The Journal of Open Source Software

ADIOS4DOLFINx: A framework for checkpointing in FEniCS

Jørgen Schartum Dokken [1¶]

1 Simula Research Laboratory, Oslo, Norway ¶ Corresponding author

DOI: 10.21105/joss.06451

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @gonsie
- @Chilipp

Submitted: 06 March 2024
Published: 30 April 2024

Summary

We introduce ADIOS4DOLFINx, a checkpointing framework for the latest version of the FEniCS project, known as DOLFINx. DOLFINx is a general framework for solving partial differential equations using the finite element method. The input to simulations using the finite element method is the computational domain (mesh), mesh markers, initial conditions, and boundary conditions. To be able to restart a simulation at any point, one has to have the capability to read in all of the aforementioned variables. The adios4dolfinx package implements all of these operations, using the Message Passing Interface (MPI) for communication across multiple processes and ADIOS2 for writing/reading data to/from file. In particular, the functionality of adios4dolfinx includes *N-to-M*-checkpointing, which means that one can store a result of a simulation that was generated with N number of processes, and read it into a program running on M processes.

Statement of need

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

ADIOS4DOLFINx enables users of the FEniCS project (Baratta et al., 2023) to store solutions during simulation, and read them in at their convenience to resume simulations at a later stage. Several checkpointing methods are implemented, including *N-to-M* checkpointing, which means saving data from a program executed with N processes, and loading it back in on M processes.

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

The framework introduces several new methods for storing partitioning information for *N-to-N*

checkpointing with arbitrary ghosting, as well as very lightweight snapshot checkpoints. A similar framework for *N-to-M* checkpointing was implemented by Ham et al. (2024) for the finite element framework Firedrake (Rathgeber et al., 2016). This frameworks differs from the one used in ADIOS4DOLFINx in several ways due to the different internal structures of DOLFINx and Firedrake.

Functionality

The software is written as a Python-extension to DOLFINx, which can be installed using the Python Package installer `pip` directly from the GitHub repository or using the ADIOS4DOLFINx from the Python Package Index (PyPI). The following features are supported:

- Snapshot checkpointing
- *N-to-M* checkpointing with mesh storage
- *N-to-M* checkpointing without mesh storage
- *N-to-N* checkpointing storing partitioning information

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

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

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

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

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

Examples

A large variety of examples covering all the functions in ADIOS4DOLFINx is available at https://jorgensd.github.io/adios4dolfinx.

Acknowledgements

We acknowledge the valuable feedback on the documentation and manuscript by Thomas M. Surowiec and Halvor Herlyng and packaging support by Min Ragan-Kelley. Additionally, we acknowledge the scientific discussion regarding feature development and code contributions by Francesco Ballarin, Henrik N. Finsberg, and Nathan Sime.

References

Baratta, I. A., Dean, J. P., Dokken, J. S., Habera, M., Hale, J., Richardson, C. N., Rognes, M. E., Scroggs, M. W., Sime, N., & Wells, G. N. (2023). *DOLFINx: The next generation FEniCS problem solving environment*. https://doi.org/10.5281/zenodo.10447666

Godoy, W. F., Podhorszki, N., Wang, R., Atkins, C., Eisenhauer, G., Gu, J., Davis, P., Choi, J., Germaschewski, K., Huck, K., Huebl, A., Kim, M., Kress, J., Kurc, T., Liu, Q., Logan, J., Mehta, K., Ostrouchov, G., Parashar, M., … Klasky, S. (2020). ADIOS 2: The adaptable input output system. A framework for high-performance data management. *SoftwareX*, *12*, 100561. https://doi.org/10.1016/j.softx.2020.100561

Habera, M., Zilian, A., Hale, J., Richardson, C. N., Blechta, J., & Dave, D. (2018). *XDMF and ParaView: checkpointing format*. https://hdl.handle.net/10993/35848

Ham, D. A., Hapla, V., Knepley, M. G., Mitchell, L., & Sagiyama, K. (2024). *Efficient n-to-m checkpointing algorithm for finite element simulations*. https://doi.org/10.48550/arXiv.2401.05868

MPI-Forum. (2012). *MPI: A Message-Passing Interface Standard. Version 3.0*. https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., Mcrae, A. T. T., Bercea, G.-T., Markall, G. R., & Kelly, P. H. J. (2016). Firedrake: Automating the finite element method by composing abstractions. *ACM Transactions on Mathematical Software*, *43*(3). https://doi.org/10.1145/2998441

Scroggs, M. W., Dokken, J. S., Richardson, C. N., & Wells, G. N. (2022). Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes. *ACM Transactions on Mathematical Software*, *48*(2). https://doi.org/10.1145/3524456