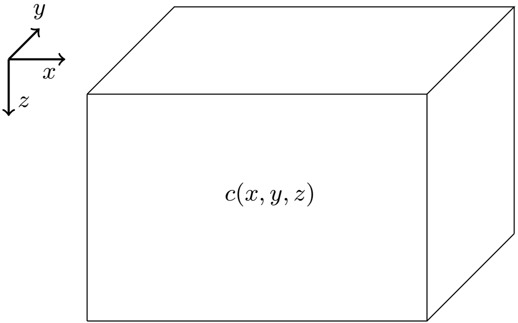
Outline

1. Introduction to Full-Waveform Inversion
2. Theory and implementation
3. Numerical experiments and benchmarks
4. Conclusions and future work

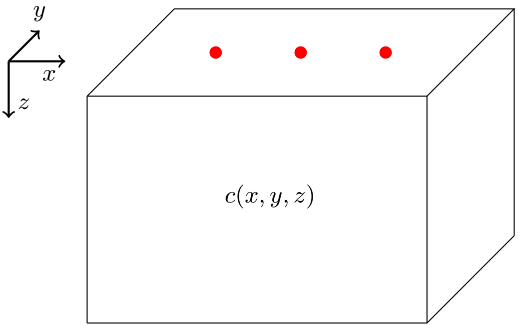
Outline

1. Introduction to Full-Waveform Inversion
2. Theory and implementation
3. Numerical experiments and benchmarks
4. Conclusions and future work

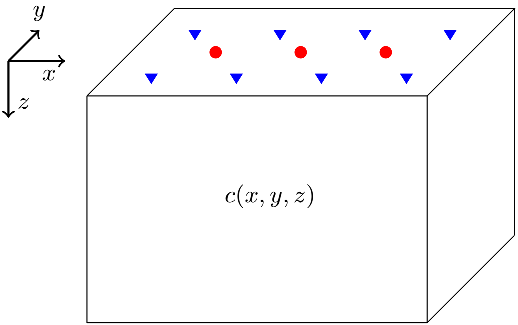
An introduction to (acoustic) Full-Waveform Inversion (FWI)



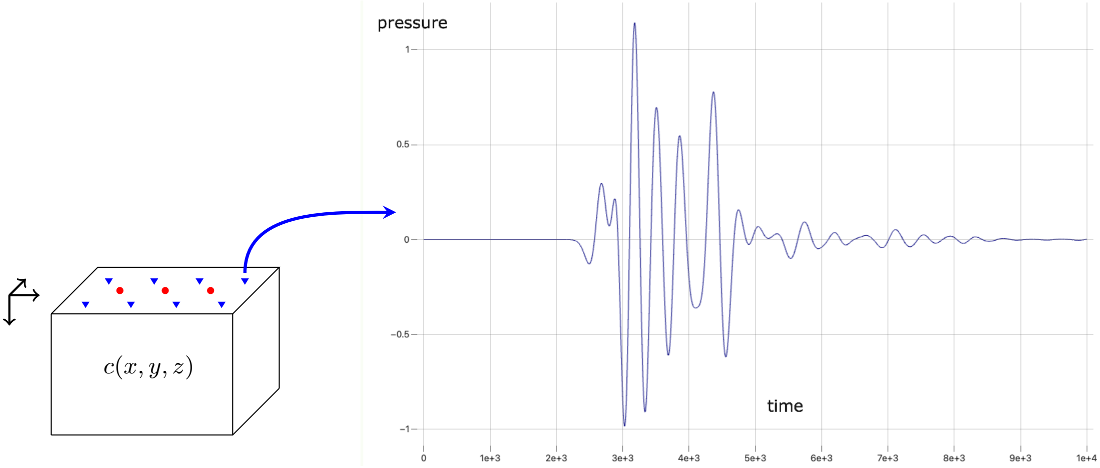
An introduction to (acoustic) Full-Waveform Inversion (FWI)



An introduction to (acoustic) Full-Waveform Inversion (FWI)



An introduction to (acoustic) Full-Waveform Inversion (FWI)



FWI applications: seismic tomography

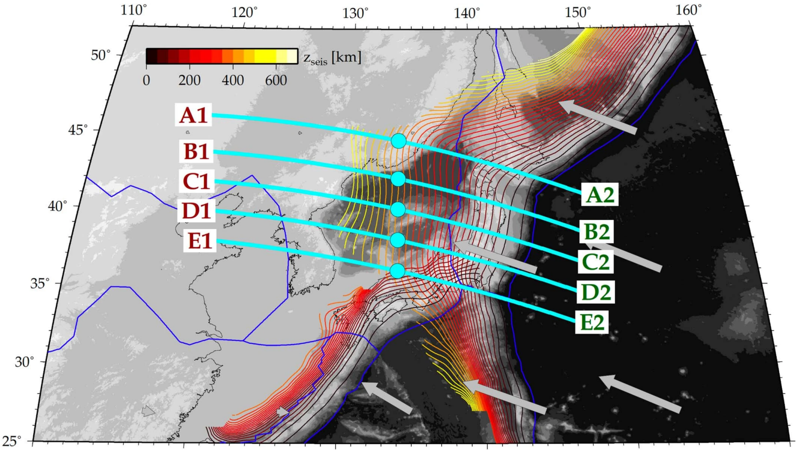


Figure from Gerya et al., 2021: Dynamic slab segmentation due to brittle–ductile damage in the outer rise
Data from Hayes et al., 2018: Slab2, a comprehensive subduction zone geometry model

FWI applications: seismic tomography

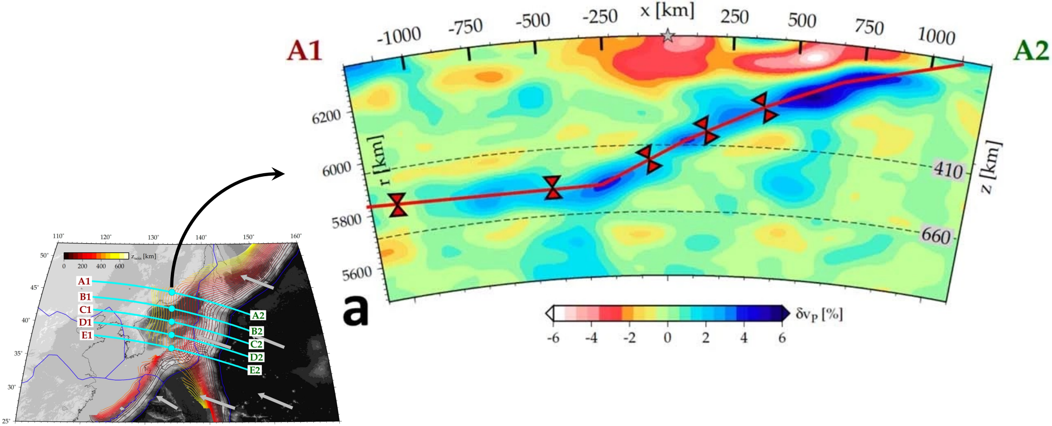
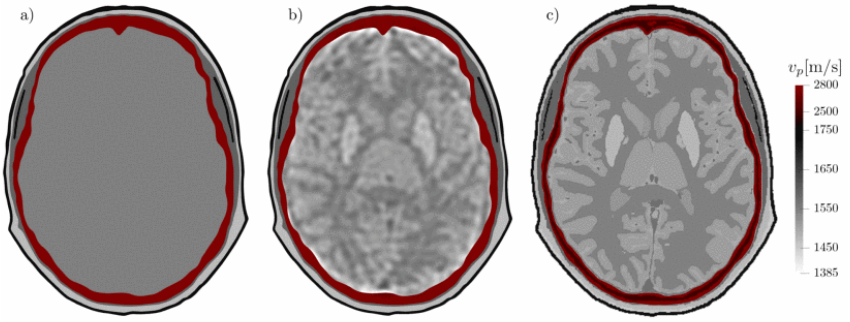


Figure from Gerya et al., 2021: Dynamic slab segmentation due to brittle–ductile damage in the outer rise
Data from Tao et al., 2018: Seismic Structure of the Upper Mantle Beneath Eastern Asia From Full Waveform Seismic Tomography

FWI applications: medical ultrasound tomography



initial model   reconstruction   target model

Figure from Marty et al., 2022: Elastic Full-Waveform Inversion for Transcranial Ultrasound Computed Tomography using Optimal Transport

Outline

1. Introduction to Full-Waveform Inversion
2. Theory and implementation
3. Numerical experiments and benchmarks
4. Conclusions and future work

Solving the forward problem: acoustic wave equation

Acoustic wave PDE with homogeneous Dirichlet BDCs

$\Omega \subset \mathbb{R}^n, t \in [0,T], p = p(\boldsymbol{x},t) : \mathbb{R}^n \times [0,T] \to \mathbb{R}, s = s(\boldsymbol{x},t) : \mathbb{R}^n \times [0,T] \to \mathbb{R}, c = c(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$

$$\begin{aligned} \frac{1}{c(\boldsymbol{x})^2}\frac{\partial^2 p(\boldsymbol{x},t)}{\partial t^2} &= \nabla_x^2\, p(\boldsymbol{x},t) + s(\boldsymbol{x},t) && , \text{ in } \overline{\Omega} \times [0,T], \\ p(\boldsymbol{x},t) &= 0 && , \text{ in } \partial\Omega \times [0,T] \end{aligned}$$

Solving the forward problem: acoustic wave equation

Acoustic wave PDE with C-PML BDCs

Let $\tilde{\partial}_i\Omega$ be and extension of $\partial\Omega$ in the direction i.

$$
\begin{aligned}
\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} &= \nabla_x^2\, p + s && \text{, in } \overline{\Omega} \times [0, T], \\
\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} &= \nabla_x^2\, p + \frac{\partial \psi_i}{\partial i} + \xi_i && \text{, in } \tilde{\partial}_i\Omega \times [0, T]
\end{aligned}
$$

Solving the forward problem: acoustic wave equation

2D acoustic wave discretization with central FD (2nd order in space and time)

$$
\begin{aligned}
p_{x,y}^{t+1} &= 2p_{x,y}^{t} - p_{x,y}^{t-1} \\
&\quad + c_{x,y}^2 \Delta t^2 \left(\frac{p_{x+1,y}^{t} - 2p_{x,y}^{t} + p_{x-1,y}^{t}}{\Delta x^2} \right) \\
&\quad + c_{x,y}^2 \Delta t^2 \left(\frac{p_{x,y+1}^{t} - 2p_{x,y}^{t} + p_{x,y-1}^{t}}{\Delta y^2} \right) \\
&\quad + c_{x,y}^2 \Delta t^2 s_{x,y}^{t} \qquad\qquad , \forall t \in [0, T], (x, y) \in \overline{\Omega}
\end{aligned}
$$

Misfit functional and minimization problem

$$\chi = \chi[p(c)] = \frac{1}{2}\sum_r (\boldsymbol{p}_r - \boldsymbol{p}_r^{\mathsf{obs}})\ \boldsymbol{C}_r^{-1}\ (\boldsymbol{p}_r - \boldsymbol{p}_r^{\mathsf{obs}})$$

Misfit functional and minimization problem

$$\chi = \chi[p(c)] = \frac{1}{2} \sum_r (\boldsymbol{p}_r - \boldsymbol{p}_r^{\textsf{obs}}) \; \boldsymbol{C}_r^{-1} \; (\boldsymbol{p}_r - \boldsymbol{p}_r^{\textsf{obs}})$$

$$\frac{\partial \chi}{\partial c_i}$$

Misfit functional and minimization problem

$$\chi = \chi[p(c)] = \frac{1}{2}\sum_r (\boldsymbol{p}_r - \boldsymbol{p}_r^{\textsf{obs}})\ \boldsymbol{C}_r^{-1}\ (\boldsymbol{p}_r - \boldsymbol{p}_r^{\textsf{obs}})$$

$$\frac{\partial \chi}{\partial c_i} \approx \frac{\chi[p(c_1,\ldots,c_i+\Delta c_i,\ldots,c_n)] - \chi[p(c_1,\ldots,c_i,\ldots,c_n)]}{\Delta c_i}$$

Misfit functional and minimization problem

$$\chi = \chi[p(c)] = \frac{1}{2} \sum_r (\boldsymbol{p}_r - \boldsymbol{p}_r^{\text{obs}}) \; \boldsymbol{C}_r^{-1} \; (\boldsymbol{p}_r - \boldsymbol{p}_r^{\text{obs}})$$

$$\frac{\partial \chi}{\partial c_i} \approx \frac{\chi[p(c_1, \ldots, c_i + \Delta c_i, \ldots, c_n)] - \chi[p(c_1, \ldots, c_i, \ldots, c_n)]}{\Delta c_i}$$

Time to compute χ on a 2000x2000 grid for 1000 time steps on 1 GPU $\approx$ 0.4 seconds

Misfit functional and minimization problem

$$\chi = \chi[p(c)] = \frac{1}{2}\sum_r (\boldsymbol{p}_r - \boldsymbol{p}_r^{\text{obs}})\ \boldsymbol{C}_r^{-1}\ (\boldsymbol{p}_r - \boldsymbol{p}_r^{\text{obs}})$$

$$\frac{\partial \chi}{\partial c_i} \approx \frac{\chi[p(c_1, \ldots, c_i + \Delta c_i, \ldots, c_n)] - \chi[p(c_1, \ldots, c_i, \ldots, c_n)]}{\Delta c_i}$$

Time to compute χ on a 2000x2000 grid for 1000 time steps on 1 GPU $\approx$ 0.4 seconds
Time to compute gradient $\rightarrow$ $16 * 10^5$ seconds $\approx$ 444 hours! NOT feasible!

Efficiently computing model parameters gradients: adjoint equation

$$G(p, c) := \nabla_x^2 \, p + s - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0$$

Efficiently computing model parameters gradients: adjoint equation

$$G(p, c) := \nabla_x^2\, p + s - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0$$

$$\mathcal{L} := \chi + \int_\Omega \int_0^T \lambda G(p, c)\; dt\; dx$$

Efficiently computing model parameters gradients: adjoint equation

$$
\begin{aligned}
G(p, c) &:= \nabla_x^2\, p + s - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0 \\
\mathcal{L} &:= \chi + \int_\Omega \int_0^T \lambda G(p, c)\ dt\ dx \\
\frac{\partial \mathcal{L}}{\partial p} &= 0
\end{aligned}
$$

Efficiently computing model parameters gradients: adjoint equation

$$
\begin{aligned}
G(p,c) &:= \nabla_x^2\, p + s - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0 \\
\mathcal{L} &:= \chi + \int_\Omega \int_0^T \lambda G(p,c)\; dt\; dx \\
\frac{\partial \mathcal{L}}{\partial p} &= 0 \xRightarrow{i.b.p.}
\end{aligned}
$$

Efficiently computing model parameters gradients: adjoint equation

$$G(p,c) := \nabla_x^2\, p + s - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0$$

$$\mathcal{L} := \chi + \int_\Omega \int_0^T \lambda G(p,c)\; dt\; dx$$

$$\frac{\partial \mathcal{L}}{\partial p} = 0 \xRightarrow{i.b.p.} \boxed{\frac{1}{c^2}\frac{\partial^2 \lambda}{\partial t^2} = \nabla_x^2\, \lambda + \tilde{s}}$$

Efficiently computing model parameters gradients: adjoint equation

$$G(p,c) := \nabla_x^2\, p + s - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0$$

$$\mathcal{L} := \chi + \int_\Omega \int_0^T \lambda G(p,c)\; dt\; dx$$

$$\frac{\partial \mathcal{L}}{\partial p} = 0 \xRightarrow{i.b.p.} \boxed{\frac{1}{c^2}\frac{\partial^2 \lambda}{\partial t^2} = \nabla_x^2\, \lambda + \tilde{s}} \quad , \text{ in } \Omega \times [T, 0]$$

Efficiently computing model parameters gradients: adjoint equation

$$G(p, c) := \nabla_x^2 \, p + s - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0$$

$$\mathcal{L} := \chi + \int_\Omega \int_0^T \lambda G(p, c) \; dt \; dx$$

$$\frac{\partial \mathcal{L}}{\partial p} = 0 \overset{i.b.p.}{\Longrightarrow} \boxed{\frac{1}{c^2}\frac{\partial^2 \lambda}{\partial t^2} = \nabla_x^2 \, \lambda + \tilde{s}} \quad \text{, in } \Omega \times [T, 0]$$

$$\tilde{\boldsymbol{s}}_r = \frac{\partial \chi}{\partial p} = \boldsymbol{C}_r^{-1} \, (\boldsymbol{p}_r - \boldsymbol{p}_r^{\text{obs}}) \quad , \forall r$$

Efficiently computing model parameters gradients: adjoint equation

$$G(p, c) := \nabla_r^2\, p + s - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0$$

$$\mathcal{L}$$

$$\frac{\partial \mathcal{L}}{\partial c_i} \overset{G \overset{!}{=} 0}{\overset{\downarrow}{=}} \frac{\partial \chi}{\partial c_i} = \frac{2}{c_i^3}\int_0^T \lambda \frac{\partial^2 p}{\partial t^2} dt$$

$$\frac{\partial \mathcal{L}}{\partial p} \qquad \times [T, 0]$$

$$\tilde{\boldsymbol{s}}_r = \frac{\partial \chi}{\partial p} = \boldsymbol{C}_r^{-1}\left(\boldsymbol{p}_r - \boldsymbol{p}_r^{\text{obs}}\right) \qquad , \forall r$$

Efficiently computing model parameters gradients: adjoint equation

Time to gradient $\approx$ 1.8 seconds!

$$G(p, c) := \nabla_r^2 \, p + s - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = 0$$

$$\mathcal{L}$$

$$\frac{\partial \mathcal{L}}{\partial c_i} \overset{G \overset{!}{=} 0}{\downarrow \atop =} \frac{\partial \chi}{\partial c_i} = \frac{2}{c_i^3} \int_0^T \lambda \frac{\partial^2 p}{\partial t^2} dt$$

$$\frac{\partial \mathcal{L}}{\partial p} \qquad \times [T, 0]$$

$$\tilde{\boldsymbol{s}}_r = \frac{\partial \chi}{\partial p} = \boldsymbol{C}_r^{-1} \left(\boldsymbol{p}_r - \boldsymbol{p}_r^{\text{obs}}\right) \qquad , \forall r$$

Acoustic FWI recipe

1. Choose initial model $c = c_0$

Acoustic FWI recipe

1. Choose initial model $c = c_0$
2. Solve acoustic wave equation to get pressure field p

Acoustic FWI recipe

1. Choose initial model $c = c_0$
2. Solve acoustic wave equation to get pressure field p
3. Compute $\chi(p)$ and adjoint source $\tilde{s}$

Acoustic FWI recipe

1. Choose initial model $c = c_0$
2. Solve acoustic wave equation to get pressure field p
3. Compute $\chi(p)$ and adjoint source $\tilde{s}$
4. Solve adjoint equation to get adjoint field λ

Acoustic FWI recipe

1. Choose initial model $c = c_0$
2. Solve acoustic wave equation to get pressure field p
3. Compute $\chi(p)$ and adjoint source $\tilde{s}$
4. Solve adjoint equation to get adjoint field λ
5. Compute $\nabla_c \chi$ while solving adjoint equation

Acoustic FWI recipe

1. Choose initial model $c = c_0$
2. Solve acoustic wave equation to get pressure field p
3. Compute $\chi(p)$ and adjoint source $\tilde{s}$
4. Solve adjoint equation to get adjoint field λ
5. Compute $\nabla_c \chi$ while solving adjoint equation
6. Update model c using $\chi(p)$ and $\nabla_c \chi$ with an optimization algorithm (GD, L-BFGS, etc...) and go back to step 2. until convergence

HMCLab.jl and SeismicWaves.jl

HMCLab

https://hmclab.science, https://ptsolvers.github.io/GPU4GEO

HMCLab.jl and SeismicWaves.jl

HMCLab

EikonalSolvers.jl

https://hmclab.science, https://ptsolvers.github.io/GPU4GEO

`HMCLab.jl` and `SeismicWaves.jl`

HMCLab

`EikonalSolvers.jl`

`MCsamplers.jl`

https://hmclab.science, https://ptsolvers.github.io/GPU4GEO

`HMCLab.jl` and `SeismicWaves.jl`

HMCLab

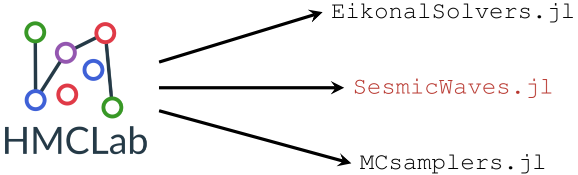
`EikonalSolvers.jl`

`SesmicWaves.jl`

`MCsamplers.jl`

https://hmclab.science, https://ptsolvers.github.io/GPU4GEO

HMCLab.jl and SeismicWaves.jl



and more...

https://hmclab.science, https://ptsolvers.github.io/GPU4GEO

`HMCLab.jl` and `SeismicWaves.jl`

HMCLab

→ `EikonalSolvers.jl`

→ `SesmicWaves.jl`

→ `MCsamplers.jl`

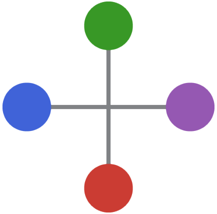
and more...

https://hmclab.science, https://ptsolvers.github.io/GPU4GEO

Code: single xPU 2D kernel w/ `ParallelStencil.jl`

```julia
@parallel_indices (i, j) function update_p(
    pold, pcur, pnew, halo, c, dt, dx, dy
)
    # pressure derivatives in space
    d2p_dx2 = (pcur[i+1, j] - 2.0 * pcur[i, j] + pcur[i-1, j]) / (dx^2)
    d2p_dy2 = (pcur[i, j+1] - 2.0 * pcur[i, j] + pcur[i, j-1]) / (dy^2)
    # update pressure
    pnew[i, j] = 2.0 * pcur[i, j] - pold[i, j] + c[i, j]^2 * dt^2 * (d2p_dx2 + d2p_dy2)

    return nothing
end
```

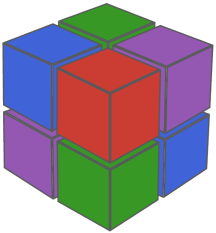


```julia
...
for it = 1:nt
    # update pressure
    @parallel (2:(nx-1), 2:(ny-1)) update_p(pold, pcur, pnew, halo, c, dt, dx, dy)
    # inject sources
    @parallel (1:nsrcs) inject_sources(pnew, dt2srctf, possrcs, it)
    # record receivers
    @parallel (1:nrecs) record_receivers(pnew, traces, posrecs, it)
    # swap pointers
    pold, pcur, pnew = pcur, pnew, pold
end
...
```

Omlin S. (CSCS), Räss L. (ETH) [https://github.com/omlins/ParallelStencil.jl]

Code: multi-xPU 2D kernel
w/ `ParallelStencil.jl` + `ImplicitGlobalGrid.jl`

```
...
for it = 1:nt
    @hide_communication b_width begin
        # update pressure
        @parallel (2:(nx-1), 2:(ny-1)) update_p(pold, pcur, pnew, halo, c, dt, dx, dy)
        # inject sources
        @parallel (1:nsrcs) inject_sources(pnew, dt2srctf, possrcs, it)
        # record receivers
        @parallel (1:nrecs) record_receivers(pnew, traces, posrecs, it)
        # exchange new pressure with other nodes
        update_halo(pnew)
    end
    # swap pointers
    pold, pcur, pnew = pcur, pnew, pold
end
...
```



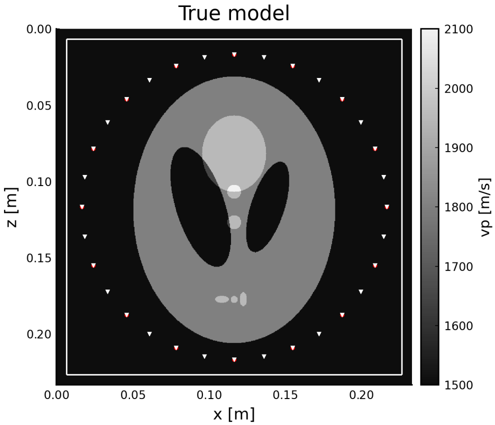
Omlin S. (CSCS), Räss L. (ETH) [https://github.com/eth-cscs/ImplicitGlobalGrid.jl]

Outline

1. Introduction to Full-Waveform Inversion
2. Theory and implementation
3. Numerical experiments and benchmarks
4. Conclusions and future work

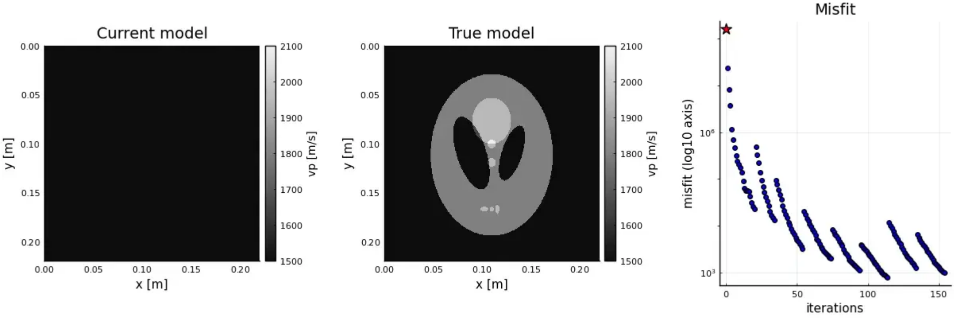
Numerical experiments: Shepp-Logan phantom inversion

Inversion setup
Sources: 16
Receivers: 32
Ricker wavelet at various frequencies
Model size: 701x701
C-PML layers: 20
Timesteps forward: 12000
Optim algo: L-BFGS

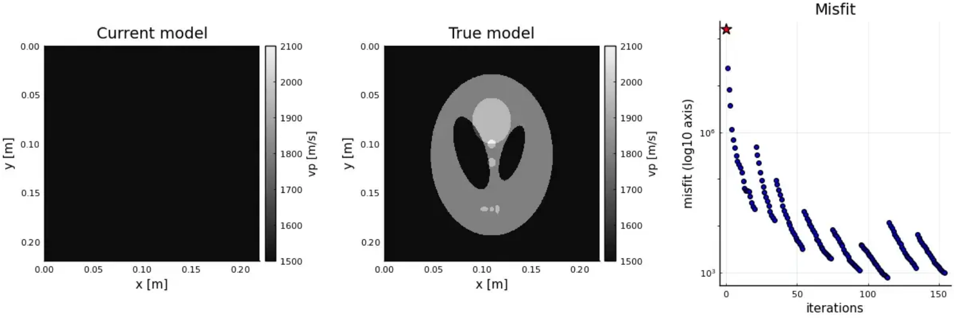


Shepp L., Logan B., 1974: The Fourier Reconstruction of a Head Section

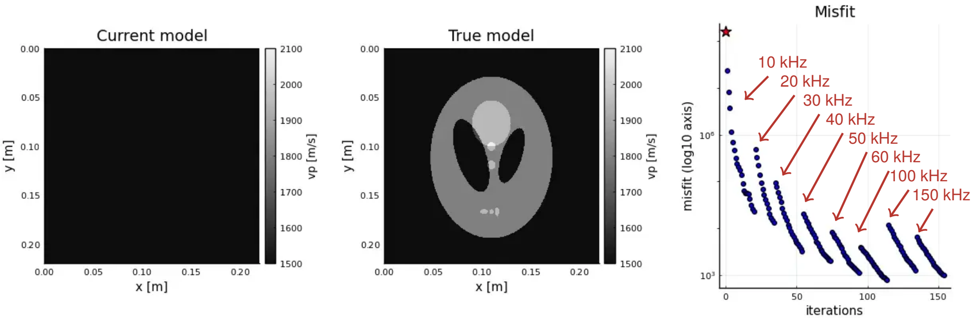
Numerical experiments: Shepp-Logan phantom inversion



Numerical experiments: Shepp-Logan phantom inversion

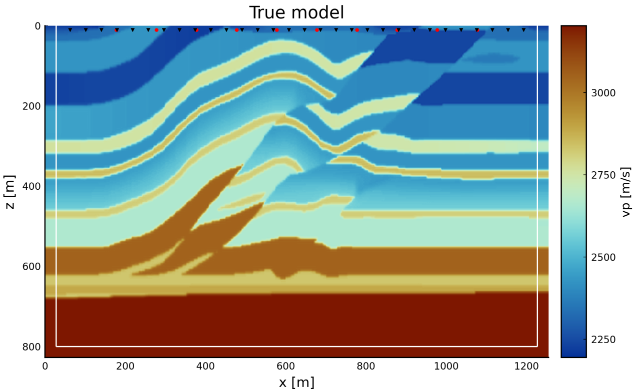


Numerical experiments: Shepp-Logan phantom inversion



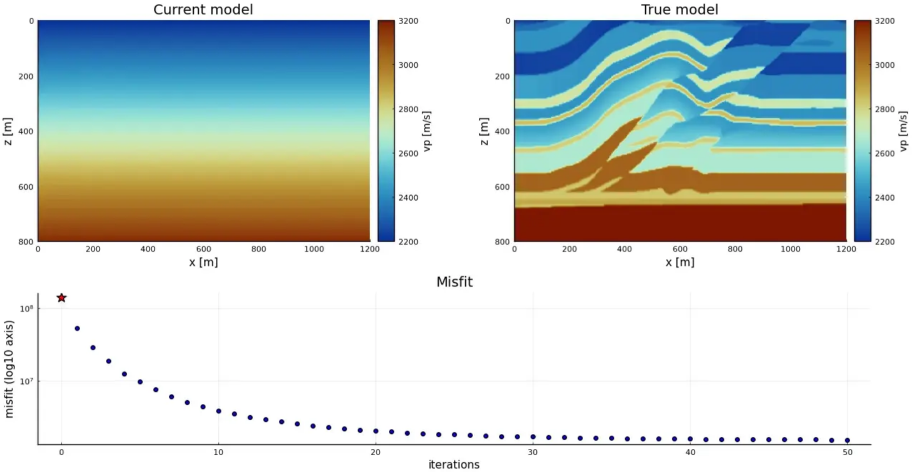
Numerical experiments: Overthrust model inversion (with correlated source noise)

Inversion setup
Sources: 10
Receivers: 30
Ricker wavelet at 12Hz
Model size: 896x594
C-PML layers: 20
Free surface BDC at top
Timesteps forward: 2500
Optim algo: L-BFGS

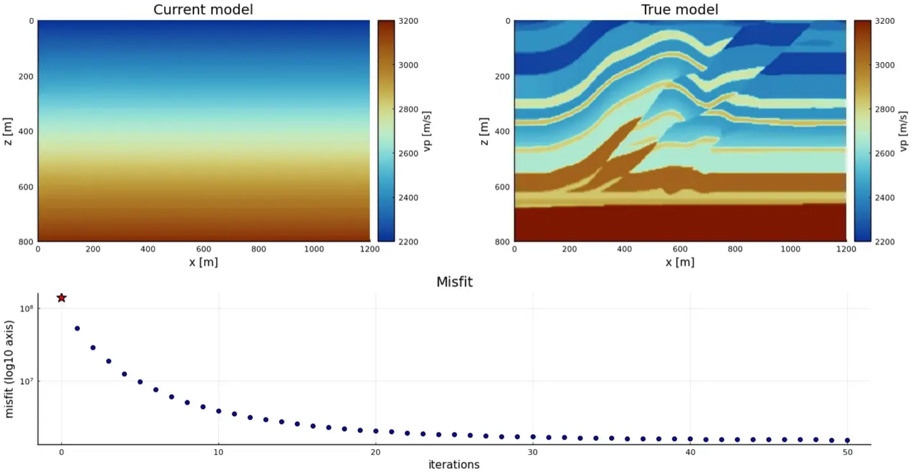


SEG/EAGE Salt and Overthrust Models [https://wiki.seg.org/wiki/SEG/EAGE_Salt_and_Overthrust_Models]

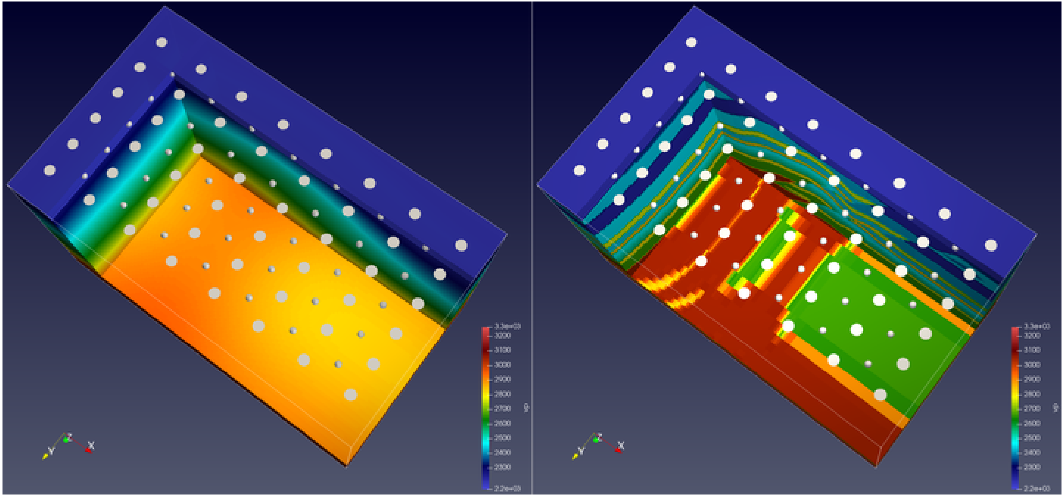
Numerical experiments: Overthrust model inversion (with correlated source noise)



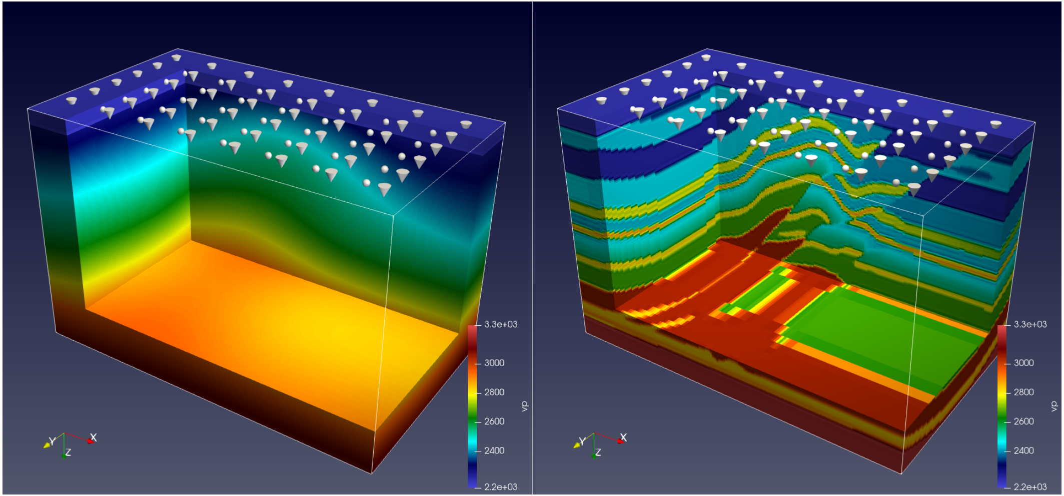
Numerical experiments: Overthrust model inversion (with correlated source noise)



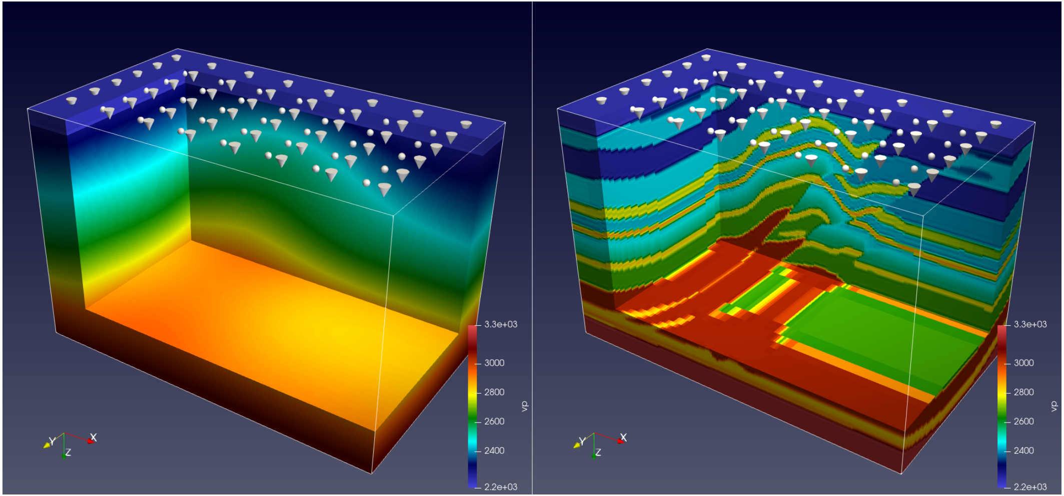
Numerical experiments: Overthrust model inversion in 3D



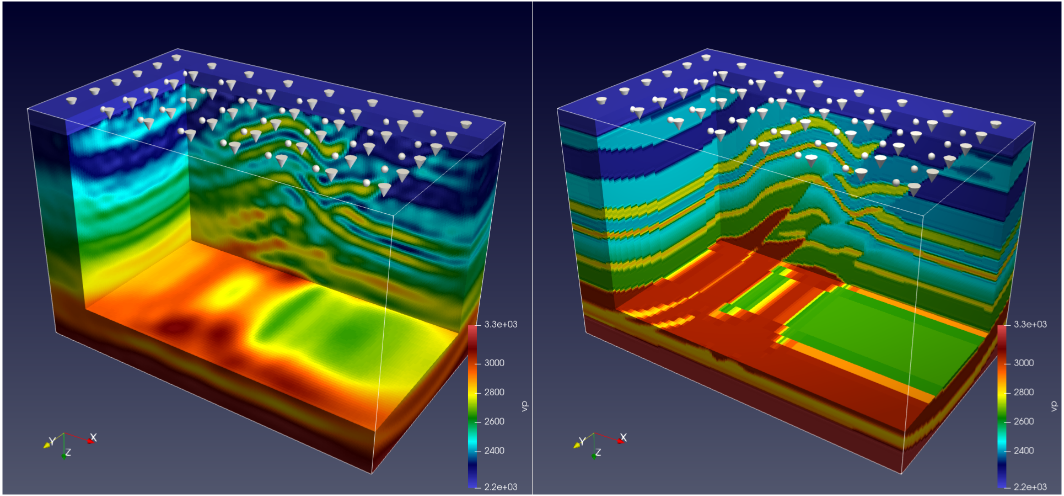
Numerical experiments: Overthrust model inversion in 3D



Numerical experiments: Overthrust model inversion in 3D



Numerical experiments: Overthrust model inversion in 3D



Benchmarks: Performance metrics

- FLOP/s-like metrics NOT adequate!
  - computation is memory bounded

Benchmarks: Performance metrics

- FLOP/s-like metrics NOT adequate!
  - computation is memory bounded
- Effective memory access A_{eff} [byte]
  - strictly needed memory loads + stores

Benchmarks: Performance metrics

- FLOP/s-like metrics NOT adequate!
  - computation is memory bounded
- Effective memory access A_{eff} [byte]
  - strictly needed memory loads + stores
  - example: acoustic 1D pressure update with $n = 10^7$ grid points (FP 64)

Benchmarks: Performance metrics

- FLOP/s-like metrics NOT adequate!
  - computation is memory bounded
- Effective memory access A_{eff} [byte]
  - strictly needed memory loads + stores
  - example: acoustic 1D pressure update with $n = 10^7$ grid points (FP 64)
    - $A_{\text{eff}} = (n + 3n) * 8 = 320$ MB

Benchmarks: Performance metrics

- FLOP/s-like metrics NOT adequate!
  - computation is memory bounded
- Effective memory access A_{eff} [byte]
  - strictly needed memory loads + stores
  - example: acoustic 1D pressure update with $n = 10^7$ grid points (FP 64)
    - $A_{\text{eff}} = (n + 3n) * 8 = 320$ MB
- Effective memory throughput: $T_{\text{eff}} = A_{\text{eff}}/t$ [byte/sec]

Benchmarks: Performance metrics

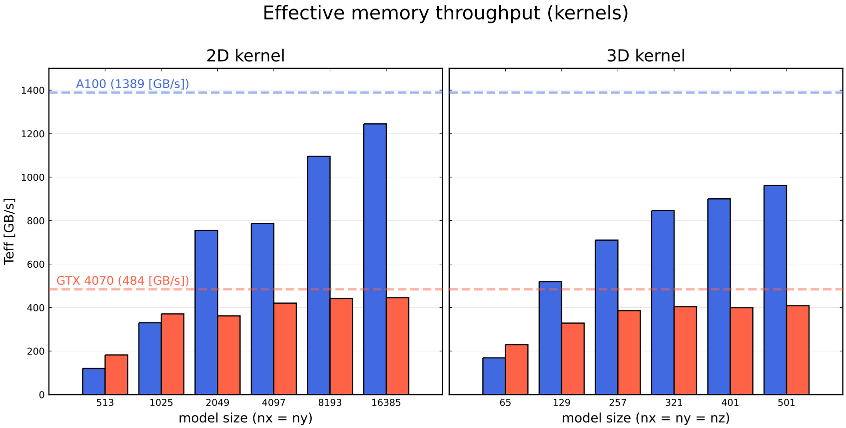
- FLOP/s-like metrics NOT adequate!
  - computation is memory bounded
- Effective memory access A_{eff} [byte]
  - strictly needed memory loads + stores
  - example: acoustic 1D pressure update with $n = 10^7$ grid points (FP 64)
    - $A_{\text{eff}} = (n + 3n) * 8 = 320$ MB
- Effective memory throughput: $T_{\text{eff}} = A_{\text{eff}}/t$ [byte/sec]
  - same example as before, suppose $t = 10^{-3}$ s

Benchmarks: Performance metrics

- FLOP/s-like metrics NOT adequate!
  - computation is memory bounded
- Effective memory access A_{eff} [byte]
  - strictly needed memory loads + stores
  - example: acoustic 1D pressure update with $n = 10^7$ grid points (FP 64)
    - $A_{\text{eff}} = (n + 3n) * 8 = 320$ MB
- Effective memory throughput: $T_{\text{eff}} = A_{\text{eff}}/t$ [byte/sec]
  - same example as before, suppose $t = 10^{-3}$ s
    - $T_{\text{eff}} = A_{\text{eff}}/t = 320$ GB/s

Benchmarks: kernels performance

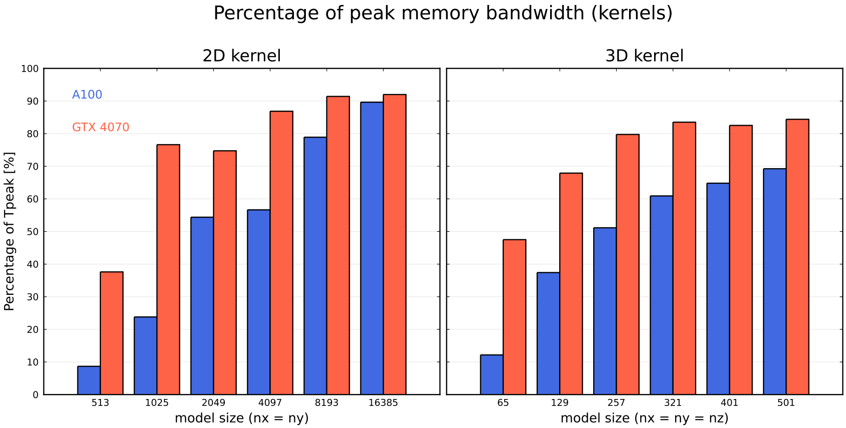
Benchmarking setup
GPUs Nvidia GTX 4070 & A100
julia version 1.8.5
flags: -O3 –check-bounds=no
CUDA version:
12.1 (for GTX 4070)
11.4 (for A100)
Peak performances measured with GPU-STREAM
20 C-PML layers in each boundary



Repeated measurements until +-5% of median execution time is within the 99% non-parametric CI

Benchmarks: kernels performance

Benchmarking setup
GPUs Nvidia GTX 4070 & A100
julia version 1.8.5
flags: -O3 –check-bounds=no
CUDA version:
12.1 (for GTX 4070)
11.4 (for A100)
Peak performances measured with GPU-STREAM
20 C-PML layers in each boundary

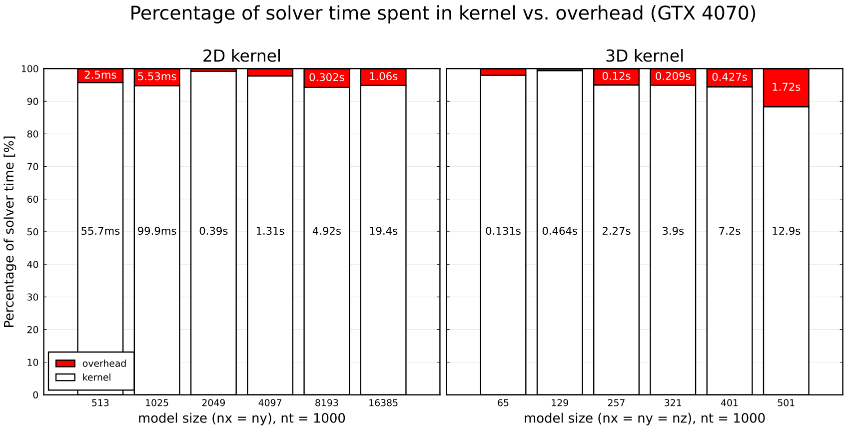


Repeated measurements until +-5% of median execution time is within the 99% non-parametric CI

Benchmarks: forward solver execution times

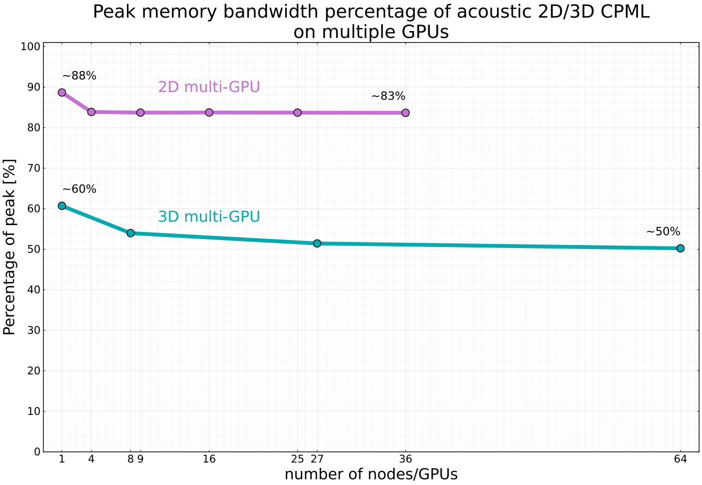
Benchmarking setup
GPU Nvidia GTX 4070
julia version 1.8.5
flags: -O3 –check-bounds=no
CUDA version: 12.1
Peak performances measured with GPU-STREAM
20 C-PML layers in each boundary

Repeated measurements until +-5% of median execution time is within the 99% non-parametric CI



Benchmarks: (preliminary) multi-GPU weak scaling

Benchmarking setup
GPUs Tesla P100 (on Piz Daint)
julia version 1.7.1
flags: -O3 –check-bounds=no
Peak performances measured with GPU-STREAM
20 C-PML layers in each boundary



Measured average time per iteration by running multiple iterations (skip first 200 iterations for 2D, skip first 19 iterations for 3D)

Outline

1. Introduction to Full-Waveform Inversion
2. Theory and implementation
3. Numerical experiments and benchmarks
4. Conclusions and future work

Conclusions

- What we have done:

Conclusions

- What we have done:
  - Efficient (and scalable) FD acoustic wave equation forward and adjoint solver...

Conclusions

- What we have done:
  - Efficient (and scalable) FD acoustic wave equation forward and adjoint solver...
  - multi-platform and portable...

Conclusions

- What we have done:
  - Efficient (and scalable) FD acoustic wave equation forward and adjoint solver...
  - multi-platform and portable...
  - using a high level language like Julia...

Conclusions

- What we have done:
  - Efficient (and scalable) FD acoustic wave equation forward and adjoint solver...
  - multi-platform and portable...
  - using a high level language like Julia...
  - with minimal HPC knowledge...

Conclusions

- What we have done:
  - Efficient (and scalable) FD acoustic wave equation forward and adjoint solver...
  - multi-platform and portable...
  - using a high level language like Julia...
  - with minimal HPC knowledge...
  - open source! (soon$^{\text{TM}}$)

Conclusions

- What we have done:
  - Efficient (and scalable) FD acoustic wave equation forward and adjoint solver...
  - multi-platform and portable...
  - using a high level language like Julia...
  - with minimal HPC knowledge...
  - open source! (soon$^{\text{TM}}$)
- Still WIP:

Conclusions

- What we have done:
  - Efficient (and scalable) FD acoustic wave equation forward and adjoint solver...
  - multi-platform and portable...
  - using a high level language like Julia...
  - with minimal HPC knowledge...
  - open source! (soon^{TM})
- Still WIP:
  - Elastic solvers

Conclusions

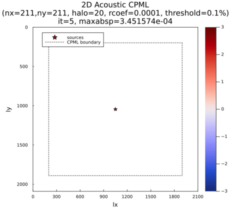
- What we have done:
  - Efficient (and scalable) FD acoustic wave equation forward and adjoint solver...
  - multi-platform and portable...
  - using a high level language like Julia...
  - with minimal HPC knowledge...
  - open source! (soon$^{\text{TM}}$)
- Still WIP:
  - Elastic solvers
  - Higher order FD stencils

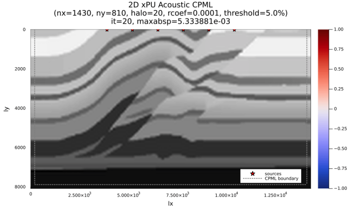
Conclusions

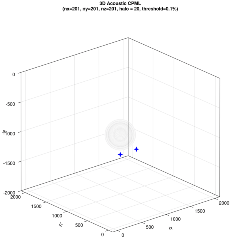
- What we have done:
  - Efficient (and scalable) FD acoustic wave equation forward and adjoint solver...
  - multi-platform and portable...
  - using a high level language like Julia...
  - with minimal HPC knowledge...
  - open source! (soon$^{\mathrm{TM}}$)
- Still WIP:
  - Elastic solvers
  - Higher order FD stencils
  - Fully fledged multi-xPU implementations

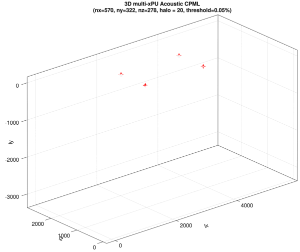
Thanks for your attention!

Giacomo Aloisi
[galoisi@student.ethz.ch]

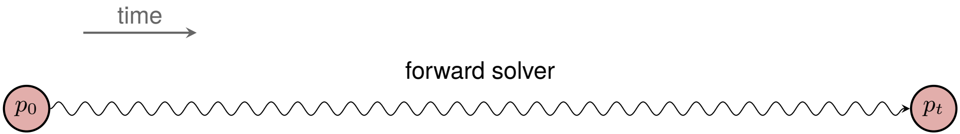




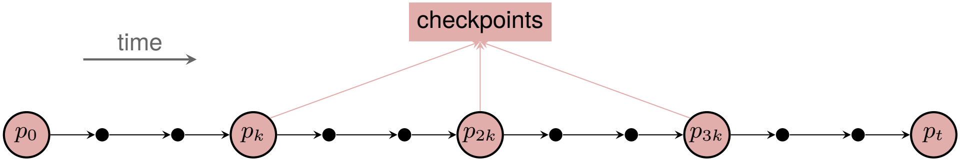




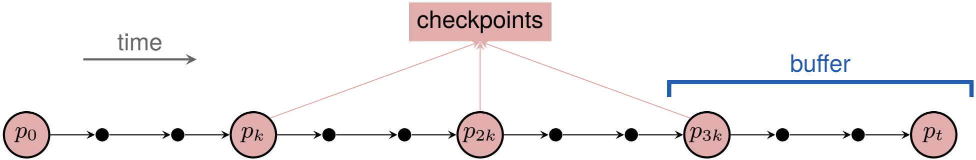
Efficiently computing model parameters gradients: checkpointing



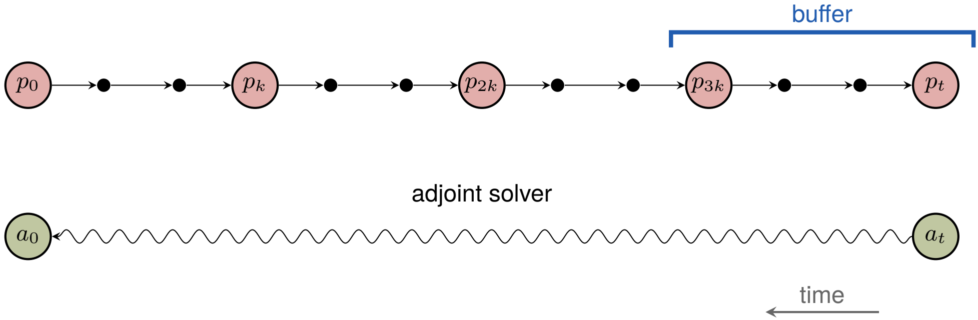
Efficiently computing model parameters gradients: checkpointing



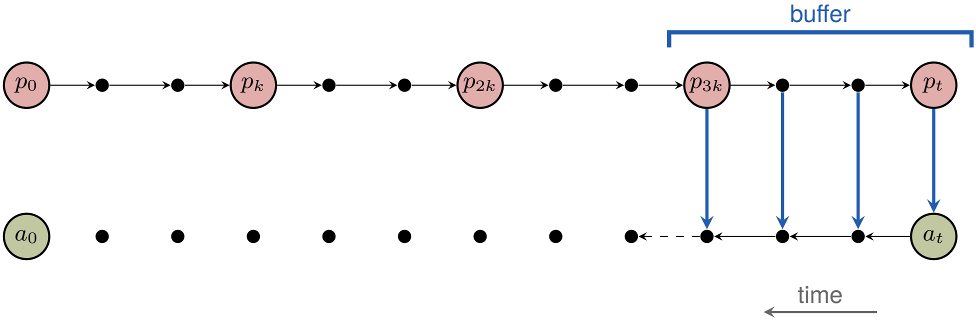
Efficiently computing model parameters gradients: checkpointing



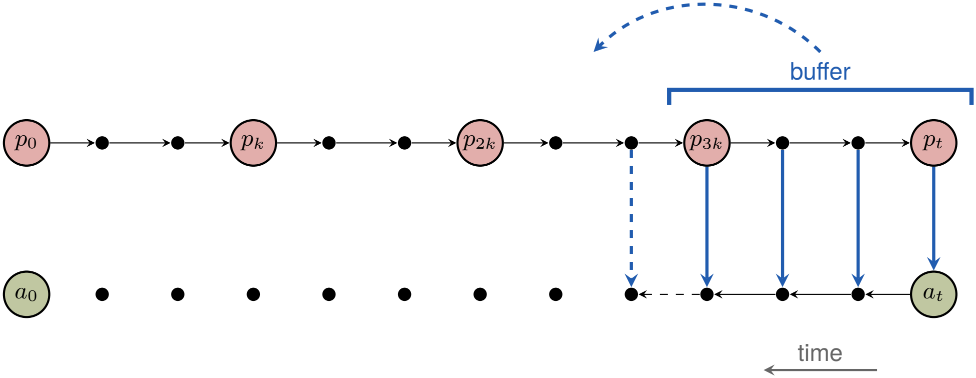
Efficiently computing model parameters gradients: checkpointing



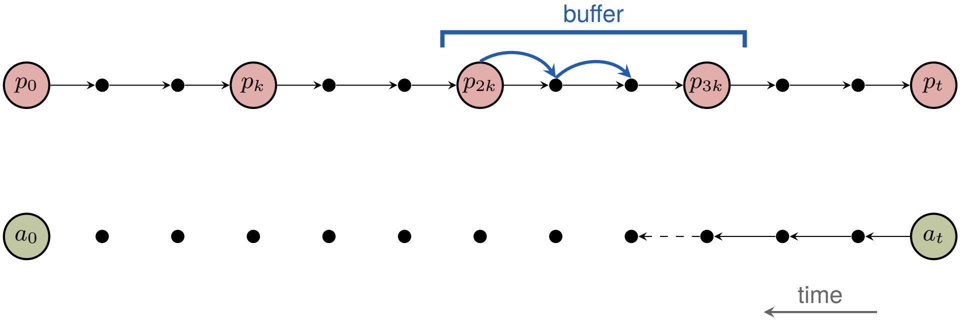
Efficiently computing model parameters gradients: checkpointing



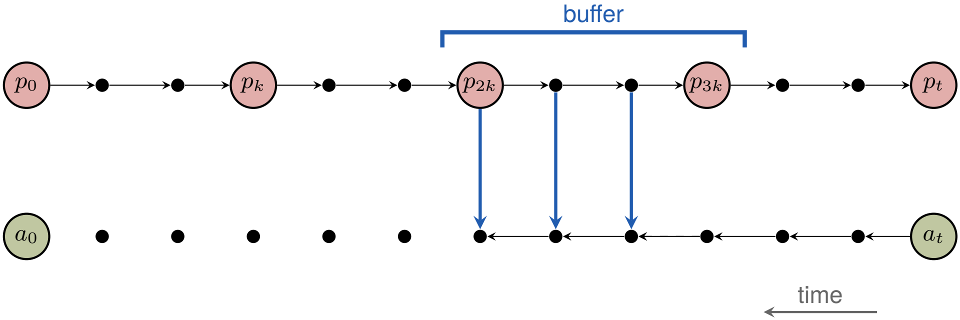
Efficiently computing model parameters gradients: checkpointing



Efficiently computing model parameters gradients: checkpointing



Efficiently computing model parameters gradients: checkpointing



Efficiently computing model parameters gradients: checkpointing