The Journal of Open Source Software

checkpoint_schedules: schedules for incremental checkpointing of adjoint simulations

Daiane I. Dolci [1], **James R. Maddison** [2], **David A. Ham** [1], **Guillaume Pallez** [3], **and Julien Herrmann** [4]

1 Department of Mathematics, Imperial College London, London, United Kingdom. **2** School of Mathematics and Maxwell Institute for Mathematical Sciences, The University of Edinburgh, United Kingdom. **3** Inria, University of Rennes, Rennes, France. **4** CNRS, IRIT, Université de Toulouse, Toulouse, France.

DOI: 10.21105/joss.06148

Software

- Review
- Repository
- Archive

Editor: Patrick Diehl

Reviewers:

- @matt-graham
- @KYANJO

Submitted: 28 September 2023
Published: 22 March 2024

Summary

checkpoint_schedules provides schedules for step-based incremental checkpointing of the adjoints to computer models. The schedules contain instructions indicating the sequence of forward and adjoint steps to be executed, and the data storage and retrieval to be performed. These instructions are independent of the model implementation, which enables the model authors to switch between checkpointing algorithms without recoding. Conversely, *checkpointing_schedules* provides developers of checkpointing algorithms a direct mechanism to convey their work to model authors. *checkpointing_schedules* has been integrated into **tlm_adjoint** (James R. Maddison et al., 2019), a Python library designed for the automated derivation of higher-order tangent-linear and adjoint models and work is ongoing to integrate it with **pyadjoint** (Mitusch et al., 2019). This package can be incorporated into other gradient solvers based on adjoint methods, regardless of the specific approach taken to generate the adjoint model.

The use of adjoint calculations to compute the gradient of a quantity of interest resulting from the solution of a system of partial differential equations (PDEs) is widespread and well-established. The resulting gradient may be employed for many purposes, including topology optimisation (Papadopoulos et al., 2021), inverse problems (Plessix, 2006), and flow control (Jansen, 2011).

Solving the adjoint to a non-linear time-dependent PDE requires the forward PDE to be solved first. The adjoint PDE is then solved in a reverse time order, but depends on the forward state. Storing the entire forward state in preparation for the adjoint calculation has a memory footprint that is linear in the number of time steps. For sufficiently large problems this will exhaust the memory of any computer system.

In contrast, checkpointing approaches store only the state required to restart the forward calculation from a limited set of steps. As the adjoint calculation progresses, the forward computation is progressively rerun from the latest available stored state up to the current adjoint step. This enables less forward state to be stored, at the expense of a higher computational cost as forward steps are run more than once. Griewank & Walther (2000) proposed a checkpointing algorithm, which is optimal under certain assumptions, including that the number of steps is known in advance, and that all the storage has equal access cost. Subsequent authors have produced checkpointing algorithms that relax these requirements in various ways, such as by accounting for different types of storage (e.g., memory and disk) or by not requiring the number of steps to be known in advance, for example Stumm & Walther (2009), Aupy et al. (2016), Schanen et al. (2016), Aupy & Herrmann (2017), Herrmann & Pallez (2020), James R. Maddison (2023), and Zhang & Constantinescu (2023).

Statement of need

This situation is typical across computational mathematics: there exists a diversity of algorithms whose applicability and optimality depends on the nature and parameters of the problem to be solved. From the perspective of users who wish to construct adjoint solvers, this creates the need to swap out different checkpointing algorithms in response to changes in the equations, discretisations, and computer systems with which they work. Those users will often lack the expertise or the time to continually reimplement additional algorithms in their framework. Further, such reimplementation work is wasteful and error-prone.

checkpointing_schedules provides a number of checkpointing algorithms accessible through a common interface, and these are interchangeable without recoding. This can be used in conjunction with an adjoint framework such as tlm_adjoint or pyadjoint and a compatible PDE framework, such as Firedrake (Ham et al., 2023) or FEniCS (Alnaes et al., 2015), to enable users to create adjoint solvers for their choice of PDE, numerical methods, and checkpointing algorithm, all without recoding the underlying algorithms.

Some of the algorithms supported by *checkpointing_schedules* have been implemented many times, while for others, such as H-Revolve the only previously published implementation was a simple proof of concept in the original paper (Herrmann & Pallez, 2020). The checkpoint schedule API provided by *checkpoint_schedules* further provides a toolkit for the implementation of further checkpoint schedules, thereby providing a direct route from algorithm developers to users.

Software description

Currently, *checkpoint_schedules* is able to generate schedules for the following checkpointing schemes: Revolve (Stumm & Walther, 2009), disk-revolve (Aupy et al., 2016), periodic-disk revolve (Aupy & Herrmann, 2017), two-level (Pringle et al., 2016), H-Revolve (Herrmann & Pallez, 2020), and mixed storage checkpointing (James R. Maddison, 2023). It also contains trivial schedules that store the entire forward state. This enables users to support adjoint calculations with or without checkpointing via a single code path.

The complete documentation for *checkpoint_schedules* is available on the Firedrake project website.

Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council [EP/W029731/1 and EP/W026066/1]. J. R. M. was supported by the Natural Environment Research Council [NE/T001607/1]. G. P. was supported in part by the French National Research Agency (ANR) in the frame of DASH (ANR-17-CE25-0004).

Author contributions

GP and JH wrote the original reference implementation of H-Revolve and related schedules originally published in Herrmann & Pallez (2020), and contributed to the fixed and enhanced version of that code that is included in *checkpoint_schedules*. DH and JM designed the original *checkpoint_schedules* API, which was implemented by DH, JM and DD. The remaining schedules were implemented by JM and DD. DD led the integration of the package, and wrote most of its documentation and test cases. Copyright headers in the respective source files record the contributors to those files.

References

Alnaes, M. S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., & Wells, G. N. (2015). The FEniCS project version 1.5. *Archive of Numerical Software*, *3*. https://doi.org/10.11588/ans.2015.100.20553

Aupy, G., & Herrmann, J. (2017). Periodicity in optimal hierarchical checkpointing schemes for adjoint computations. *Optimization Methods and Software*, *32*(3), 594–624. https://doi.org/10.1080/10556788.2016.1230612

Aupy, G., Herrmann, J., Hovland, P., & Robert, Y. (2016). Optimal multistage algorithm for adjoint computation. *SIAM Journal on Scientific Computing*, *38*(3), C232–C255. https://doi.org/10.1145/347837.347846

Griewank, A., & Walther, A. (2000). Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software (TOMS)*, *26*(1), 19–45. https://doi.org/10.1145/347837.347846

Ham, D. A., Kelly, P. H. J., Mitchell, L., Cotter, C. J., Kirby, R. C., Sagiyama, K., Bouziani, N., Vorderwuelbecke, S., Gregory, T. J., Betteridge, J., Shapero, D. R., Nixon-Hill, R. W., Ward, C. J., Farrell, P. E., Brubeck, P. D., Marsden, I., Gibson, T. H., Homolya, M., Sun, T., … Markall, G. R. (2023). *Firedrake user manual* (First edition). Imperial College London; University of Oxford; Baylor University; University of Washington. https://doi.org/10.25561/104839

Herrmann, J., & Pallez, G. (2020). H-revolve: A framework for adjoint computation on synchronous hierarchical platforms. *ACM Transactions on Mathematical Software (TOMS)*, *46*(2), 1–25. https://doi.org/10.1145/3378672

Jansen, J. D. (2011). Adjoint-based optimization of multi-phase flow through porous media – a review. *Computers & Fluids*, *46*(1), 40–51. https://doi.org/10.1016/j.compfluid.2010.09.039

Maddison, James R. (2023). On the implementation of checkpointing with high-level algorithmic differentiation. *arXiv Preprint arXiv:2305.09568*. https://doi.org/10.48550/arXiv.2305.09568

Maddison, James R., Goldberg, D. N., & Goddard, B. D. (2019). Automated calculation of higher order partial differential equation constrained derivative information. *SIAM Journal on Scientific Computing*, *41*(5), C417–C445. https://doi.org/10.1137/18M1209465

Mitusch, S. K., Funke, S. W., & Dokken, J. S. (2019). Dolfin-adjoint 2018.1: Automated adjoints for FEniCS and Firedrake. *Journal of Open Source Software*, *4*(38), 1292. https://doi.org/10.21105/joss.01292

Papadopoulos, I. P., Farrell, P. E., & Surowiec, T. M. (2021). Computing multiple solutions of topology optimization problems. *SIAM Journal on Scientific Computing*, *43*(3), A1555–A1582. https://doi.org/10.1137/20M1326209

Plessix, R.-E. (2006). A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. *Geophys. J. Int*, *167*, 495–503. https://doi.org/10.1111/j.1365-246X.2006.02978.x

Pringle, G., Jones, D. C., Goswami, S., Narayanan, S. H. K., & Goldberg, D. (2016). *Providing the ARCHER community with adjoint modelling tools for high-performance oceanographic and cryospheric computation*. EPCC. https://nora.nerc.ac.uk/id/eprint/516314/

Schanen, M., Marin, O., Zhang, H., & Anitescu, M. (2016). Asynchronous two-level checkpointing scheme for large-scale adjoints in the spectral-element solver Nek5000. *Procedia Computer Science*, *80*, 1147–1158. https://doi.org/10.1016/j.procs.2016.05.444

Stumm, P., & Walther, A. (2009). Multistage approaches for optimal offline checkpointing. *SIAM Journal on Scientific Computing*, *31*(3), 1946–1967. https://doi.org/10.1137/080718036

Zhang, H., & Constantinescu, E. M. (2023). Optimal checkpointing for adjoint multistage time-stepping schemes. *Journal of Computational Science*, *66*, 101913. https://doi.org/10.1016/j.jocs.2022.101913