Inversion with Devito: Trading off memory and compute with PyRevolve

Navjot Kukreja[1] (With contributions from a lot of people!)

STMI Workshop
Research Centre for Gas Innovation
Universidade de São Paulo

October 3, 2019

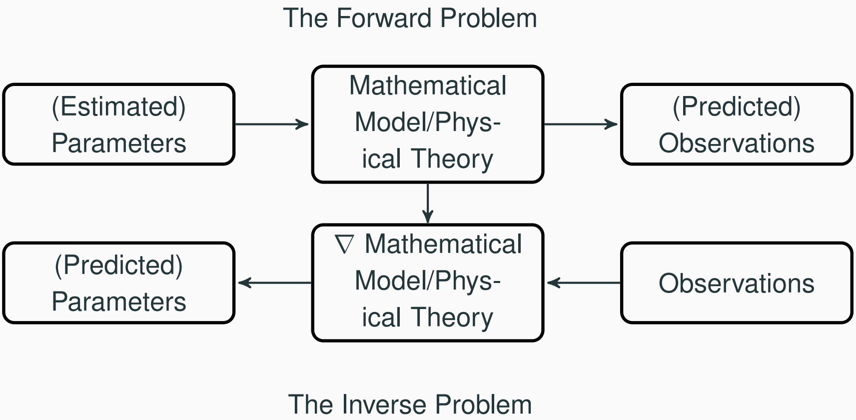
[1]Department of Earth Science and Engineering, Imperial College London, UK

Introduction

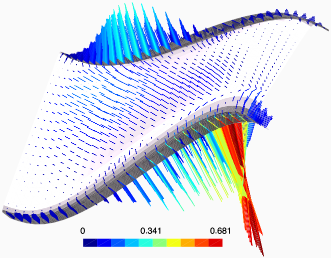
Checkpointing

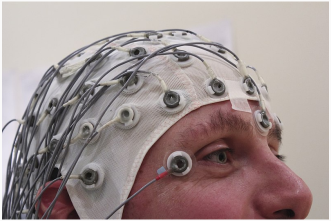
Compression

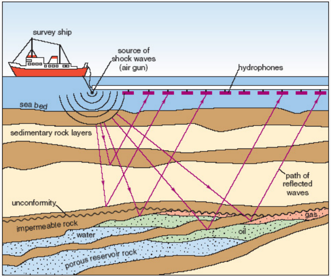
Inverse Problems

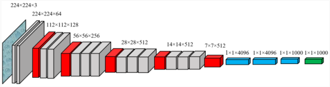


Applications

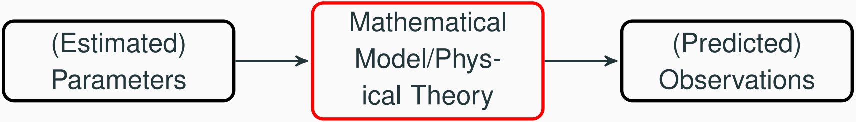








The Forward Problem

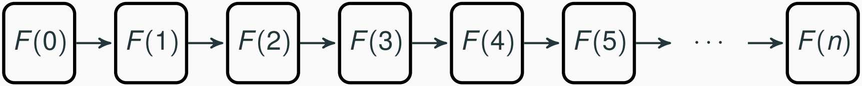


Devito[1]

[1] Navjot Kukreja et al. "Devito: Automated fast finite difference computation". In: *2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC)*. IEEE. 2016, pp. 11–19.

Data flow

Data flow for forward problem:



Raising the abstraction with Devito

$$\begin{cases} m\frac{d^2 u(x,t)}{dt^2} - \nabla^2 u(x,t) = q_s \\ u(.,0) = 0 \\ \frac{du(x,t)}{dt}\Big|_{t=0} = 0 \end{cases}$$

↓

```
pde = m * u.dt2 - u.laplace
stencil = Eq(u.forward, solve(pde, u.forward)[0])
fwd_op = Operator([stencil], ...)
```

↓

```
void finite_difference_solver(...) {
  //...impenetrable "performance␣optimised" code
}
```

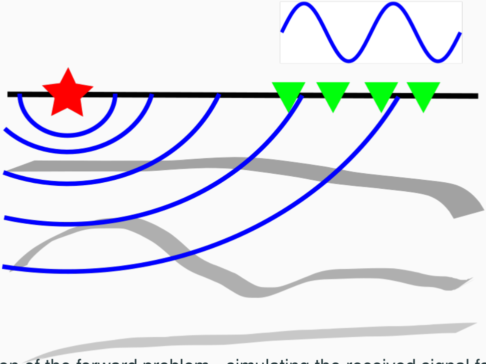
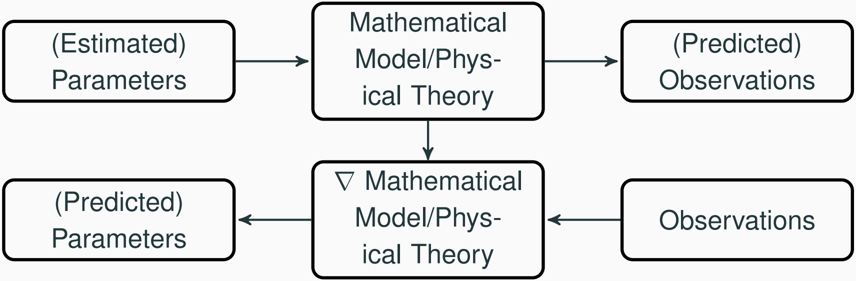


Figure 1: Illustration of the forward problem - simulating the received signal for a given structure

Inverse Problems

The Forward Problem



The Inverse Problem

Full Waveform Inversion

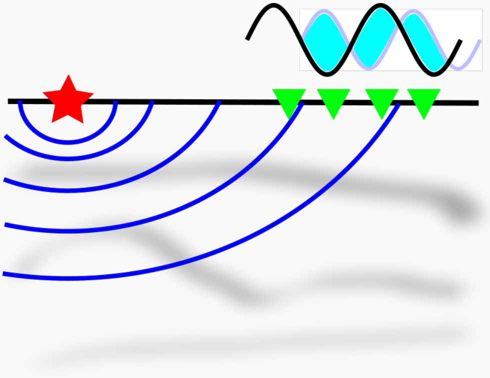


Figure 2: Illustration of full waveform inversion - initial guess

Full Waveform Inversion

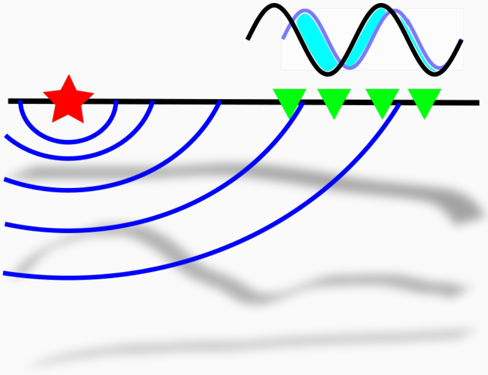


Figure 3: Illustration of full waveform inversion - in progress

Full Waveform Inversion

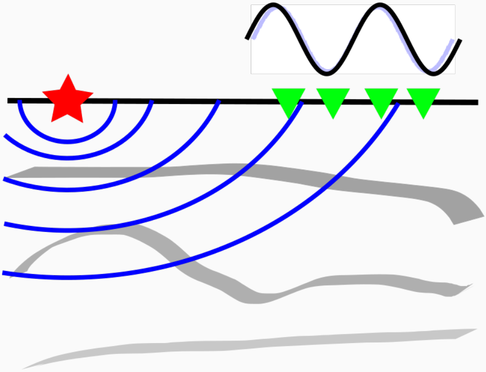
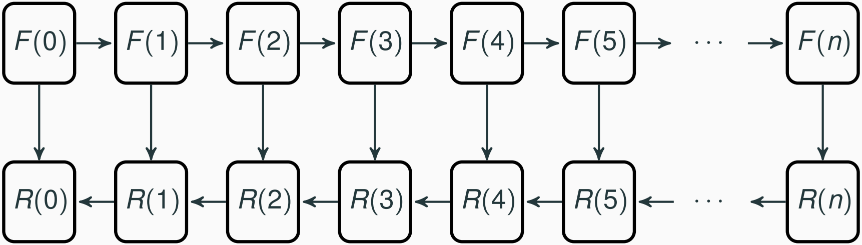


Figure 4: Illustration of full waveform inversion - convergence

Data flow

Data flow for gradient calculation:



Adjoint mode - store all timesteps

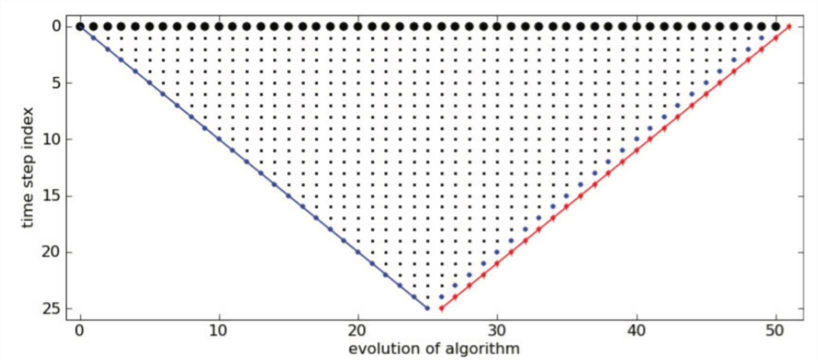


Figure 5: Progression of the adjoint computation with wall-clock time on the x-axis and simulation time on the y-axis. Each vertical cross-section represents the status at that time. The dots represent checkpoints stored in memory - here a checkpoint is stored at each time step. Image Source:[2]

[2] Qiqi Wang, Parviz Moin, and Gianluca Iaccarino. "Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation". In: *SIAM Journal on Scientific Computing* 31.4 (2009), pp. 2549–2567.

Dealing with high memory requirements

- For a typical 3D problem [3] the grid has $287 \times 881 \times 881$ points in single precision, meaning each timestep is 900 MB.
- Such a problem would be run for 2500 timesteps, meaning the total memory required for a *naive* adjoint run would be 2.3 TB.
- One strategy to fit this into existing computer architectures would be domain decomposition
  - Might end up wasting computational power in order to use more memory
  - Communication overhead might start dominating soon, especially in a cloud environment
- Another technique, that is domain-specific, is saving the wavefields only on the boundaries of the domain and recomputing from there
  - Only works for a very small number of cases, i.e. where the equation is time-reversible
- Compression
- Checkpointing

[3] e.g. Overthrust model

Introduction

Checkpointing

Compression

Adjoint mode - checkpointed

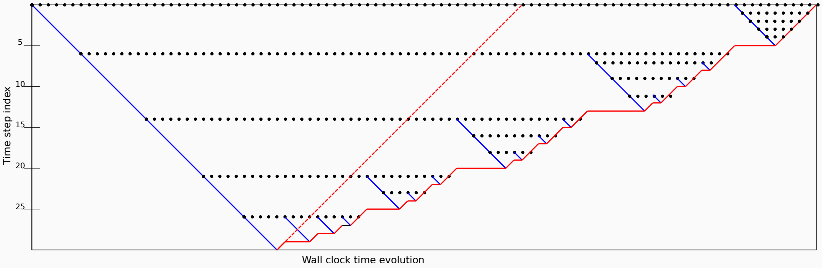


Figure 6: Progression of the adjoint computation with wall-clock time on the x-axis and the simulation time on the y-axis. In this case the number of checkpoints is less than the timesteps, hence there is some recomputation involved.

Checkpointing - Revolve

For the problem where:

1. Number of steps is known in advance
2. Only one level of memory available
3. Checkpoint sizes are uniform
4. Saving/retrieving a checkpoint takes no time
5. Computational cost of the steps is uniform
6. Cost of restarting operators is zero

the optimal algorithm, Revolve, was given by[4]. Given a certain number of steps and a given amount of memory, Revolve provides the start-stop-restart schedule that minimises the amount of recomputation.

---

[4] Andreas Griewank and Andrea Walther. "Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation". In: *ACM Transactions on Mathematical Software (TOMS)* 26.1 (2000), pp. 19–45.

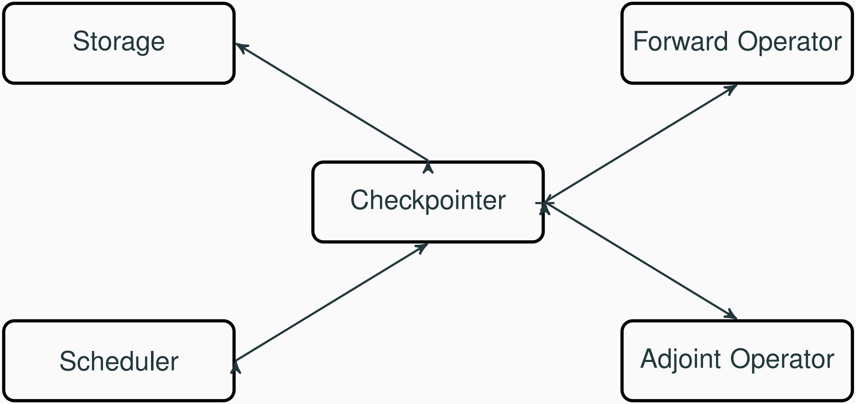
Checkpointing - Separation of concerns

Given the different kinds of checkpointing algorithms that apply to different kinds of problems and in different scenarios, it makes sense to have a library/tool manage checkpointing for Separation of Concerns.

PyRevolve[5]

[5] Navjot Kukreja et al. "High-level python abstractions for optimal checkpointing in inversion problems". In: *arXiv preprint arXiv:1802.02474* (2018).

PyRevolve


Storage
Forward Operator
Checkpointer
Scheduler
Adjoint Operator

Modelling the performance of Revolve

For a simple forward-adjoint computation with no recomputation, time to solution would be:

$$T_N(N) = 2 \cdot C \cdot N \tag{1}$$

where C is the time taken to compute a single timestep of either the forward or the adjoint mode (assume they're the same for now) and N is the number of timesteps. Revolve introduces overheads:

- O_R the overhead due to the recomputation involved
- O_S the overhead due to repeatedly storing/loading checkpoints

Time to solution under Revolve is hence:

$$T_R(N, M) = 2 \cdot C \cdot N + O_R(N, M) + O_S(N, M) \tag{2}$$

Modelling the performance of Revolve

The storage overhead IS ZERO! (according to[6])

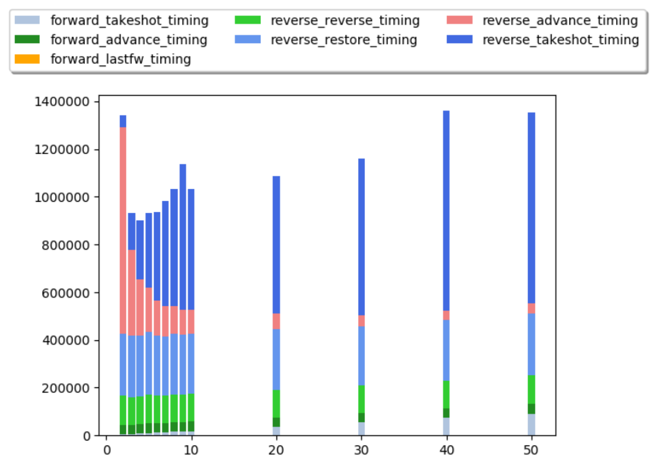


Figure 7: Time spent in different actions during a Revolve-based forward-adjoint computation

[6]Griewank and Walther, "Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational

Keeping the storage overhead in mind

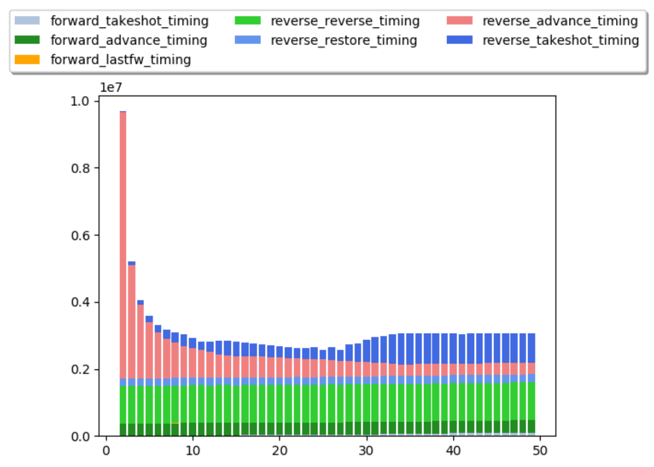


Figure 8: Updated Revolve timings after a modification to remove (some) redundant copies

Modelling the performance of Revolve

We can already use this simple performance model to answer some questions: If we have slow but infinite memory, how many checkpoints should we store?

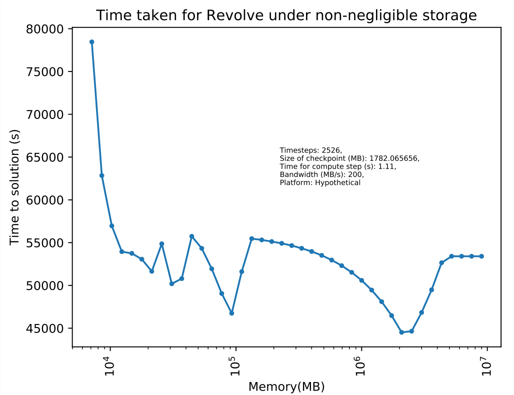


Figure 9: Time to solution when using different amounts of memory in Revolve, but when dealing with slow memory

Memory-compute tradeoffs

- Revolve/Checkpointing is a way to trade off memory and compute. So is compression.
- Compression has been used to compress the entire forward trajectory in past work.
- With domain-specific compression algorithms (we live inside a DSL so do not want to make too many assumptions about our problem)
- But which one is a better choice?
- Checkpointing + Compression can allow you to store more checkpoints, hence do less recomputation. i.e. can we combine the two?
- Is any of this even worth the trouble?

Trying Compression

Is **F** (compression ratio) constant? I tried ZFP[7] to find out.

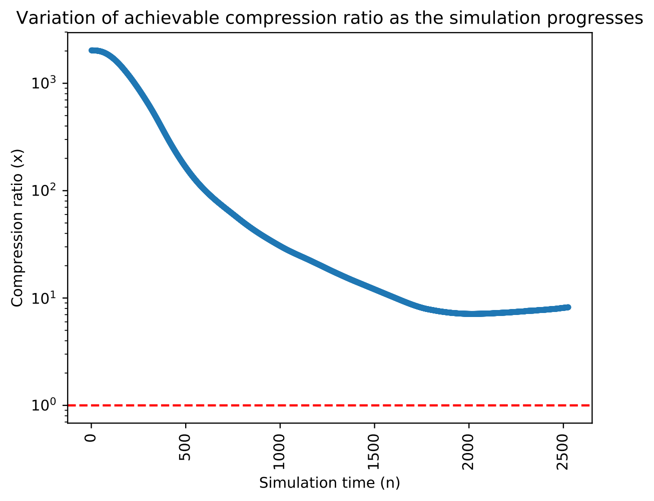


Figure 10: Compression ratios achieved when I tried to compress every timestep of a seismic problem setup.

[7] Peter Lindstrom. “Fixed-rate compressed floating-point arrays”. In: *IEEE transactions on visualization and computer graphics* 20.12 (2014), pp. 2674–2683.

Reference wavefield

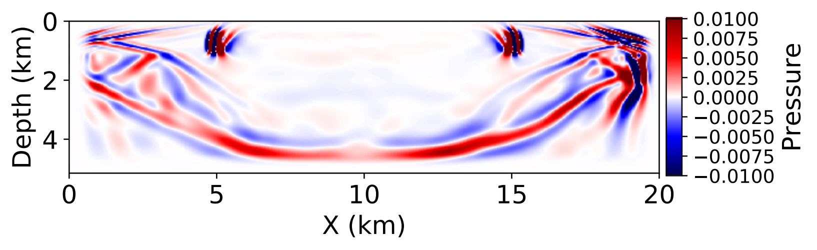


Figure 11: Cross-section of the wavefield used as a reference sample for compression and decompression. This field was formed after a Ricker wavelet source was placed at the surface of the model and the wave propagated for 2500 timesteps. This is a vertical (x-z) cross-section of a 3D field, taken at the y source location

Lossless 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: 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.

Tolerance settings within ZFP

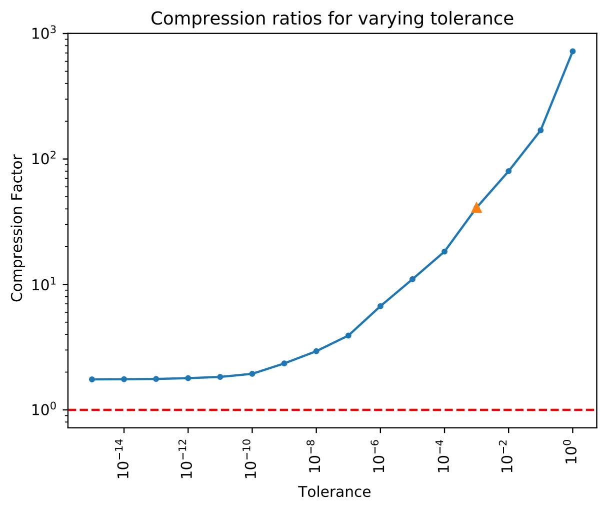


Figure 12: Compression ratios achieved on compressing the wavefield. We define compression ratio as the ratio between the size of the uncompressed data and the compressed data. The dashed line represents no compression. The highlighted point corresponds to the setting used for the other results here unless otherwise specified.

Errors introduced during compression-decompression

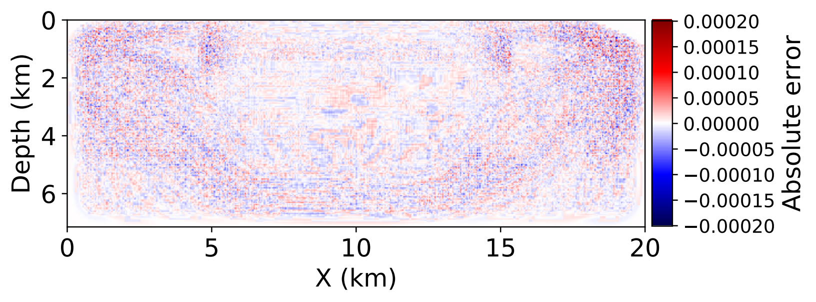


Figure 13: Cross-section of the field that shows errors introduced during compression and decompression using the fixed-tolerance mode. It is interesting to note that the errors are more or less evenly distributed across the domain with only slight variations corresponding to the wave amplitude. A small block-like structure characteristic of ZFP can be seen.

Checkpoint error impact

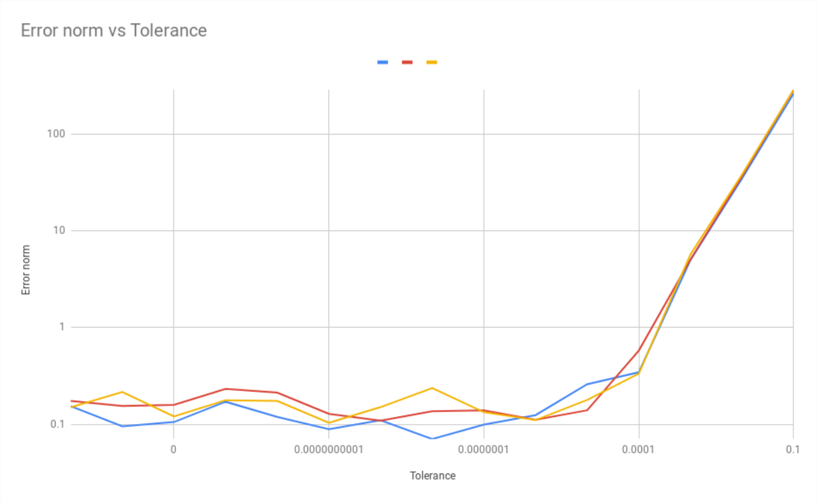


Figure 14: L2 norm of gradient error for different settings of lossy compression

Compression-Revolve performance model

The performance model can now be extended to include compression. With compression, the storage overhead goes up:

$$\mathbf{O}_{SC}(N,M) = \mathbf{W}(N, M \cdot F) \cdot \left(\frac{2 \cdot \mathbf{S}}{\mathbf{F} \cdot \mathbf{B}} + t_c\right) + N \cdot \left(\frac{2 \cdot \mathbf{S}}{\mathbf{F} \cdot \mathbf{B}} + t_d\right) \qquad (3)$$

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.

Understanding the model

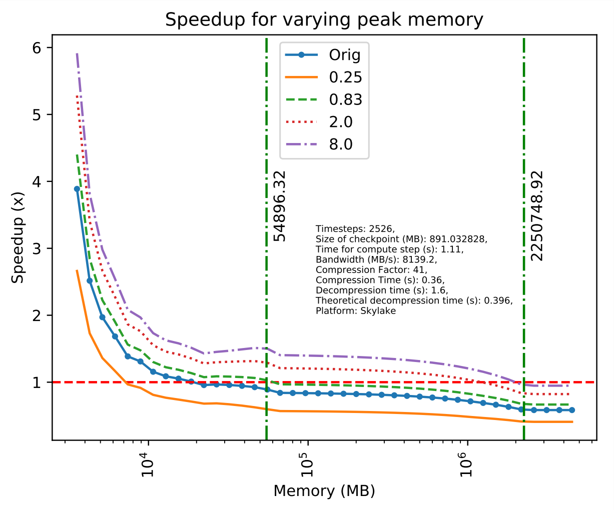


Figure 15: The speedups predicted by the performance model for varying memory. The baseline (1.0) is the performance of a Revolve-only implementation under the same conditions. The different curves represent kernels with differing compute times (represented here as a factor of the sum of compression and decompression times).

Future work

- Compression
  - Optimal scheduling under compression
  - Also include SZ in the study
  - Study acceptable error tolerances (application dependent)
  - Extend to multi-level checkpointing
- Complex data dependencies (e.g. higher order in time, subsampling)
- Cost of restarting operators

Thank you

Thank you [8]

Questions?

[8] This work was funded by the Intel Parallel Computing Centre at Imperial College London and EPSRC EP/R029423/1

Problem setup

- Grid size: $881 \times 881 \times 287$
- SEG Overthrust model
- Ricker source placed in the x-y centre of the domain, just below the surface
- 2500 timesteps

Scheduling strategy

One of the assumptions of Revolve is that all checkpoints are the same size. This is clearly not true under (lossy) compression. Hence this breaks the optimality of Revolve. One possible suggestion to improve this schedule is:

- Use a(n) (underestimating) heuristic for compression ratio (**F**) for a checkpoint at a given timestep
- Report this pessimistic **M** to Revolve, one checkpoint at a time, to get the location of the next checkpoint.
- When storing checkpoint, track how much memory left after actual compression to calculate a new **M** for Revolve.

Understanding the model

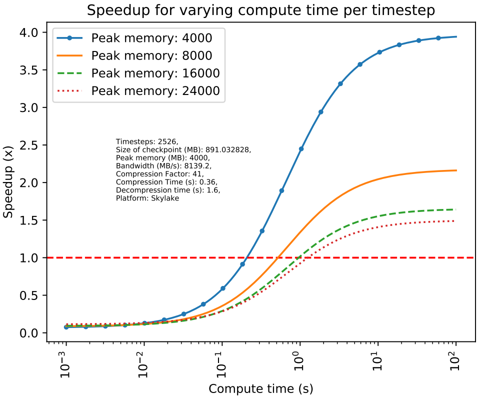


Figure 16: The speedups predicted by the performance model for varying compute cost. The baseline (1.0) is the performance of a Revolve-only implementation under the same conditions. The benefits of compression drop rapidly if the computational cost of the kernel that generated the data is much lower than the cost of compressing the data. For increasing computational costs, the benefits are bounded.

Understanding the model

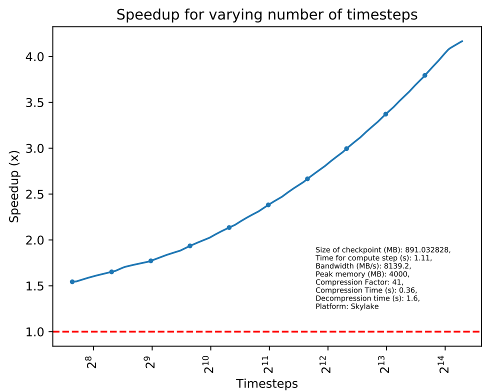


Figure 17: The speedups predicted by the performance model for varying number of timesteps to be reversed. The baseline (1.0) is the performance of a Revolve-only implementation under the same conditions. It can be seen that compression becomes more beneficial as the number of timesteps is increased.

Checkpointing - Multistage

For the problem where:

1. Number of steps is known in advance
2. ~~Only one level of memory available~~
3. Checkpoint sizes are uniform
4. Saving/retrieving a checkpoint (to first level memory) takes no time (zero-cost checkpointing)
5. Computational cost of the steps is uniform
6. Cost of restarting operators is zero

the optimal algorithm was given by[9].

[9] Guillaume Aupy et al. "Optimal multistage algorithm for adjoint computation". In: *SIAM Journal on Scientific Computing* 38.3 (2016), pp. C232–C255.

Checkpointing - Online Multistage

For the problem where:

1. ~~Number of steps is known in advance~~
2. ~~Only one level of memory available~~
3. Checkpoint sizes are uniform
4. Saving/retrieving a checkpoint (to first level memory) takes no time (zero-cost checkpointing)
5. Computational cost of the steps is uniform
6. Cost of restarting operators is zero

the optimal algorithm was given by[10] and[11].

---

[10] Michel Schanen et al. “Asynchronous two-level checkpointing scheme for large-scale adjoints in the spectral-element solver Nek5000”. In: *Procedia Computer Science* 80 (2016), pp. 1147–1158.

[11] Guillaume Aupy and Julien Herrmann. “Periodicity in optimal hierarchical checkpointing schemes for adjoint computations”. In: *Optimization Methods and Software* 32.3 (2017), pp. 594–624.

Understanding the model

- The first vertical line at 55GB 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 optimisations 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 novelty here.