# Lossy Checkpoint Compression in Full Waveform Inversion: a case study with ZFPv0.5.5 and the Overthrust Model

Navjot Kukreja[1], Jan Hückelheim[2], Mathias Louboutin[3], John Washbourne[4], Paul H.J. Kelly[5], and Gerard J. Gorman[6]

[1]Department of Computer Science, University of Liverpool
[2]Argonne National Laboratory
[3]Georgia Institute of Technology
[4]Chevron Corporation
[5]Department of Computing, Imperial College London
[6]Department of Earth Science and Engineering, Imperial College London

**Correspondence:** Navjot Kukreja (n.kukreja@liverpool.ac.uk)

**Abstract.** This paper proposes a new method that combines checkpointing methods with error-controlled lossy compression for large-scale high-performance Full-Waveform Inversion (FWI), an inverse problem commonly used in geophysical exploration. This combination can significantly reduce data movement, allowing a reduction in run time as well as peak memory.

In the Exascale computing era, frequent data transfer (e.g., memory bandwidth, PCIe bandwidth for GPUs, or network) is the performance bottleneck rather than the peak FLOPS of the processing unit.

Like many other adjoint-based optimization problems, FWI is costly in terms of the number of floating-point operations, large memory footprint during backpropagation, and data transfer overheads. Past work for adjoint methods has developed checkpointing methods that reduce the peak memory requirements during backpropagation at the cost of additional floating-point computations.

Combining this traditional checkpointing with error-controlled lossy compression, we explore the three-way tradeoff between memory, precision, and time to solution. We investigate how approximation errors introduced by lossy compression of the forward solution impact the objective function gradient and final inverted solution. Empirical results from these numerical experiments indicate that high lossy-compression rates (compression factors ranging up to 100) have a relatively minor impact on convergence rates and the quality of the final solution.

**Keywords.** Lossy compression, Full waveform inversion, checkpointing, memory

## 1 Introduction

Full-waveform inversion (FWI) is an adjoint-based optimization problem used in seismic imaging to infer the earth's subsurface structure and physical parameters (Virieux and Operto, 2009). Here, a PDE describing the propagation of a wave is solved repeatedly, while modifying the estimate of the physical properties of the system so that a simulated signal matches the signal recorded in a physical experiment. The compute and memory requirements for this and similar PDE-constrained optimization problems can readily push the world's top supercomputers to their limits. Table 1 estimates the computational requirements of an FWI problem on the SEAM Model (Fehler and Keliher, 2011). Although the grid-spacing and timestep interval depends on various problem-specific factors, we can do a back-of-the-envelope calculation to appreciate the scale of FWI. To estimate the number of operations per grid point, we use the acoustic anisotropic equation in a Tilted-Transverse Isotropic medium (Zhang et al., 2011), which is commonly used today in commercial FWI. Such a problem would require almost 90 days of continuous execution at 1 PFLOP/s. The memory requirements for this problem are also prohibitively high. As can be seen in Table 1, the gradient computation step is responsible for this problem's high memory requirement, and the focus of this paper is to reduce that requirement.

The FWI algorithm is explained in more detail in Section 2. It is important to note that despite the similar terminology, the checkpointing we refer to in this paper is not done for resilience or failure recovery. This is the checkpointing from automatic-differentiation theory, with the objective of reducing the memory footprint of a large computation by trading recomputation for storage. This checkpointing occurs completely in memory (i.e. RAM), and the discussion in this paper is orthogonal to any issues around Disk Input/Output.

| **Description** | **Number** | **Peak Memory** | **No. of Flops** |
|---|---|---|---|
| Single grid point (TTI) | 1 | 8 bytes | 6300 |
| Complete grid | $1000 \times 1000 \times 1000$ | 8GB | $6.3 \times 10^{12}$ |
| Forward propagation (timesteps) | 10000 | 24GB | $6.3 \times 10^{16}$ |
| Gradient Computation | 2 (FW+REV)[1] | 80TB | $1.26 \times 10^{17}$ |
| Shots | 10000 | 80TB | $1.26 \times 10^{21}$ |
| Optim. Iterations | 20 | 80TB | $2.52 \times 10^{22}$ |

**Table 1.** Estimated computational requirements of a Full-Waveform Inversion problem based on the SEAM model (Fehler and Keliher, 2011), a large scale industry standard geophysical model that is used to benchmark FWI. Note that real-world FWI problems are likely to be larger. [1]A gradient computation involves a forward simulation followed by a reverse/adjoint computation. For simplicity we assume the same size of computation during the forward/adjoint pass.

### 1.1 FWI and other similar problems

FWI is similar to other inverse problems like brain-imaging (Guasch et al., 2020), shape optimization (Jameson et al., 1998), and even training a neural network. When training a neural network, the activations calculated when propagating forward along the network need to be stored in memory and used later during backpropagation. The size of the corresponding computation in a neural network depends on the depth of the network and, more importantly, the input size. We assume the input is an image for the purpose of this exposition. For typical input image sizes of less than $500 \times 500$ px, the computation per data point is relatively (to FWI) small, both in the number of operations and memory required. This is compensated by processing in *mini-batches*, where multiple data points are processed at the same time. This batch dimension's size is usually adjusted to *fill* *up* the target hardware to its capacity (and no more). This is the standard method of managing the memory requirements of a neural network training pipeline. However, for an input image that is large enough, or a network that is deep enough, it is seen that the input image, network weights, and network activations together require more memory than available on a single node, even for a single input image (*batchsize* $= 1$ ). We previously addressed this issue in the context of neural networks (Kukreja et al., 2019b). In this paper we address the same issue for FWI.

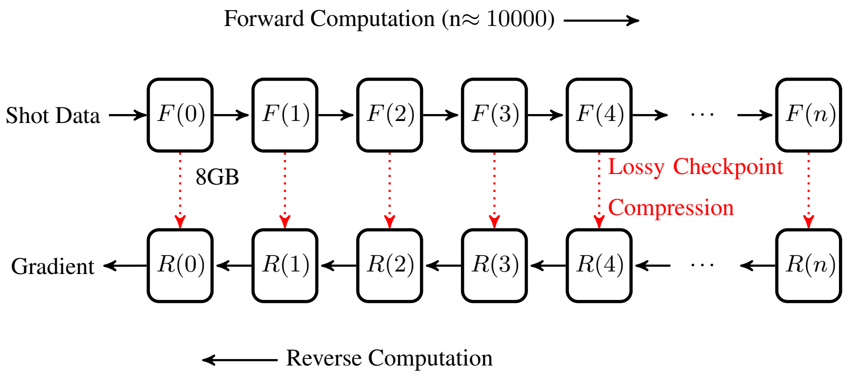


**Figure 1.** An illustration of the approach presented in this paper. Checkpoints are compressed using lossy compression to combine lossy compression and the checkpoint-recompute strategies.

Many algorithmic optimizations/approximations are commonly applied to reduce the computational load from the numbers calculated in Table 1. These optimizations could either reduce the number of operations or the amount of memory required. In this paper, we shall focus on the high memory footprint of this problem. One standard approach is to save the field at only the boundaries and reconstruct the rest of the field from the boundaries to reduce the memory footprint. However, the applicability of this method is limited to time-reversible PDEs. In this work, we use the isotropic acoustic equation as the example. Although this equation is time-reversible, many other variations used in practice are not. For this reason, we do not discuss this method in this paper.

A commonly used method to deal with the problem of this large memory footprint is domain-decomposition over MPI, where the computational domain is split into subdomains over multiple compute nodes to use their memory. The efficacy of this method depends on the ability to hide the MPI-communication overhead behind the computations within the subdomain. For effective communication-computation overlap, the subdomains should be big enough that the computations within the subdomain take at least as long as the MPI communication. This places a lower bound on subdomain size (and hence peak memory consumption per MPI rank) that is a function of the network interconnect - this lower bound might be too large for slow interconnects e.g. on cloud systems.

Some iterative frequency domain methods, e.g. Knibbe et al. (2014), can alleviate the memory limit but are not competitive with time-domain methods in the total time to solution.

Hybrid methods that combine time-domain methods, as well as frequency-domain methods, have also been tried (Witte et al., 2019). However, this approach can be challenging because the application code must decide the user's discrete set of frequencies to achieve a target accuracy.

In the following subsections, we discuss three techniques that are commonly used to alleviate this memory pressure - namely numerical approximations, checkpointing, and data compression. The common element in these techniques is that all three solve the problem of high memory requirement by increasing the operational intensity of the computation - doing more computations per byte transferred from memory. With the gap between memory and computational speeds growing wider as we move into the exaflop era, we expect to use such techniques to increase moving forward.

## 1.2 Approximate methods

There has been some recent work on alternate floating-point representations (Chatelain et al., 2019), although we are not aware of this technique being applied to FWI. Within FWI, many approximate methods exist, including *On-the-fly Fourier transforms* (Witte et al., 2019). However, it is not clear whether this method can provide fine-tuned bounds on the solution's accuracy. In contrast, other completely frequency-domain formulations can provide clearer bounds (van Leeuwen and Herrmann, 2014), but as previously discussed, this comes at the cost of a much higher computational complexity. In this paper, we restrict ourselves to time-domain approaches only.

Another approximation commonly applied to reduce the memory pressure in FWI in the time domain is subsampling. Here, the timestep-rate of the gradient computation (See equation 4) is decoupled from the timestep-rate of the adjoint wavefield computation, with one gradient timestep for every $n$ adjoint steps. This reduces the memory footprint by a factor of $n$, since only one-in-$n$ values of the forward wavefield need to be stored. The Nyquist theorem is commonly cited as the justification for this sort of subsampling. However, the Nyquist theorem only provides a lower bound on the error - it is unclear whether an upper bound on the error has been established on this method. Although more thorough empirical measurements of the errors induced in subsampling have been done before (Louboutin and Herrmann, 2015), we do a brief empirical study in Section 4.7 as a baseline to compare the error with our method.

## 1.3 Checkpointing

Instead of storing the wavefield at every timestep during the forward computation, it is possible to store it at a subset of the timesteps only. During the following computation that proceeds in a reverse order to calculate the gradient, if the forward wavefield is required at a timestep that was not stored, it can be recovered by restarting the forward computation from the last available timestep. This is commonly known as checkpointing. Algorithms have been developed to define the optimal checkpointing schedule involving forward, store, backward, load, and recompute events under different assumptions (Griewank and Walther, 2000; Wang et al., 2009; Aupy and Herrmann, 2017). This technique has also been applied to FWI-like computations (Symes, 2007).

In previous work, we introduced the open-source software pyRevolve, a Python module that can automatically manage the checkpointing strategies under different scenarios with minimal modification to the computations code (Kukreja et al., 2018).

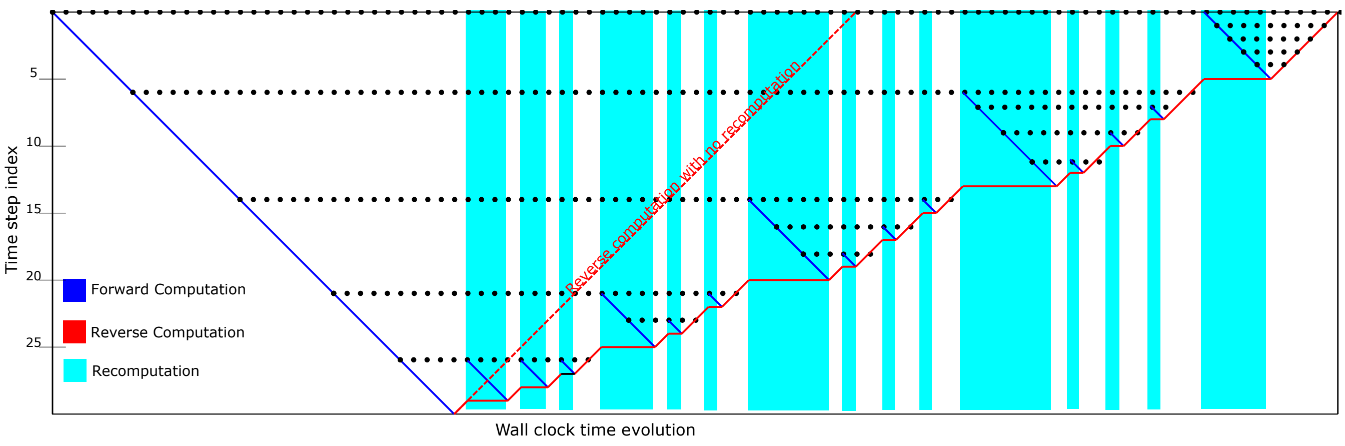


**Figure 2.** Schematic of the checkpointing strategy. Wall-clock time is on the horizontal axis, while the vertical axis represents simulation time. The blue line represents forward computation. The dotted red line represents how the reverse computation would have proceeded after the forward computation, had there been enough memory to store all the necessary checkpoints. Checkpoints are shown as the black dots. The reverse computation under the checkpointing strategy is shown as the solid red line. It can be seen that the reverse computation proceeds only where the results of the forward computation are available. When not available, the forward computation is restarted from the last available checkpoint to recompute the results of the forward - shown here as the blue shaded regions.

For this work, we extended pyRevolve to integrate lossy compression. For details on the computational implementation of lossy checkpoint compression, as well as a comparison of different compression algorithms based on their compression as well as overheads, please refer to our previous publication (Kukreja et al., 2019a).

The most significant advantage of checkpointing is that the numerical result remains unchanged by applying this technique. Note that we will shortly combine this technique with lossy compression which might introduce an error, but checkpointing alone is expected to maintain bitwise equivalence. Another advantage is that the increase in run time incurred by the recomputation is predictable.

## 1.4 Data Compression

Compression or bit-rate reduction is a concept originally from signal processing. It involves representing information in fewer bits than the original representation. Since there is usually some computation required to go from one representation to another, compression can be seen as a memory-compute tradeoff.

Perhaps the most commonly known and used compression algorithm is ZLib (from GZip) (Deutsch and Gailly, 1996). ZLib is a lossless compression algorithm, i.e., the data recovered after compressing-decompressing is an exact replica of the original data before compression. Although ZLib is targeted at text data, which is one-dimensional and often has predictable repetition, other lossless compression algorithms are designed for other kinds of data. One example is FPZIP (Lindstrom et al., 2017), which is a lossless compression algorithm for multidimensional floating-point data.

For floating-point data, another possibility is lossy compression, where the compressed-decompressed data is not exactly the same as the original data, but a close approximation. The precision of this approximation is often set by the user of the compression algorithm. Two popular algorithms in this class are SZ (Di and Cappello, 2016) and ZFP (Lindstrom, 2014).

Compression has often been used to reduce the memory footprint of adjoint computations in the past, including Weiser and Götschel (2012); Boehm et al. (2016); Marin et al. (2016). However, all these studies use hand-rolled compression algorithms specific to the corresponding task - Weiser and Götschel (2012) focusses on parabolic equations, Boehm et al. (2016) focusses on wave propagation like us, and Marin et al. (2016) focusses on fluid flow. All three use their own lossy compression algorithm to compress the entire time history, and look at checkpointing as an alternative to lossy compression. In this paper we use a more general floating-point compression algorithm - ZFP. Since this compressor has been extensively used and studied across different domains and has implementations for various hardware platforms - this lends a sense of trust in this compressor, increasing the relevance of our work. None of the previously mentioned studies combine compression and checkpointing, as we do here.

Cyr et al. (2015) performs numerical experiments to study the propagation of errors through an adjoint problem using compression methods like PCA. However, they do not consider the combination of checkpointing and compression in a single strategy.

Floating-point can be seen as a *compressed representation* that is not entirely precise. However, the errors introduced by the floating-point representation are already accounted for in the standard numerical analysis as noise. The errors introduced by ZFP's compression of fields are more subtle since the compression loss is pattern sensitive. Hence we tackle it empirically here.

### 1.5 Contributions

The last few sections discussed some existing methods that allow trade-offs that are useful in solving FWI on limited resources. While checkpointing allows a trade-off between computational time and memory, compression allows a trade-off between memory and accuracy. This work combines these three approaches into one three-way trade-off.

In previous work (Kukreja et al., 2019a), we have shown that it is possible to accelerate generic adjoint-based computations (of which FWI is a subset), by using lossy compression on the checkpoints. For a given checkpoint absolute error tolerance (*atol* ), compression may or may not accelerate the computation. The performance model from Kukreja et al. (2019a) helps us answer this question a priori, i.e., without running any computations.

In this work, we evaluate this method on the specific problem of FWI, specifically the solver convergence and accuracy. To this end, we conduct an empirical study of:

1. Propagation of errors when starting from a lossy checkpoint.

2. Effect of checkpoint errors on the gradient computation.

3. Effect of decimation/subsampling on the gradient computation.

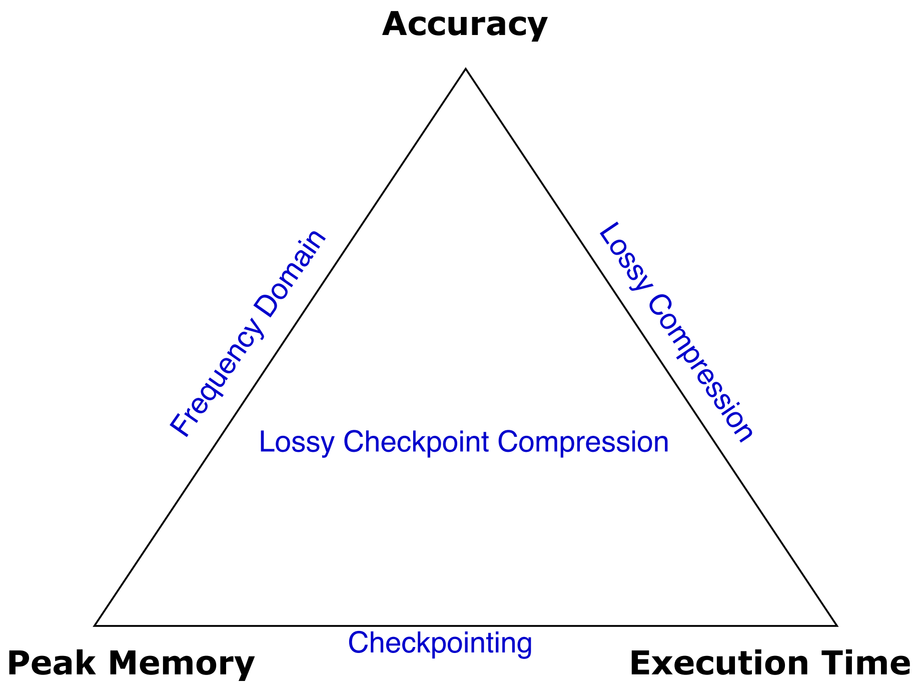


**Figure 3.** Schematic of the three-way tradeoff presented in this paper. With the use of checkpointing, it was possible to trade off memory and execution time (the horizontal line). With the use of compression alone, it was possible to trade off memory and accuracy. The combined approach presented in this work provides a novel three-way tradeoff.

4. Accumulation of errors through the stacking of multiple shots.
5. Effect of the lossy gradient on the convergence of FWI.

The rest of the paper is organized as follows. Section 2 gives an overview of FWI. This is followed by a description of our experimental setup in Section 3. Next, Section 4 presents the results, followed by a discussion in Section 5. Finally we present our conclusions and future work.

## 2 Full Waveform Inversion

FWI is designed to numerically simulate a seismic survey experiment and invert for the earth parameters that best explain the observations. In the physical experiment, a ship sends an acoustic impulse through the water by triggering an explosion. The waves created as a result of this impulse travel through the water into the earth's subsurface. The reflections and turning components of these waves are recorded by an array of receivers being dragged in tow by the ship. A recording of one signal sent and the corresponding signals received at each of the receiver locations is called a shot. A single experiment typically consists of 10000 shots.

Having recorded this collection of data ($\mathbf{d}_{\text{obs}}$), the next step is the numerical simulation. This starts with a wave equation. Many equations exist that can describe the propagation of a sound wave through a medium - the choice is usually a trade-off between accuracy and computational complexity. We mention here the simplest such equation, that describes isotropic acoustic wave propagation:

$$\mathbf{m}(x)\frac{\partial^2 \mathbf{u}(t,x)}{\partial t^2} - \nabla^2 \mathbf{u}(t,x) = \mathbf{q}_s(t,x), \tag{1}$$

where $\mathbf{m}(x) = \frac{1}{\mathbf{c}^2(x)}$ is the squared slowness, $\mathbf{c}(x)$ the spatially dependent speed of sound, $\mathbf{u}(t,x)$ is the pressure wavefield, $\nabla^2 \mathbf{u}(t,x)$ denotes the laplacian of the wavefield and $\mathbf{q}_s(t,x)$ is a source term. Solving Equation 1 for a given $\mathbf{m}$ and $\mathbf{q}_s$ can give us the *simulated* signal that *would be* received at the receivers. Specifically, the *simulated data* can be written as:

$$\mathbf{d}_{\text{sim}} = \mathbf{P}_r \mathbf{u} = \mathbf{P}_r \mathbf{A}(\mathbf{m})^{-1} \mathbf{P}_s^\top \mathbf{q}_s \tag{2}$$

where $\mathbf{P}_r$ is the measurement operator that restricts the full wavefield to the receivers locations, $\mathbf{A}(\mathbf{m})$ is the linear operator that is the discretization of the operator corresponding to Equation 1, and $\mathbf{P}_s$ is a linear operator that injects a localized source ($\mathbf{q}_s$) into the computational grid.

Using this, it is possible to set up an optimization problem that aims to find the value of $\mathbf{m}$ that minimizes the difference between the simulated signal ($\mathbf{d}_{\text{sim}}$) and the observed signal ($\mathbf{d}_{\text{obs}}$):

$$\underset{\mathbf{m}}{\operatorname{argmin}}\, \Phi_s(\mathbf{m}) = \frac{1}{2} \left\| \mathbf{d}_{\text{sim}} - \mathbf{d}_{\text{obs}} \right\|_2^2 . \tag{3}$$

This objective function $\Phi_s(\mathbf{m})$ can be minimized using a gradient descent method. The gradient can be computed as follows:

$$\nabla \Phi_s(\mathbf{m}) = \sum_{t=1}^{n_t} \mathbf{u}[\mathbf{t}] \mathbf{v}_{tt}[\mathbf{t}] = \mathbf{J}^T \delta \mathbf{d} \tag{4}$$

where $\mathbf{u}[\mathbf{t}]$ is the wavefield from Equation 1 and $\mathbf{v}_{tt}[\mathbf{t}]$ is the second-derivative of the adjoint field (Tarantola, 1984). The adjoint field is computed by solving an adjoint equation *backwards* in time. The appropriate adjoint equation is a result of the choice of the forward equation. In this example, we chose the acoustic isotropic equation (Equation 1), which is self-adjoint. However, it is not always trivial to derive the adjoint equation corresponding to a chosen forward equation (Hückelheim et al., 2019). This adjoint computation can only be started once the forward computation (i.e. the one involving Equation 1) is complete. Commonly, this is done by storing the intermediate values of $\mathbf{u}$ during the forward computation, then starting the adjoint computation to get values of $\mathbf{v}$, and using that and the previously calculated $\mathbf{u}$ to directly calculate $\nabla \Phi_s(\mathbf{m})$ in the same loop. This need to store the intermediate values of $\mathbf{u}$ during the forward computation is the source of the high memory footprint of this method.

The computation described in the previous paragraph is for a single shot and must be repeated for every shot, and the final gradient is calculated by averaging the gradients calculated for the individual shots. This is repeated for every iteration of the minimization. This entire minimization problem is one step of a multi-grid method that starts by inverting only the low frequency components on a coarse grid, and adding higher frequency components which require finer grids over successive inversions.

## 3 Experimental setup

### 3.1 Reference Problem

We use Devito (Kukreja et al., 2016; Luporini et al., 2018; Louboutin et al., 2019) to build an acoustic wave propagation experiment. The velocity model was initialized using the SEG Overthrust model. This velocity model was then smoothed using a Gaussian function to simulate a starting guess for a complete FWI problem. The original domain was surrounded by a 40 point deep absorbing boundary layer. This led to a total of $287 \times 881 \times 881$ grid points. This was run for $4000$ms with a step of $1.75$ms, making 2286 timesteps. The spatial domain was discretized on a grid with a grid spacing of 20m, and the discretization was 16th-order in space and second-order in time. We used 80 shots for our experiments with the sources placed along the x-dimension, spaced equally and just under the water surface. The shots were generated by modeling a Ricker source of peak frequency 8Hz.

The FWI experiments were run on Azure's Kubernetes Service, using the framework described in (Zhang et al., 2021). Each cloud worker instance processed only one shot at a time, although there were 20 cloud worker instances running in parallel. The processing of each shot started by reading shot data previously written to Azure Blob Storage, straight to the cloud worker instance. This was followed by a long gradient computation step (approximately 10 minutes). Due to this, we posit that file input/output did not play a role in the experiments discussed here.

Following the method outlined in Peters et al. (2019), we avoid inverse crime by generating the shots using a variation of Equation 1 that includes density, while using Equation 1 for inversion. The gradient was scaled by dividing by the norm of the original gradient in the first iteration. This problem solved in double precision is what we shall refer to as the *reference problem* in the rest of this paper. Note that this reference solution itself has many sources of error, including floating-point arithmetic and the discretization itself.

As can be seen in Section 2, there is only one wavefield in the isotropic acoustic equation, the pressure field, which is called $u$ in the equations. This is the field we compress in all of the experiments. The lossy-compression algorithm accepts a-priori a value for absolute error tolerance (*atol* ) and is expected to respect this tolerance during the compression-decompression cycle. We verify this as part of our experiments. Note that the word *absolute* is used here from the point-of-view of the compression - we are choosing an absolute value of error tolerance. The absolute value of acceptable error will probably change for waves of different amplitudes. Peak Signal to Noise Ratio can be used to calibrate the results here across different pressure amplitudes.

### 3.2 Error metrics

Let $F(i,j,k)$ be the original field (i.e. before any compression/loss), and $G(i,j,k)$ be the field recovered after lossy compression of $F(i,j,k)$, followed by decompression. We report errors using the following metrics:

**PSNR** : Peak Signal to Noise Ratio, we define this as:

$$\text{PSNR (dB)} = 10\log_{10}\frac{R^2}{\text{MSE}}, \tag{5}$$

where $R$ is the range of values in the field to be compressed, and MSE is the *mean squared error* between the reference and the lossy field. More precisely,

$$\text{MSE} = \frac{1}{mnp}\sum_{k=0}^{p}\sum_{j=0}^{n}\sum_{i=0}^{m}[F(i,j,k) - G(i,j,k)]^2, \tag{6}$$

and $R = \max(F(i,j,k)) - \min(F(i,j,k))$.

**Angle** : We treat $F(i,j,k)$ and $G(i,j,k)$ as vectors and calculate the angle between them as follows:

$$cos\theta = \frac{\overrightarrow{F}\cdot\overrightarrow{G}}{\left\|\overrightarrow{F}\right\|\cdot\left\|\overrightarrow{G}\right\|} \tag{7}$$

**Error Norms** : We also report some errors by defining the error vector $E(i,j,k)$, as $F(i,j,k) - G(i,j,k)$, and reporting $L_2$ and $L_\infty$ norms of this vector.

## 4 Results

### 4.1 Evolution of compressibility

To understand the evolution of compressibility, we compress every timestep of the reference problem using the same compression setting and report on the achieved compression factor as a function of the timestep.

This is shown in Figure 4. It can be seen that in the beginning the field is highly compressible since it consists of mostly zeros. The compressibility is worst towards the end of the simulation when the wave has reached most of the domain.

Therefore we pick the last timestep as the reference for further experiments. A 2D cross section of this snapshot is shown in Figure 5. Figure 6 shows a histogram of pressure values in order to give us an idea of the absolute pressure amplitude present in the wavefield being compressed. This plot can be used to give context to the absolute error tolerances in the rest of the results.

### 4.2 Direct compression

To understand the direct effects of compression, we compressed the reference wavefield using different absolute tolerance (*atol* ) settings and observed the errors incurred as a function of *atol* . The error is a tensor of the same shape as the original field and results from subtracting the reference field and the lossy field. Figure 7 shows the Peak Signal-to-Noise Ratio achieved for each *atol* setting. Figure A1 in the appendix shows some additional norms for this error tensor.

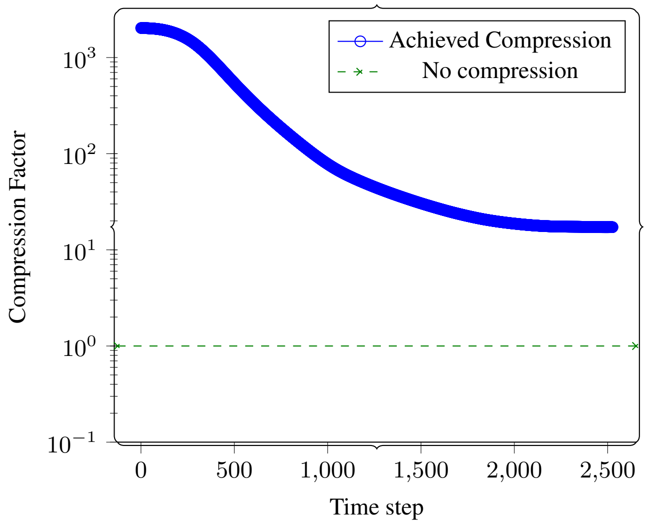


**Figure 4.** Evolution of compressibility through the simulation. We compressed every time step of the reference problem using an absolute error tolerance (*atol* ) setting of $10^{-4}$. The Compression factor achieved is plotted here as a function of the timestep number. Higher is more compression. Dotted line represents *no compression*. We can see that the first few timesteps are compressible to 1000x - since they are mostly zeros. The achievable compression factor drops as the wave propagates through the domain and seems to stabilize to  20x towards the end. We pick the last time step as the reference field for further experiments.

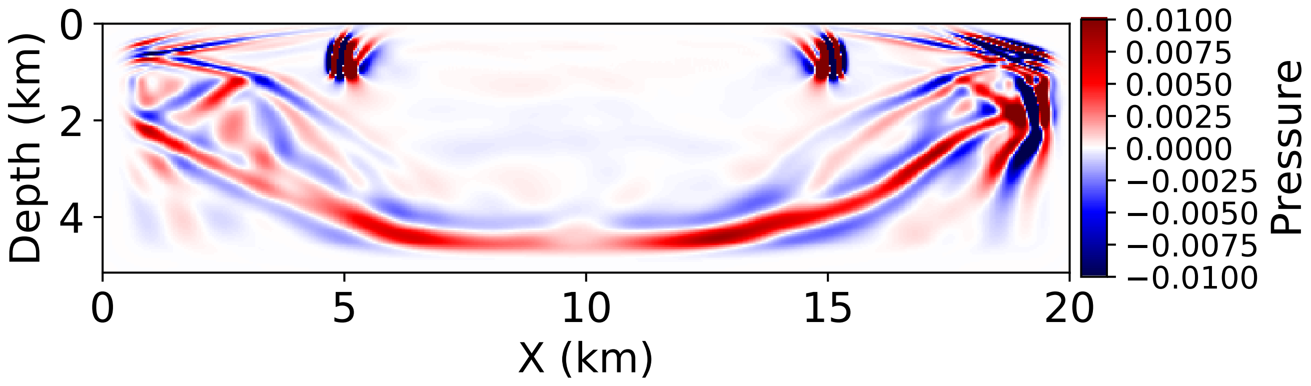


**Figure 5.** A 2D slice of the last time step of the reference solution. The wave has spread through most of the domain. The values in this figure have been thresholded to $\pm 0.01$ for easy visualization. For pressure amplitudes, see Figure 6.

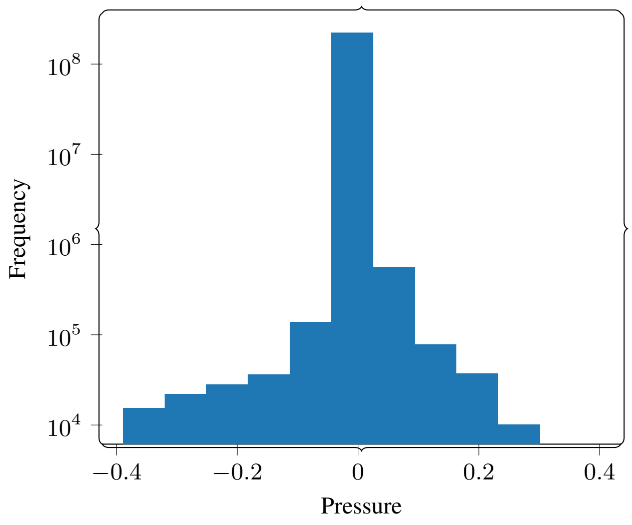


**Figure 6.** A histogram of pressure values in the wavefield shown in Figure 5. These values can be used to give context to the absolute error tolerances used in the rest of the experiments.

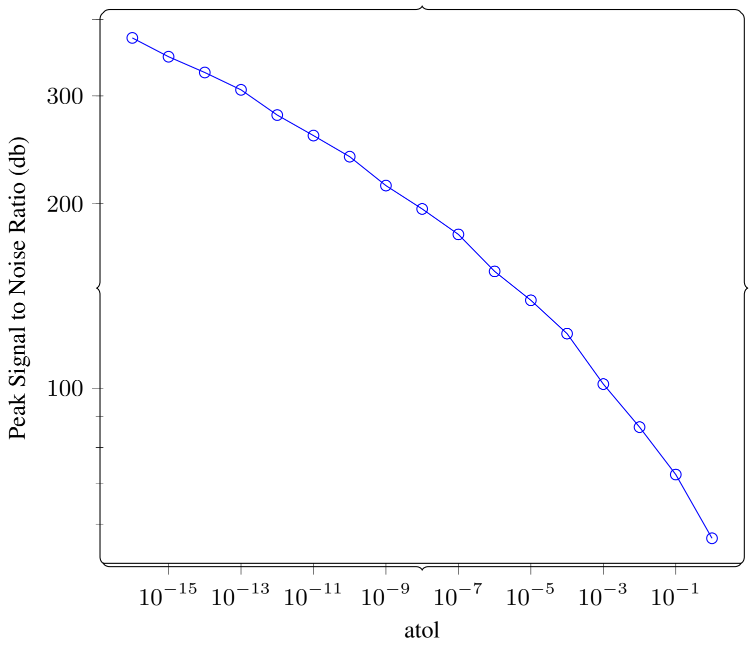


**Figure 7.** Direct compression: We compress the wave field at the last time step of the reference solution using different *atol* settings and report the Peak Signal to noise ratio (PSNR) achieved. Higher PSNR is lower error. The PSNR is very high for low absolute error tolerance (*atol* ) and drops predictably as *atol* is increased. See Figure A1 for more metrics on this comparison.

## 4.3 Forward propagation

Next, we ran the simulation for 550 steps and compressed the field's final state after this time. We then compress and decompress this through the lossy compression algorithm, getting two checkpoints - a reference checkpoint and a lossy checkpoint. We then restarted the simulation from step 550, comparing the progression of the simulation restarted from the lossy checkpoint vs. a *reference* simulation that was started from the original checkpoint. We run this test for another 250 timesteps.

Figure 8 shows the evolution of the PSNR between wavefields at corresponding timesteps as a function of the number of timesteps evolved. There is a slight initial drop followed by essentially constant values for the 250 timesteps propagated. This tells us that the numerical method is robust to the error induced by lossy compression and the error does not appear to be forcing the system to a different solution.

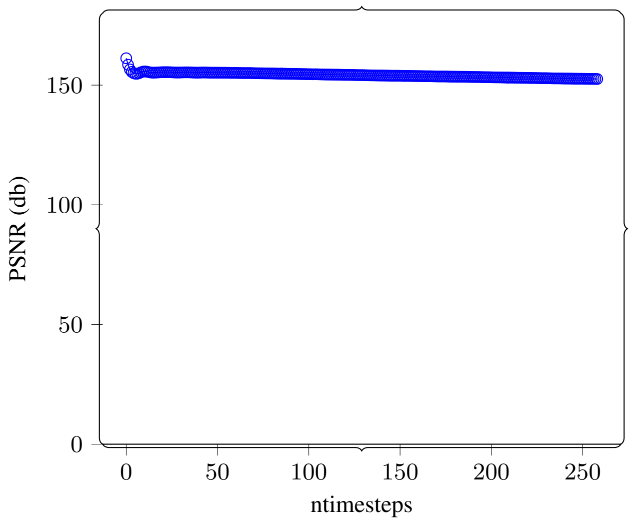


**Figure 8.** Forward propagation: We stop the simulation after about 500 timesteps. We then compress the state of the wavefield at this point using *atol* $= 10^{-6}$. We then continue the simulation from the lossy checkpoint and compare with the reference version. Here we show the error as PSNR. In these plots, higher is better, and lower implies more error. The PSNR dropped slightly over the first few timesteps and then remains essentially constant after.

## 4.4 Gradient computation

In this experiment, we do the complete gradient computation, as shown in Figure 1 - once for the reference problem and for a few different lossy settings. We measured the error in the gradient computation as a function of *atol* , using the same compression settings for all checkpoints.

In Figure 9, we report the error between the reference gradient and the *lossy gradient* as PSNR (See Section 3.2 for definition). The PSNR remains unchanged until an *atol* of $10^{-6}$, and even at its lowest, is at a very high value of 80.

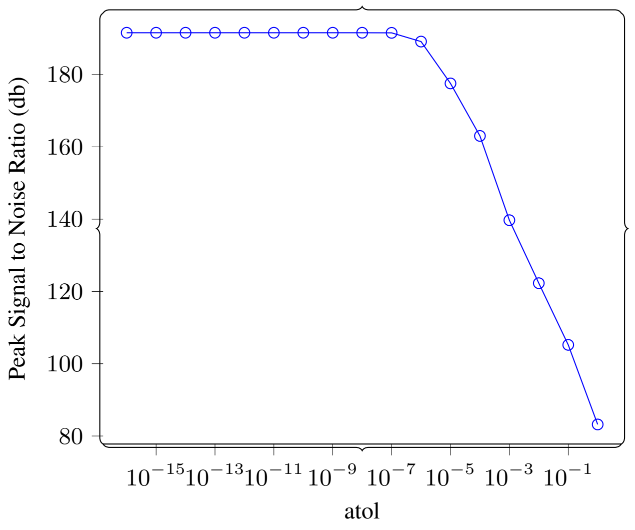


**Figure 9.** Gradient computation: In this experiment we carry out the full forward-reverse computation to get a gradient for a single shot, while compressing the checkpoints at different *atol* settings. This plot shows the PSNR of reference vs lossy gradient as a function of *atol* on the lossy checkpoints. We can see that the PSNR remains unchanged until about *atol* $= 10^{-6}$ and is very high even at very high values of *atol* .

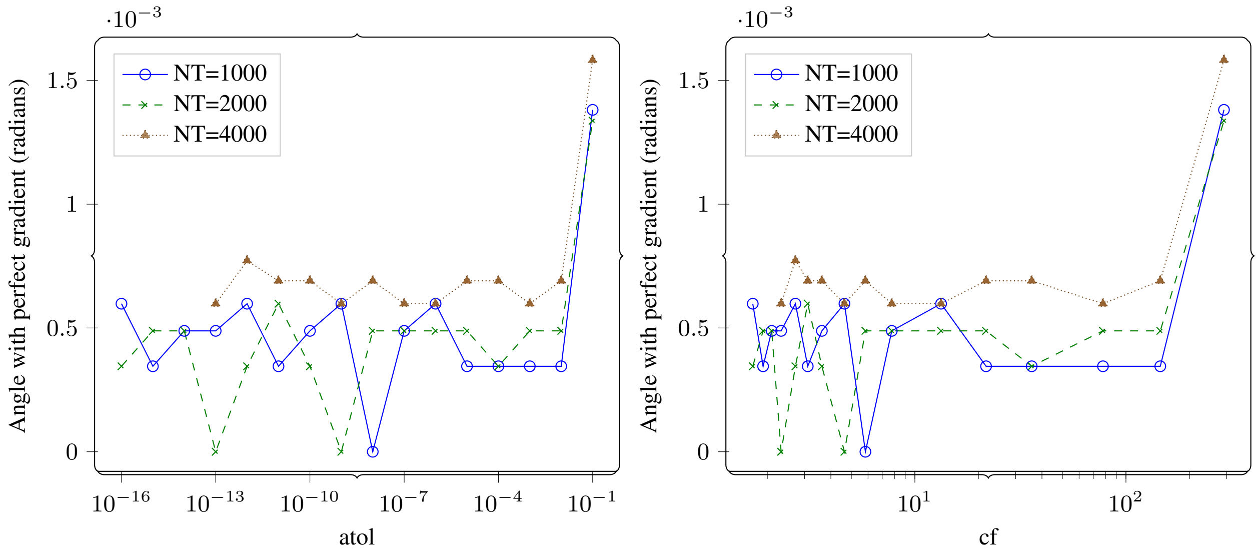


**Figure 10.** Gradient computation: Angle between the lossy gradient vector and the reference gradient vector (in radians) vs *atol* (left) and vs compression factor (right). If the lossy gradient vector was pointing in a significantly different direction as compared to the reference gradient, we could expect to see that on this plot. The angles are quite small. The number of timesteps do not affect the result by much. The results are also resilient to increasing *atol* up to $10^{-2}$. Compression factors of over 100x do not seem to significantly distort the results either.

In Figure 10, we look at the same experiment, but through a different metric. Here we report the angle between the reference gradient vector and the lossy gradient vector. We see that the error remains constant at a very small value of $5 \times 10^{-4}$ radian up to *atol* $= 10^{-2}$. This same plot also shows that the number of timesteps do not appear to change the error by much (for a constant number of checkpoints).

In Figure 11, we report the error as $L_\infty$ and $L_2$ norms, with respect to achieved compression factor. This makes it easier to compare with the subsampling strategy presented later (See Figure 19).

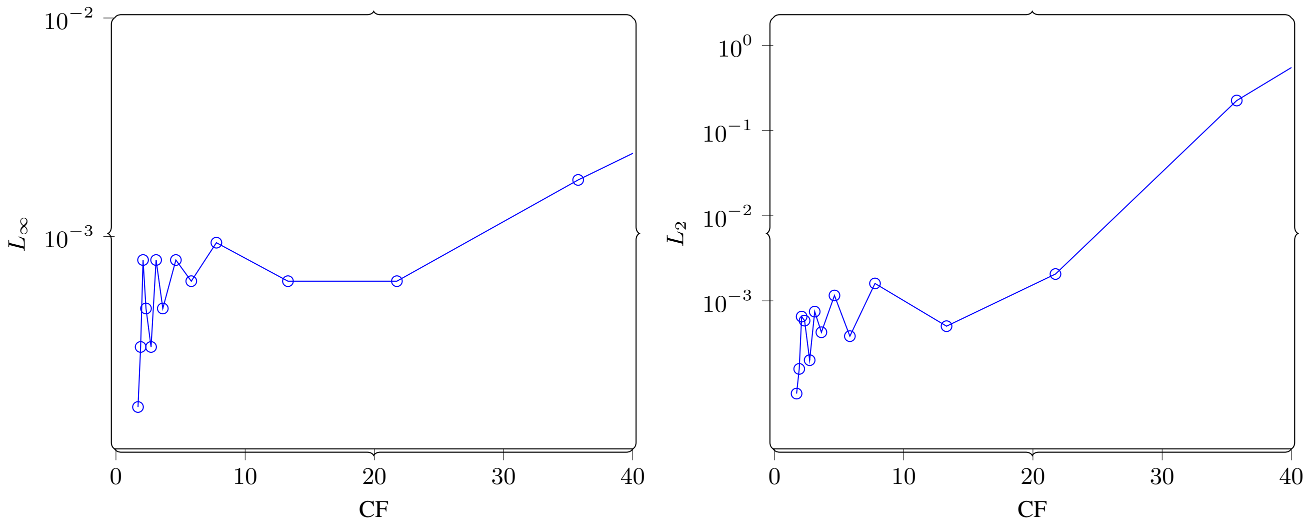


**Figure 11.** Gradient error: $L_\infty$ (left) and $L_2$ (right) norms of the gradient error as a function of the achieved compression factor (CF). It can be seen that error is negligible in the range of CF up to 16. Figure 19 repeats a part of these results to aid comparison with the subsampling strategy.

We also repeated the gradient computation for different number of checkpoints. Fewer checkpoints would apply lossy compression at fewer points in the calculation, while the amount of recomputation would be higher. More checkpoints would require less recomputation, but would apply lossy compression at more points during the calculation. Figure 12 shows the results of this experiment. We see that the PSNR is largely dependent on the value of *atol* , and does not change much when the number of checkpoints is changed by two orders of magnitude.

It can be seen from the plots that the errors induced in the checkpoint compression do not propagate significantly until the gradient computation step. In fact, the *atol* compression setting does not affect the error in the gradient computation until a cutoff point. It is likely that the cross-correlation step in the gradient computation is acting as an error-correcting step since the adjoint computation continues at the same precision as before - the only errors introduced are in the values from the forward computation used in the cross-correlation step (the dotted arrows in Figure 1).

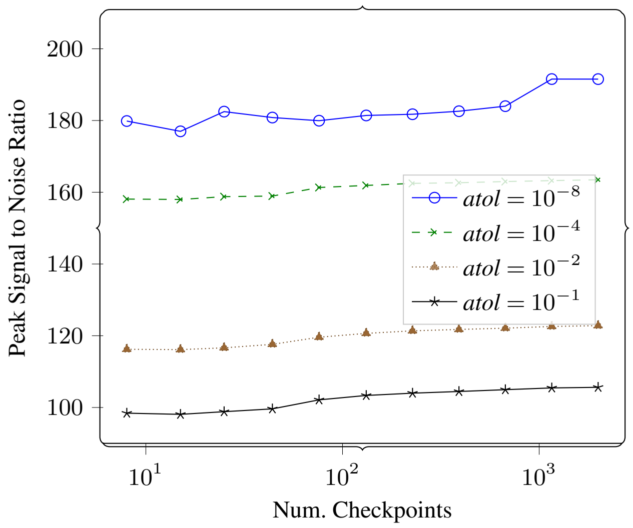


**Figure 12.** Gradient error: In this plot we measure the effect of varying number of checkpoints on the error in the gradient. We report PSNR of lossy vs reference gradient as a function of number of checkpoints, for four different compression settings.

## 4.5 Stacking

After gradient computation on a single shot, the next step in FWI is the accumulation of the gradients for individual shots by averaging them into a single gradient. We call this stacking. In this experiment we studied the accumulation of errors through this stacking process. Figure 13 shows the error in the gradient computation (compared to a similarly processed reference problem) as a function of the number of shots.

This plot shows us that the errors across the different shots are not adding up and the cumulative error is not growing with the number of shots - except for the compression setting of $atol = 10^{-1}$, which is chosen as an example of unreasonably high compression.

## 4.6 Convergence

Finally, we measure the effect of an approximate gradient on the convergence of the FWI problem. In practice, FWI is run for only a few iterations at a time as a fine-tuning step interspersed with other imaging steps. Here we run a fixed number of FWI iterations (30) to make it easier to compare different experiments. To make this a practical test problem, we extract a 2D slice from the original 3D velocity model and run a 2D FWI instead of 3D. We compare the convergence trajectory with the reference problem and report. For reference, Figure 14 shows the known true velocity model for this problem. Figure 15 shows the final velocity model after running a *reference* FWI for 30 iterations. Figure 17 shows the final velocity model after running FWI with compression enabled at different *atol* settings - also for 30 iterations.

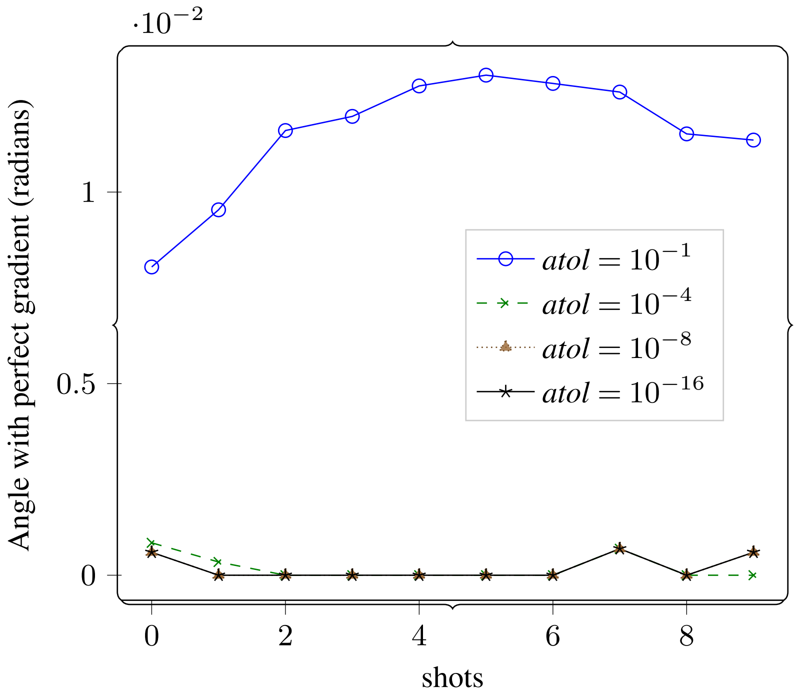


**Figure 13.** Shot stacking: The gradient is first computed for each individual shot and then added up for all the shots. In this experiment we measure the propagation of errors through this step. This plot shows that while errors do have the potential to accumulate through the step - as can be seen from the curve for $atol = 10^{-1}$, for compression settings that are useful otherwise, the errors do not accumulate significantly.

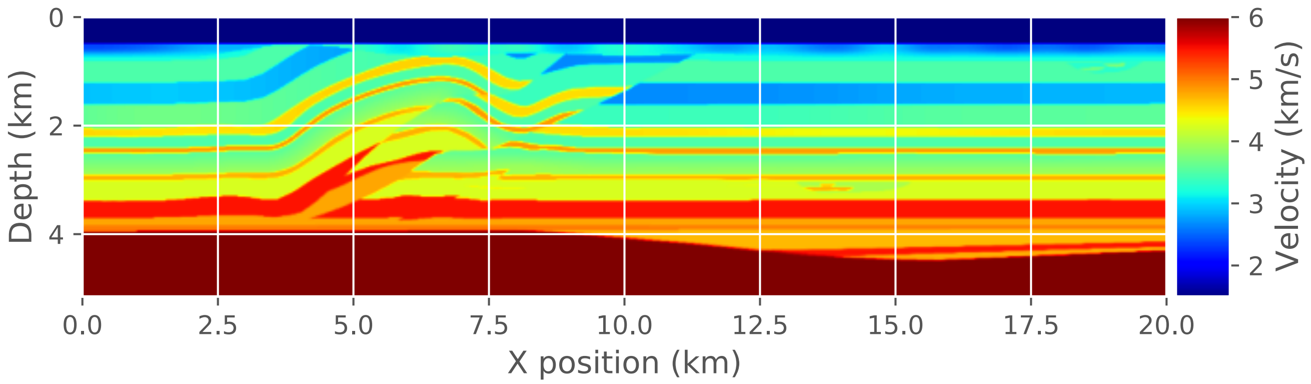


**Figure 14.** True model used for FWI

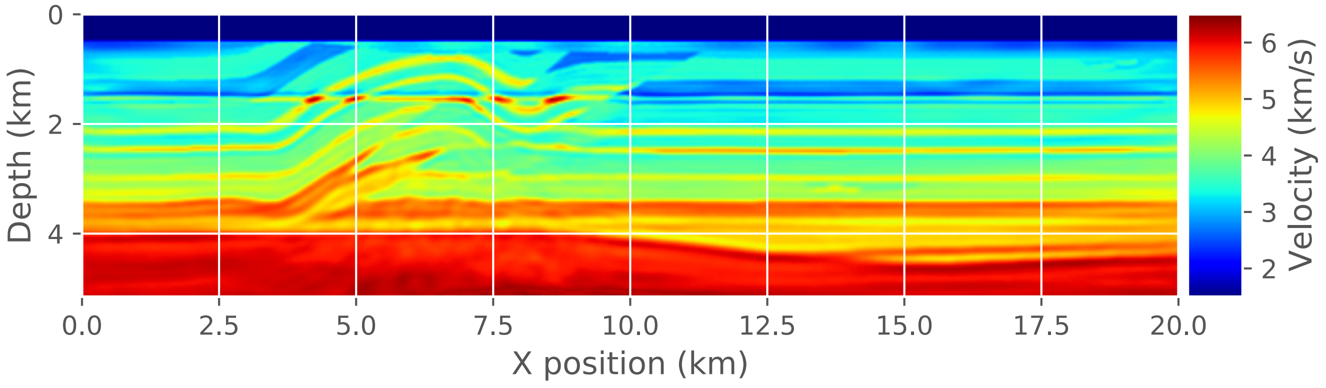


**Figure 15.** Reference solution for the complete FWI problem. This is the solution after running *reference* FWI for 30 iterations

Figure 16 shows the convergence trajectory - the objective function value as a function of the iteration number. We show this convergence trajectory for 4 different compression settings. It can be seen that the compressed version does indeed follow a very similar trajectory as the original problem.

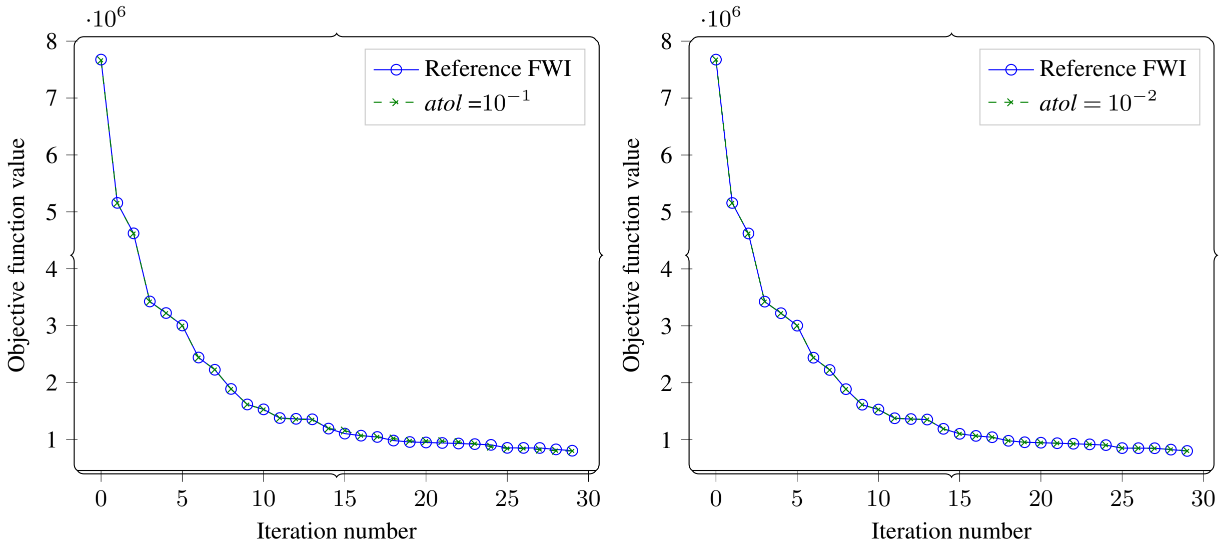


**Figure 16.** Convergence: As the last experiment, we run complete FWI to convergence (up to max 30 iterations). Here we show the convergence profiles for *atol* $= 10^{-1}$ (left) and *atol* $= 10^{-2}$ (right) vs the reference problem. The reference curve is so closely followed by the lossy curve that the reference curve is hidden behind.

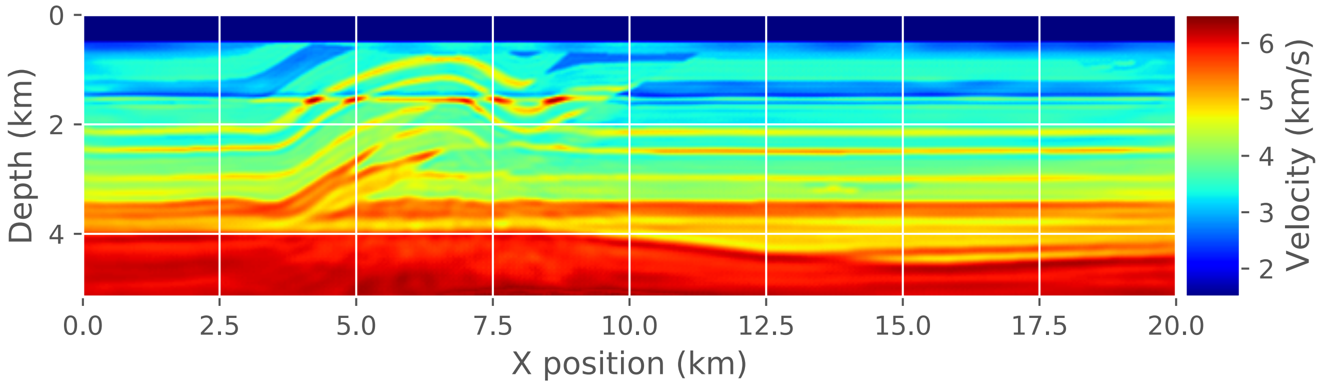


**Figure 17.** Final image after running FWI *atol* $= 10^{-1}$. It is visually indistinguishable from the reference solution in Figure 15.

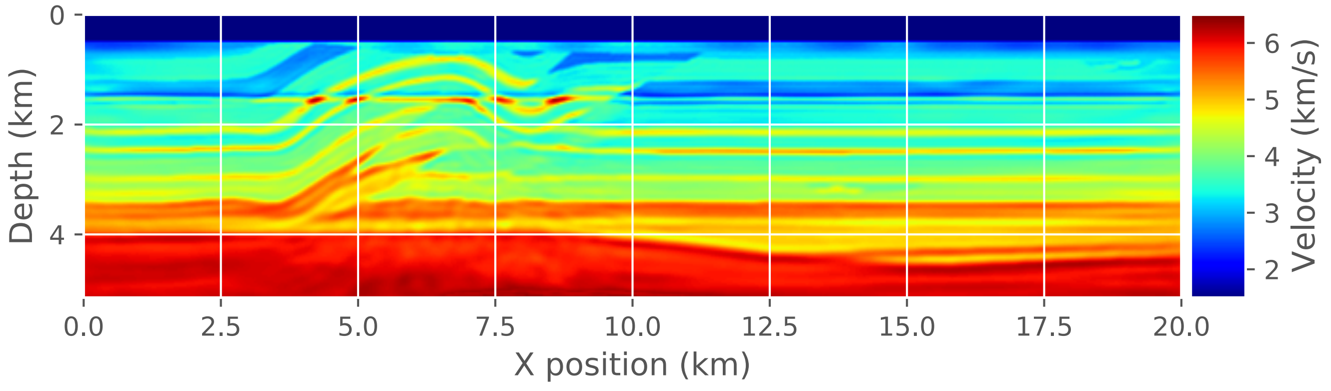


**Figure 18.** Final image after running FWI *atol* $= 10^{-2}$. It is visually indistinguishable from the reference solution in Figure 15.

## 4.7 Subsampling

As a comparison baseline, we also use subsampling to reduce the memory footprint as a separate experiment and track the errors. Subsampling is another method used to reduce the memory footprint of FWI, often used in industry. The method is set up so that the forward and adjoint computations continue at the same time stepping as the reference problem. However, the gradient computation is now not done at the same rate - it is reduced by a factor *f*. We plot results for varying *f*.

Figure 19 shows some error metrics as a function of the compression factor $f$.

Comparing Figure 19 with Figure 11, it can be seen that the proposed method produces significantly smaller errors for similar compression factors.

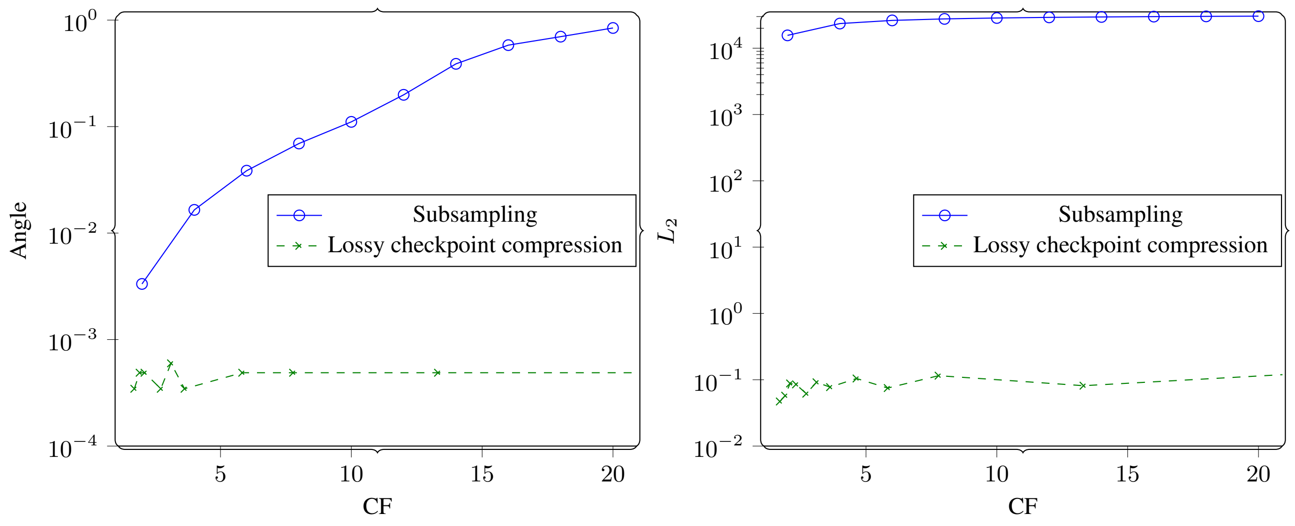


**Figure 19.** Subsampling: We set up an experiment with subsampling as a baseline for comparison. Subsampling is when the gradient computation is carried at a lower timestepping than the simulation itself. This requires less data to be carried over from the forward to the reverse computation at the cost of solution accuracy so is comparable to lossy checkpoint compression. This plot shows the angle between the lossy gradient and the reference gradient versus the compression factor $CF$ (left) and $L_2$ norm of gradient error versus the compression factor $CF$ (right) for this experiment. Compare this to the errors in Figure 11 that apply for lossy checkpoint compression.

## 5 Discussion

The results indicate that significant lossy compression can be applied to the checkpoints before the solution is adversely affected. This is an interesting result because, while it is common to see approximate methods leading to approximate solutions, this is not what we see in our results - the solution error doesn't change much for large compression error. This being an empirical study, we can only speculate on the reasons for this. We know that in the proposed method, the adjoint computation is not affected at all - the only effect is in the wavefield carried over from the forward computation to the gradient computation step. Since the gradient computation is a cross-correlation, we only expect correlated signals to grow in magnitude in the gradient calculation and when gradients are stacked. The optimization steps are likely to be error-correcting as well since even with an approximate gradient (*atol* $> 10^{-4}$), the convergence trajectory and the final results do not appear to change much - indicating that the errors in the gradient might be canceling out over successive iterations. There is even the possibility that these *errors* in the gradient act as a regularization (Tao et al., 2019). This is likely since we know from Diffenderfer et al. (2019), that ZFP's errors are likely to *smoothen* the field being compressed. We also know from Tao et al. (2019) that ZFP's errors are likely to be (close-to) normally distributed with 0 mean, which reinforces the idea that this is likely to act as a regularizer. We only tried this with the ZFP compression algorithm in this work. ZFP being a generic floating-point compression algorithm, is likely to be more broadly applicable than application-specific compressors. This makes it more broadly useful for Devito, which was the DSL that provided the context for this work. Some clear choices for the next compressors to try would be SZ

(Di and Cappello, 2016) - which is also a generic floating-point compression library, and the application-specific compressors from Weiser and Götschel (2012); Boehm et al. (2016); Marin et al. (2016). A different compression algorithm would change:

- the error distribution,
- the compression/decompression times, and
- the achieved compression factors.

Based on the experiments from Tao et al. (2019), we would expect the errors in SZ to be (nearly) uniformly distributed with a 0 mean. It would be interesting to see the effect this new distribution has on the method we describe here. If a new compressor can achieve higher compression factors than ZFP (for illustration), in less compression/decompression time than ZFP, then it will clearly speed up the application relative to ZFP. In reality, the relationship is likely to be more complex, and the performance model from Kukreja et al. (2019a) helps compare the performance of various compressors on this problem without running the full problem. The number of checkpoints has some effect on the error - more checkpoints incur less error for the same compression setting - as would be expected. Since we showed the benefits of compression for inversion, the expected speedup does not depend on the medium that varies between iterations from extremely smooth to very heterogeneous. While we focused on acoustic waves in this work for simplicity, different physics should not impact the compression factor due to the strong similarity between the solutions of different wave equations. However, different physics might require more fields in the solution - increasing the memory requirements, while also increasing the computational requirements. Whether this increase favours compression or recomputation more depends on the operational intensity of the specific wave equation kernel. The choice of misfit function is also not expected to impact our results since the wavefield does not depend on the misfit function. A more thorough study will, however, be necessary to generalize our results to other problem domains such as computational fluid dynamics that involve drastically different solutions.

Our method accepts an acceptable error tolerance as input for every gradient evaluation. We expect this to be provided as part of an adaptive optimization scheme that requires approximate gradients in the first few iterations of the optimization, and progressively more accurate gradients as the solution approaches the optimum. Such a scheme was previously described in Blanchet et al. (2019). Implementing such an adaptive optimizer and using it in practice is ongoing work.

## 6 Conclusions and Future Work

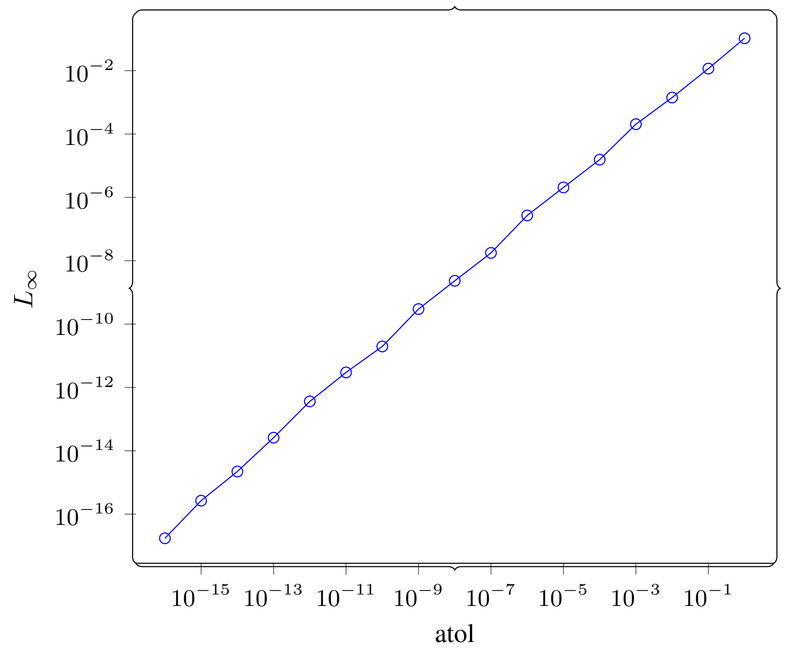
In the preceding sections, we have shown that using lossy compression, high compression factors can be achieved without significantly impacting the convergence or final solution of the inversion solver. This is a very promising result for the use of lossy compression in FWI. The use of compression in large computations like this is especially important in the exascale era, where the gap between computing and memory speed is increasingly large. Compression can reduce the strain on the memory bandwidth by trading it off for extra computation - this is especially useful since modern CPUs are hard to saturate with low OI computations.

In future work, we would like to study the interaction between compression errors and the velocity model for which FWI is being solved, as well as the source frequency. We would also like to compare multiple lossy compression algorithms e.g., *SZ*.

*Code and data availability.* The data used was from the Overthrust model, provided by SEG/EAGE (Aminzadeh and Brac, 1997). The code for the scripts used here (Kukreja, 2020), the Devito DSL (Luporini et al., 2020), and pyzfp (Kukreja et al., 2020) is all available online through Zenodo.

## Appendix A: Additional Results

### A1 Direct compression



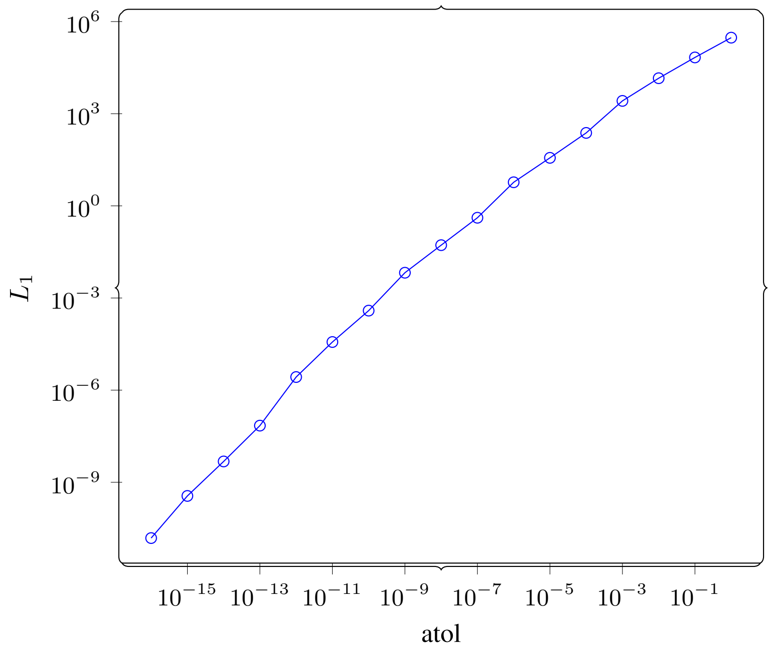


**Figure A1.** Direct compression: On the top, $L_\infty$ norm of error versus *atol* . This plot verifies that ZFP respects the tolerance we set. On the bottom, $L_1$ norm of error versus *atol* . From the difference in magnitude between the $L_\infty$ plot and this one, we can see how the error is spread across the domain.

## A2 Gradient Computation

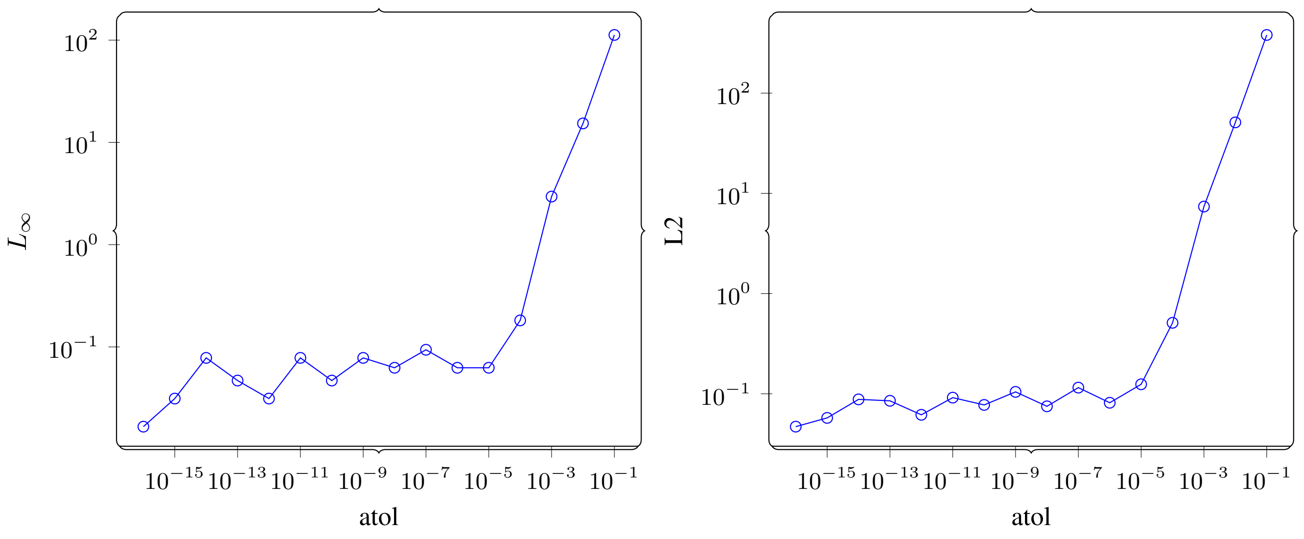


**Figure A2.** Gradient computation: $L_\infty$ norm of gradient error versus *atol* (left) and $L_2$ norm of gradient error versus *atol* . It can be seen that the error stays almost constant and very low up to a threshold value of $10^{-4}$

*Author contributions.* Most of the code and experimentation were done by Navjot. The experiments were planned between Jan and Navjot. Mathias helped set up meaningful experiments. John contributed in finetuning the experiments and the presentation of results. Paul and Gerard gave the overall direction of the work. Everybody contributed to the writing.

*Competing interests.* The authors have no competing interests to declare

*Acknowledgements.* This work was funded in part by support from the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. This research was carried out with the support of Georgia Research Alliance and partners of the ML4Seismic Center.

## References

Aminzadeh, F. and Brac, J.: SEG/EAGE 3-D Overthrust Models., https://doi.org/10.5281/zenodo.4252588, https://doi.org/10.5281/zenodo.4252588, 1997.

Aupy, G. and Herrmann, J.: Periodicity in optimal hierarchical checkpointing schemes for adjoint computations, Optimization Methods and Software, 32, 594–624, 2017.

Blanchet, J., Cartis, C., Menickelly, M., and Scheinberg, K.: Convergence rate analysis of a stochastic trust-region method via supermartingales, INFORMS journal on optimization, 1, 92–119, 2019.

Boehm, C., Hanzich, M., de la Puente, J., and Fichtner, A.: Wavefield compression for adjoint methods in full-waveform inversion, Geophysics, 81, R385–R397, 2016.

Chatelain, Y., Petit, E., de Oliveira Castro, P., Lartigue, G., and Defour, D.: Automatic exploration of reduced floating-point representations in iterative methods, in: European Conference on Parallel Processing, pp. 481–494, Springer, 2019.

Cyr, E. C., Shadid, J., and Wildey, T.: Towards efficient backward-in-time adjoint computations using data compression techniques, Computer Methods in Applied Mechanics and Engineering, 288, 24–44, 2015.

Deutsch, P. and Gailly, J.-L.: Zlib compressed data format specification version 3.3, Tech. rep., RFC 1950, May, 1996.

Di, S. and Cappello, F.: Fast error-bounded lossy HPC data compression with SZ, in: 2016 ieee international parallel and distributed processing symposium (ipdps), pp. 730–739, IEEE, 2016.

Diffenderfer, J., Fox, A. L., Hittinger, J. A., Sanders, G., and Lindstrom, P. G.: Error analysis of zfp compression for floating-point data, SIAM Journal on Scientific Computing, 41, A1867–A1898, 2019.

Fehler, M. and Keliher, P. J.: SEAM phase 1: Challenges of subsalt imaging in tertiary basins, with emphasis on deepwater Gulf of Mexico, Society of Exploration Geophysicists, 2011.

Griewank, A. and Walther, A.: Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation, ACM Transactions on Mathematical Software (TOMS), 26, 19–45, 2000.

Guasch, L., Agudo, O. C., Tang, M.-X., Nachev, P., and Warner, M.: Full-waveform inversion imaging of the human brain, npj Digital Medicine, 3, 1–12, 2020.

Hückelheim, J., Kukreja, N., Narayanan, S. H. K., Luporini, F., Gorman, G., and Hovland, P.: Automatic differentiation for adjoint stencil loops, in: Proceedings of the 48th International Conference on Parallel Processing, pp. 1–10, 2019.

Jameson, A., Martinelli, L., and Pierce, N.: Optimum aerodynamic design using the Navier–Stokes equations, Theoretical and computational fluid dynamics, 10, 213–237, 1998.

Knibbe, H., Mulder, W., Oosterlee, C., and Vuik, C.: Closing the performance gap between an iterative frequency-domain solver and an explicit time-domain scheme for 3D migration on parallel architectures, Geophysics, 79, S47–S61, 2014.

Kukreja, N.: navjotk/error_propagation: v0.1, https://doi.org/10.5281/zenodo.4247199, https://doi.org/10.5281/zenodo.4247199, 2020.

Kukreja, N., Louboutin, M., Vieira, F., Luporini, F., Lange, M., and Gorman, G.: Devito: Automated fast finite difference computation, in: 2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC), pp. 11–19, IEEE, 2016.

Kukreja, N., Hückelheim, J., Lange, M., Louboutin, M., Walther, A., Funke, S. W., and Gorman, G.: High-level python abstractions for optimal checkpointing in inversion problems, arXiv preprint arXiv:1802.02474, 2018.

Kukreja, N., Hückelheim, J., Louboutin, M., Hovland, P., and Gorman, G.: Combining Checkpointing and Data Compression to Accelerate Adjoint-Based Optimization Problems, in: European Conference on Parallel Processing, pp. 87–100, Springer, 2019a.

Kukreja, N., Shilova, A., Beaumont, O., Huckelheim, J., Ferrier, N., Hovland, P., and Gorman, G.: Training on the Edge: The why and the how, in: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 899–903, IEEE, 2019b.

Kukreja, N., Greaves, T., Gorman, G., and Wade, D.: navjotk/pyzfp: Dummy release to force Zenodo archive, https://doi.org/10.5281/zenodo.4252530, https://doi.org/10.5281/zenodo.4252530, 2020.

Lindstrom, P.: Fixed-rate compressed floating-point arrays, IEEE transactions on visualization and computer graphics, 20, 2674–2683, 2014.

Lindstrom, P. G. et al.: FPZIP, Tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2017.

Louboutin, M. and Herrmann, F. J.: Time compressively sampled full-waveform inversion with stochastic optimization, in: SEG Technical Program Expanded Abstracts 2015, pp. 5153–5157, Society of Exploration Geophysicists, 2015.

Louboutin, M., Lange, M., Luporini, F., Kukreja, N., Witte, P. A., Herrmann, F. J., Velesko, P., and Gorman, G. J.: Devito (v3. 1.0): an embedded domain-specific language for finite differences and geophysical exploration, Geoscientific Model Development, 12, 1165–1187, 2019.

Luporini, F., Lange, M., Louboutin, M., Kukreja, N., Hückelheim, J., Yount, C., Witte, P., Kelly, P. H., Gorman, G. J., and Herrmann, F. J.: Architecture and performance of Devito, a system for automated stencil computation, arXiv preprint arXiv:1807.03032, 2018.

Luporini, F., Louboutin, M., Lange, M., Kukreja, N., rhodrin, Bisbas, G., Pandolfo, V., Cavalcante, L., tjb900, Gorman, G., Mickus, V., Bruno, M., Kazakas, P., Dinneen, C., Mojica, O., von Conta, G. S., Greaves, T., SSHz, EdCaunt, de Souza, J. F., Speglich, J. H., Jr., T. A., Jan, Witte, P., BlockSprintZIf, gamdow, Hester, K., Rami, L., Washbourne, R., and vkrGitHub: devitocodes/devito: v4.2.3, https://doi.org/10.5281/zenodo.3973710, https://doi.org/10.5281/zenodo.3973710, 2020.

Marin, O., Schanen, M., and Fischer, P.: Large-scale lossy data compression based on an a priori error estimator in a spectral element code, Tech. rep., ANL/MCS-P6024-0616, 2016.

Peters, B., Smithyman, B. R., and Herrmann, F. J.: Projection methods and applications for seismic nonlinear inverse problems with multiple constraints, Geophysics, 84, R251–R269, 2019.

Symes, W. W.: Reverse time migration with optimal checkpointing, Geophysics, 72, SM213–SM221, 2007.

Tao, D., Di, S., Guo, H., Chen, Z., and Cappello, F.: Z-checker: A framework for assessing lossy compression of scientific data, The International Journal of High Performance Computing Applications, 33, 285–303, 2019.

Tarantola, A.: Inversion of seismic reflection data in the acoustic approximation, Geophysics, 49, 1259–1266, 1984.

van Leeuwen, T. and Herrmann, F. J.: 3D frequency-domain seismic inversion with controlled sloppiness, SIAM Journal on Scientific Computing, 36, S192–S217, 2014.

Virieux, J. and Operto, S.: An overview of full-waveform inversion in exploration geophysics, Geophysics, 74, WCC1–WCC26, 2009.

Wang, Q., Moin, P., and Iaccarino, G.: Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation, SIAM Journal on Scientific Computing, 31, 2549–2567, 2009.

Weiser, M. and Götschel, S.: State trajectory compression for optimal control with parabolic PDEs, SIAM Journal on Scientific Computing, 34, A161–A184, 2012.

Witte, P. A., Louboutin, M., Luporini, F., Gorman, G. J., and Herrmann, F. J.: Compressive least-squares migration with on-the-fly Fourier transforms, Geophysics, 84, 1–76, 2019.

Zhang, Q., Iordanescu, G., Tok, W. H., Brandsberg-Dahl, S., Srinivasan, H. K., Chandra, R., Kukreja, N., and Gorman, G.: Hyperwavve: A cloud-native solution for hyperscale seismic imaging on Azure, in: First International Meeting for Applied Geoscience & Energy, pp. 782–786, Society of Exploration Geophysicists, 2021.

Zhang, Y., Zhang, H., and Zhang, G.: A stable TTI reverse time migration and its implementation, GEOPHYSICS, 76, WA3–WA11, https://doi.org/10.1190/1.3554411, http://dx.doi.org/10.1190/1.3554411, 2011.