Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

FEniCS–preCICE: Coupling FEniCS to other simulation software

Benjamin Rodenberg [a,*], Ishaan Desai [b], Richard Hertrich [a], Alexander Jaust [c], Benjamin Uekermann [b]

[a] *Scientific Computing in Computer Science, Department of Informatics, Technical University of Munich, Germany*
[b] *Usability and Sustainability of Simulation Software, Institute for Parallel and Distributed Systems, University of Stuttgart, Germany*
[c] *Simulation of Large Systems, Institute for Parallel and Distributed Systems, University of Stuttgart, Germany*

A R T I C L E   I N F O

Article history:
Received 20 March 2021
Received in revised form 23 July 2021
Accepted 8 September 2021

Keywords:
FEniCS
Fluid–structure interaction
Conjugate heat transfer
Multiphysics
Coupled problems
Finite element method
preCICE

A B S T R A C T

The new software FEniCS–preCICE is a middle software layer, sitting in between the existing finite-element library FEniCS and the coupling library preCICE. The middle layer simplifies coupling (existing) FEniCS application codes to other simulation software via preCICE. To this end, FEniCS–preCICE converts between FEniCS and preCICE mesh and data structures, provides easy-to-use coupling conditions, and manages data checkpointing for implicit coupling. The new software is a library itself and follows a FEniCS-native style. Only a few lines of additional code are necessary to prepare a FEniCS application code for coupling. We illustrate the functionality of FEniCS–preCICE by two examples: a FEniCS heat conduction code coupled to OpenFOAM and a FEniCS linear elasticity code coupled to SU2. The results of both scenarios are compared with other simulation software showing good agreement.

Code metadata

Current code version	v1.2.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00054
Code Ocean compute capsule	https://codeocean.com/capsule/4816993
Legal Code License	LGPL-3.0
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	FEniCS, pyprecice, numpy, scipy
If available Link to developer documentation/manual	https://www.precice.org/adapter-fenics.html
Support	https://precice.discourse.group/

1. Motivation and significance

Enabling simulations to play a significant role in answering the great research challenges of our time – let it be nuclear fusion, personalized medicine, or climate prediction – requires the efficient interplay of diverse simulation software [1]. Ideally, single simulation components may be treated as black boxes and may easily be plugged together or exchanged. The software preCICE [2] helps to do exactly that: It can be used to glue together arbitrarily many of such black-box simulation components. In the following, we refer to these components as *participants* of a coupled simulation. preCICE focuses mainly on mesh-based discretizations of PDE models as participants. The finite element software FEniCS [3,4] is a popular choice to solve such PDE models in rather compact Python scripts. We call such a Python script a *FEniCS application code*. In this paper, we develop and document a new software called FEniCS–preCICE adapter, which allows researchers to easily couple FEniCS application codes to other simulation software by using preCICE.

preCICE itself is a library. Thus, for coupling, a participant needs to call the API of preCICE in its source code. Due to the high abstraction level of the preCICE API, the necessary changes to a participant's source code are minimally invasive. These changes are typically realized in a so-called *adapter* – an additional class, module, or callback library of the participant's source code. An adapter defines the coupling meshes, handles coupling boundary conditions, and realizes the steering of the coupling. Till

* Corresponding author.
E-mail address: benjamin.rodenberg@in.tum.de (Benjamin Rodenberg).

https://doi.org/10.1016/j.softx.2021.100807

around 2016, preCICE users had to write their own adapters for codes they wanted to couple. As many users coupled the same codes, they had to solve the same problems, and continuously reinvented the wheel. Therefore, official, stand-alone, general-purpose adapters for widely used software (e.g. for OpenFOAM [5, 6] or SU2 [7]) were first introduced in [8] to end the waste of precious human development resources. These developments have significantly contributed to the usability and popularity of preCICE and, thus, to the process of scientific discovery.

In this paper, we follow the same lines of argument and introduce a stand-alone, general-purpose adapter for FEniCS. We follow a FEniCS-native style, which makes the entry barrier for users of FEniCS as low as possible. The adapter can be easily integrated into existing FEniCS codes. The distributed-memory parallelization of FEniCS is supported out of the box. FEniCS itself is not a single simulation code, but a library as well. This makes a general-purpose adapter a challenging task. We, therefore, restrict the generality: we focus on 2D problems and study time-dependent fluid–structure interaction (FSI) and conjugate heat transfer (CHT) as examples. The adapter can also be applied to volume-coupled problems,[1] on the one hand, and can easily be extended to 3D problems,[2] on the other hand.

FEniCS has already been used to solve FSI, or more general multi-physics problems in a *monolithic* fashion [9–11]. To this end, the library multiphenics provides FEniCS-tools to solve multi-physics problems.[3] The interaction of the different physical phenomena is modelled through a large coupled equation system that is solved using FEniCS. Very often conforming meshes are required across the different physical domains. We, however, follow a *partitioned* approach, which allows to combine several specialized single-physics participants to solve overall multi-physics problems. FEniCS has already been coupled in a partitioned fashion to reaktoro [12] to simulate reactive transport or to other instances of FEniCS to allow for non-conforming meshes in FSI [13]. Recently, the generic coupling software MUI [14] has been used to realize an FSI coupling between OpenFOAM and FEniCS [15]. A review of other generic coupling software can, for example, be found in [2]. In this paper, we present a software for coupling of FEniCS-based applications to arbitrary other simulation software via preCICE.

This paper follows the suggested structure of *SoftwareX*: We describe the software in detail in Section 2. This includes an overview of the software architecture, a detailed description of the functionality of the software, an usage example for its API, and a brief overview of used testing approaches. We give two illustrative use cases of the FEniCS–preCICE adapter in Section 3: a conjugate heat transfer simulation with FEniCS and OpenFOAM and a fluid–structure interaction with FEniCS and SU2. Afterwards, we discuss the impact of the new software in Section 4.

2. Software description

The FEniCS–preCICE adapter might be considered an unusual research software. It is neither a stand-alone single program, nor a library that can be used in such a program. Instead, it is a middle–layer software between two large software packages: the finite element library FEniCS and the coupling library preCICE. Furthermore, a coupled simulation, by definition, consists of multiple participants, which brings even more software packages to the table. When describing the FEniCS–preCICE adapter, it is, therefore, essential to describe how the new software interfaces with these other software packages. In Section 2.1, we give an overview of the overall software architecture alongside a brief introduction to the individual packages: preCICE, FEniCS, and the FEniCS–preCICE adapter. Afterwards, in Section 2.2, we give detailed information on the API of the FEniCS–preCICE adapter and its functionality. A short example code in Section 2.3 completes the software description. Last, in Section 2.4, we explain how the new software is tested.

2.1. Software architecture

Fig. 1 gives an overview on how all software layers play together in a coupled simulation using the FEniCS–preCICE adapter. The user provides a FEniCS application code (`solver.py`), which uses FEniCS (`import fenics`) for solving a certain PDE model with a finite element method. Additionally, the application code imports the FEniCS–preCICE adapter (`import fenicsprecice`), which is a Python package itself, for coupling to other simulation software. The adapter imports preCICE (`import precice`) – more specifically the Python bindings of preCICE. Finally, preCICE handles the coupling to other simulation software, for example OpenFOAM or SU2. Let us have a closer look at the individual packages.

preCICE provides three building blocks for coupling mesh-based PDE solvers: (1) Methods for data mapping between non-matching meshes. (2) Fixed-point acceleration methods to stabilize coupled equation systems. (3) Communication between participants, which are separate executables, potentially running on different nodes in a heterogeneous compute cluster. preCICE is a library and follows a peer-to-peer coupling concept — no central server-like entity is required. The library is written in C++, but also offers bindings in Python generated from the C++ API with Cython [16]. As input and output arguments, the Python bindings use primitive Python data types as well as NumPy arrays [17]. preCICE is configured at run time through an `xml` file describing a complete coupled simulation setup.[4]

FEniCS is a finite element Python package with an extensive C++ implementation, which is named DOLFIN, under the hood. Its API uses a high abstraction level. FEniCS provides functionality and data structures for generating and managing meshes. On top of meshes, various finite element spaces can be created. Weak forms of arbitrary PDEs can then be defined in a very compact notation. In Section 2.3, we give a brief code example, which illustrates the FEniCS API. The FEniCS–preCICE adapter has been tested and developed using FEniCS 2019.1.0. For more information on FEniCS, we refer the reader to [4].

FEniCS–preCICE provides an API for coupling a FEniCS application code using preCICE. This means that the FEniCS application code calls the FEniCS–preCICE adapter, which in turn calls preCICE. Some of the API calls of the FEniCS–preCICE adapter are simply redirected to preCICE. For other API calls, the adapter provides substantial functionality, for example for conversion of data structures. We give a detailed explanation of the API of the adapter in Section 2.2. The adapter is configured through a `json` file, which describes what data is written and read by the adapter. For simplification, currently only one read and one write data field on a single coupling mesh is technically supported by the adapter, whereas the preCICE API offers more flexibility. An extension in this direction is planned. Furthermore, FEniCS–preCICE

[1] A volume-coupled FEniCS example is available here: https://github.com/precice/tutorials/pull/219.

[2] A working prototype of the adapter extended to 3D scenarios can be found in a pull request (github.com/precice/fenics-adapter/pull/133). In this prototype every function which involves handling of vector data is now modified to handle either 2D or 3D data depending on the dimension of the problem. This prototype has been tested for a 3D serial FSI scenario (see github.com/precice/tutorials/pull/222).

[3] https://mathlab.sissa.it/multiphenics.

[4] See https://www.precice.org/configuration-overview.html for details on configuration of preCICE.

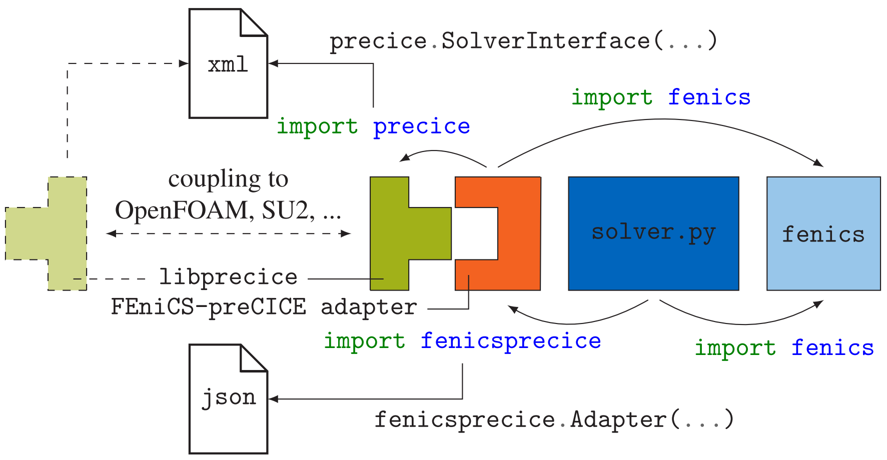


Fig. 1. Overview of the software architecture. From left to right: preCICE, FEniCS–preCICE adapter, application code `solver.py`, and FEniCS.

was only tested with continuous Galerkin first– and second–order finite elements, while FEniCS provides a broad range of available finite element spaces, including higher order and discontinuous Galerkin.

2.2. Software functionalities

The FEniCS–preCICE adapter uses a central `Adapter` object as handle to its API. We list the important parts of the API in Fig. 2 and describe individual member functions of `Adapter` step by step in the following. To follow all implementation details of this section requires some previous knowledge of FEniCS.

Adapter initialization (lines 4–13). The constructor of `Adapter` creates and configures the adapter object. The object is initialized by calling `initialize`, where the `coupling_subdomain` argument is the domain boundary where data should be coupled. Two optional arguments exist to configure the coupling: For one-way coupling, either a `read_function_space` or a `write_object` must be provided — depending on whether the participant reads or writes data. For two-way coupling, both arguments are required.

A FEniCS `FunctionSpace` is provided for the `read_function_space` and the `write_object` to provide the coupling mesh as well as the type of function (scalar/vector, order) to the adapter. If the user wants to provide initial write data, a FEniCS `Function` can be provided as `write_object`.

Data access (line 15–20). The user can call `write_data` and `read_data` to write data to or read data from preCICE, respectively. The argument `write_function` provided to `write_data` is a FEniCS `Function`, which the adapter samples at the coupling mesh. Calling `read_data` returns a dict `data` that contains coupling mesh points and associated data. It can be used to directly update coupling expressions or point sources.

Coupling consistent quantities via expressions (lines 22–28). To enforce coupling boundary conditions, we need to distinguish two types of coupling data: quantities that require consistent data mapping (e.g. temperature, heat flux, or force per unit volume) and quantities that require conservative data mapping (e.g. forces). Let us first consider boundary conditions for consistent quantities. For this case, FEniCS' `Expression` has proven to be the right tool. This object is very flexible and can be used in many different ways — not only for boundary conditions. The FEniCS book [4] gives many examples how to use an expression for both, Dirichlet and Neumann boundary conditions. An essential Dirichlet boundary condition, for instance, can be defined by FEniCS' `DirichletBC`. However, expressions can also be used directly in the weak form, for instance for Neumann boundary conditions (`... + expr * v * ds`) or for volume terms (`... + expr * v * dx`). Here, `expr` is an `Expression`, `v` the test function, and `ds` and `dx` are surface and volume integration elements. Normally, for boundary conditions, expressions are explicitly given as symbolic expressions.[5] For coupling boundaries, however, such a continuous representation needs to be constructed from the nodal values given by preCICE. To this end, the adapter provides a `CouplingExpression` which inherits from FEniCS' `UserExpression`. The continuous representation is constructed using an interpolation routine following the approach suggested in [18], where first a polynomial least-squares fit is constructed followed by a radial-basis function interpolation. For the radial-basis function interpolation, we use routines from SciPy [19]. This interpolation is not to be confused with the data mapping that preCICE uses. An uninitialized `CouplingExpression` is created and returned by `create_coupling_expression`. It is initialized or updated by calling `update_coupling_expression` and providing the dict `data` obtained from `read_data`.

Coupling conservative quantities via point sources (lines 30–32). Boundary conditions for conservative quantities can be realized by point-wise boundary conditions [20]. To this end, FEniCS offers `PointSource(V, p, magnitude)`, which can be applied to the equation system by `PointSource.apply(rhs)`. Here, `V` is a function space, `p` the coordinates where the point source is applied, and `magnitude` the magnitude of the point source. `get_point_sources` allows the user to obtain point sources at the individual coupling mesh points; again, by providing the dict `data` obtained from `read_data`. Please note that, contrary to a `CouplingExpression`, which is created once and then updated via a pointer-like access pattern, a `PointSource` is just overwritten.

Checkpointing (lines 34–40). For implicit coupling (also known as strong or tight coupling), each time step (or even multiple time steps within a so-called time window) needs to be repeated iteratively until the coupling residual drops below a defined threshold. preCICE handles the convergence measurement, the iteration control, and the acceleration of the implicit coupling loop. The responsibility of the adapter is to provide a mechanism to go backwards in time. This is realized by storing and retrieving checkpoints of the complete solver state. To

[5] e.g., `Expression('sin(x[0]) + cos(x[1])')`

```python
1  import precice  # python bindings of preCICE
2
3  class Adapter:
4      def __init__(self, adapter_config_filename='precice-adapter-config.json'):
5          # create adapter object and interface to preCICE python bindings
6          self._interface = precice.Interface(...)
7
8      def initialize(self, coupling_subdomain, read_function_space=None, write_object=None):
9          # initialize coupling mesh and initialize data in preCICE
10         precice_dt = self._interface.initialize()
11         ...
12         self._interface.initialize_data()
13         return precice_dt  # returns maximum allowed timestep size
14
15     def read_data(self):
16         # creates a dictionary with coupling vertices and reads associated data from preCICE
17         return data
18
19     def write_data(self, write_function):
20         # sample write_function and write data to preCICE
21
22     def create_coupling_expression(self):
23         # create uninitialized CouplingExpression
24         return CouplingExpression(...)
25
26     def update_coupling_expression(self, coupling_expression, data):
27         # get nodal data and vertex coordinates from dictionary data
28         coupling_expression.update_boundary_data(nodal_data, x_coordinates, y_coordinates)
29
30     def get_point_sources(self, data):
31         # creates one list of PointSources per dimensions from given coupling data
32         return x_PointSources, y_PointSources
33
34     def store_checkpoint(self, user_u, t, n):
35         self._checkpoint = SolverState(user_u.copy(), t, n)
36         self._interface.mark_action_fulfilled(precice.action_write_iteration_checkpoint())
37
38     def retrieve_checkpoint(self):
39         self._interface.mark_action_fulfilled(precice.action_read_iteration_checkpoint())
40         return self._checkpoint.get_state()
41
42     def advance(self, dt):
43         return self._interface.advance(dt)
```

Fig. 2. API of the FEniCS-preCICE adapter (excerpt). The actual implementation is sketched, whenever it is short enough.

this end, the adapter offers methods `store_checkpoint` and `retrieve_checkpoint`, which are designed such that a user cannot accidentally destroy or overwrite checkpoints.

Steering (lines 42–43). Steering methods, which allow to control the time and the coupling loop (`advance` and several others), are all directly forwarded to preCICE. We refer to the documentation of the Python bindings of preCICE for details.[6]

Parallelization. FEniCS supports distributed-memory parallelization based on MPI:

```
mpiexec -np numberOfRanks python3 solver.py
```

The FEniCS–preCICE adapter directly supports this parallelization. We want to note a few necessary implementation details. In most cases, the domain decomposition of FEniCS does not yield a situation where all parallel ranks are located at the coupling boundary. Nevertheless, to allow for a single implementation for serial and parallel cases, the adapter object is created and initialized on all ranks. If the domain of a rank is not connected to the coupling boundary, the rank is considered as an inactive rank from the perspective of the adapter. A further technical challenge results from the ghost communication layers in FEniCS. As standard for the finite element method, vertices at the domain boundaries are duplicated on all connected ranks. However, only one rank has the ownership of a specific vertex. At the coupling boundary, only the rank which owns a vertex defines it as part of the coupling mesh for preCICE. Thus, in the adapter, each rank can only read new values from preCICE on the vertices it owns. The reconstruction of the coupling boundary condition, however, requires values also at non-owned vertices. To exchange these values between ranks, the adapter uses its own communication step after reading values from preCICE. The MPI wrapper shipped with FEniCS cannot be used here because the communication needs to take place after data is read from preCICE and before it is updated in FEniCS. The current parallel implementation, however, only supports boundary conditions defined in the form of expressions. Generating point source objects in parallel cannot be supported due to a known problem in FEniCS.[7] Additionally, the parallelization currently only supports 2D cases. For 3D parallel cases, nodes on the coupling interface may generally be shared by more than two processes, which requires an extension of our current treatment of ghost communication layers.

6 https://github.com/precice/python-bindings.

7 https://fenicsproject.discourse.group/t/4729.

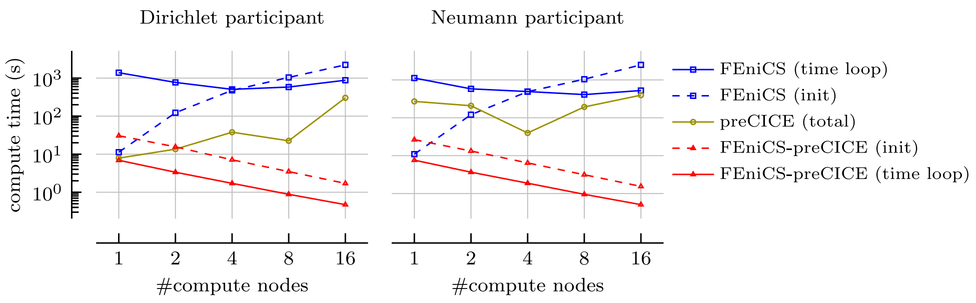


Fig. 3. Strong scaling of the example test case described in Section 2.3 with a mesh resolution of 3000 × 3000 for each participant and 10 computed time steps. The time for the Dirichlet participant is shown on the left and the time for the Neumann participant on the right. Each participant uses #compute nodes with 28 processes each. The total runtime is split up into the contributions of each preCICE, FEniCS–preCICE, and FEniCS. For FEniCS–preCICE and FEniCS, we show time spent for initialization (init) and time inside the time loop. For preCICE, however, a proper differentiation is challenging [21]. We show instead the total runtime, which is affected by intra-participant load imbalances of FEniCS and inter-participant load imbalances of the coupled setup. In fact, the Dirichlet participant requires more time per time step due to the additional computation of the heat flux. For a proper study of the performance and scalability of preCICE, we refer to the literature [21,22]. We observe that, while the scalability of FEniCS suffers from the still relatively small amount of unknowns, the time spent in the adapter scales well for both initialization and per time step and is negligible compared to FEniCS and preCICE.

Due to the pure peer-to-peer concept and fully–parallel implementations of data mapping and fixed-point acceleration, preCICE is capable of efficiently supporting parallel coupled simulations on ten thousands of MPI ranks [21,22]. To investigate whether the new middle layer FEniCS–preCICE introduces any detrimental overhead, we perform a strong scaling study, shown in Fig. 3. We observe that the time spent in the adapter shows good scalability and is negligible compared to the amount of time spent in the solver.

2.3. Example: A simple coupled heat transfer solver

To complete the explanation of the software architecture and the API in the last sections, we now show a simple but almost complete example application code `solver.py`. The example we study is the time-dependent heat equation and is borrowed from the FEniCS tutorials [23].

We define one edge of the squared domain as a Dirichlet coupling boundary. Through this boundary, our example code could be coupled to another FEniCS application code solving the time-dependent heat equation as well, which would render the overall problem into a simple partitioned heat equation. Alternatively, the coupling partner could also be a fluid solver, which would result in a conjugate heat transfer scenario. We show the results of such an example in Section 3.1. The setup is illustrated in Fig. 4. In the following, we first look briefly at the adapter configuration and, afterwards, we discuss the application code.

2.3.1. Configuration of the adapter

Fig. 5 shows the configuration file of our example. The coupling participant is named `FEniCS` and the associated coupling mesh `FEniCSMesh`. The participant reads temperature values from preCICE and uses them to construct a Dirichlet boundary condition at the coupling boundary. Finally, the participant writes heat flux values to preCICE, which are explicitly computed as gradients of the temperature field.

2.3.2. Application code

Fig. 6 shows the almost complete application code of our example. Let us first have a look at the FEniCS code parts. Lines 5 and 6 define the geometry and mesh. Afterwards, lines 9 and 10 define the (quadratic) function space as well as trial and test functions, which are then used in line 25 to define the weak form of the time-dependent heat equation. For time integration, an implicit Euler scheme is used. Furthermore, lines 12 and 13 define Dirichlet boundary conditions at the non-coupling boundary. The discretized linear system is solved in every time step in line 35.

The highlighted lines show the code related to the coupling. We import FEniCS–preCICE in line 2. Line 7 defines a subdomain as coupling boundary. Then, lines 15 to 18 create the handle to the adapter, initialize it with a read and write function space used for two-way coupling, and create the Dirichlet coupling boundary condition. In line 28, the time loop control is handed over to preCICE. If required, data checkpoints are written in lines 29 and 30 and read in lines 40 to 43. Temperature values are read from preCICE in line 32 and used to update the coupling boundary condition in line 33. After solving the linear system, heat fluxes are extracted from the new solution (using the `VectorFunctionSpace` from line 11) and written to preCICE in lines 36 and 37. Then, finally, the actual coupling is advanced. Here, preCICE returns an upper limit for the next time step size, which is enforced in line 34.

2.4. Testing

The complex software architecture of the FEniCS–preCICE adapter, as introduced in Section 2.1, makes testing the software more involved than testing most other research software. First, the FEniCS–preCICE adapter is a middle software layer — it is called by a FEniCS application code and calls in turn preCICE (via its Python bindings). Second, a coupled simulation needs at least two participants, thus at least two layered software stacks, of which the FEniCS–preCICE adapter could be part of one or both. Both concepts are not unusual in software engineering in general. Thus, there are known techniques for testing, such as *mocking* [24]. The concepts are, however, uncommon for research codes, which typically only consist of stand-alone executables or libraries. We apply two different approaches to test the FEniCS–preCICE adapter.

The first approach tests the software in a complete coupled setup. We couple two FEniCS application codes to keep the number of dependencies small. We take the example we just introduced in the previous section and couple it with its counterpart – a time-dependent heat equation with a Neumann coupling boundary condition. We could then compare the results of this partitioned heat equation with those of a single-domain heat equation in FEniCS. We follow, however, an even simpler approach (similar to the FEniCS tutorials [23]): We construct a manufactured solution with linear dependency in time and quadratic dependency in space. As we use P^2 finite elements and first-order time integration, the discretization can recover the exact solution for arbitrary (coarse) space and time resolutions.[8] The only

[8] Refer to [25, Section 4.1] for details on the discretization.

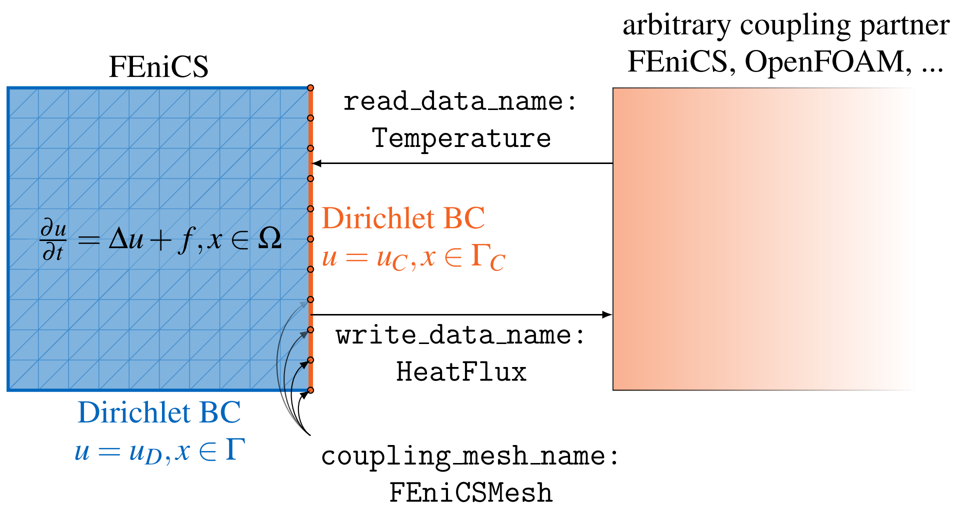


Fig. 4. Unit square Ω, where heat equation is solved by the FEniCS participant. Dirichlet boundary conditions are applied on Γ. Coupling happens on Γ_C, where another Dirichlet boundary condition is applied and heat flux is sampled to establish the coupling.

```
{
"participant_name": "FEniCS",
"config_file_name": "precice-config.xml",
"interface": {
    "coupling_mesh_name": "FEniCSMesh",
    "write_data_name": "HeatFlux",
    "read_data_name": "Temperature"
  }
}
```

Fig. 5. Adapter configuration file `precice-adapter-config.json`.

remaining error component is the coupling error, which can be controlled by the coupling convergence measures in every time step. Tightening the thresholds reduces the error until reaching machine precision.[9] In preCICE nomenclature, we call such a test, where multiple participants and preCICE are a dependency, a system test.

The second approach to test the FEniCS–preCICE adapter regards the software as an isolated unit — independent of any other coupled participant and even independent of preCICE itself. To this end, we use a *mocked up* version of the Python bindings of preCICE. This allows us to test individual functions of the adapter, which then call the mocked up dummy implementation instead of preCICE itself. To illustrate the concept, Fig. 7 gives an example on how to test the adapter function `read_data`. Line 2 imports the mock object `MockedPrecice`. In line 4, we use `patch` from `unittest.mock`[10] to replace the real implementation of preCICE with the mock object. This means that in line 8 and indirectly in line 9, not the real preCICE, but the mock object is imported. The test can be run without preCICE being installed on the test system. In line 13, we use `MagicMock` to define the behaviour of the mocked preCICE function `read_block_scalar_data`. We then can test whether `read_data` converts to and returns the correct data in line 22. `MagicMock` also allows us to record the arguments a mocked function receives. We, finally, use this functionality in line 28 to also test whether preCICE receives the correct input arguments from the adapter.

3. Illustrative examples

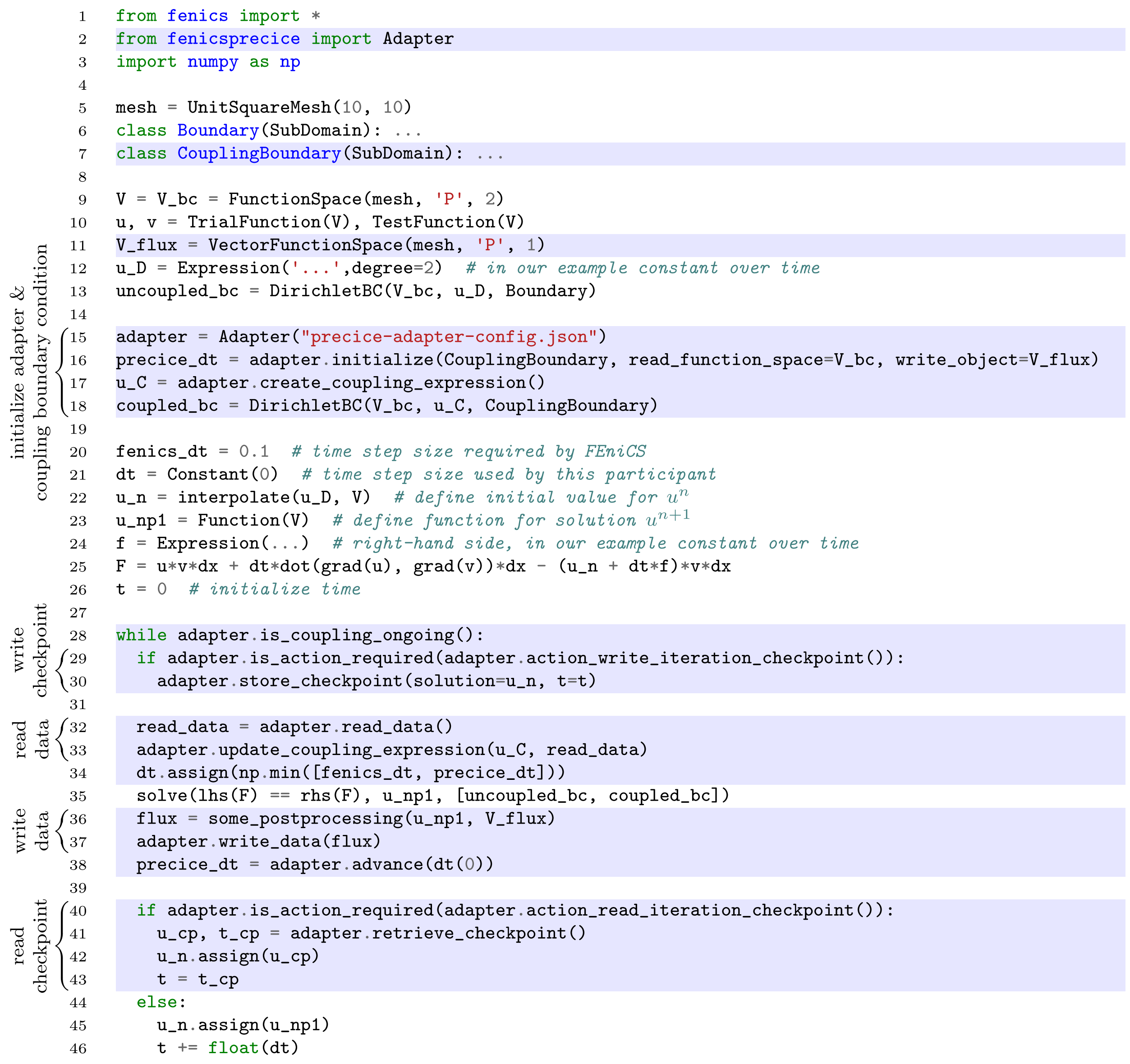
We give two examples to showcase how the FEniCS–preCICE adapter can be used in practice. First, in Section 3.1, a heat conduction solver in FEniCS is coupled to an OpenFOAM fluid solver for conjugate heat transfer (CHT). Afterwards, in Section 3.2, a linear elasticity solver in FEniCS is coupled to an SU2 fluid solver for fluid–structure interaction (FSI). We deliberately pick two different applications of FEniCS (heat transfer and structural mechanics) and two different coupling partners (OpenFOAM, SU2) to show the wide applicability of the new adapter. For both setups, we use simple geometries. These geometrical setups are offered as preCICE tutorials for many different participant combinations (FEniCS, OpenFOAM, SU2, deal.II, Nutils, code_aster, CalculiX). FEniCS–preCICE adapter release v1.2.0[11] with preCICE release v2.2.0[12] and the Python bindings release v2.2.0.2[13] is used in these examples. All software that is needed for the examples presented below is part of the preCICE distribution v202104.1.0[14] – except the FEniCS–preCICE adapter v1.2.0.

3.1. Conjugate heat transfer with FEniCS and OpenFOAM

As a simple CHT test case, we consider a flow over a heated plate[15] inspired by [26]. Fig. 8(a) sketches the domain and lists

9 Case is provided under https://github.com/precice/tutorials/tree/v202104.1.1/partitioned-heat-conduction.
10 https://docs.python.org/3/library/unittest.mock.html.

11 https://github.com/precice/fenics-adapter/releases/tag/v1.2.0
12 https://github.com/precice/precice/releases/tag/v2.2.0
13 https://github.com/precice/python-bindings/releases/tag/v2.2.0.2
14 https://github.com/precice/vm/releases/tag/v202104.1.0
15 https://github.com/precice/tutorials/tree/v202104.1.1/flow-over-heated-plate.



```python
from fenics import *
from fenicsprecice import Adapter
import numpy as np

mesh = UnitSquareMesh(10, 10)
class Boundary(SubDomain): ...
class CouplingBoundary(SubDomain): ...

V = V_bc = FunctionSpace(mesh, 'P', 2)
u, v = TrialFunction(V), TestFunction(V)
V_flux = VectorFunctionSpace(mesh, 'P', 1)
u_D = Expression('...',degree=2)  # in our example constant over time
uncoupled_bc = DirichletBC(V_bc, u_D, Boundary)

adapter = Adapter("precice-adapter-config.json")
precice_dt = adapter.initialize(CouplingBoundary, read_function_space=V_bc, write_object=V_flux)
u_C = adapter.create_coupling_expression()
coupled_bc = DirichletBC(V_bc, u_C, CouplingBoundary)

fenics_dt = 0.1  # time step size required by FEniCS
dt = Constant(0)  # time step size used by this participant
u_n = interpolate(u_D, V)  # define initial value for u^n
u_np1 = Function(V)  # define function for solution u^{n+1}
f = Expression(...)  # right-hand side, in our example constant over time
F = u*v*dx + dt*dot(grad(u), grad(v))*dx - (u_n + dt*f)*v*dx
t = 0  # initialize time

while adapter.is_coupling_ongoing():
  if adapter.is_action_required(adapter.action_write_iteration_checkpoint()):
    adapter.store_checkpoint(solution=u_n, t=t)

  read_data = adapter.read_data()
  adapter.update_coupling_expression(u_C, read_data)
  dt.assign(np.min([fenics_dt, precice_dt]))
  solve(lhs(F) == rhs(F), u_np1, [uncoupled_bc, coupled_bc])
  flux = some_postprocessing(u_np1, V_flux)
  adapter.write_data(flux)
  precice_dt = adapter.advance(dt(0))

  if adapter.is_action_required(adapter.action_read_iteration_checkpoint()):
    u_cp, t_cp = adapter.retrieve_checkpoint()
    u_n.assign(u_cp)
    t = t_cp
  else:
    u_n.assign(u_np1)
    t += float(dt)
```

Fig. 6. Coupled heat equation FEniCS application code. The calls to FEniCS–preCICE are highlighted, while the remaining lines represent a simplified version of the original heat equation example from the FEniCS tutorials [23].

all physical parameters. Heat conduction in the plate in the lower part of the domain is simulated with FEniCS, using a very similar application code as already explained in Section 2.3. On top of the plate, we simulate a fluid flow from left to right with the OpenFOAM [5,6] solver `buoyantPimpleFoam`. To couple OpenFOAM, we make use of the OpenFOAM-preCICE adapter[16] [27]. We use a Dirichlet–Neumann coupling: The solid participant (FEniCS) receives temperature values from the fluid participant and uses them as Dirichlet boundary condition at the coupling interface. The fluid participant (OpenFOAM) receives heat flux values from the solid participant and uses them as Neumann boundary condition. For the Dirichlet boundary condition in FEniCS, we use a FEniCS expression as described in Section 2.2. To map coupling data between non-matching meshes at the coupling interface, we use a nearest-neighbour mapping (in preCICE). Fig. 8(b) shows the temperature distribution of the coupled simulation after approaching steady-state. To verify our results, we compare the OpenFOAM-FEniCS coupling to an already existing OpenFOAM-OpenFOAM coupling. For comparable mesh resolutions, the results match very well, see Fig. 8(c).

3.2. Fluid–structure interaction with FEniCS and SU2

As a simple FSI test case, we consider a wall-mounted elastic flap in a channel flow.[17] Fig. 9(a) sketches the domain and lists all physical parameters. To simulate the elastic flap, we use a linear elasticity code in FEniCS, which was developed in the Bachelor's thesis of Richard Hertrich [20] following an example from [28]. In the fluid domain, we use the compressible Euler solver of SU2 [7]. To couple SU2, we make use of the SU2-preCICE adapter[18] [29]. We again use a Dirichlet–Neumann coupling: The fluid participant (SU2) receives displacement values from the solid participant, computes velocity values from the displacement

16 https://github.com/precice/openfoam-adapter/releases/tag/v1.0.0.

17 https://github.com/precice/tutorials/tree/v202104.1.1/perpendicular-flap.

18 https://github.com/precice/su2-adapter/tree/ab84387.

```python
from unittest import TestCase
import tests.MockedPrecice

@patch.dict('sys.modules', **{'precice': tests.MockedPrecice})
class TestReadData(TestCase):
  def test_scalar_read(self):
    from unittest.mock import MagicMock
    from precice import Interface
    import fenicsprecice

    # mock preCICE API
    dummy_data = np.arange(...)  # hard-coded dummy values
    Interface.read_block_scalar_data = MagicMock(return_value=dummy_data)

    # initialize adapter
    adapter = fenicsprecice.Adapter(self.dummy_config)
    [...]
    # call adapter API
    read_data = adapter.read_data()

    # (1) check whether expected output is received
    assert(read_data == expected_read_data)

    # (2) check whether preCICE API was called with expected arguments
    recorded_args = Interface.read_block_scalar_data.call_args[0]
    expected_args = ...  # hard-coded
    for arg, expected_arg in zip(recorded_args, expected_args):
      assert(arg == expected_arg)
```

Fig. 7. Testing the individual function `read_data` of the FEniCS–preCICE adapter with a mocked preCICE implementation.

values, and uses the velocity values as Dirichlet boundary condition at the coupling interface. The solid participant (FEniCS) receives force values from the fluid participant and uses them as Neumann boundary condition. For the Neumann boundary condition in FEniCS, we now use point sources, as described in Section 2.2. To map coupling data between non-matching meshes at the coupling interface, we again use a nearest-neighbour mapping (in preCICE). Fig. 9(b) shows the fluid velocity at maximum deformation of the beam. To verify our results, we compare the SU2-FEniCS coupling to an already existing SU2-deal.II coupling. We use deal.II v9.2 [30] and the preCICE-deal.II adapter.[19] For comparable mesh resolutions, the results match once again very well, see Fig. 9(c).

4. Impact

The FEniCS–preCICE adapter enables the coupling of existing FEniCS application codes to other simulation software in only a few lines of code. In particular, this holds for simulation software for which preCICE adapters already exist such as OpenFOAM, SU2, or deal.II. Application scientists can now focus on coupled problems from the physical perspective and let the FEniCS–preCICE adapter handle the technical aspects: converting mesh and data structures, handling coupling conditions, or checkpointing for implicit coupling.

The increase in opportunities works in both directions: Not only existing FEniCS users can now easily connect to the preCICE community and other simulation software, but also other communities (e.g., the large OpenFOAM community) can now directly benefit from FEniCS.

We want to illustrate the range of opportunities with three examples:

- Within the collaborative research centre 1313 at University of Stuttgart [31] several porous media applications are studied, such as the hydromechanical coupling of fractures and porous media. The current implementation is based on FEniCS and preCICE [32], as it lowers the implementation hurdles compared to previous implementations based on DUNE [33]. Furthermore, basing the implementation on FEniCS and preCICE immediately enables parallel computing capabilities. The current realization, however, uses the preCICE API directly from the application code. Using the new FEniCS–preCICE adapter instead will lead to a more idiomatic incorporation of preCICE into the coupled FEniCS codes, allowing the researchers to rather focus on the implementation and evaluation of their models than the coupling itself. Some extensions of the adapter will be necessary, however, to tackle mixed-dimensional problems as they appear in hydromechanical coupling simulations.
- Together with researchers from Helmholtz-Zentrum Geesthacht and HSU Hamburg, we are currently investigating the coupling of electro-chemistry models to fracture mechanics to simulate corrosion.[20] For both fields, promising models [34] or codes in FEniCS [35] exist. In contrast to both examples in Section 3, a volume coupling will be necessary. However, since the adapter treats coupling conditions as general FEniCS `Expressions` volume coupling is already supported by the adapter.
- The layered design created by the FEniCS–preCICE adapter makes it also possible to easily prototype and test new numerical coupling algorithms, such as quasi-Newton waveform iteration in [25]. Here, an additional layer between the adapter and preCICE was used to implement and test a waveform data structure separately.

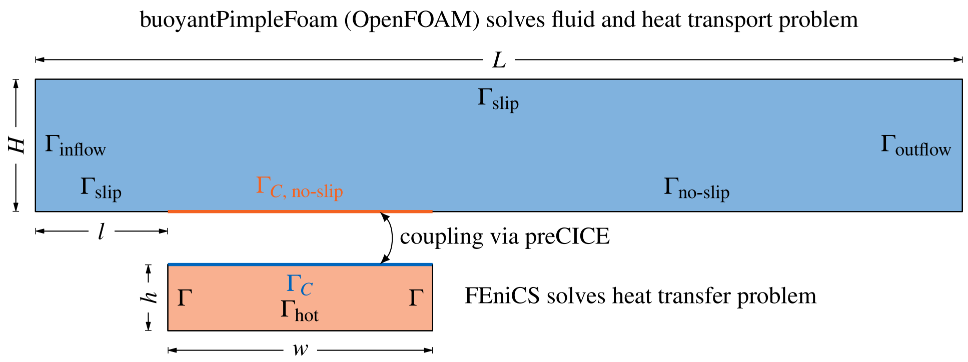
In view of the rapid growth of the preCICE community and the popularity of FEniCS, we expect significant interest in the adapter in the next years. At this point in time (September 2021), preCICE is used by more than 100 research groups,[21] spread over academia, non-academic research centres, and industry.

[19] https://github.com/precice/dealii-adapter/tree/685508e.

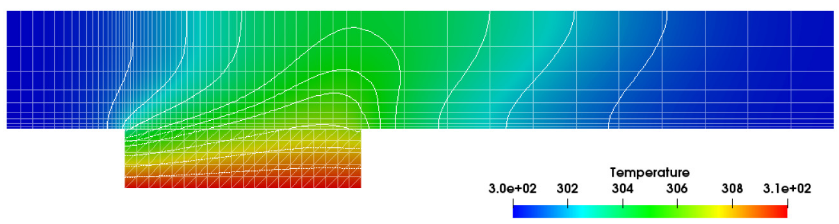
[20] A prototype implementation is available at https://github.com/uekerman/Coupled-Brittle-Fracture.

[21] Some preCICE users wrote testimonials on https://www.precice.org.

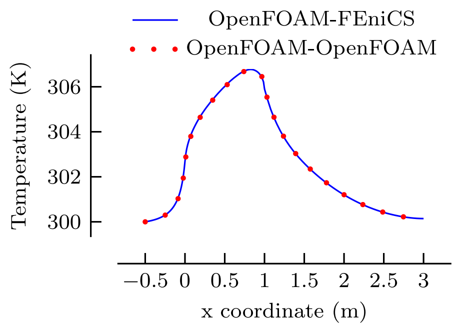
Dimensions	
L	3.50 m
H	0.50 m
l	0.50 m
h	0.25 m
w	1.00 m
z	0.40 m
Fluid	
u_∞	0.1 m/s
T_∞	300 K
g	9.81 m/s^2
k_F	100 W/(m K)
c_p	5000 J/(kg K)
μ	$2 \cdot 10^{-4}$ kg/(m s)
Solid	
T_{hot}	310 K
α	1 m^2/s
k_S	100 W/(m K)


buoyantPimpleFoam (OpenFOAM) solves fluid and heat transport problem
L
Γslip
H
Γinflow
Γoutflow
Γslip
ΓC, no-slip
Γno-slip
l
coupling via preCICE
h
Γ
ΓC
Γhot
Γ
FEniCS solves heat transfer problem
w

(a) Geometric setup and physical parameters. All boundaries except the inflow and outflow boundary (Γ_{inflow}, Γ_{outflow}), the hot bottom of the plate (Γ_{hot}) and the coupling interface (Γ_C) are insulated.



(b) Steady-state temperature distribution. For better visualization, the depicted meshes are 5 times coarser than in reality.



(c) Steady-state temperature profile along a line 0.01m above and parallel to the coupling interface $\Gamma_{C,\text{no-slip}}$. Both profiles differ by a relative l^2 error of $7.64 \cdot 10^{-5}$.

Fig. 8. Conjugate heat transfer testcase.

5. Conclusions

The new software FEniCS–preCICE allows the coupling of FEniCS application codes to other simulation software via preCICE. Our motivation was to make such external coupling as easy as possible for existing FEniCS users. To this end, we hid technical coupling complexity, such as parallel data structures, data conversion, or checkpointing, within the new middle software layer. We were able to implement FEniCS–preCICE as a library and to follow a FEniCS-native style. This allows users of FEniCS to couple existing FEniCS application codes to other simulation software by only adding a few easy-to-understand lines of code. We illustrated the potential by coupling 2D FEniCS application codes to OpenFOAM and SU2 to simulate conjugate heat transfer and fluid–structure interaction problems. The design of the new software already envisions future extensions to other coupled problems. Current work in preCICE itself targets mesh-particle coupling. The current version of FEniCS–preCICE should already allow us to carry this functionality over to FEniCS once released in preCICE. The impact of the new software should be significant. We already mentioned two projects that plan to use the software in the future: coupling fracture mechanics to porous media flow and coupling fracture mechanics to electro-chemistry models. FEniCS itself is currently undergoing a major redesign named DOLFIN-X. Initial investigation showed that it should be possible to easily port FEniCS–preCICE to DOLFIN-X, which we want to realize in future work.

Declaration of competing interest

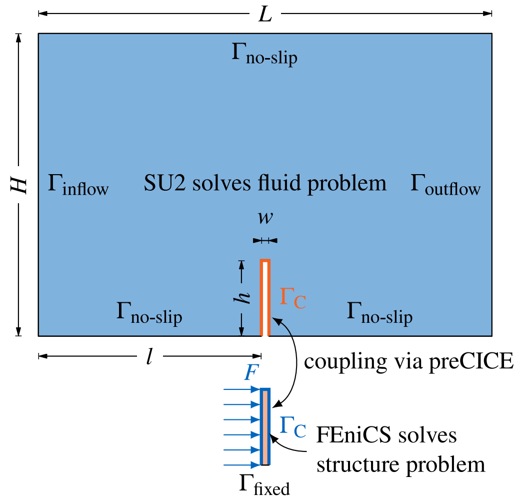
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

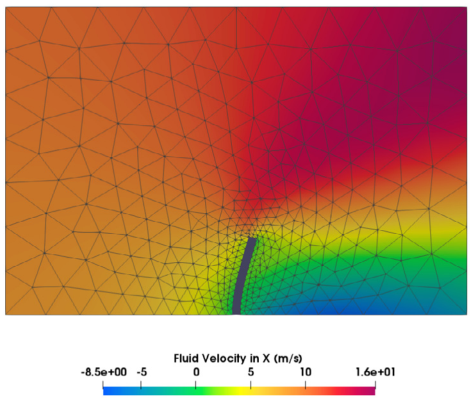
We thank the group of Claus Führer and Philipp Birken at Lund University for hosting Benjamin Rodenberg in Lund, where the first prototype of the adapter was developed. Additionally, we thank Peter Meisrimel from Lund University for technical support and helpful discussions regarding FEniCS. Furthermore, we thank the reviewers of this article for their critical evaluation of the paper and the software and for their valuable feedback.

Benjamin Rodenberg's research stay in Lund was funded by SPPEXA, the DFG (Deutsche Forschungsgemeinschaft, German Research Foundation) Priority Program 1648 – *Software for Exascale Computing*. The work of Benjamin Uekermann has been funded by the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 754462 and by the DFG Cluster of Excellence EXC 2075 *Data-Integrated Simulation Science (SimTech)*, project number 390740016. The work of Ishaan Desai has been funded by the DFG project *preDOM*, project number 391150578, and by *SimTech*. The work of Alexander Jaust has been funded by the DFG SFB 1313, project number 327154368.

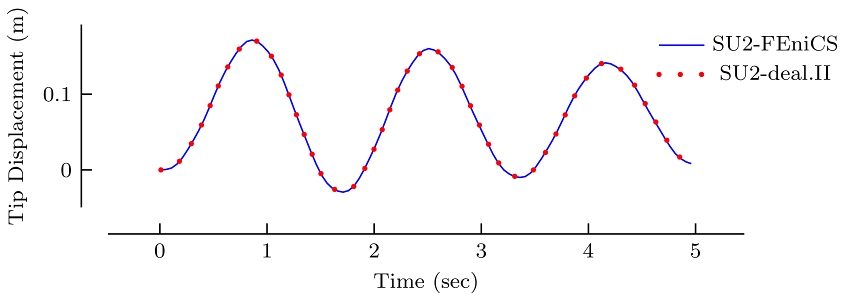
Dimensions	
L	6.00 m
H	4.00 m
l	2.95 m
h	1.00 m
w	0.10 m
Fluid	
Ma_∞	0.01
p_∞	101325 Pa
T_∞	288.15 K
α (AOA)	0°
Solid	
E	$4 \cdot 10^6$ N/m^2
ν_s	$3 \cdot 10^{-1}$
ρ_s	$3 \cdot 10^3$ kg/m^3


L
$\Gamma_{\text{no-slip}}$
H
Γ_{inflow}
SU2 solves fluid problem
Γ_{outflow}
w
h
Γ_C
$\Gamma_{\text{no-slip}}$
$\Gamma_{\text{no-slip}}$
l
coupling via preCICE
F
Γ_C
FEniCS solves structure problem
Γ_{fixed}

(a) Geometric setup and physical parameters



(b) Velocity field of fluid domain



(c) Tip displacement in x direction of the elastic flap over time. Both curves differ by a relative l^2 error of $8.27 \cdot 10^{-3}$

Fig. 9. Fluid–structure interaction test case.

References

[1] Keyes DE, McInnes LC, Woodward C, Gropp W, Myra E, Pernice M, et al. Multiphysics simulations: Challenges and opportunities. Int J High Perform Comput Appl 2013;27(1):4–83. http://dx.doi.org/10.1177/1094342012468181.

[2] Bungartz H-J, Lindner F, Gatzhammer B, Mehl M, Scheufele K, Shukaev A, Uekermann B. preCICE – A fully parallel library for multi-physics surface coupling. Comput & Fluids 2016;141:250–8. http://dx.doi.org/10.1016/j.compfluid.2016.04.003, Advances in Fluid-Structure Interaction.

[3] Alnæs M, Blechta J, Hake J, Johansson A, Kehlet B, Logg A, et al. The FEniCS project version 1.5. Arch Numer Softw 2015;3(100). http://dx.doi.org/10.11588/ans.2015.100.20553.

[4] Logg A, Mardal K-A, Wells GN. Automated solution of differential equations by the finite element method. Springer; 2012, http://dx.doi.org/10.1007/978-3-642-23099-8.

[5] Weller HG, Tabor G, Jasak H, Fureby C. A tensorial approach to computational continuum mechanics using object-oriented techniques. Comput Phys 1998;12(6):620–31. http://dx.doi.org/10.1063/1.168744.

[6] Jasak H. OpenFOAM: Open source CFD in research and industry. Int J Nav Archit Ocean Eng 2009;1(2):89–94. http://dx.doi.org/10.2478/IJNAOE-2013-0011.

[7] Economon T, Palacios F, Copeland S, Lukaczyk TW, Alonso J. SU2: An open-source suite for multiphysics simulation and design. AIAA J 2016;54(3):828–46. http://dx.doi.org/10.2514/1.J053813.

[8] Uekermann B, Bungartz H-J, Yau LC, Chourdakis G, Rusch A. Official preCICE adapters for standard open-source solvers. In: Proceedings of the 7th GACM colloquium on computational mechanics for young scientists from academia; 2017. URL https://www.gacm2017.uni-stuttgart.de/registration/Upload/ExtendedAbstracts/ExtendedAbstract_0138.pdf.

[9] Catty CD, Rognes ME. Mixed-dimensional coupled finite elements in FEniCS. 2019, http://dx.doi.org/10.5281/zenodo.3557448.

[10] Hoffman J, Jansson J, Degirmenci NC, Spühler JH, Vilela De Abreu R, Jansson N, Larcher A. FEniCS-HPC: Coupled multiphysics in computational fluid dynamics. In: Di Napoli E, Hermanns M-A, Iliev H, Lintermann A, Peyser A, editors. High-performance scientific computing. Cham: Springer International Publishing; 2017, p. 58–69. http://dx.doi.org/10.1007/978-3-319-53862-4_6.

[11] Bergersen AW, Slyngstad A, Gjertsen S, Valen-Sendstad K. turtleFSI : A robust and monolithic FEniCS-based fluid-structure interaction solver. J Open Source Softw 2020;5(50):2089. http://dx.doi.org/10.21105/joss.02089.

[12] Damiani LH, Kosakowski G, Glaus MA, Churakov SV. A framework for reactive transport modeling using FEniCS–Reaktoro: governing equations and benchmarking results. Comput Geosci 2020;24(3):1071–85. http://dx.doi.org/10.1007/s10596-019-09919-3.

[13] Massing A, Larson MG, Logg A, Rognes ME. A Nitsche-based cut finite element method for a fluid-structure interaction problem. Commun Appl Math Comput Sci 2015;10(2):97–120. http://dx.doi.org/10.2140/camcos.2015.10.97.

[14] Tang Y-H, Kudo S, Bian X, Li Z, Karniadakis GE. Multiscale universal interface: a concurrent framework for coupling heterogeneous solvers. J Comput Phys 2015;297:13–31. http://dx.doi.org/10.1016/j.jcp.2015.05.004.

[15] Liu W, Wang W, Skillen A, Longshaw S, Moulinec C, Emerson D. A parallel partitioned approach on fluid-structure interaction simulation using the multiscale universal interface coupling library. In: 14th WCCM-ECCOMAS Congress 2020, vol. 1400; 2021. http://dx.doi.org/10.23967/wccm-eccomas.2020.272.

[16] Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn DS, Smith K. Cython: The best of both worlds. Comput Sci Eng 2011;13(2):31–9. http://dx.doi.org/10.1109/MCSE.2010.118.

[17] van der Walt S, Colbert S, Varoquaux G. The NumPy array: A structure for efficient numerical computation. Comput Sci Eng 2011;13(2):22–30. http://dx.doi.org/10.1109/MCSE.2011.37.

[18] Lindner F, Mehl M, Uekermann B. Radial basis function interpolation for black-box multi-physics simulations. In: Proceedings of the VII international conference on coupled problems in science and engineering. CIMNE; 2017, p. 50–61, URL http://hdl.handle.net/2117/190255.

[19] Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat Methods 2020;17:261–72. http://dx.doi.org/10.1038/s41592-019-0686-2.

[20] Hertrich R. Partitioned fluid structure interaction: Coupling FEniCS and OpenFOAM via preCICE. [B.Sc. thesis], Technical University of Munich; 2019, URL https://mediatum.ub.tum.de/1520579.

[21] Bungartz H-J, Lindner F, Mehl M, Scheufele K, Shukaev A, Uekermann B. Partitioned fluid-structure-acoustics interaction on distributed data – coupling via preCICE. In: Bungartz H-J, Neumann P, Nagel EW, editors. Software for exa-scale computing – SPPEXA 2013-2015. Springer; 2016, p. 239–66. http://dx.doi.org/10.1007/978-3-319-40528-5_11.

[22] Totounferoush A, Simonis F, Uekermann B, Schulte M. Efficient and scalable initialization of partitioned coupled simulations with preCICE. Algorithms 2021;14(6). http://dx.doi.org/10.3390/a14060166.

[23] Langtangen HP, Logg A. Solving PDEs in Python - the FEniCS tutorial I. Cham: Springer International Publishing; 2016, http://dx.doi.org/10.1007/978-3-319-52462-7.

[24] Fowler M. Mocks aren't stubs. 2007, URL http://martinfowler.com/articles/mocksArentStubs.html.

[25] Rüth B, Uekermann B, Mehl M, Birken P, Monge A, Bungartz H-J. Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications. Internat J Numer Methods Engrg 2021;122(19):5236–57. http://dx.doi.org/10.1002/nme.6443.

[26] Vynnycky M, Kimura S, Kanev K, Pop I. Forced convection heat transfer from a flat plate: the conjugate problem. Int J Heat Mass Transfer 1998;41(1):45–59. http://dx.doi.org/10.1016/S0017-9310(97)00113-0.

[27] Chourdakis G. A general OpenFOAM adapter for the coupling library preCICE. [M.Sc. thesis], Technical University of Munich; 2017, URL https://mediatum.ub.tum.de/1462269.

[28] Bleyer J. Numerical tours of computational mechanics with FEniCS. Zenodo; 2018, http://dx.doi.org/10.5281/zenodo.1287832.

[29] Rusch A. Extending SU2 to fluid-structure interaction via preCICE. [B.Sc. thesis], Technical University of Munich; 2016, URL https://mediatum.ub.tum.de/1461810.

[30] Arndt D, Bangerth W, Blais B, Clevenger TC, Fehling M, Grayver AV, et al. The `deal.II` library, version 9.2. J Numer Math 2020;28(3):131–46. http://dx.doi.org/10.1515/jnma-2020-0043.

[31] Sonderforschungsbereich 1313, University of Stuttgart. 2020, URL https://www.sfb1313.uni-stuttgart.de/ [Accessed 30 September 2021].

[32] Schmidt P, Jaust A, Steeb H, Mehl M. Simulation of flow in deformable fractures using a quasi-Newton based partitioned coupling approach. 2021, URL https://arxiv.org/abs/2104.05815.

[33] Schmidt P, Steeb H. Numerical aspects of hydro-mechanical coupling of fluid-filled fractures using hybrid-dimensional element formulations and non-conformal meshes. GEM - Int J Geomath 2019;10(1):14. http://dx.doi.org/10.1007/s13137-019-0127-5.

[34] Höche D. Simulation of corrosion product deposit layer growth on bare magnesium galvanically coupled to aluminum. J Electrochem Soc 2014;162(1):C1–C11. http://dx.doi.org/10.1149/2.0071501jes.

[35] Hirshikesh, Natarajan S, Annabattula RK. A FEniCS implementation of the phase field method for quasi-static brittle fracture. Front Struct Civ Eng 2019;13:380–96. http://dx.doi.org/10.1007/s11709-018-0471-9.