# Combining Checkpointing and Data Compression to Accelerate Adjoint-Based Optimization Problems

Navjot Kukreja[1(✉)], Jan Hückelheim[1], Mathias Louboutin[2], Paul Hovland[3], and Gerard Gorman[1]

[1] Imperial College London, London, UK
nkukreja@imperial.ac.uk
[2] Georgia Institute of Technology, Atlanta, GA, USA
[3] Argonne National Laboratory, Lemont, IL, USA

**Abstract.** Seismic inversion and imaging are adjoint-based optimization problems that process up to terabytes of data, regularly exceeding the memory capacity of available computers. Data compression is an effective strategy to reduce this memory requirement by a certain factor, particularly if some loss in accuracy is acceptable. A popular alternative is checkpointing, where data is stored at selected points in time, and values at other times are recomputed as needed from the last stored state. This allows arbitrarily large adjoint computations with limited memory, at the cost of additional recomputations.

In this paper, we combine compression and checkpointing for the first time to compute a realistic seismic inversion. The combination of checkpointing and compression allows larger adjoint computations compared to using only compression, and reduces the recomputation overhead significantly compared to using only checkpointing.

**Keywords:** Checkpointing · Compression · Adjoints · Inversion · Seismic

## 1 Introduction

### 1.1 Adjoint-Based Optimization

Adjoint-based optimization problems typically consist of a simulation that is run forward in simulation time, producing data that is used in reverse order by a subsequent adjoint computation that is run backwards in simulation time. Many important numerical problems in science and engineering use adjoints and follow this pattern.

Since the data for each of the computed timestep in the forward simulation will be used later in the adjoint computation, it would be prudent to store it in memory until it is required again. However, the total size of this data can


R. Yahyapour (Ed.): Euro-Par 2019, LNCS 11725, pp. 87–100, 2019.
https://doi.org/10.1007/978-3-030-29400-7_7

often run into tens of terabytes, exceeding the memory capacity of most computer systems. Previous work has studied recomputation or data compression strategies to work around this problem. In this paper we investigate a combination of compression and recomputation.

### 1.2 Example Adjoint Problem: Seismic Inversion

Seismic inversion typically involves the simulation of the propagation of seismic waves through the earth's subsurface, followed by a comparison with data from field measurements. The model of the subsurface is iteratively improved by minimizing the misfit between simulated data and field measurement in an adjoint optimization problem [18]. The data collected in an offshore survey typically consists of a number of "shots" - each of these shots corresponding to different locations of sources and receivers. Often the gradient is computed for each of these shots independently on a single cluster compute node, and then collated across all the shots to form a single model update. The processing across shots is thereby easily parallelized and requires only little communication, followed by a long period of independent computation (typically around 10–100 min). Since the number of shots is typically of the order of $10^4$, clusters can often be fully utilized even if individual shots are only processed on a single node.

### 1.3 Memory Requirements

A number of strategies have been studied to cope with the amount of data that occurs in adjoint computations - perhaps the simplest is to store all data to a disk, to be read later by the adjoint pass in reverse order. However, the computation often takes much less time than the disk read and write, hence leaving disk speed as a bottleneck.

Domain decomposition, where a single shot may be distributed across more than one node, is often used not only to distribute the computational workload across more processors, but also across more memory. While this strategy is very powerful, the number of compute nodes and therefore the amount of memory that can be used efficiently is limited, for example by communication overheads that start to dominate as the domain is split into increasingly small pieces. Secondly, this strategy can be wasteful if the need for memory causes more nodes to be used than can be completely utilized for computation. Lastly, this method is not well suited for cloud-based setups since it can complicate the setup and performance will suffer due to the slow inter-node communication.

Checkpointing is yet another strategy to reduce the memory overhead. Only a subset of the timesteps during the forward pass is stored. Other timesteps are recomputed when needed by restarting the forward pass from the last available stored state. We discuss this strategy in Sect. 3. Previous work has applied checkpointing to seismic imaging and inversion problems [9,20]. An alternative is data compression, which is discussed in Sect. 2.

In this paper, we extend the previous studies by *combining* checkpointing and compression. This is obviously useful when the data does not fit in the available

memory even after compression, for example for very large adjoint problems, or for problems where the required accuracy limits the achievable compression ratios.

Compared to the use of only checkpointing without compression, this combined method often improves performance. This is a consequence of the reduced size of stored timesteps, allowing more timesteps to be stored during the forward computation. This in turn reduces the amount of recomputation that needs to be performed. On the other hand, the compression and decompression itself takes time. The answer to the question "does compression pay off?", depends on a number of factors including - available memory, the required precision, the time taken to compress and decompress, and the achieved compression factors, and various problem specific parameters like computational intensity of the kernel involved in the forward and adjoint computations, and the number of timesteps.

Hence, the answer to the compression question depends not only on the problem one is solving (within seismic inversion, there are numerous variations of the wave equation that may be solved), but also the hardware specifics of the machine on which it is being solved. In fact, as we will see in Sect. 5, the answer might even change during the solution process of an individual problem. This brings up the need to predict whether compression pays off in a given scenario, without incurring significant overheads in answering this question. To this end, we present a performance model that answers that question.

### 1.4 Summary of Contributions

In this paper, we study

- the use of different compression algorithms to seismic data including six lossless and a lossy compression algorithm for floating point data,
- a performance model for checkpointing alone, taking into account the time taken to read and write checkpoints, and
- an online performance model to predict whether compression would speed up an optimization problem.

## 2 Compression Algorithms

Data compression is increasingly used to reduce the memory footprint of scientific applications. This has been accelerated by the advent of special purpose compression algorithms for floating-point scientific data, such as ZFP or SZ [10,14].

Lossless algorithms guarantee that the exact original data can be recovered during decompression, whereas lossy algorithms introduce an error, but often guarantee that the error does not exceed certain absolute or relative error metrics. Typically, lossy compression is more effective in reducing the data size. Most popular compression packages offer various settings that allow a tradeoff between compression ratio, accuracy, and compression and decompression time.

Another difference we observed between lossless and lossy compression algorithms was that the lossless compression algorithms we evaluated tended to interpret all data as one-dimensional series only while SZ and ZFP, being designed for scientific data, take the dimensionality into account directly. This makes a difference in the case of a wavefield, for example, where the data to be compressed corresponds to a smoothly varying function in (two or) three dimensions and interpreting this three-dimensional data as one-dimensional would completely miss the smoothness and predictability of the data values.

It is worth noting that another data reduction strategy is to typecast values into a lower precision format, for example, from double precision to single precision. This can be seen as a computationally cheap lossy compression algorithm with a compression ratio of 2.

Perhaps counterintuitively, compression can not only reduce the memory footprint, but also speed up an application. Previous work has observed that the compression and decompression time can be less than the time saved from the reduction in data that needs to be communicated across MPI nodes or between a GPU and a host computer [17].

One way of using compression in adjoint-based methods is to compress all timesteps during the forward pass. If the compression ratio is sufficient to fit the compressed data in memory, compression can serve as an *alternate strategy* to checkpointing. Previous work has discussed this in the context of computational fluid dynamics [7,16] and seismic inversion using compression algorithms specifically designed for the respective applications [6,8].

Since the time spent on compressing and decompressing data is often non-negligible, this raises the question whether the computational time is better spent on this compression and decompression, or on the recomputation involved in the more traditional checkpointing approach. This question was previously answered to a limited extent for the above scenario where compression is an alternative to checkpointing, in a specific application [7]. We discuss this in more detail in Sect. 4.

### 2.1 Lossless Compression

Blosc is a library that provides optimized high-performance implementations of various lossless compressors, sometimes beyond their corresponding reference implementations [2]. For our experiments we use this library through its python interface. The library includes implementations for six different lossless compression algorithms, namely ZLIB, ZSTD, BLOSCLZ, LZ4, LZ4HC and Snappy. All these algorithms look at the data as a one-dimensional stream of bits and at least the blosc implementations have a limit on the size of the one-dimensional array that can be compressed in one call. Therefore we use the python package *blosc-pack*, which is a wrapper over the blosc library, to implement *chunking*, i.e. breaking up the stream into chunks of a chosen size, which are compressed one at a time.

### 2.2 Lossy Compression

We use the lossy compression package ZFP [14] written in C. To use ZFP from python, we developed a python wrapper for the reference implementation of ZFP[1]. ZFP supports three compression modes, namely fixed tolerance, fixed precision and fixed rate. The fixed-tolerance mode limits the absolute error, while the fixed-precision mode limits the error as a ratio of the range of values in the array to be compressed. The fixed-rate mode achieves a guaranteed compression ratio requested by the user, but does not provide any bounds on accuracy loss.

The fixed-rate mode could make our implementation more straightforward by offering a predictable size of compressed checkpoints, but the lack of error bounds makes this option less attractive. Moreover, ZFP claims to achieve the best "compression efficiency" in the fixed-tolerance mode, and we thus chose to focus on this mode.

SZ [10] is a more recently developed compression library, also focussed on lossy compression of floating-point scientific data, also developed in C. While we have also written a python wrapper for the reference implementation of SZ[2], a thorough comparison of ZFP and SZ remains future work.

## 3 Checkpointing Performance Model

As previously mentioned, checkpointing is a strategy to store selected timesteps, and recompute others when needed. The question which checkpoints should be stored to get the best tradeoff between recomputation time and memory footprint was answered in a provably optimal way by the Revolve checkpointing algorithm [11]. Revolve makes certain assumptions, for example that all timesteps have the same compute cost and storage size, the number of timesteps is known a priori, and there is only one level of memory (e.g. RAM) that is restricted in size, but very fast. Other authors have subsequently developed extensions to Revolve that are optimal under different conditions [4,19]. We focus in this paper on the classic Revolve algorithm, and store all checkpoints in RAM.

In this section, we build on the ideas introduced in [19] to build a performance model that predicts the runtime of an adjoint computation using Revolve checkpointing. We call the time taken by a single forward computational step $C_F$ and correspondingly, the time taken by a single backward step $C_R$. For a simulation with $\mathbf{N}$ timesteps, the minimum wall time required for the full forward-adjoint evaluation is given by

$$T_N = \mathbf{C_F} \cdot \mathbf{N} + \mathbf{C_R} \cdot \mathbf{N} \tag{1}$$

If the size of a single timestep in memory is given by $\mathbf{S}$, this requires a memory of at least size $\mathbf{S} \cdot \mathbf{N}$. If sufficient memory is available, no checkpointing or compression is needed.

If the memory is smaller than $\mathbf{S} \cdot \mathbf{N}$, Revolve provides a strategy to solve for the adjoint field by storing a subset of the $\mathbf{N}$ total checkpoints and recompute

[1] To be released open source on publication.
[2] Also to be released open source upon publication.

the remaining ones. The overhead introduced by this method can be broken down into the recomputation overhead $\mathbf{O}_R$ and the storage overhead $\mathbf{O}_S$. The recomputation overhead is the amount of time spent in recomputation, given by

$$\mathbf{O}_R(N, M) = p(N, M) \cdot \mathbf{C_F}, \tag{2}$$

where $p(N, M)$ is the minimum number of recomputed steps from [11], given as

$$p(N, M) = \begin{cases} N(N-1)/2, & \text{if } M = 1 \\ \min\limits_{1<=\widetilde{N}<=N} \{\widetilde{N} + p(\widetilde{N}, M) + p(N - \widetilde{N}, M - 1)\}, & \text{if } M > 1 \end{cases} \tag{3}$$

where $M$ is the number of checkpoints that can be stored in memory. Note that for $M \geq N$, $\mathbf{O}_R$ would be zero. For $M < N$, $\mathbf{O}_R$ grows rapidly as M is reduced relative to N.

In an ideal implementation, the storage overhead $\mathbf{O}_S$ might be zero, since the computation could be done "in-place", but in practice, checkpoints are generally stored in a separate section of memory and they need to be transferred to a "computational" section of the memory where the computation is performed, and then the results copied back to the checkpointing memory. This copying is a common feature of checkpointing implementations, and might pose a non-trivial overhead when the computation involved in a single timestep is not very large. This storage overhead is given by:

$$\mathbf{O}_{SR}(N, M) = \mathbf{W}(N, M) \cdot \frac{\mathbf{S}}{\mathbf{B}} + \mathbf{N} \cdot \frac{\mathbf{S}}{\mathbf{B}} \tag{4}$$

where $\mathbf{W}$ is the total number of times Revolve writes checkpoints for a single run, $\mathbf{N}$ is the number of times checkpoints are read, and $\mathbf{B}$ is the bandwidth at which these copies happen. The total time to solution becomes

$$T_R = \mathbf{C_F} \cdot \mathbf{N} + \mathbf{C_R} \cdot \mathbf{N} + \mathbf{O}_R(N, M) + \mathbf{O}_{SR}(N, M) \tag{5}$$

## 4 Performance Model Including Compression

By using compression, the size of each checkpoint is reduced and the number of checkpoints available is increased ($M$ in Eq. 3). This reduces the recomputation overhead $\mathbf{O}_R$, while at the same time adding overheads related to compression and decompression in $\mathbf{O}_S$. To be beneficial, the reduction in $\mathbf{O}_R$ must offset the increase in $\mathbf{O}_{SR}$, leading to an overall decrease in the time to solution $T$.

Our performance model assumes that the compression algorithm behaves uniformly across the different time steps of the simulation, i.e. that we get the same compression ratio, compression time and decompression time, no matter which of the $N$ possible checkpoints we try to compress/decompress. The storage overhead now becomes

$$\begin{aligned} \mathbf{O}_{SR}(N, M) =& \mathbf{W}(N, M \cdot F) \cdot \left( \frac{\mathbf{S}}{\mathbf{F} \cdot \mathbf{B}} + t_c \right) \\ &+ \mathbf{N} \cdot \left( \frac{\mathbf{S}}{\mathbf{F} \cdot \mathbf{B}} + t_d \right) \end{aligned} \tag{6}$$

where **F** is the compression ratio (i.e. the ratio between the uncompressed and compressed checkpoint), and $t_c$ and $t_d$ are compression and decompression times, respectively. At the same time, the recomputation overhead decreases because **F** times more checkpoints are now available.

## 5 Acceptable Errors and Convergence

Our performance model is agnostic of the specific optimization problem being solved. We envision it being used in a generic checkpointing runtime that manages the checkpointed execution of an optimization problem, and accepts an acceptable error tolerance as an input parameter for each gradient evaluation and determines whether or not compression can pay off for that iteration. For this reason, we do not discuss in this paper whether or not a certain accuracy is acceptable for any given application.

We note that there is some previous work in this area, discussing for example the effect of bounded pointwise errors in a multi-dimensional field on computed numerical derivatives, for ZFP [1]. In the context of seismic inversion, other work discusses accuracy requirements in optimization loops, and notes that high accuracy is only needed when already close to a minimum [6,13]. There has been previous work on choosing the most appropriate compression algorithm under some circumstances [21]., and work that addresses convergence guarantees of trust-region based optimization methods in the presence of gradients that are only known with a probability $p$ [5].

Despite all this previous work, for most practical adjoint optimization applications, the relationship between accuracy (whether caused by roundoff, compression or truncation errors) and convergence remains a field of ongoing research.

## 6 Problem and Test Case

We use Devito [15] to solve forward and adjoint wave equation problems. Devito is a domain-specific language that enables the rapid development of finite-difference solvers from a high-level description of partial differential equations. The simplest version of the seismic wave equation is the acoustic isotropic wave equation defined as:

$$m(x)\frac{\partial^2 u(t,x)}{\partial t^2} - \nabla^2 u(t,x) = q(t,x), \tag{7}$$

where $m(x) = \frac{1}{c^2(x)}$ is the squared slowness, $c(x)$ the spatially dependent speed of sound, $u(t,x)$ is the pressure wavefield, $\nabla^2 u(t,x)$ denotes the laplacian of the wavefield and $q(t,x)$ is a source term.

The solution to Eq. 7 forms the forward problem. The seismic inversion problem minimizes the misfit between simulated and observed signal given by:

$$\min_m \phi_s(m) = \frac{1}{2}\left\|d_{sim} - d_{obs}\right\|_2^2. \tag{8}$$

We call the kernel derived from a basic finite difference formulation of Eq. 7, the OT2 kernel because it is second-order accurate in time. We also use another formulation from [15], which is 4th-order accurate in time. We call this the OT4 kernel.

This optimization problem is usually solved using gradient based methods such as steepest descent, where the gradient is computed using the adjoint-state method.

The values of $m(x)$ used in this work are derived from the Overthrust model [3] over a grid of $287 \times 881 \times 881$ points, including an absorbing layer of 40 points on each side. The grid spacing is 25 m in space. The propagation time is 4 s that corresponds to 2526 timesteps. The wave field at the final time is shown in Fig. 2a. The uncompressed size of this single time step field is just under 900 MB. If one were to store all the timesteps, this would require 2.3 TB of memory.

To implement Revolve with Devito, we use pyRevolve [12] which is a python library to manage the execution of checkpointed adjoint computations. The performance model in Sect. 3 assumes that the implementation is similar to pyRevolve, which stores a checkpoint by copying a portion of the operator's working memory to the checkpointing memory and similarly loads a checkpoint by copying from the checkpointing memory to the operator's working memory.

For benchmarking we used a dual-socket Intel(R) Xeon(R) Platinum 8180M @ 2.50 Ghz (28 cores each) (skylake).

## 7 Results and Discussion

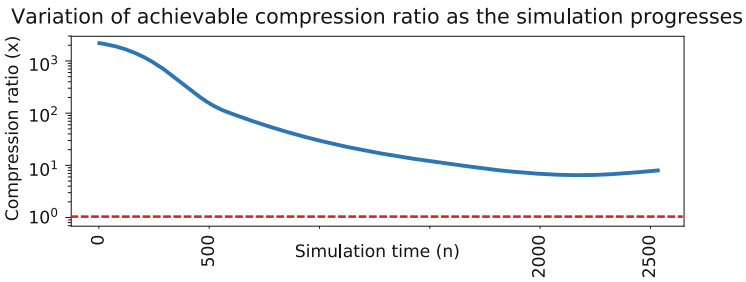



**Fig. 1.** Compression ratios achieved on compressing different time steps. Every timestep from 1 to 2526 was compressed and plotted.

To understand the compressibility of the data produced in a typical wave-propagation simulation, we ran a simulation as per the setup described in Sect. 6, and tried to compress every single timestep. For this we chose ZFP in fixed tolerance mode at some arbitrary tolerance level. We noted the compression ratios achieved at every timestep. As Fig. 1 shows, the initial timesteps are much easier to compress than the later ones. This is not surprising since most wave simulations start with the field at rest, i.e. filled with zeros. As the wave reaches more

parts of the domain, the field becomes less compressible until it achieves a stable state when the wave has reached most of the domain.

If the simulation had started with the field already oscillating in a wave, it is likely that the compressibility curve for that simulation would be flat. This tells us that the compressibility of the last timestep of the solution is representative of the worst-case compressibility and hence we used the last timestep as our reference for comparison of compression in the rest of the analysis.

**Table 1.** Some results from trying out all possible compressors and settings in blosc. We selected the best compression ratio seen for each compressor. "Setting" here is the choice between speed and compression, where 0 is fastest and 9 is highest compression.

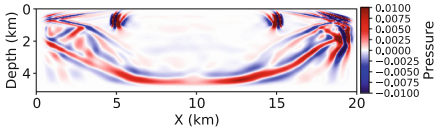
| Compressor | Chunk size (bytes) | Shuffle Mode | Setting | Compression time (ms) | Decompression time (ms) | Compression Ratio |
|---|---|---|---|---|---|---|
| BloscLZ | 1048576 | SHUFFLE | 6 | 4249.44 | 1288.86 | 1.188 |
| LZ4 | 2965280 | SHUFFLE | 4 | 1371.26 | 920.98 | 1.199 |
| LZ4HC | 2097152 | SHUFFLE | 8 | 31245.16 | 926.69 | 1.265 |
| ZLib | 524288 | SHUFFLE | 7 | 30218.81 | 2470.04 | 1.291 |
| ZStd | 524288 | SHUFFLE | 9 | 117238.76 | 1477.34 | 1.312 |

Table 1 shows the compression ratios and times for a few different lossless compressors and their corresponding settings. As can be seen, the compression factors achieved, and the time taken to compress and decompress can vary significantly, but it is hard to say whether this compression could be used to speed up the inversion problem.

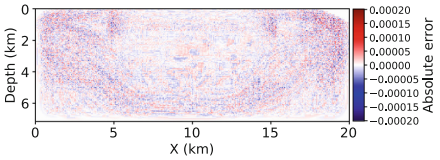
Figure 3a shows compression ratios for different tolerance settings for the fixed-tolerance mode of ZFP. The point highlighted here was the setting used to compress all timesteps in Fig. 1. Figure 2b shows the spatial distribution of the errors after compression and decompression, compared to the original field, for this setting. Table 3b shows the effect of different levels of pointwise absolute error on the overall error in the gradient evaluation. We can see that the error in the gradient evaluation does not explode.

To validate the revolve-only performance model, Fig. 4a shows the predicted runtime for a variety of peak memory constraints along with measured runtime for the same scenario. Figure 4b shows a comparison of predicted and measured runtimes for the OT2 kernel with compression enabled. Figure 4c repeats this experiment for the OT4 kernel which has a higher computational complexity. It can be seen that the model is able to predict the real performance very closely in all three cases.

We have now seen that the performance model from Sect. 4 is effective at predicting the runtime of adjoint computations. To study the performance model, we first visualize it along the axis of available memory, comparing the predicted performance of the chosen compression scheme with the predicted performance of a Revolve-only adjoint implementation. This is shown in Fig. 5 where we

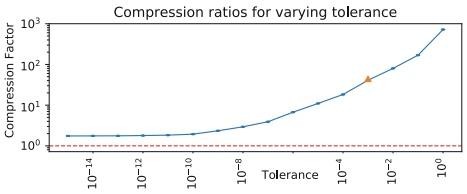



(a) Reference wavefield for compression and decompression.




(b) Errors introduced during compression and decompression using the fixed-tolerance mode.

**Fig. 2.** This field was formed after a Ricker wavelet source was placed at the surface of the model and the wave propagated for 2500 timesteps. This is a vertical (x-z) cross-section of a 3D field, taken at the $y$ source location. It is interesting to note that the errors are more or less evenly distributed across the domain with only slight variations corresponding to the wave amplitude (from Figure a). A small block-like structure characteristic of ZFP can be seen.




(a) Effect of tolerance on Compression Ratio

| Tolerance | Gradient error |
|---|---|
| 0.1 | 662.905 |
| 0.01 | 70.619 |
| 0.001 | 10.485 |
| 0.0001 | 0.763 |
| $10^{-5}$ | 0.194 |
| $10^{-6}$ | 0.154 |
| $10^{-7}$ | 0.151 |

(b) Effect of tolerance on Gradient error

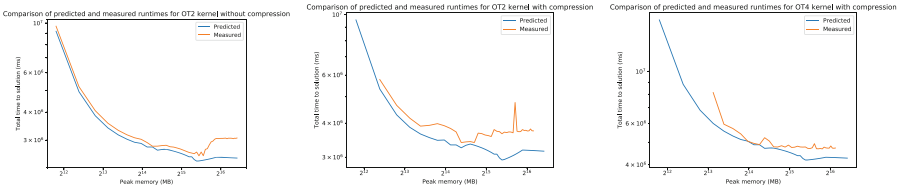
**Fig. 3.** Effect of tolerance settings of ZFP in fixed-tolerance mode on Compression ratio (left) and final gradient evaluation (right). We define compression ratio as the ratio between the size of the uncompressed data and the compressed data. The dashed line represents no compression. The highlighted point corresponds to the setting used for the other results here unless otherwise specified. The gradient error (right) is the 2-norm of the error tensor in the gradient, as compared with an exact computation.

can distinguish three different scenarios, depending on the amount of available memory.

1. If the memory is insufficient even with compression to store the entire trajectory, one can either use checkpointing only, or combine checkpointing with compression. This is the left section of the figure.
2. If the available memory is not sufficient to store the uncompressed trajectory, but large enough to store the entire compressed trajectory, we compare two possible strategies: Either use compression only, or use checkpointing only. This is the middle section of the figure.
3. If the available system memory is large enough to hold the entire uncompressed trajectory, neither compression nor checkpointing is necessary. This is the right section of the figure.

The second scenario was studied in previous work [7], while the combined method is also applicable to the first scenario, for which previous work has only used checkpointing without compression.

We can identify a number of factors that make compression more likely to be beneficial compared to pure checkpointing: A very small system memory size and a large number of time steps lead to a rapidly increasing recompute factor,




(a) OT2, No compression (b) OT2, Compression (c) OT4, Compression

**Fig. 4.** Predicted vs measured runtimes for two different kernels (OT2 and OT4), with and without compression. This shows that the performance model can predict the runtime effectively. The compression setting used was ZFP with absolute error tolerance set to $10^{-6}$

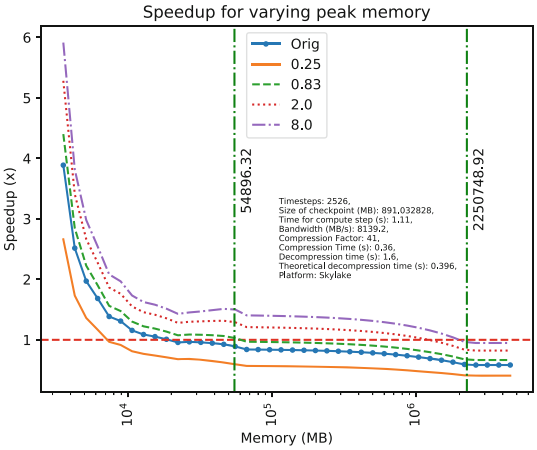
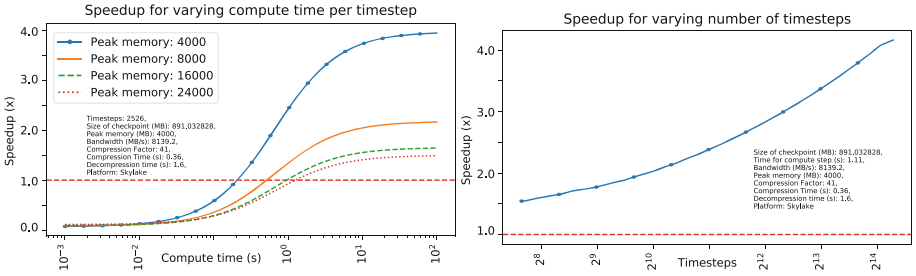



**Fig. 5.** The speedups predicted by the performance model for varying memory. The baseline (1.0) is the performance of a Revolve-only implementation under the same conditions. The different curves represent kernels with differing compute times (represented here as a factor of the sum of compression and decompression times). The first vertical line at 53 GB marks the spot where the compressed wavefield can completely fit in memory and Revolve is unnecessary if using compression. The second vertical line at 2.2 TB marks the spot where the entire uncompressed wavefield can fit in memory and neither Revolve nor compression is necessary. The region to the right is where these optimizations are not necessary or relevant. The middle region has been the subject of past studies using compression in adjoint problems. The region to the left is the focus of this paper.

and compression can substantially reduce this recompute factor. This can be seen in Figs. 5 and 6b.




(a) Varying Compute (b) Varying timesteps

**Fig. 6.** The speedups predicted by the performance model for varying compute cost (left) and number of timesteps (right). The baseline (1.0) is the performance of a Revolve-only implementation under the same conditions. The benefits of compression drop rapidly if the computational cost of the kernel that generated the data is much lower than the cost of compressing the data. For increasing computational costs, the benefits are bounded. It can be seen that compression becomes more beneficial as the number of timesteps is increased.

The extent to which the recompute factor affects the overall runtime also depends on the cost to compute each individual time step. If the compute cost per time step is large compared to the compression and decompression cost, then compression is also likely to be beneficial, as shown in Fig. 6a. As the time per time step increases and the compression cost becomes negligible, we observe that the ratio between the runtime of the combined method and that of pure checkpointing is only determined by the difference in recompute factors.

## 8 Conclusions and Future Work

We used compression to reduce the computational overhead of checkpointing in an adjoint computation used in seismic inversion. We developed a performance model that computes whether or not the combination of compression and checkpointing will outperform pure checkpointing or pure compression in a variety of scenarios, depending on the available memory size, computational intensity of the application, and compression ratio and throughput of the compression algorithm. In future work, we plan to extend this work by

- further exploring the relationship between pointwise error bounds in compression and the overall error of the adjoint gradient evaluation,
- extending our performance model to support non-uniform compression ratios, as would be expected for example if the initial wave field is smoother and therefore more easily compressible,

- studying strategies where different compression settings (or even no compression) is used for a subset of time steps,
- exploring compression and multi-level checkpointing, including SSD or hard drives in addition to RAM storage,
- and finally by developing checkpointing strategies that are optimal even if the size of checkpoints post-compression varies and is not known a priori.

**Acknowledgments.** This work was funded by the Intel Parallel Computing Centre at Imperial College London and EPSRC EP/R029423/1. This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics and Computer Science programs under contract number DE-AC02-06CH11357. We would also like to acknowledge the support from the SINBAD II project and the member organizations of the SINBAD Consortium.

We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

This paper benefited from discussions with Kaiyuan Huo, Fabio Luporini, Thomas Matthews, Paul Kelly, Oana Marin.

## References

1. https://computation.llnl.gov/projects/floating-point-compression/zfp-and-derivatives
2. Alted, F.: Why modern cpus are starving and what can be done about it. Comput. Sci. Eng. **12**(2), 68 (2010)
3. Aminzadeh, F., Burkhard, N., Long, J., Kunz, T., Duclos, P.: Three dimensional SEG/EAGE models–an update. Lead. Edge **15**(2), 131–134 (1996)
4. Aupy, G., Herrmann, J., Hovland, P., Robert, Y.: Optimal multistage algorithm for adjoint computation. SIAM J. Sci. Comput. **38**(3), C232–C255 (2016)
5. Blanchet, J., Cartis, C., Menickelly, M., Scheinberg, K.: Convergence rate analysis of a stochastic trust region method for nonconvex optimization. arXiv preprint arXiv:1609.07428 (2016)
6. Boehm, C., Hanzich, M., de la Puente, J., Fichtner, A.: Wavefield compression for adjoint methods in full-waveform inversion. Geophysics **81**(6), R385–R397 (2016)
7. Cyr, E.C., Shadid, J., Wildey, T.: Towards efficient backward-in-time adjoint computations using data compression techniques. Comput. Methods Appl. Mech. Eng. **288**, 24–44 (2015)
8. Dalmau, F.R., Hanzich, M., de la Puente, J., Gutiérrez, N.: Lossy data compression with DCT transforms. In: EAGE Workshop on High Performance Computing for Upstream (2014)
9. Datta, D., Appelhans, D., Evangelinos, C., Jordan, K.: An asynchronous two-level checkpointing method to solve adjoint problems on hierarchical memory spaces. Comput. Sci. Eng. **20**(4), 39–55 (2018)
10. Di, S., Tao, D., Liang, X., Cappello, F.: Efficient lossy compression for scientific data based on pointwise relative error bound. IEEE Trans. Parallel Distrib. Syst. **30**(2), 331–345 (2018)
11. Griewank, A., Walther, A.: Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. ACM Trans. Math. Softw. (TOMS) **26**(1), 19–45 (2000)

12. Kukreja, N., Hückelheim, J., Lange, M., Louboutin, M., Walther, A., Funke, S.W., Gorman, G.: High-level python abstractions for optimal checkpointing in inversion problems. arXiv preprint arXiv:1802.02474 (2018)
13. van Leeuwen, T., Herrmann, F.J.: 3d frequency-domain seismic inversion with controlled sloppiness. SIAM J. Sci. Comput. **36**(5), S192–S217 (2014)
14. Lindstrom, P.: Fixed-rate compressed floating-point arrays. IEEE Trans. Visual Comput. Graphics **20**(12), 2674–2683 (2014)
15. Louboutin, M., Lange, M., Luporini, F., Kukreja, N., Witte, P.A., Herrmann, F.J., Velesko, P., Gorman, G.J.: Devito: an embedded domain-specific language for finite differences and geophysical exploration. CoRR abs/1808.01995, August 2018. https://arxiv.org/abs/1808.01995
16. Marin, O., Schanen, M., Fischer, P.: Large-scale lossy data compression based on an a priori error estimator in a spectral element code. Technical report, ANL/MCS-P6024-0616 (2016)
17. O'Neil, M.A., Burtscher, M.: Floating-point data compression at 75 gb/s on a GPU. In: Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units. ACM (2011). https://doi.org/10.1145/1964179.1964189
18. Plessix, R.E.: A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. Geophys. J. Int. **167**(2), 495–503 (2006)
19. Stumm, P., Walther, A.: Multistage approaches for optimal offline checkpointing. SIAM J. Sci. Comput. **31**(3), 1946–1967 (2009)
20. Symes, W.W.: Reverse time migration with optimal checkpointing. Geophysics **72**(5), SM213–SM221 (2007)
21. Tao, D., Di, S., Liang, X., Chen, Z., Cappello, F.: Optimizing lossy compression rate-distortion from automatic online selection between sz and zfp. arXiv preprint arXiv:1806.08901 (2018)