




1 ADIOS4DOLFINx: A framework for checkpointing in 2 FEniCS

3 **Jørgen Schartum Dokken**  ¹ ¶

4 1 Simula Research Laboratory ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright
and release the work under a
Creative Commons Attribution 4.0
International License ([CC BY 4.0](#)).

5 Summary

6 We introduce a checkpointing framework for the latest version of the FEniCS project, known as
7 DOLFINx. The framework leverages the data-centric approach of DOLFINx along with a state
8 of the art adaptable Input/Output system called ADIOS2. Several variations of checkpointing
9 are supported, including *N-to-M* checkpointing of function data, storage of mesh partitioning
10 information for *N-to-N* checkpointing and snapshot checkpointing for RAM reduction during
11 simulation. All MPI operations are using MPI-3 Neighborhood collectives.

12 Statement of need

13 The ability to start, stop and resume simulations is becoming increasingly important with the
14 growing use of supercomputers for solving scientific and engineering problems. A rising number
15 of large scale problems are deployed on high performance, memory distributed computing
16 systems and users tend to run more demanding simulations. These are often non-linear and
17 time-dependent, which typically amounts to thousands of CPU hours. As it might uncover
18 bugs and unphysical solutions, the ability to run parts of the simulation, inspect the result
19 and then resume simulation becomes a key factor to enable efficient development. If this is
20 discovered early on, the simulation can be terminated saving the developer time, money and
21 energy-usage.

22 The proposed framework enables users of the FEniCS project ([Baratta et al., 2023](#)) to store
23 solutions during simulation, and read them in at their convenience to resume simulations at a
24 later stage. Several checkpointing methods are implemented, including *N-to-M* checkpointing,
25 which means saving data from a program executed with *N* processes, and loading it back in on
26 *M* processes.

27 Functionality for *N-to-M* checkpointing was implemented for the old version of DOLFIN by
28 ([Habera et al., 2018](#)). However, this functionality is not present in the newest version of
29 the FEniCS Project ([Baratta et al., 2023](#)). The storage principles in the ADIOS4DOLFINx
30 are based on the ideas present in this implementation. However, the implementation for
31 non-Lagrangian finite element spaces vastly differs, due to the usage of dof-permutations
32 ([Scroggs et al., 2022](#)). Additionally, all global MPI-calls in the old implementation have been
33 reimplemented with scalable MPI-communication using the MPI-3 Neighborhood Collectives
34 ([MPI-Forum, 2012](#)).

35 The framework introduces several new methods for storing partitioning information for *N-to-N*
36 checkpointing with arbitrary ghosting, as well as very lightweight snapshot checkpoints. A
37 similar framework for *N-to-M* checkpointing was implemented by ([Ham et al., 2024](#)) for the
38 finite element framework Firedrake ([Rathgeber et al., 2016](#)). This frameworks differs from
39 the one used in ADIOS4DOLFINx in several ways due to the different internal structures of
40 DOLFINx and Firedrake.

41 **Functionality**

42 The software is written as a Python-extension to DOLFINx, which can be installed using the
43 Python Package installer pip directly from the Github repository or using the [ADIOS4DOLFINx](#)
44 from the Python Package Index. The following features are supported:

- 45 ▪ Snapshot checkpointing
- 46 ▪ *N-to-M* checkpointing with mesh storage
- 47 ▪ *N-to-M* checkpointing without mesh storage
- 48 ▪ *N-to-N* checkpointing storing partitioning information

49 A *snapshot checkpoint* is a checkpoint that is only valid during the run of a simulation. It is
50 lightweight (only stores the local portion of the global dof array to file), and is stored using the
51 *Local Array* feature in ADIOS2 ([Godoy et al., 2020](#)) to store data local to the MPI process.
52 This feature is intended for use-cases where many solutions have to be aggregated to the end
53 of a simulation to some post-processing step, or as a fall-back mechanism when restarting a
54 diverging iterative solver.

55 A *N-to-M* checkpoint is a checkpoint that can be written with N processes and read back in
56 with M processes. Two versions of this checkpoint is supported; One where storage of the
57 mesh is required and without mesh storage. The reasoning for such a split is that when a
58 mesh is read into DOLFINx and passed to an appropriate partitioner, the ordering mesh nodes
59 (coordinates) and connectivity (cells) is changed. Writing these back into *global arrays* requires
60 MPI communication to ensure contiguous writing of data.

61 The *N-to-M* checkpoint with mesh storage exclusively writes contiguous chunks of data owned
62 by the current process to an ADIOS2 *Global Array* that can be read in with a different number
63 of processes at a later stage. This operation requires no MPI-communication.

64 In many cases, the input mesh might stem from an external mesh generator and is stored
65 together with mesh entity markers in an external file, for instance an XDMF-file. To avoid
66 duplication of this mesh data, a stand-alone file that can be associated with the XDMF file for
67 a later restart can be created. This method requires some MPI neighborhood collective calls
68 to move data from the process that currently owns it to the relevant process for that stores it
69 as a *Global Array* in contiguous chunks. Both *N-to-M* checkpoint routines uses the same API
70 to read in checkpoints at a later instance.

71 In certain scenarios, mesh partitioning might be time-consuming, as a developer is running the
72 same problem over and over again with the same number of processes. As DOLFINx supports
73 custom partitioning ([Baratta et al., 2023](#)), we use this feature to read in partition data from a
74 previous run. As opposed to the checkpoints in the old version of DOLFIN, these checkpoints
75 handle any ghosting, that being a custom ghosting provided by the user, or the shared-facet
76 mode provided by DOLFINx.

77 **Examples**

78 A large variety of examples covering all the functions in adios4dolfinx is available at <https://jorgensd.github.io/adios4dolfinx>.

80 **Acknowledgements**

81 We acknowledge the valuable feedback on the documentation and manuscript by Thomas M.
82 Surowiec and Halvor Herlyng. Additionally, we acknowledge the scientific discussion regarding
83 feature development and code contributions by Henrik N. Finsberg and Francesco Ballarin.

84 **References**

- 85 Baratta, I. A., Dean, J. P., Dokken, J. S., Habera, M., Hale, J., Richardson, C. N., Rognes,
86 M. E., Scroggs, M. W., Sime, N., & Wells, G. N. (2023). *DOLFINx: The next generation*
87 *FEniCS problem solving environment*. <https://doi.org/10.5281/zenodo.10447666>
- 88 Godoy, W. F., Podhorszki, N., Wang, R., Atkins, C., Eisenhauer, G., Gu, J., Davis, P., Choi, J.,
89 Germaschewski, K., Huck, K., Huebl, A., Kim, M., Kress, J., Kurc, T., Liu, Q., Logan, J.,
90 Mehta, K., Ostrouchov, G., Parashar, M., ... Klasky, S. (2020). ADIOS 2: The adaptable
91 input output system. A framework for high-performance data management. *SoftwareX*, 12,
92 100561. <https://doi.org/10.1016/j.softx.2020.100561>
- 93 Habera, M., Zilian, A., Hale, J., Richardson, C. N., Blechta, J., & Dave, D. (2018). *XDMF*
94 *and ParaView: checkpointing format*. <https://hdl.handle.net/10993/35848>
- 95 Ham, D. A., Hapla, V., Knepley, M. G., Mitchell, L., & Sagiya, K. (2024). *Efficient n-to-m*
96 *checkpointing algorithm for finite element simulations*. [https://doi.org/10.48550/arXiv.](https://doi.org/10.48550/arXiv.2401.05868)
97 [2401.05868](https://doi.org/10.48550/arXiv.2401.05868)
- 98 MPI-Forum. (2012). *MPI: A Message-Passing Interface Standard. Version 3.0*. [https:](https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf)
99 [//www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf](https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf)
- 100 Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., Mcrae, A. T. T., Bercea,
101 G.-T., Markall, G. R., & Kelly, P. H. J. (2016). Firedrake: Automating the finite element
102 method by composing abstractions. *ACM Trans. Math. Softw.*, 43(3). [https://doi.org/10.](https://doi.org/10.1145/2998441)
103 [1145/2998441](https://doi.org/10.1145/2998441)
- 104 Scroggs, M. W., Dokken, J. S., Richardson, C. N., & Wells, G. N. (2022). Construction of
105 arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell
106 meshes. *ACM Trans. Math. Softw.*, 48(2). <https://doi.org/10.1145/3524456>