High-level abstractions for checkpointing in PDE-constrained optimisation

Navjot Kukreja[1] [a]

Also:
Jan Hückelheim[1] Michael Lange[2] Mathias Louboutin[3] Andrea Walther[4]
Simon W. Funke[5] Gerard J. Gorman[1]

July 10, 2018

[1] Department of Earth Science and Engineering, Imperial College London, UK
[2] European Centre for Medium-range Weather Forecasts, Reading, UK
[3] Georgia Institute of Technology, United States of America
[4] University of Paderborn, Germany
[5] Simula Laboratories, Norway

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

Seismic Imaging - Motivation

Full Waveform Inversion (FWI): A PDE-constrained optimisation problem to understand the earth

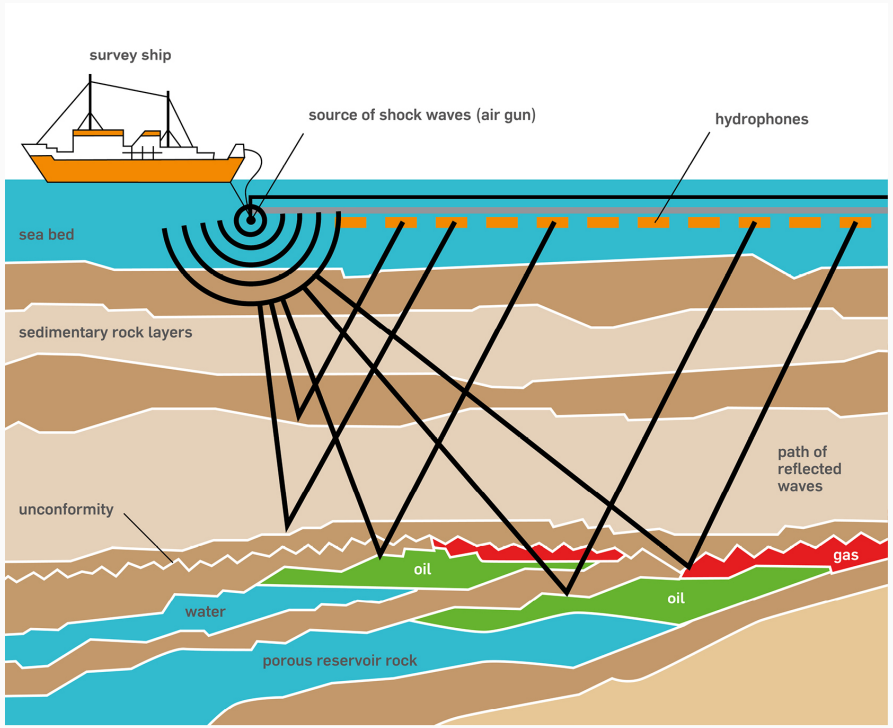


Figure 1: Offshore seismic survey

Source: http://www.open.edu/openlearn/science-maths-technology/science/environmental-science/earths-physical-resources-petroleum/content-section-3.2.1

Problem Statement - the Forward Problem

Given source signal q_s (at a given location) and the earth's physical parameters m, the wave propagation can be simulated using the equation:

$$\begin{cases} m\frac{d^2u(x,t)}{dt^2} - \nabla^2 u(x,t) = q_s \\ u(.,0) = 0 \\ \frac{du(x,t)}{dt}|_{t=0} = 0 \end{cases} \tag{1}$$

Sample Devito [1] code to setup a forward operator ($F(i)$):

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

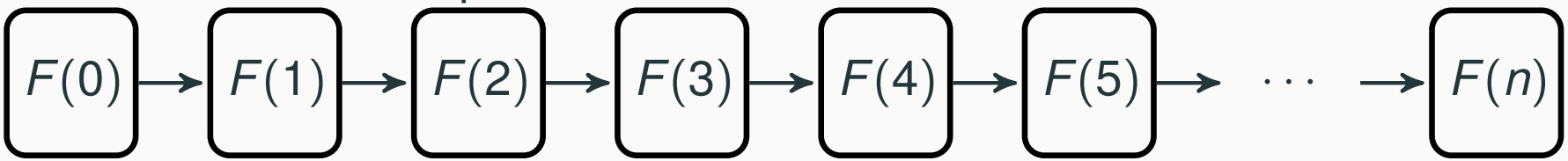
which can be called using:

```
fwp_op.apply(t_start, t_end)
```

[1] Devito (Michael Lange et al. [2017]) is a DSL for rapid development of finite-difference simulations.

Data flow

Data flow for forward problem:



The function u describes the entire wavefield. The signal received at the specific (given) receiver locations could be seen as:

$$d_{sim} = P_r u = P_r A(m)^{-1} P_s^T q_s \tag{2}$$

where A is the action of the equation 1, P_r is the receiver restriction operator, and P_s is the source projection operator.

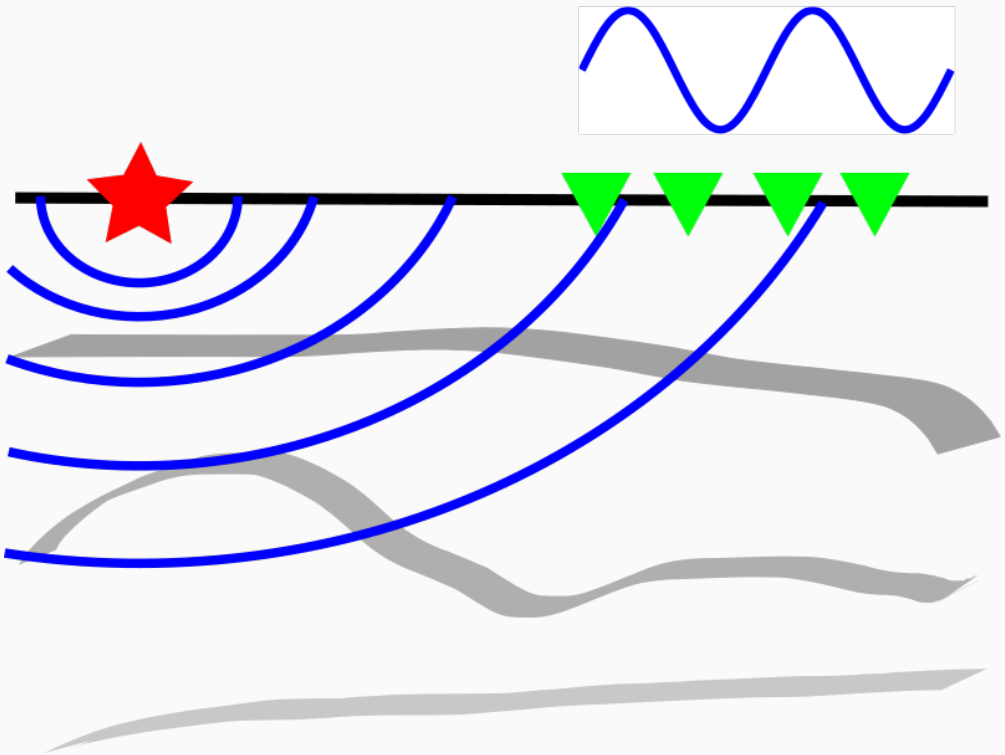


Figure 2: Illustration of the forward problem - simulating the received signal for a given structure

Full Waveform Inversion

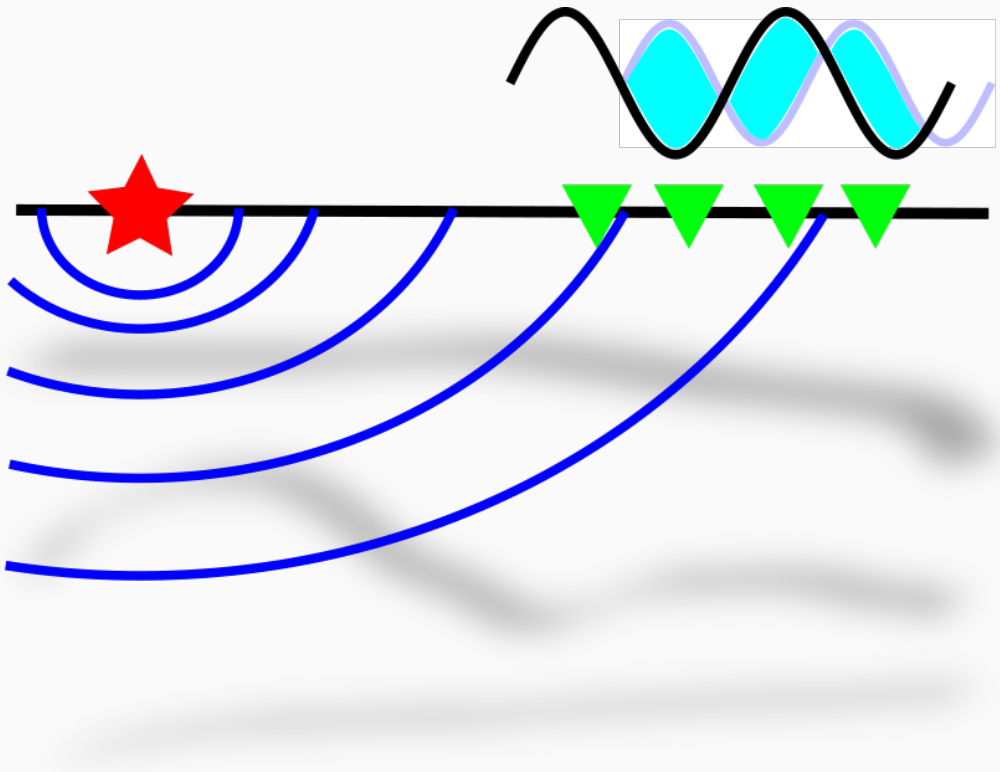


Figure 3: Illustration of full waveform inversion - initial guess

Full Waveform Inversion

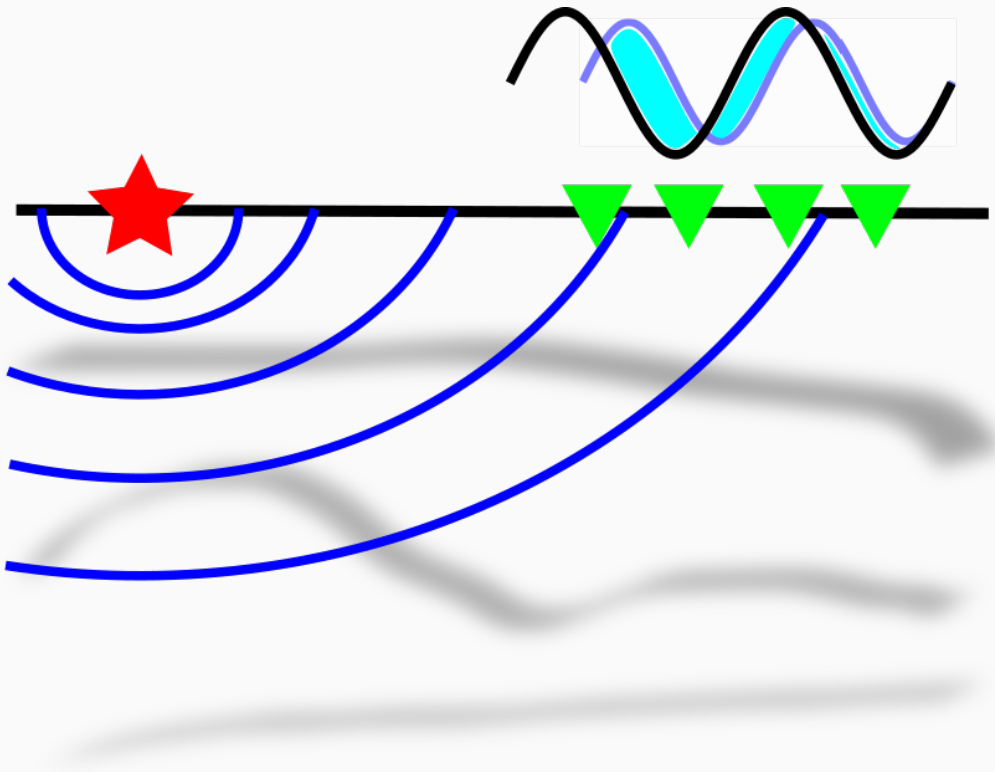


Figure 4: Illustration of full waveform inversion - in progress

Full Waveform Inversion

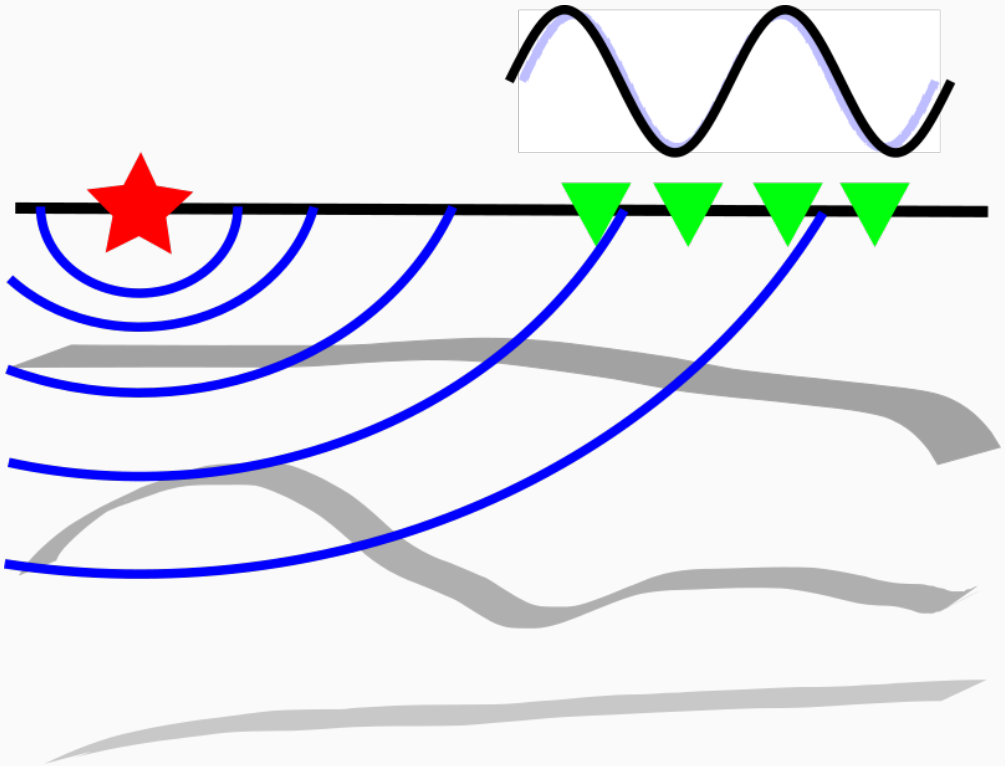


Figure 5: Illustration of full waveform inversion - convergence

Problem Statement - Full Waveform Inversion

FWI can be defined as Virieux and Operto [2009]:

$$\underset{\mathbf{m}}{\text{minimize}}\, \Phi_s(\mathbf{m}) = \frac{1}{2} \|\mathbf{d}_{sim} - \mathbf{d_{obs}}\|_2^2 \tag{3}$$

The gradient of the objective function $\Phi_s(\mathbf{m})$ with respect to the model parameter $\mathbf{m}$ is given by Plessix [2006]:

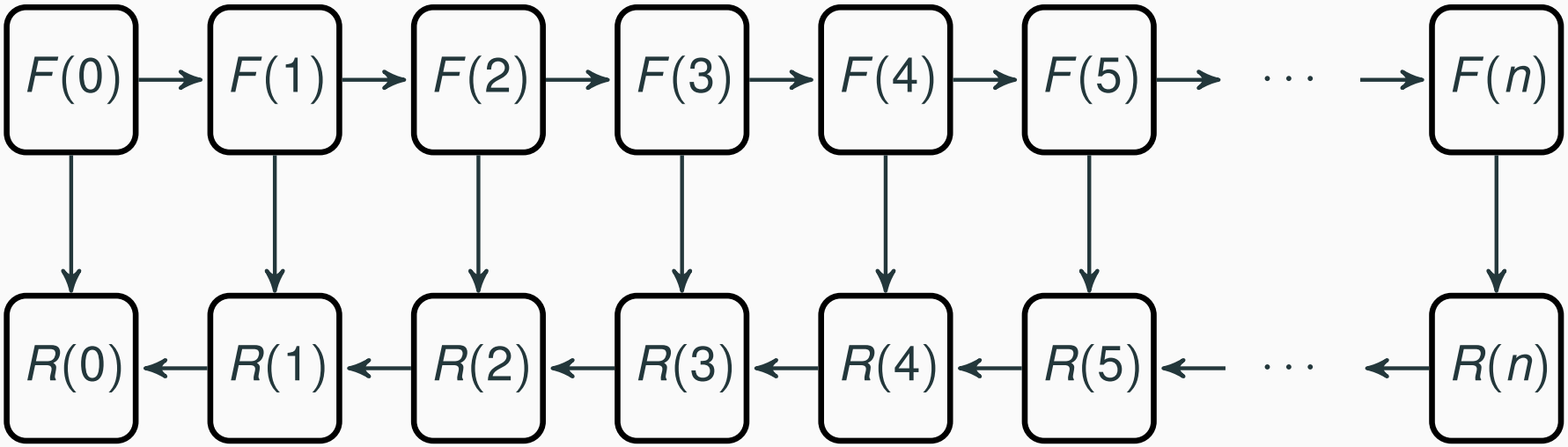
$$\nabla \Phi_s(\mathbf{m}) = \sum_{t=1}^{n_t} \mathbf{u}[\mathbf{t}]\mathbf{v}_{tt}[\mathbf{t}] \tag{4}$$

where $\mathbf{u}[\mathbf{t}]$ is the wavefield in the forward problem and $\mathbf{v}_{tt}[\mathbf{t}]$ is the second-derivative of the adjoint (reverse) field. The reverse operator ($R(i)$):

```
rev_op = Operator(...)
```

Data flow

Data flow for gradient calculation:



Adjoint mode - store all timesteps

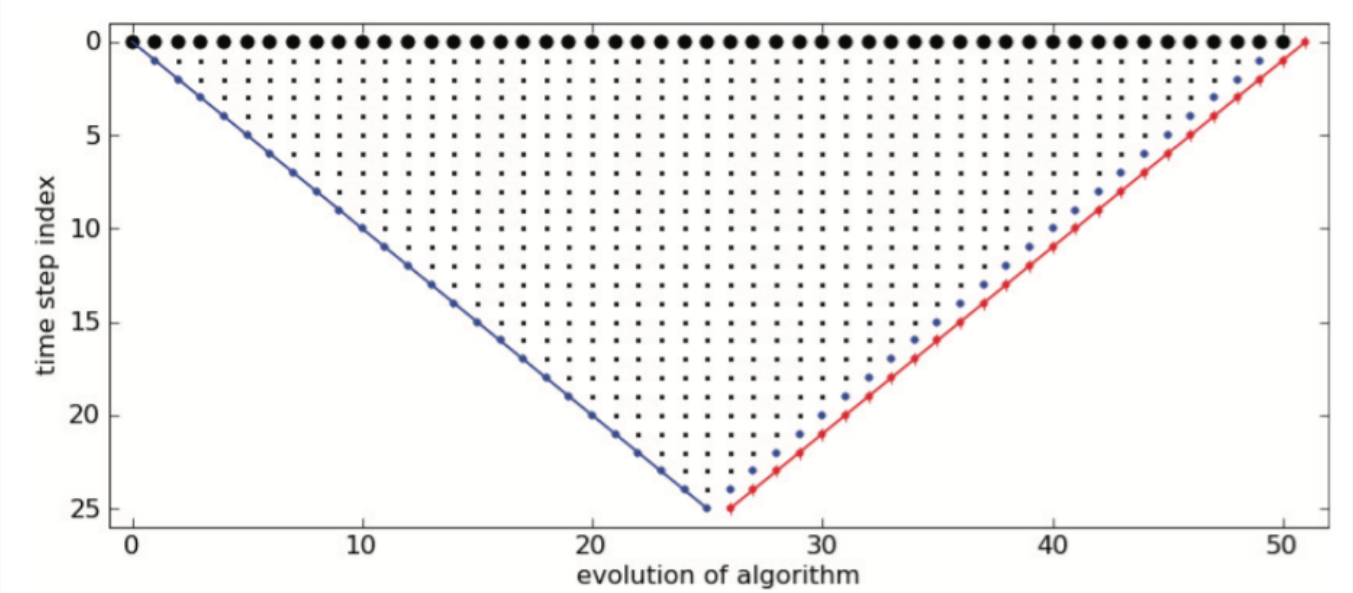


Figure 6: Progression of the adjoint computation with wall-clock time on the x-axis and simulation time on the y-axis. Each vertical cross-section represents the status at that time. The dots represent checkpoints stored in memory - in this case, a checkpoint is stored at each time step.
Image Source: Wang et al. [2009]

Adjoint mode - checkpointed

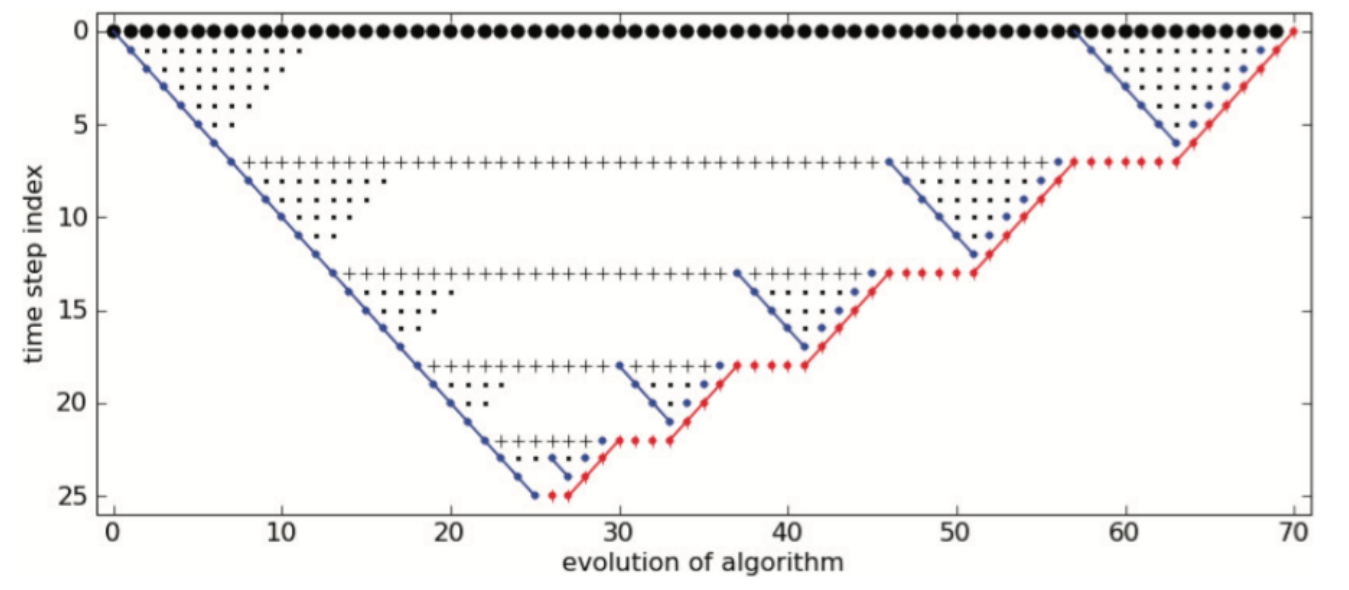


Figure 7: Progression of the adjoint computation with wall-clock time on the x-axis and the simulation time on the y-axis. In this case the number of checkpoints is less than the timesteps, hence there is some recomputation involved.
Image Source: Wang et al. [2009]

Checkpointing - the schedule

The peak memory consumption can be reduced by storing only a subset of intermediate results and recomputing the others when required – this is known as *Checkpointing*.

A *checkpointing schedule* gives:

- which intermediate results should be stored during the initial forward computation
- During the backward computation, how the stored checkpoints are used to restart the forward computation interleaved with the backward computation.

Optimal schedules

Given a problem with n steps, and a given amount of memory, what is the checkpointing schedule that minimises the recomputation? i.e. an optimal schedule

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 Griewank and Walther [2000]. 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.

Checkpointing - Online

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 Wang et al. [2009].

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 Aupy et al. [2016].

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 Schanen et al. [2016] and Aupy and Herrmann [2017].

Checkpointing - comparison

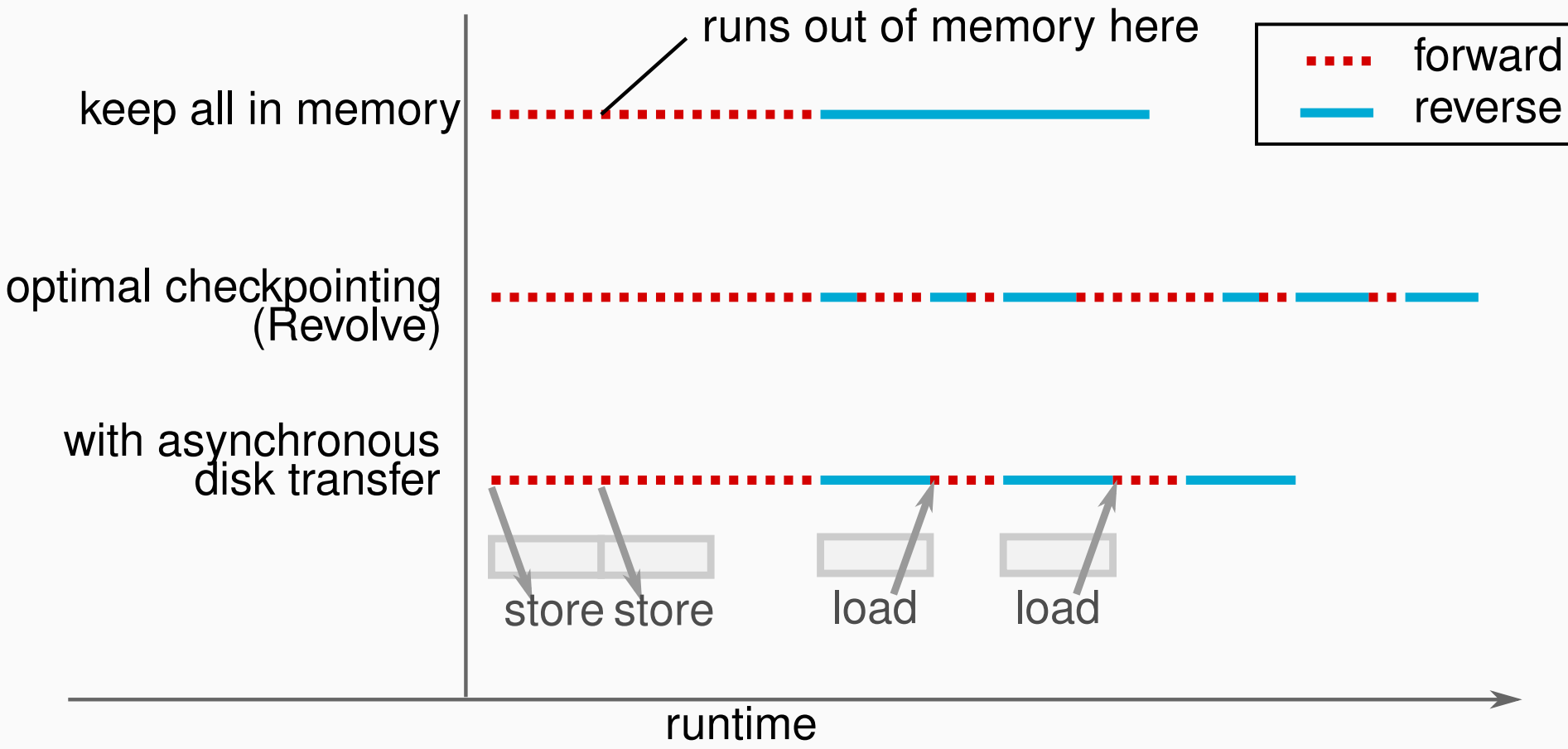


Figure 8: Timeline of events for conventional adjoint, Revolve checkpointing, and asynchronous multistage checkpointing.
Image Source: Kukreja et al. [2018]

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

PyRevolve

- PyRevolve [1] is an open-source library to manage checkpointing within python
- Based on the original Revolve library
- New API based on callbacks in python makes very few assumptions about the Operators being checkpointed on the one hand, and the checkpointing algorithm on the other.
- It only requires Operators that are able to start and stop at any timestep as provided in the arguments and a deep-copy [2] method that can be used to save/load checkpoints.

[1] https://github.com/opesci/pyrevolve

[2] This may violate assumption 4

Sample code to use checkpointing with PyRevolve:

```
# Initialize the two Operators as required
fwd_op = Operator(...)
rev_op = Operator(...)

# Which fields need checkpointing
checkpointer = Checkpointer([u])
# Initialize PyRevolve - will allocate memory
revolver = Revolver(checkpointer, fwd_op, rev_op, nt,
    n_checkpoints)

# Run the simulation in forward mode
# Stopping to take checkpoints
revolver.apply_forward()

# Do something with the result of the forward

# Reverse mode - automatically handle restarts
revolver.apply_reverse()
```

Checkpointing - Practical considerations

However, some assumptions are hard to realise in practice.

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

Zero-cost checkpointing

Most implementations of checkpointing (incuding Tapenade [1]) involve heavy use of deep copies, assuming that these are free.

However, this assumption only holds true when time required for a computational step $F(i)$ is much larger than time required to copy the result of $F(i)$ to checkpointing memory.

The deep copies introduce significant overheads when the computation used to calculate $F(i)$ is a small one (memory-bound).

Can we avoid deep copies?

[1] A popular algorithmic-differentiation tool that does automatic checkpointing (Hascoët and Pascual [2013])

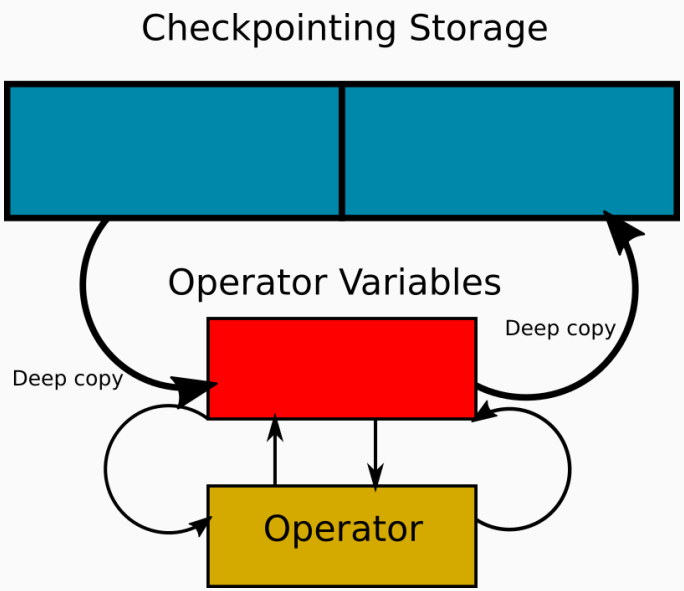
Why is the deep copy required?

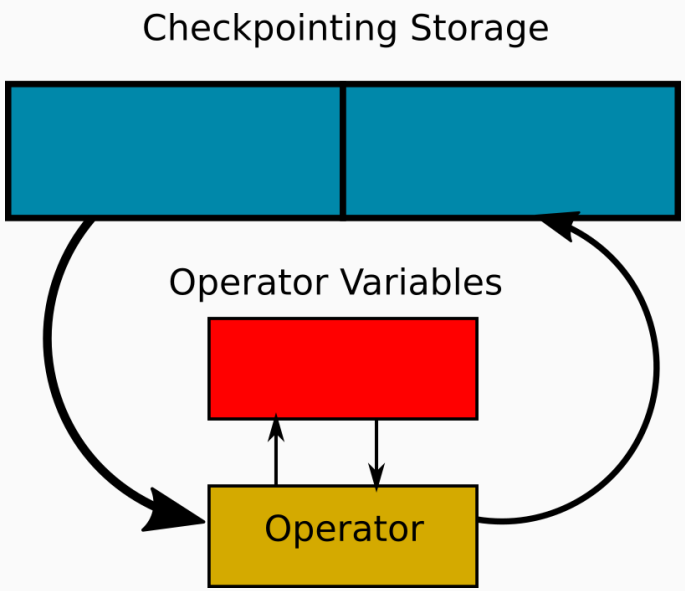
The computation is typically designed to work on a myriad of variables consisting of scalars, arrays, structs etc.

The deep copies move the data from these variables into the checkpoint storage and back.

Could this be done asynchronously?
Even if a subset of the variables could be used to proceed the compute while the others are dirty, being a memory-bound computation, it would compete with the asynchronous copy for memory bandwidth.

Zero-cost checkpointing





A section of a forward-recomputation during checkpointing:

```
...
inbuffer <= deep_copy(ckp[k])
operator(inbuffer, outbuffer)
ckp[k+1] <= deep_copy(outbuffer)
...
```

Changed implementation:

```
...
operator(ckp[k], ckp[k+1])
...
```

Zero-cost checkpointing

To satisfy Revolve's zero-cost-checkpointing assumption (assumption 4), we changed the Operator definition from:

```
Operator.apply(t_start, t_end)
```

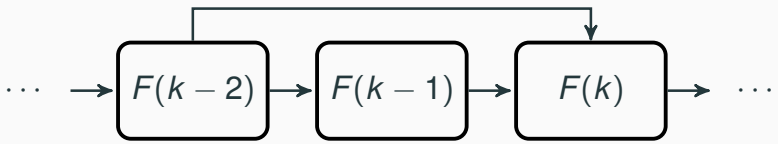
to:

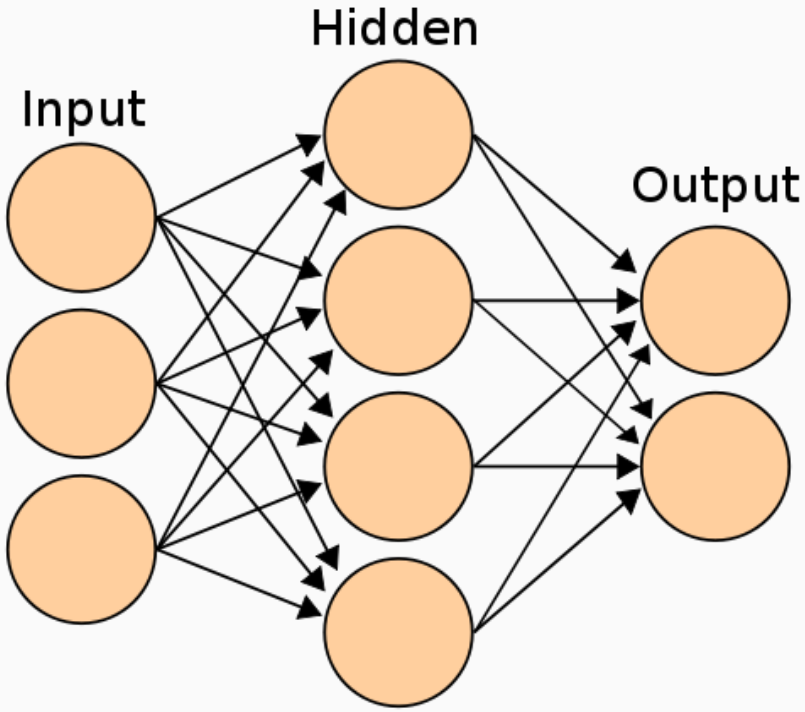
```
Operator.apply(t_start, t_end, inbuffer, outbuffer)
```

and, as a result, removed the deep copy required at every checkpoint save/restore.

Other open problems

- Cost of restarting operators (e.g. time-tiling, function-call overheads)
- Complex data dependencies (e.g. higher order in time, subsampling)
- non-uniform computational cost (e.g. deep learning networks)
- non-uniform checkpoint size (e.g. lossy compression, wavefront-tracking optimisation for seismic)





Thank you

Thank you

Questions?

References i

G. Aupy and J. Herrmann. Periodicity in optimal hierarchical checkpointing schemes for adjoint computations. *Optimization Methods and Software*, 32(3):594–624, 2017.

G. Aupy, J. Herrmann, P. Hovland, and Y. Robert. Optimal multistage algorithm for adjoint computation. *SIAM Journal on Scientific Computing*, 38(3):C232–C255, 2016.

A. Griewank and A. Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):19–45, 2000.

References ii

L. Hascoët and V. Pascual. The Tapenade Automatic Differentiation tool: Principles, Model, and Specification. *ACM Transactions On Mathematical Software*, 39(3), 2013. URL http://dx.doi.org/10.1145/2450153.2450158.

N. Kukreja, J. Hückelheim, and G. J. Gorman. Backpropagation for long sequences: beyond memory constraints with constant overheads. *arXiv preprint arXiv:1806.01117*, 2018.

Michael Lange, Navjot Kukreja, Fabio Luporini, Mathias Louboutin, Charles Yount, Jan Hückelheim, and Gerard J. Gorman. Optimised finite difference computation from symbolic equations. In Katy Huff, David Lippa, Dillon Niederhut, and M. Pacer, editors, *Proceedings of the 16th Python in Science Conference*, pages 89 – 97, 2017. doi: 10.25080/shinma-7f4c6e7-00d.

References iii

R.-E. Plessix. A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. *Geophysical Journal International*, 167(2):495–503, 2006.

M. Schanen, O. Marin, H. Zhang, and M. Anitescu. Asynchronous two-level checkpointing scheme for large-scale adjoints in the spectral-element solver nek5000. *Procedia Computer Science*, 80: 1147–1158, 2016.

J. Virieux and S. Operto. An overview of full-waveform inversion in exploration geophysics. *Geophysics*, 74(6):WCC1–WCC26, 2009.

Q. Wang, P. Moin, and G. Iaccarino. Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. *SIAM Journal on Scientific Computing*, 31(4):2549–2567, 2009.