Engineering with Computers (2025) 41:1051–1084
https://doi.org/10.1007/s00366-024-01974-4

ORIGINAL ARTICLE

multiRegionFoam: A Unified Multiphysics Framework for Multi-Region Coupled Continuum-Physical Problems

Heba Alkafri[1] · Constantin Habes[1] · Mohammed Elwardi Fadeli[1] · Steffen Hess[2] · Steven B. Beale[2,3] · Shidong Zhang[2] · Hrvoje Jasak[4] · Holger Marschall[1]

Received: 9 July 2023 / Accepted: 22 March 2024 / Published online: 19 October 2024

Abstract

This paper presents a unified framework, called multiRegionFoam, for solving multiphysics problems of the multi-region coupling type within OpenFOAM (FOAM-extend). It is intended to supersede the existing solver with the same name. The design of the new framework is modular, allowing users to assemble a multiphysics problem region-by-region and coupling conditions interface-by-interface. The present approach allows users to choose between deploying either monolithic or partitioned interface coupling for each individual transport equation. The formulation of boundary conditions is generalised in the sense that their implementation is based on the mathematical jump/transmission conditions in the most general form for tensors of any rank. The present contribution focuses on the underlying mathematical model for interface coupled multiphysics problems, as well as on the software design and resulting code structure that enable a flexible and modular approach. Finally, deployment for different multi-region coupling cases is demonstrated, including conjugate heat, multiphase flows and fuel-cells. Source code: multiRegionFoam v1.1 [1], repository https://bitbucket.org/hmarschall/multiregionfoam/.

Article highlights

- A novel multiphysics framework, called multiRegionFoam, has been developed for solving multi-region coupled problems in OpenFOAM.
- The design of the framework allows for a modular multiphysics setup with freedom of choice on the coupling strategy (partitioned vs. monolithic).
- Extension of the general transport equation by interface conditions enables a unified coupling approach.

Keywords Multiphysics · Interface coupling · Multi-region problems · OpenFOAM

1 Introduction

Interface-coupled multi-region problems like fluid–structure interactions, conjugate heat and mass transfer or multi-phase flow problems represent a subgroup of multiphysics problems of high relevance in engineering. Analysis of the underlying continuum-physical models reveals inherent structural similarities, which can be exploited in software design and method development to devise a unified computational multiphysics framework for multi-region coupled continuum-physical problems over a broad spectrum of applications.

✉ Constantin Habes
constantin.habes@tu-darmstadt.de

✉ Holger Marschall
holger.marschall@tu-darmstadt.de

[1] Department of Mathematics, Computational Multiphase Flow Group, Technische Universität Darmstadt, Peter-Grünberg-Straße 10, 64287 Darmstadt, Hessen, Germany

[2] Forschungszentrum Jülich GmbH, Institute of Energy and Climate Research (IEK-9, IEK-13, IEK-14), Wilhelm-Johnen-Straße, 52428 Jülich, Nordrhein-Westfalen, Germany

[3] Department of Mechanical and Materials Engineering, Queen's University, Stuart Street, Kingston, ON K7L 3N6, Canada

[4] The Cavendish Laboratory, Department of Physics, University of Cambridge, JJ Thomson Avenue, Cambridge CB3 0HE, UK

We have developed a novel unified solver framework for computational multiphysics of the multi-region coupling type, i.e. transport processes coupled across region boundaries/interfaces. The code design is such that a multiphysics problem can be assembled region-by-region, and the coupling conditions interface-by-interface. Both monolithic and partitioned coupling for each individual transport equation can be applied as desired by the user. Fluid flow problems are dealt with using SIMPLE, PISO and PIMPLE pressure–velocity algorithms—with loops of predictor and corrector steps across regions. The code is implemented as a C++ library in OpenFOAM (FOAM-extend) for computational continuum physics [2] and follows the principles of object-oriented programming.

We incorporate user-defined types of regions representing sub-domains of specific physics i.e. a set of transport equations. Such a region type should govern a meaningful subset of physics specific to the region—such as e.g. fluid flow, solid mechanics, species and/or energy transport—and can be combined with others. This results into a modular concept and allows to assemble a multiphysics problem region-by-region. The coupling and communication between regions is realised in a modular fashion where interface-specific physics as well as interpolation/mapping methods are accessible in boundary conditions. The implementation is readily parallelised for large scale computations in domain decomposition mode for runs on distributed-memory parallel computer architectures.

Literature survey

There are numerous open-source solutions to cope with multi-region coupled problems. However, the majority of codes are dedicated to specific problems, and thus are following a domain-driven design. In consequence, they cannot be easily adapted to other multi-region coupling problems. Available proprietary simulation codes, on the other-hand-side, often do provide platforms to solve for a broad range of multi-region coupling problems. However, being proprietary the source-codes are non-accessible to the community with the consequence of limited flexibility and/or extensibility, particularly when it comes to very specific engineering applications in technology niches. To alleviate these limitations, (mostly) open-source multi-code coupling approaches have been devised. Examples are the `ADVENTURE_Coupler` (ADVanced ENgineering analysis Tool for Ultra large REal world) [3], `MpCCI` (Mesh-based parallel Code Coupling Interface) [4, 5], `OpenPALM` (Projet d'Assimilation par Logiciel Multimethodes) [6, 7], the `OASIS` coupler [8, 9], `PIKE` (Physics Integration KErnels) [10] as a part of the Trilinos library [11], `preCICE` [12, 13], and `FEniCS` [14–16]. With these coupling software packages, multiple distinct simulations codes are coupled in a co-simulation run—each with an own specialisation coping with the physics in one specific region of the multi-region domain. Note that such code-to-code coupling frameworks for co-simulations are particularly outside the scope of this work, since they inherently only provide partitioned coupling strategies and thus suffer from stability and/or efficiency issues when it comes to challenging, e.g. numerically stiff, coupling problems.

In what follows, we attempt to provide a comprehensive yet concise overview over available open-source multi-region coupling software in the literature, highlighting their coverage of applications.

Alya [17, 18] is a Fortran and C based code developed at Barcelona Supercomputing Center, Spain. It solves coupled multiphysics problems using high performance computing techniques for distributed and shared memory supercomputers. The simulations involve the solution of partial differential equations in an unstructured mesh using finite element methods. Indeed it provides region coupling in a single code environment as well as partitioned multi-code coupling using existing code. Examples of implementation for fluid–structure interaction (FSI) and cojugate heat transfer (CHT) problems, among other applications, are found in [19] and [17].

code_saturne [20] is developed primarily by Électricité de France R&D (EDF) for computational fluid dynamics (CFD) applications. It is written in C and Fortran and relies on finite volume discretisation. It can be coupled with other codes, using its Parallel and Locator Exchange library (PLE) [21], for instance, with SYRTHES [22], a code for transient thermal simulations in solids, to model CHT problems [23]. It also has a module for arbitrary Lagrangian Eulerian (ALE) interface tracking in the frame of fluid–structure interaction [24].

deal.II [25] is a C++ library intended to serve as an interface to the complex data structures and algorithms required for solving partial differential equations using adaptive finite element methods. It is deployed in multiphysics simulations for various applications including FSI in ALE formulation [26], and numerous others [27].

MOOSE stands for Multiphysics Object-Oriented Simulation Environment [28]. It is an open-source C++ based code developed at the Idaho National Laboratory, enabling parallel multiphysics simulation. It uses a finite ele-

ment framework and supports segregated and fully implicit volumetric coupling as well as partitioned interface coupling. Fluid dynamics, heat transfer, and fluid–structure interaction are some of the applications where MOOSE is used [29, 30].

OpenFOAM [2] (Open Field Operation And Manipulation) is an open-source C++ library for computational continuum physics (CCP) including computational fluid dynamics (CFD) based on the finite volume method (FVM) with support for dynamic meshes of general topology (unstructured meshes). Within OpenFOAM, numerous application-specific multi-region frameworks are available mostly from its active developer community. For instance, `solids4foam` [31] has been developed for FSI simulations, `openFuelCell` [32] for modelling fuel cells, `chtMultiRegionFoam` [33] for CHT problems, and `catalyticfoam` [34] for reactive flows at surfaces. Note that besides the above application-specific solutions to interface-coupled multiphysics, OpenFOAM (OpenFOAM-dev) has undergone refactoring towards a "new modular solver framework" [35], in which so-called solver modules can be selected for coupled multi-region simulations. However, with only one solver module selectable for each region, the resulting framework forces the user to develop modules covering the full set of physics in a region. This hinders flexibility and introduces unnecessary complexity. Moreover, interfacial physics has not been modularised and generalised at all. Both aspects have been subject to the present work.

Yales2 [36] aims at solving two-phase combustion from primary atomization to pollutant prediction on massive complex meshes. It is developed at CORIA-CFD using C++ and Fortran. It uses a finite volume solver for multiphysics problems in fluid dynamics, with support for ALE within FSI context [37] in addition to the possibility for multi-code coupling, such as for CHT applications [38] through the OpenPALM library [39].

While the focus here is on open-source, there are also various proprietary software packages developed for similar purposes, such as COMSOL Multiphysics [40], FEATool [41], Fluent [42], and LS-DYNA [43].

Aim and objective

This contribution aims to provide a unified and versatile framework to cope with multiphysics problems of the multi-region coupling type in OpenFOAM. Particular emphasis is put on

- freedom in the choice of coupling strategies—i.e. monolithic and partitioned coupling is at the choice of the user for each individual transport equation,
- ease and flexibility in assembling multiphysics problems—by means of use-defined region types, which also can be superimposed, leading to a multiphysics setup that can be assembled in a modular fashion,
- support of established predictor-corrector based solution algorithms—e.g. to support pressure–velocity, or magnetohydrodynamics in fluid flow across regions,
- strict physics-driven design which allows to clearly separate material models from balance equations—by rigorously exploiting the common mathematical structure of transport equations *and* interface jump and transmission (flux) conditions.

With this, it is hoped to enable substantial coverage over a spectrum of different multiphysics problems of the multi-region coupling type, and to leverage significant synergies among different, so far disjoint domain-expert communities, such that improvements and fixes from one community can directly benefit others.

In the remainder, detailed information on the generic mathematical formulation of the sharp-interface model is given (Sect. 2) which is exploited at the software design stage in the code structure to arrive at a unified multiphysics framework (Sect. 5). Eventually, its deployment for different multi-region coupling cases is demonstrated (Sect. 7).

2 Generic sharp interface model

We aim to provide a concise self-contained derivation of the sharp interface model in its generic formulation as it emerges from balance considerations of conserved quantities. The mathematical procedure will also yield the well-known general transport equation introduced by Spalding in [44]. Thus, the following can also be seen as its extension to general transport equations in multiple domains coupled across sharp interfaces separating the domains. For this, we shall closely follow [45–48].

Starting point of this derivation is a conservation equation in its generic form,

$$\frac{D\Phi}{Dt} = J + S, \tag{1}$$

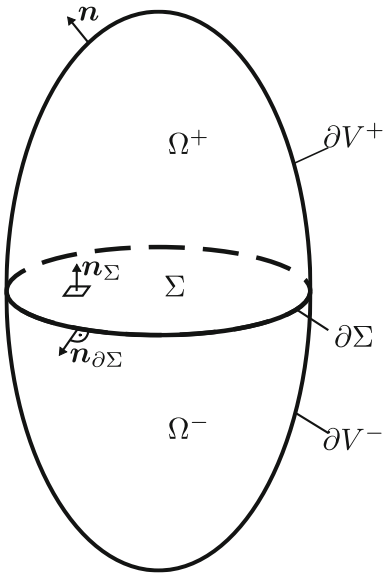


Fig. 1 Material volume V with two subdomains Ω^+ and Ω^- separated by a sharp interface Σ

with the left hand side of (1) being the material derivative of an extensive quantity Φ (e.g. mass, momentum, energy), J denoting the flux term and S being sources/sinks of Φ. We assume this generic conservation equation to be valid in a material control volume $V(t) = \Omega^+(t) \cup \Omega^-(t)$, being composed out of two subdomains Ω^+ and Ω^- (see Fig. 1). The two subdomains are separated by a deformable sharp interface $\Sigma(t) = \partial\Omega^+(t) \cap \partial\Omega^-(t)$ which leads to the fact that $V(t)$ is bounded by $\partial V(t) = \partial V^+(t) \cup \partial V^-(t) \cup \partial\Sigma(t)$, where $\partial V^\pm(t) = \partial\Omega^\pm(t) \backslash \Sigma(t)$. Furthermore, we define that the interface normal $\boldsymbol{n}_\Sigma$ always points from Ω^- to Ω^+ and the interface edge normal $\boldsymbol{n}_{\partial\Sigma}$ points out of Σ.

The sharp interface $\Sigma(t)$ considered here can be seen as a simplification of a transition layer of finite thickness between adjacent domains with different physical properties. Therefore, the interface is not necessarily mass-less and can store conserved quantities which are accounted for in the following by the appearance of so-called surface excess quantities [45]. Thus, the extensive quantity can be written as a volume integral,

$$\Phi = \int_{V(t)} \rho\phi \mathrm{d}\boldsymbol{x} + \int_{\Sigma(t)} \rho^\Sigma \phi^\Sigma \mathrm{d}\boldsymbol{x}, \tag{2}$$

where ρ is the mass density in $\Omega^\pm$, ϕ is the mass density-related volume specific density of the extensive quantity in the bulk and ρ^Σ and ϕ^Σ are their respective area specific excess quantities defined on the interface [46]. The flux term can be expressed through

$$J = -\int_{\partial V(t)} \boldsymbol{j} \cdot \boldsymbol{n} \mathrm{d}s - \int_{\partial\Sigma(t)} \boldsymbol{j}^\Sigma \cdot \boldsymbol{n}_{\partial\Sigma} \mathrm{d}l, \tag{3}$$

with the first integral being a surface integral over the bulk flux density $\boldsymbol{j}$ across the boundary of $V(t)$ and the second integral being a line integral over the interface flux density $\boldsymbol{j}^\Sigma$ across the interface boundary. A similar expression as (2) can be formulated for the source/sink term

$$S = \int_{V(t)} s \mathrm{d}\boldsymbol{x} + \int_{\Sigma(t)} s^\Sigma \mathrm{d}s\,. \tag{4}$$

Here s is the source/sink density field in the bulk and s^Σ is its respective counterpart on the interface. Inserting (2), (3) and (4) into (1) gives

$$\begin{aligned} &\frac{D}{Dt}\int_{V(t)} \rho\phi \mathrm{d}\boldsymbol{x} + \frac{D}{Dt}\int_{\Sigma(t)} \rho^\Sigma \phi^\Sigma \mathrm{d}s \\ &= -\int_{\partial V(t)} \boldsymbol{j} \cdot \boldsymbol{n} \mathrm{d}s - \int_{\partial\Sigma(t)} \boldsymbol{j}^\Sigma \cdot \boldsymbol{n}_{\partial\Sigma} \mathrm{d}l \\ &\quad + \int_{V(t)\backslash\Sigma(t)} s \mathrm{d}\boldsymbol{x} + \int_{\Sigma(t)} s^\Sigma \mathrm{d}s\,. \end{aligned} \tag{5}$$

The generalized transport theorem, [49]

$$\begin{aligned} &\frac{D}{Dt}\int_{V(t)} \rho\phi \mathrm{d}\boldsymbol{x} \\ &= \int_{V(t)\backslash\Sigma(t)} \frac{\partial\rho\phi}{\partial t} \mathrm{d}\boldsymbol{x} + \int_{V(t)\backslash\Sigma(t)} \nabla\cdot(\rho\phi\boldsymbol{u}) \mathrm{d}\boldsymbol{x} \\ &\quad + \int_{\Sigma(t)} [\![\rho\phi(\boldsymbol{u} - \boldsymbol{u}^\Sigma)]\!] \cdot \boldsymbol{n}_\Sigma \mathrm{d}s\,, \end{aligned} \tag{6}$$

can be applied to the first term on the left hand side of (5), where the jump brackets are defined as

$$\begin{aligned} [\![\phi]\!](t,\boldsymbol{x}) &= \lim_{h\to 0^+} \left[\phi\left(t, \boldsymbol{x} + h\boldsymbol{n}_\Sigma\right) - \phi\left(t, \boldsymbol{x} - h\boldsymbol{n}_\Sigma\right)\right], \\ &\boldsymbol{x} \in \Sigma\,. \end{aligned} \tag{7}$$

Furthermore, $\boldsymbol{u}$ is the velocity field in the bulk and $\boldsymbol{u}^\Sigma$ is the interface velocity field which—in the general case of a fluid interface—can have contributions in both normal and tangential direction to the interface. The second term on the left hand side of (5) can be reformulated through the use of the surface transport theorem [46]

$$\begin{aligned} \frac{D}{Dt}\int_{\Sigma(t)} \rho^\Sigma \phi^\Sigma \mathrm{d}s &= \int_{\Sigma(t)} \frac{\partial\rho^\Sigma\phi^\Sigma}{\partial t} \mathrm{d}s \\ &\quad + \int_{\Sigma(t)} \nabla_\Sigma \cdot (\rho^\Sigma \phi^\Sigma \boldsymbol{u}^\Sigma) \mathrm{d}s. \end{aligned} \tag{8}$$

Here the interface Nabla operator is defined by $\nabla_\Sigma = [\mathbf{I} - \boldsymbol{n}_\Sigma \otimes \boldsymbol{n}_\Sigma] \cdot \nabla$, with interface divergence of a vector $\nabla_\Sigma \cdot \boldsymbol{y} = tr(\nabla_\Sigma \boldsymbol{y})$ being the trace of the interface gradient [47]. Using the two-phase divergence theorem [46], the

first term on the right hand side of (5) can be written as

$$-\int_{\partial V(t)} \boldsymbol{j}\cdot\boldsymbol{n}\mathrm{d}s = -\int_{V(t)\setminus\Sigma(t)} \nabla\cdot\boldsymbol{j}\mathrm{d}\boldsymbol{x} - \int_{\Sigma(t)} [\![\boldsymbol{j}]\!]\cdot\boldsymbol{n}_\Sigma \mathrm{d}s. \quad (9)$$

Similarly, the second term on the right hand side of (5) can be expressed using the surface divergence theorem [50]

$$-\int_{\partial\Sigma(t)} \boldsymbol{j}^\Sigma\cdot\boldsymbol{n}_{\partial\Sigma}\mathrm{d}l = -\int_{\Sigma(t)} \nabla_\Sigma\cdot\boldsymbol{j}^\Sigma\mathrm{d}s. \quad (10)$$

Substituting (6), (8), (9) and (10) back into (5) yields

$$\begin{aligned}&\int_{V(t)\setminus\Sigma(t)} \left[\frac{\partial(\rho\phi)}{\partial t} + \nabla\cdot(\rho\phi\boldsymbol{u}) + \nabla\cdot\boldsymbol{j} - s\right]\mathrm{d}\boldsymbol{x}\\ &+\int_{\Sigma(t)} \left[\frac{\partial(\rho^\Sigma\phi^\Sigma)}{\partial t} + \nabla_\Sigma\cdot(\rho^\Sigma\phi^\Sigma\boldsymbol{u}^\Sigma) + \nabla_\Sigma\cdot\boldsymbol{j}^\Sigma\right.\\ &\left.+[\![\rho\phi(\boldsymbol{u}-\boldsymbol{u}^\Sigma)+\boldsymbol{j}]\!]\cdot\boldsymbol{n}_\Sigma - s^\Sigma\right]\mathrm{d}s = 0. \quad (11)\end{aligned}$$

Localizing this expression to points inside the bulk results in the well-known general transport equation

$$\frac{\partial\rho\phi}{\partial t} + \nabla\cdot(\rho\phi\boldsymbol{u}) = -\nabla\cdot\boldsymbol{j} + s \quad (12)$$

introduced by Spalding [44]. Doing the same for points on the interface yields

$$\begin{aligned}&\frac{\partial\left(\rho^\Sigma\phi^\Sigma\right)}{\partial t} + \nabla_\Sigma\cdot(\rho^\Sigma\phi^\Sigma\boldsymbol{u}^\Sigma)\\ &+[\![\rho\phi(\boldsymbol{u}-\boldsymbol{u}^\Sigma)+\boldsymbol{j}]\!]\cdot\boldsymbol{n}_\Sigma = -\nabla_\Sigma\cdot\boldsymbol{j}^\Sigma + s^\Sigma\,, \quad (13)\end{aligned}$$

which is called the interface transmission or flux condition. This equation relates the transport of the surface excess quantities to the jump in convective and diffusive fluxes from the adjacent regions across the interface. By specifying ϕ, $\boldsymbol{j}$, s and their corresponding surface excess quantities it is then possible to obtain the respective transport equations and transmission conditions for mass, momentum, energy and entropy of a continuum to describe. An example of such specifications for a system containing multiple fluid phases that are separated by a sharp, mass-less and capillary interface are given in Table 1. In this table, p represents the pressure, $\boldsymbol{\tau}$ is the deviatoric stress tensor, $\boldsymbol{g}$ is the gravitational vector, e is the internal energy and $\boldsymbol{q}$ is the heat flux vector. The interfacial stress tensor $\boldsymbol{\tau}^\Sigma$ accounts for the interfacial tension.

3 Interface-coupling

Within the sharp interface model framework, adjacent regions are coupled with each other through the interface transmission condition (13). Note that (13) does not represent a boundary condition for the coupling of the primitive fields (unconserved quantities like pressure, velocity or temperature for which the governing equations are solved for and which are used to describe the state of each region). However, it can be reformulated when considering a generic primitive field f [51] to match with

$$[\![\Gamma\nabla f]\!]\cdot\boldsymbol{n}_\Sigma = \mathcal{F}. \quad (14)$$

Here, Γ typically denotes a constant diffusivity but can also be a dependent function of other variables. The jump of the interfacial flux of f (flux discontinuity) is denoted by $\mathcal{F}$, which might also be a dependent function of other variables. When linearised appropriately, (14) can be used as a coupling Neumann boundary condition on either side of the interface.

In addition, one also needs to account for jumps of the primitive fields f at the interface in order to fully describe the interfacial coupling. Such jumps of the primitive fields arise from closure, e.g. when applying methods of rational continuum mechanics which require the second law of thermodynamics to be satisfied in every case—see [45] for a rigorous treatise on this subject. Jump conditions can also be written in a generic form [51],

$$[\![f]\!] = \mathcal{J}\,, \quad (15)$$

where $\mathcal{J}$ represents the interfacial jump of f and may be a dependent function of primitive transport variables. In a linearised form (15) can be applied as a coupling Dirichlet boundary condition on either side of the interface.

When it comes to the algorithmic aspect of region-to-region coupling, monolithic coupling and partitioned coupling types are to be distinguished. In monolithic coupling methods, the coupled equations for each region are assembled and solved simultaneously in one system accounting

Table 1 Example of the specifications of the general transport equation and the generic interface transmission condition describing multiple fluid phases that are separated by a sharp, mass-less and capillary interface

	ϕ	$\boldsymbol{j}$	s	ϕ^Σ	$\boldsymbol{j}^\Sigma$	s^Σ
Mass	1	0	0	0	0	0
Momentum	$\boldsymbol{u}$	$p\mathbf{I}-\boldsymbol{\tau}$	$\rho\boldsymbol{g}$	0	$-\boldsymbol{\tau}^\Sigma$	0
Energy	$e+\frac{\boldsymbol{u}^2}{2}$	$\boldsymbol{q}+(p\mathbf{I}-\boldsymbol{\tau})\cdot\boldsymbol{u}$	b	0	$-\boldsymbol{\tau}^\Sigma\cdot\boldsymbol{u}^\Sigma$	0

for the coupling conditions implicitly. In contrast, in partitioned coupling methods, the equations of each region are solved separately, updating the coupling conditions using iterations [52]. These partitioned methods are most often based on so-called non-overlapping domain decomposition methods—also known as Schwarz methods—and can be of different types [53, 54]. One such type is the Dirichlet-Neumann-Algorithm, which is implemented in the presented framework. Here, each interface is assigned a coupling Dirichlet boundary condition on one side and a coupling Neumann boundary condition on the other side. Since partitioned coupling algorithms in general can lack convergence, acceleration methods are needed during the update procedure [55]. These acceleration methods can either be based on relaxation of the boundary condition values or on quasi-Newton procedures. The `multiRegionFoam` framework implements a fixed and an Aitken relaxation method as well as the IQN-ILS (Interface Quasi Newton Inverse Least Squares) procedure. The interested reader is referred to [55–57] for more details.

4 Notes on OpenFOAM

In the following, we attempt to set out details regarding two essential features OpenFOAM provides *by design*, namely the object registry and runtime selection. Both are crucial to understand at this point, since `multiRegionFoam` leverages them so as to devise a flexible and modular approach for computational multiphysics problems of the multi-region coupling type.

4.1 Run-time selection

Typically a domain expert will be concerned with developing models and/or testing different combinations of models. For this purpose, in the strict object-oriented programming paradigm of OpenFOAM, models are implemented as classes answering to the same interface so they are encapsulated and re-usable. Note that the term 'model' is used here in the broadest sense, e.g. for the choice of linear solvers, or the selection of discretisation schemes, etc. Then, compiling such model classes into shared objects (model libraries) has the advantage that the selection of models can be deferred until run-time (compared to compile-time in traditional factory methods). Together with Run-Time Selection (RTS) tables, models can then be selected from dynamically loaded shared libraries. Such an approach provides ultimate extensibility, since new models can be just added to a RTS table at run-time (by loading shared libraries dynamically) and become available to the top-level solvers just like the models of the same kind from the legacy code. The basic idea of a Run-Time Selection Table is to use a combination of static member variables and methods as well as templates to declare a Hash-Table in the base classes which all child classes register to automatically [58]. It leverages the fact that when a new shared library is loaded, all static variables are immediately initialized, which is exploited to call code that inserts the type name to the parent class's table of models. The parent class's side basically manages a dynamic 'v-table' to construction methods of child classes. Traditionally, a static method (called New) looks up the requested model (provided by user input), constructs the object with its concrete type, but returns a pointer to the base type.

4.2 Object registry

In OpenFOAM the object registry can be thought of as a kind of database that stores information about each object registered to it. It provides a way to keep track of all relevant objects created in a simulation, making it easier to access and manipulate them during runtime. An object registry is implemented using a hash table which belongs to C++ data structures that store key-value pairs. In this case, the keys are strings representing the names of the objects, and the values are references to the objects themselves. All classes in OpenFOAM inheriting from `objectRegistry` represent such an object registry. The most notable ones are the classes `Time`, providing read access to e.g. mesh objects, and `mesh`, providing read access to e.g. field objects.

To look up an object in the registry, OpenFOAM uses a technique called string hashing. If the object is found, its reference is returned. If the object is not found, an exception is thrown indicating that the object does not exist in the registry. In essence, the mechanism relies on two ingredients to check for existence of a requested object and return a reference to it (see Listing 1):

- the object's name, which is assumed to be unique in a single database, and
- the object's type, which is passed as a template argument to the lookup member function. Dynamic casts are simply used to check whether the `regIOobject` object of requested name is also of requested type.

This hierarchy enables flexible access to objects across multiple libraries. These objects are typically declared at the main scope in solver code and persist throughout the solver's execution. The object registration mechanism facilitates obtaining references to these objects from any shared library, provided there is access to the corresponding database (refer to Sect. 4.1 for the significance of this access level). This mechanism effectively eliminates the requirement to pass lengthy lists into class constructors, which otherwise impairs maintainability, extensibility, and generality.

Listing 1 Object lookup example in OpenFOAM

```
// Lookup the velocity field (of type volVectorField) from the mesh
const volVectorField& U = mesh.lookupObject<volVectorField>("U");
```

5 Code structure and design

In its essence, `multiRegionFoam` is a unified framework for multiphysics simulations of region-to-region coupling type. This coverage of distinct region and interfacial physics requires combinatorial flexibility. Therefore, it is designed with attention to the following complementary aspects, in many areas going beyond basic requirements of domain-driven software development in research & development:

- **Usability**. Often domain experts are developing research software using their substantial knowledge on details of the continuum-physical model specific to their own area. To leverage this potential, we have followed the domain-driven software design approach. Utmost attention has been devoted to devise a modular framework for both region- and interface-specific physics which can be developed as entities on their own right. Our aim has indeed been to keep the differences between implementing a module to writing a domain-specific top-level solver in OpenFOAM as low as possible.
- **Understandability**. We have aimed to devise a complete and organized software fabric with a concise, clear as well as descriptive terminology for names of classes, data and functions. This has been motivated by the wish that when presenting `multiRegionFoam` to an engineer not familiar with the code before, basic functionality and principles of use should be easily comprehended.
- **Generality**. Recognising inherent structural similarities which multiphysics problems of the multi-region coupling type have in common, we have devoted significant effort in a general mathematical formulation as foundation of the software design. This has led to a unified framework for multi-region coupling with coverage over a wide range of numerous coupled continuum-physics problems from various distinct fields.
- **Extensibility**. The structure of `multiRegionFoam` has been purposely designed to allow the flexible and non-intrusive addition of new capabilities or functionality. For instance, it is straightforward to add new modules for region- and interface-specific physics and to complement the set of provided coupling algorithms and coupled boundary conditions if needed.
- **Maintainability**. `multiRegionFoam` makes comprehensive use of modern C++, such as classes (encapsulation, inheritance and composition), virtual functions (dynamic polymorphism), and operator overloading. We paid attention to enforcing consistent encapsulations of class families under common interfaces being as small as possible. Classes are minimal in size and as low in complexity as possible, so as to fulfill their (single) task. Moreover, special attention has been paid to avoid code-repetition by means of templating.
- **Robustness**. Substantial efforts have been devoted to provide different coupling strategies. The approach enables to deploy both monolithic and partitioned coupling at the user's choice for each continuum-physical transport equation. This enables to devise the solution strategy to coupled multiphysics problems of region-to-region coupling type on an abstraction level, allowing to have numerical robustness (stability and convergence) in mind despite dealing with substantial model complexity.
- **Parallel Efficiency**. We ensure the possibility to efficiently deploy `multiRegionFoam` on distributed-memory parallel computer architectures on high-performance computing clusters by providing means for straightforward domain-decomposition. In particular, the interface-to-interface communication layer is developed such that it can be used in coupled boundary conditions for data transfer both in serial and parallel in a versatile manner providing multiple mapping strategies.

5.1 Main class structure

We have followed a strictly object-oriented programming paradigm underlying a layered software design. Fig. 2 depicts the class structure following the unified modeling language (UML) class diagram convention [59]. The building blocks of the code structure are two fundamental classes; `regionType` and `regionInterfaceType` which are base classes providing common functionalities that any type of region or interface would require regardless of the simulated physics. Examples of such functionalities for regions include assembling the equations that specify their physical behaviour, correcting the region's material properties and moving its mesh. Similarly for interfaces, examples are administering the protocols for communication and coupling between regions, allowing for deploying various mapping methods, and implementing generic coupled boundary conditions (Sect. 5.4).

From the aforementioned base classes, a code with modular design is devised which relies on the run-time selection mechanism (Sect. 4.1) in which the fundamental inheritance in C++ plays a crucial role. This enables the creation of

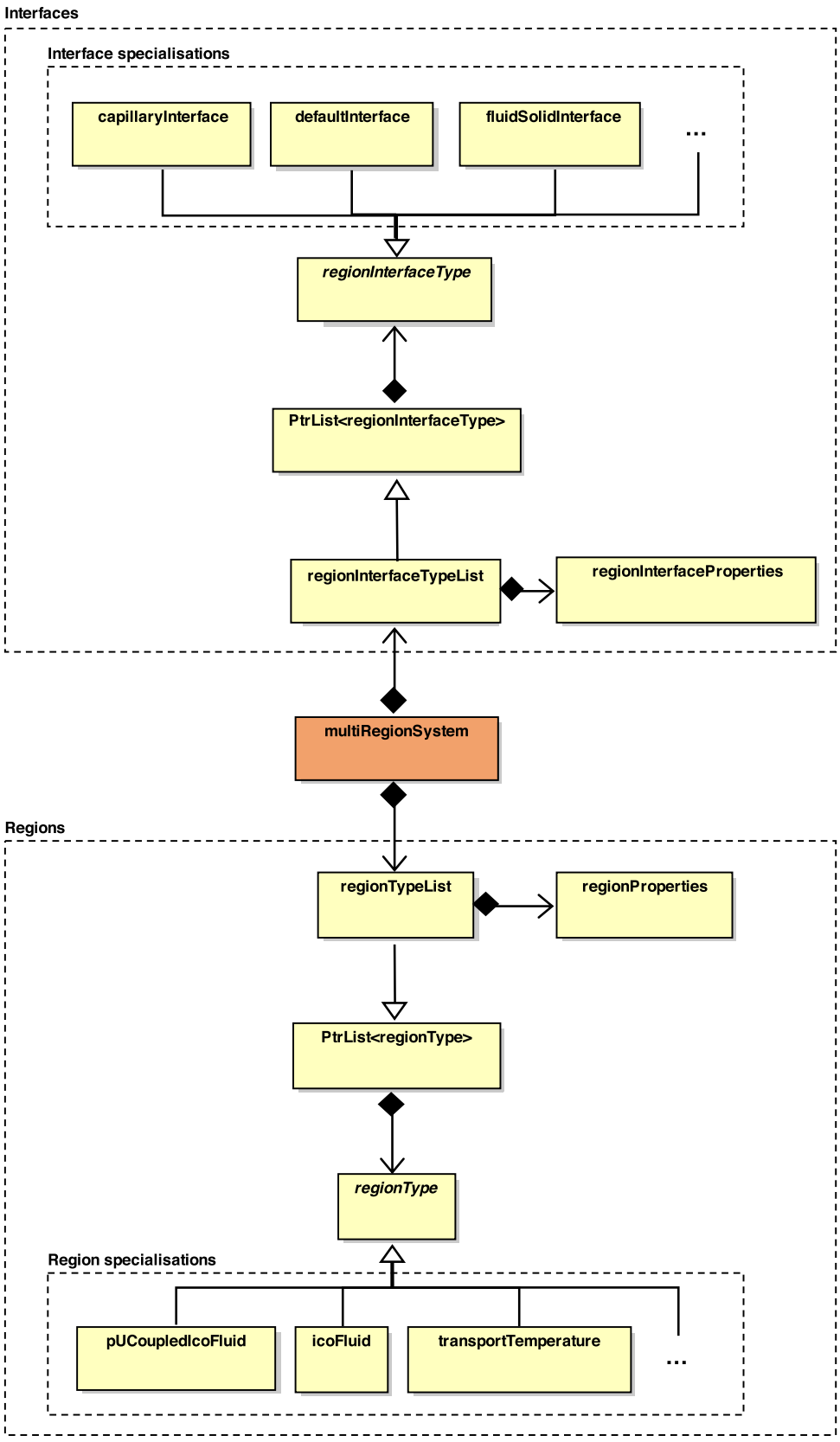


Fig. 2 General structure of the `multiRegionFoam` framework

Listing 2 Object lookup or read helper function

```
template <class T>
inline autoPtr<T> lookupOrRead
(
    const fvMesh& mesh,
    const word& fldName,
    const bool& read=true,
    const bool& write=true,
    const tmp<T> fld = tmp<T>(nullptr)
);
```

derived classes that inherit all common functionalities and extend them to account for additional physical processes at the respective region or interface by defining specialised fields or equations. The inheritance hierarchy is indicated in Fig. 2 by a solid line with a hollow arrowhead pointing from derived to base classes. Currently some specialised region types are implemented such as `icoFluid` for transient flow of incompressible fluid, `conductTemperature` and `transportTemperature` for thermal transport inside the fluid and the solid, respectively. Also different region interface types are already available, like the `capillaryInterface` which is a fluid-fluid interface type accounting for surface tension and surfactant transport and `heatTransferInterface` accounting for heat transfer between adjacent regions. Additional region or interface types can be easily added with this design. Moreover, different region types and region interface types can be superimposed. This is made possible by utilising the functionality of the object registry (Sect. 4.2) and the helper function `lookupOrRead` in Listing 2, so that all superimposed region types have access to the fields present in one region even if they are defined in another one.

The information of the regions and interfaces, such as name, type and the settings for the interfacial coupling algorithm are specified by the user at run-time via the dictionaries `multiRegionProperties` and `regionInterfaceProperties` (see Listings 4 and 3 for a simple conjugate heat transfer (CHT) problem whose setup is also displayed in Fig. 3). The regions and interfaces information is read and stored by the `regionProperties` and `regionInterfaceProperties` classes which are utilised by the `regionTypeList` and `regionInterfaceTypeList` classes to instantiate the regions and the interfaces. The `multiRegionSystem` class builds on these lists to create the multi-region system of equations. These composition relationships are indicated by a solid line with a filled diamond in Fig. 2. It orchestrates the solution process by either applying monolithic or partitioned approaches and therefore acts as the main class interface to the top-level solver `multiRegionFoam`.

5.2 Solution of the coupled system

The set of equations describing the physical behaviour are defined in the specialisations of the `regionType` class in the `setCoupledEqns` function as illustrated in Listing 5 for the transport temperature equation in a fluid region. An instance of the equation system is stored in a hash pointer table, `HashPtrTable`. In this case, it is called `fvScalarMatrices` which points to a finite volume matrix system of scalar type. Other types include vector, tensor, or symmetric tensor matrices, as well as block coupled types `fvBlockMatrix`, as indicated in Listing 6. A unique name identifier is used as a key for the hash table where the same name pattern is used among all types of regions for straightforward access to all coupled equations later when the system is set up and solved.

The multi-region system comprising all such physics-specific equations, defined in different specialisations of region types, is assembled and solved in the `multiRegionSystem` class via the `solve` function shown in Listing 7. The auto pointers `interfaces_` and `regions_` which point to `regionInterfaceTypeList` and `regionTypeList` classes (cf. Fig. 2) provide access to the list of all regions and their interfaces, including the functions defined in their respective specialised classes. For example, using `interfaces_->detach()`, the detach boundary mesh modifier is applied to the interfaces, which triggers the recalculation of the cell to cell/ cell to face distances and interpolation weights at the coupled boundaries in preparation for partitioned coupling. Furthermore, `regions_->solveRegion()` checks for possible individual physics solution requirement in each region. Another function, `solvePIMPLE` (see Sect. 5.3), is dedicated for solving pressure–velocity systems, if any, using the PIMPLE algorithm. A distinction of cases is made based on whether the pressure and velocity fields are coupled across the interface or not. For example, solving the pressure–velocity system in the fluid region in the case of heat transfer between fluid and solid, versus the case of pressure–velocity interfacial coupling between two fluids as in a rising bubble scenario. Moreover, the coupling of the pressure and the velocity fields is made separate from other fields in order

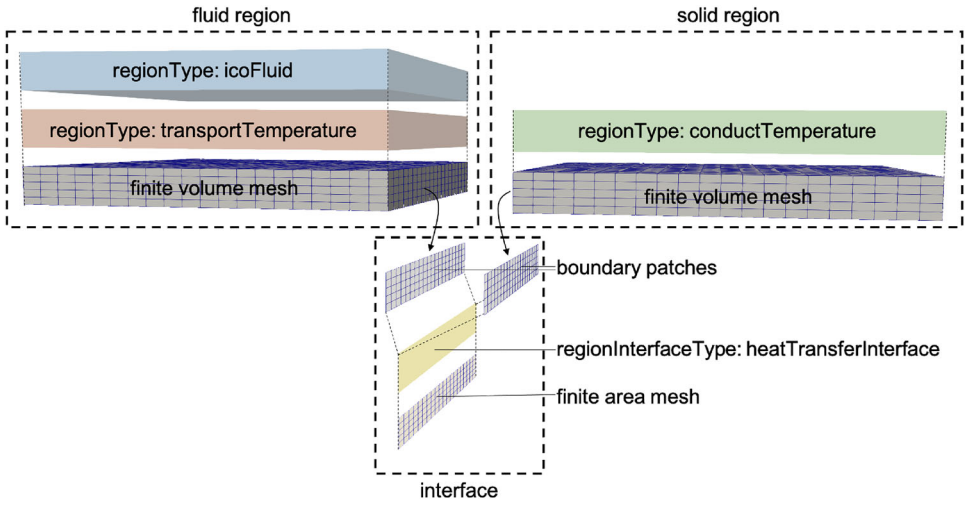


Fig. 3 Example of a heat transfer multi region system being composed out of two regions. The physics of the fluid region is described by the superposition of the two `regionTypes` `icoFluid` and `transportTemperature` while the physics of the solid region is described by the `regionType` `conductTemperature` with a `heatTransferInterface` coupling both regions

Listing 3 `multiRegionProperties` dictionary for a simple CHT problem

```
regions
(
    ( fluid (icoFluid transport Temperature) )
    ( solid (conductTemperature) )
);

DNA // Dirichlet-Neumann Algorithm controls
{
    T
    {
        maxCoupleIter 20;
        residualControl
        {
            maxJumpRes 1e-07;
            outputJumpResField no;
            maxFluxRes 1e-07;
            outputFluxResField no;
        }
    }
}
```

to allow for mesh motion using the Arbitrary Lagrangian–Eulerian (ALE) interface tracking method. For other fields, the system of equations is created and solved under the `assembleAndSolveEqns` function for partitioned coupling and the `assembleAndSolveCoupledMatrix` for monolithic coupling. Each of these functions loops over the list of the partitioned or monolithic coupled fields from the lists `partitionedCoupledFldNames` or `monolithicCoupledFldNames`, respectively. These are `hashedWordLists` reading coupled fields as specified in the `regionInterfaceProperties` dictionary (Listing 4). Note that when the partitioned Dirichlet-Neumann Algorithm (DNA) (Sect. 3) is selected, a convergence criteria is added. It is based on the interface residuals (see Section 4.4.2 in [51]) which are computed in the generic coupled boundary condition classes (Sect. 5.4). `dnaControl` has access to these coupled boundary conditions and residuals for all interfaces. The DNA controlled fields and the termination criteria are specified by the user in the `multiRegionProperties` dictionary as illustrated for the temperature field `T` in Listing 3 under the `DNA` entry The assembly and solution of the system of coupled equations in partitioned mode is shown in Listing 8 which represents the `assembleAndSolveEqns` function of the `multiRegionSystem` class. The function iterates through all regions and obtains a non-constant reference, called `rg`, to the current region. It also constructs a unique name `matrixSystemName` for the equation system for the coupled field in this region. This name matches the pattern of the key assigned for the hash pointer table defined within the `setCoupledEqns` function in the respective specialised region type as mentioned earlier (see Listing 5). The coupled equation matrix defined in the current region is retrieved from the hashed table using the `getCoupledEqn` function which is a member of the `regionType` class. It is defined using a template, which allows it to be specialised for different types of matrices where the type is deduced from the input `matrixSystemName`. The use of templatisation is

Listing 4 `regionInterfaceProperties` dictionary for a simple CHT problem

```
partitionedCoupledPatches
(
    fluidsolid
    {
        interfaceType heatTransferInterface;

        coupledPatchPair
        (
            (fluid bottom)
            (solid top)
        );

        coupledFields
        (
            T
        );

        heatTransferInterfaceCoeffs{}
    }
);
monolithicCoupledPatches();

curvatureCorrectedSurfacePatches 0();
interpolatorUpdateFrequency 1;
interfaceTransferMethod directMap;
directMapCoeffs{}
GGICoeffs{}
```

Listing 5 `setCoupledEqns` of `transportTemperature` region type

```
void Foam::regionTypes::transportTemperature::setCoupledEqns()
{
    //- Create temperature equation system
    TEqn =
    (
        rho_*cp_
       *(
            fvm::ddt(T())
          + fvm::div(phi_(), T())
        )
     ==
        fvm::laplacian(kappa_(), T())
    );
    //- Store equation system in appropriate HashPtrTable
    fvScalarMatrices.set
    (
        T_().name()
      + mesh().name() + "Mesh"
      + transportTemperature::typeName + "Type"
      + "Eqn",
        &TEqn()
    );
}
```

Listing 6 coupled governing equations

```
//- coupled governing equations
HashPtrTable<fvMatrix<scalar> > fvScalarMatrices;
HashPtrTable<fvMatrix<vector> > fvVectorMatrices;
HashPtrTable<fvMatrix<symmTensor> > fvSymmTensorMatrices;
HashPtrTable<fvMatrix<tensor> > fvTensorMatrices;
HashPtrTable<fvBlockMatrix<vector4> > fvVector4Matrices;
```

Listing 7 `solve` function of `multiRegionSystem`

```
void Foam::multiRegionSystem::solve()
{
    //- Detach boundary mesh modifier
    interfaces_->detach();

    //- Solve individual region physics
    regions_->solveRegion();
    //- Check if at least one region implements PIMPLE loop
    //  and solve pressure-velocity system if so
    if (regions_->usesPimple())
    {
        regions_->solvePIMPLE();
    }

    //- Solve region-region coupling (partitioned)
    forAll (partitionedCoupledFldNames_, fldI)
    {
        //- Get name of field which is currently partitioned coupled
        word fldName = partitionedCoupledFldNames_[fldI];
        //- Solve pressure-velocity system using PIMPLE
        if (fldName == "pUPimple")
        {
            while (dnaControls_[fldName]->loop())
            {
                // PIMPLE p-U-coupling
                regions_->solvePIMPLE();
                // ALE mesh motion corrector
                regions_->meshMotionCorrector();
                // Update interface inherent physics
                interfaces_->update();
            }
        }
        else
        {
            //- Solve other partitioned coupled fields
            while (dnaControls_[fldName]->loop())
            {
            assembleAndSolveEqns<fvMatrix, scalar>(fldName);
                // fields of higher tensor rank ...
            }
        }
    }
    //- Attach boundary mesh modifier
    interfaces_->attach();

    //- Solve region-region coupling (monolithic)
    forAll (monolithicCoupledFldNames_, fldI)
    {
        //- Get name of field which is currently monolithic coupled
        word fldName = monolithicCoupledFldNames_[fldI];
        //- Assemble and solve block matrix for monolithic couple fields
         assembleAndSolveCoupledMatrix<fvMatrix, scalar>
        (
            monolithicCoupledScalarFlds_,fldName
        );
        // fields of higher tensor rank ...
    }
}
```

particularly beneficial here to avoid writing multiple identical classes differing only in type. The system of equations is then assembled and solved with the option to perform individual post-solve actions that are implemented in each region.

For monolithic coupling, the procedure starts with attaching the meshes at the interfaces between adjacent regions using the attach boundary mesh modifier `interfaces_->attach()` function as indicated previously in Listing 7. This mesh modifier again triggers the recalculation of the cell to cell/ cell to face distances and interpolation weights at the coupled boundaries in preparation for monolithic coupling. Then, as for partitioned coupling, the `assembleAndSolveCoupledMatrix` function creates and solves the coupled system. However, it is now repre-

Listing 8 `assembleAndSolveEqns` of `multiRegionSystem`

```
template< template<class> class M, class T>
void Foam::multiRegionSystem::assembleAndSolveEqns
(
    word fldName
) const
{
    forAll (regions_(), regI)
    {
        //- Getting non const access to current region
        regionType& rg = const_cast <regionType&>(regions_()[regI]);

        //- Unique name of equation system
        word matrixSystemName =
        (
            fldName
          + rg.mesh().name() + "Mesh"
          + rg.regionTypeName() + "Type"
          + "Eqn"
        );

        //- Sanity checks
        // ...

        //- Get equation from region
        M<T>& eqn = rg.getCoupledEqn<M,T>(matrixSystemName);

        //- Relax equation
        rg.relaxEqn<T>(eqn);

        //- Solve equation
        eqn.solve();

        //- Post solve actions
        rg.postSolve();
    }
}
```

sented as a block coupled finite volume matrix using the `coupledFvMatrix` approach [60] as illustrated in Listing 9. This is a block matrix system that is initialised with the number of monolithic coupled regions. The equations to be loaded into the `coupledFvMatrix` are obtained from the respective regions using the `getCoupledEqn` function analogously to the partitioned approach by iterating over the regions and collecting equations by their unique name. Unlike partitioned coupling, the retrieved coupled equations are not directly solved but instead inserted into the block matrix system. They are first appended into the dynamic list of equations `eqns` which holds pointers to the equation matrices for all regions. The `new` operator dynamically allocates memory for each instance of a coupled equation appended to the list. Finally, the equations are solved in the block matrix system simultaneously in an implicit manner using the linear equation solver specified by the solution dictionary of the first region's mesh for the given coupled field name.

5.3 Pressure–velocity coupling

The pressure–velocity coupling is solved in a semi-implicit manner using the PIMPLE algorithm which is a combination of the well known Pressure-Implicit with Splitting of Operators (PISO) algorithm [61] and a more consistent version of the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) algorithm [62]. This involves a predictor-corrector procedure in which a tentative solution for the velocity is obtained in the predictor step while the pressure is updated in the corrector step. The standard implementations of these algorithms in OpenFOAM are refactored and integrated into the class structure of `multiRegionFoam` (see Fig. 2). The code is implemented in the `icoFluid` region type class which needs to be specified by the user in the `multiRegionProperties` dictionary (Listing 3). The code block for each of the steps of the PIMPLE algorithm are defined in the `momentumPredictor` and the `pressureCorrector` functions. These are called in the `regionTypeList` class by utilising the concept of operator overloading in C++, as illustrated in Listing 10. The `solvePIMPLE()` function represents the skeleton of the PIMPLE algorithm which outlines the outer and the inner loops and the execution across regions. In order to ensure the consistency of the pressure–velocity coupling, each step of the solution procedure is performed for all regions before the next step is carried out.

Listing 9 `assembleAndSolveCoupledMatrix` of `multiRegionSystem`

```
template< template<class> class M, class T>
void Foam::multiRegionSystem::assembleAndSolveCoupledMatrix
(
    PtrList<GeometricField<T, fvPatch Field, volMesh> >& flds,
    word fldName
) const
{
    //- Get number of monolitic coupled regions per field name
    //  and return if there are none
    // ...

    //- Initialise block matrix system
    //  with number of monolitic coupled regions
    coupledFvMatrix<T> coupledEqns(nEqns);

    //- Assemble all matrices one-by-one and combine them into the
    //  block matrix system
    label nReg = 0;
    DynamicList<M<T>* > eqns;
    forAll (regions_(), regI)
    {
        //- Getting non const access to current region
        regionType& rg = const_cast<regionType&>(regions_()[regI]);

        //- Unique name of equation system
        word matrixSystemName =
        (
            fldName
          + rg.mesh().name() + "Mesh"
          + rg.regionTypeName() + "Type"
          + "Eqn"
        );

        //- Get equation from region
        M<T>& eqn = rg.getCoupledEqn<M,T>(matrixSystemName);

        //- Insert matrix into block matrix system
        eqns.append(new M<T>(eqn));
        coupledEqns.set(nReg, eqns[nReg]);
        nReg++;
    }

    //- Solve block matrix system
    coupledEqns.solve
    (
        regions_()[0].mesh().solutionDict().solver(fldName + "coupled")
    );

    // Post solve actions for monolithic coupled fields
    // ...
}
```

5.4 Generic boundary conditions for partitioned coupling

As described in Sect. 3, the solution of the multi-region system using a partitioned approach is achieved via the Dirichlet-Neumann-Algorithm. This requires specifying boundary conditions on the connecting boundaries of the sub-regions where a Dirichlet condition is applied on one side while a Neumann condition is applied on the other one. For this purpose, a set of generic boundary conditions are devised exploiting the fact that these conditions have a general mathematical formulation (Sect. 2). Fig. 4 shows how the implementation is integrated into the structure of `multiRegionFoam`.

The generic boundary conditions are derived from the standard OpenFOAM Dirichlet and Neumann boundary conditions, namely `fixedValueFvPatchField` and `fixedGradientFvPatchField`. This results in two classes `genericRegionCoupledValueFvPatchField` and `genericRegionCoupledFluxFvPatchField` which implement the interfacial jump and transmission conditions, respectively. Utilising templates, these conditions are made agnostic to the actual physics that the jumps and fluxes originate from and rather receive this information from the relevant specialisation of `regionInterfaceTypes`. The derived coupled boundary conditions also inherit from the `interfaceToInterfaceCoupleManager` class which gives access to the neighbour region data,

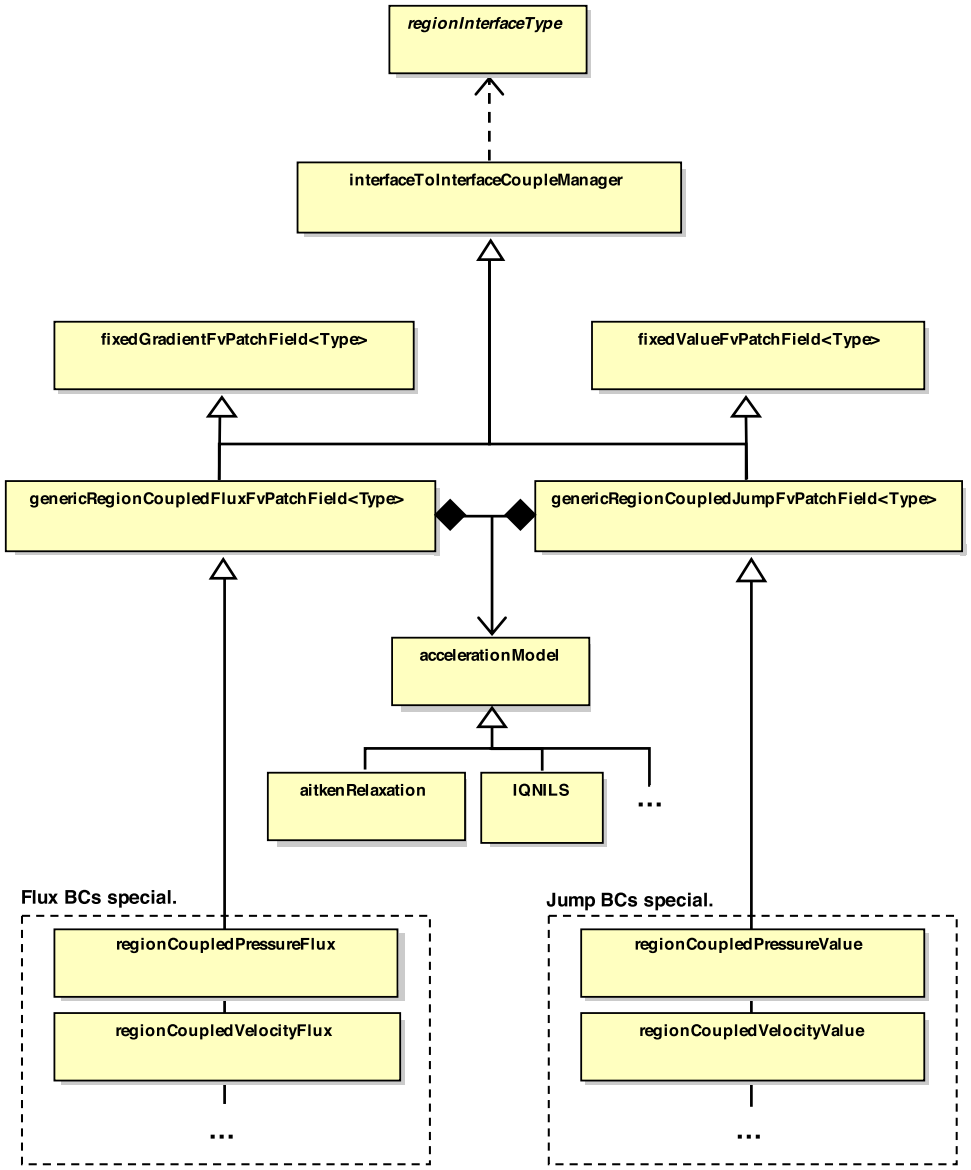


Fig. 4 General structure of the generic coupled boundary condition

such as the mesh and patch names, and has access to the `regionInterfaceType` on which the coupled boundary condition is applied. For their use in the partitioned interface coupling solution approach they are also able to utilize different convergence acceleration methods including, besides others, the Aitken relaxation method [55] and the Interface Quasi-Newton Inverse Least-Squares method (IQN-ILS) [63].

5.5 Parallelisation

In OpenFOAM, parallelization is generally implemented through the domain decomposition method. This typically requires dividing the solution domain into sub-regions in order for each of them to be solved on separate CPU cores. The standard Message Passing Interface (MPI) [64] is then utilised to establish communication between the processors. Although the proposed multi-region framework already requires subdivision of the computational mesh into multiple domains, these sub-domains are not convenient for efficient parallel computing, especially because a unified framework is sought. For example, if partitioned coupling is selected, then only one region will be solved at a time, or if the regions vary significantly in size in some scenarios, this will lead to an unbalanced distribution of load over the processors. Instead, all regions are decomposed into the same number of sub-domains but not necessarily using the same decomposition method. Fig. 5 depicts an example of two interface coupled regions representing gas bubble and liquid where both phases are decomposed into two sub-domains but one is decomposed vertically while the other is decomposed horizontally. This causes the two patches, that form

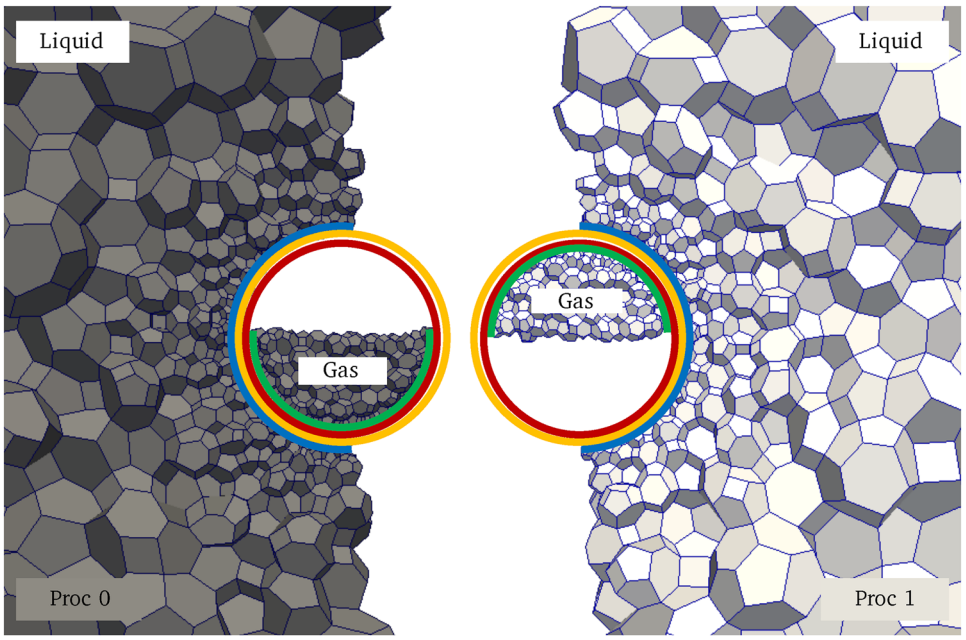


Fig. 5 Local (blue, green) and global (red, orange) poly patches of a gas and a liquid region decomposition [51]

```
testSuite
├── baseCase
│   ├── ...
│   ├── Allclean ..................................... (Listing 12)
│   ├── Allrun ....................................... (Listing 12)
│   └── test_template.py ............................. (Listing 11)
├── tests
│   ├── monolithic
│   │   ├── Pr_0.01_Re_500_k_1
│   │   ├── Pr_0.01_Re_500_k_5
│   │   └── ...
│   └── partitioned_Aitken
│       ├── Pr_0.01_Re_500_k_1
│       └── ...
├── pytest.ini
├── conftest.py
├── genCases.py ...................................... (Listing 15)
├── getResults.py .................................... (Listing 14)
├── run
└── clean
```

Fig. 6 Directory structure for the testing framework

the two sides of the interface, to be non-entirely overlapping (cf. blue and green lines in Fig. 5) which makes the fields mapping between the two sides impossible. To overcome this situation, the *global face zone* approach, introduced by Cardiff et al. [31] and Tuković et al. [65], is used. It suggests that each processor is given access to the entire interface patch via the so-called `globalPolyPatch` (red and orange lines in Fig. 5) which resembles the union of the non-overlapping boundaries, holding copies of their data and allowing for mapping the fields from one global patch to the other. These global patches also facilitate the implementation of the coupled boundary conditions. To adapt this concept in `multiRegionFoam`, the `globalPolyPatch` is incorporated into the `regionInterfaceType` class, i.e. it includes a pair of the local interface patches as well as pair of global patches.

6 Automated test harness

In order to validate and evaluate different aspects of `multiRegionFoam`, an automated test framework is

Listing 10 `solvePIMPLE` of `regionTypeList`

```
void Foam::regionTypeList::solvePIMPLE()
{
    //- Get the number of outer correctors that should be performed
    //  during the PIMPLE procedure (We do not have a top-level
    //  mesh. Construct fvSolution for the runTime instead.)
    fvSolution solutionDict(runTime_);
    const dictionary& pimple = solutionDict.subDict("PIMPLE");
    int nOuterCorr(readInt(pimple.lookup("nOuterCorrectors")));

    //- PIMPLE loop
    for (int oCorr=0; oCorr<nOuterCorr; oCorr++)
    {
        forAll(*this, i)
        {
            this->operator[](i).prePredictor();
        }

        forAll(*this, i)
        {
            this->operator[](i).momentumPredictor();
        }

        forAll(*this, i)
        {
            this->operator[](i).pressureCorrector();
        }
    }
}
```

Listing 11 Test case completion check using `oftest` and `pytest` libraries

```
def test_completed(run_reset_case):
    log = oftest.path_log()
    assert oftest.case_status(log) == 'completed'
```

Listing 12 `oftest` requirement for `Allrun` & `Allclean` scripts

```
cd ${0%/*} || exit 1 # Run from this directory
# ...
```

deployed. The main goal is to automatically run a suite of test cases and report on the results. This includes testing whether each simulation was successfully executed and completed as well as checking the obtained results against some criteria, such as a reference solution or an error threshold. The testing framework is also useful for parameter studies, continuous integration, and testing of further developments or new features. The tests are performed using Python while utilising the `oftest` library [66] which is a test framework for OpenFOAM cases available as open source under the GPLv3 License. It makes use of the `pytest` library [67] and `pytest-xdist` [68] plugin to run the tests across multiple CPUs. The implementation is illustrated for the flow over a heated plate scenario (see Sect. 7.1). The main study investigates the interface-coupling approach (monolithic/partitioned), where the latter could be performed with different acceleration methods (cf. Sect. 3). Each coupling approach entails a sub-study of the parameters Re, Pr, and k which correspond to the Reynolds number, Prandtl number, and thermal conductivity ratio, respectively. The directory structure of the test suite is depicted in Fig. 6, emphasizing the necessary files for the test setup and execution. An overview of these files and the testing process is described below:

- A `baseCase` folder is required which, in addition to the standard OpenFOAM case files, contains the `test_template.py` script provided in Listing 11. In this script the `test_completed` function adds the case to the pool of tests, returns the status of completion (passed/failed) in the terminal, and leaves the simulation log file in the specified path. The `run_reset_case` module from `oftest` library is responsible for running and cleaning the case according to the standard `Allrun` and `Allclean` files. These must include the requirement in Listing 12.

Listing 13 `run` script to automate the execution of all the testing steps

```
./clean
python genCases.py
pytest --no-clean-up -n auto tests
python getResults.py
```

- The `tests` folder includes the variations of the base case which are generated using any preferred tool, such as `PyFoam` [69] or its additional module `CaseFOAM` [70]. The latter is particularly advantageous when the study involves subcases associated with each of the primary ones. Thus, it is used in this scenario as illustrated in Listing 15, which shows the content of the `genCases.py` script.
- The results, typically generated from a post processing utility, are accessed and analyzed in the `getResults.py` file. A snippet is provided in Listing 14 where the `CaseFOAM` built-in function `positional_field` is used to retrieve and combine the results from all tests into a single data structure. This can be further analyzed using standard Python libraries for data management and visualisation.
- The files `pytest.ini` and `conftest.py` are configuration files for global settings of the test execution, as well as custom options specifying, for example, the number of time steps or whether the test cases are cleaned or kept after running.
- With the required files in place and a sourced OpenFOAM, the `run` script (Listing 13) automates the execution of all the steps. The option `-n auto` indicates using as many processors as available, while `–no-clean-up` specifies not to clean the case after run.

7 Examples of usage

A collection of test cases is presented in this section to showcase the usage of `multiRegionFoam`, including validation studies where analytical or experimental results are available in the literature, in addition to advanced industrial applications.

7.1 Forced convection heat transfer from a flat plate

In this case, an incompressible laminar flow over a flat plate is considered, following the description from Vynnycky et al. [71]. Their numerical results as well as their derived reference solution, using boundary-layer theory, are used for validation. A schematic sketch of the case setup is shown in Fig. 7. A fluid of uniform temperature T_∞ and velocity U_∞ flows over a plate of finite thickness that is held at constant temperature $T_s > T_\infty$.

Table 2 Boundary conditions for the flow over a heated plate

Boundary	Thermal	Velocity
Fluid		
inlet	300 K	$(1\ 0\ 0)^\top$ m/s
bottom	coupled	$(0\ 0\ 0)^\top$ m/s
slip-bottom (before the plate)	zeroGradient	zeroGradient
noSlip-bottom (after the plate)	zeroGradient	$(0\ 0\ 0)^\top$ m/s
Outlet, top	zeroGradient	zeroGradient
Solid		
top	coupled	–
bottom	310 K	–
left, right	zeroGradient	–

Table 4 Parameters for the flow over a heated plate simulation

Re	Pr	k
500	0.01	1, 5, 20
0000	0.01	1, 5, 20
500	100	1, 5, 20

Table 5 Thermophysical properties of the fluid and solid for the flow over a heated plate

Property	Symbol	Unit	Solid	Fluid
Density	ρ	kg/m^3	1	1
Dynamic viscosity	μ	kg/ms	–	$\rho_f U_\infty L/\mathrm{Re}$
Thermal conductivity	k	W/m · K	100	k_s/k
Specific heat capacity	c_p	J/kg · K	100	$k_f \mathrm{Pr}/\mu$

The computational mesh is depicted in Fig. 8. It covers the fluid and solid regions, both consisting of hexahedral elements. In order to capture the thermal and viscous boundary layers in the fluid, mesh grading is used to obtain a finer mesh at the fluid–solid interface and the leading edge of the plate. The two regions are coupled at the fluid–solid interface with a `heatTransferInterface regionInterfaceType` which consists of the meshes boundary patches, namely the "bottom" patch from the fluid and the "top" patch from the solid. Table 3 shows the thermal coupled boundary conditions used in `multiRegionFoam`. A Dirichlet condition is applied at the fluid side of the fluid–solid interface while a Neumann condition is specified at the

Listing 14 Snippet of `getResults.py` to retrieve the tests results using `CaseFOAM`

```
# Specify baseCaseDir and caseStructure as in genCases.py  (cf. Listing 15)
# Specify directories in the postProcessing folder to retrieve data from
TdataDir = 'sets/fluid'
Tdata = casefoam.positional_field (TdataDir, 'DataFile_T.xy', 10,
                                   caseStructure, baseCaseDir)
# Rename the columns
Tdata.columns = ['x','T','coupling method','parameters']

# Calculate a new column (theta) from the temperature
Tdata['theta'] = (Tdata['T']-300)/(310-300)
```

Table 3 Coupled thermal boundary conditions for the flow over a heated plate simulation

Region	Boundary	Partitioned	Monolithic
Fluid	bottom	regionCoupledTemperatureJump	monolithicTemperature
Solid	top	regionCoupledHeatFlux	monolithicTemperature

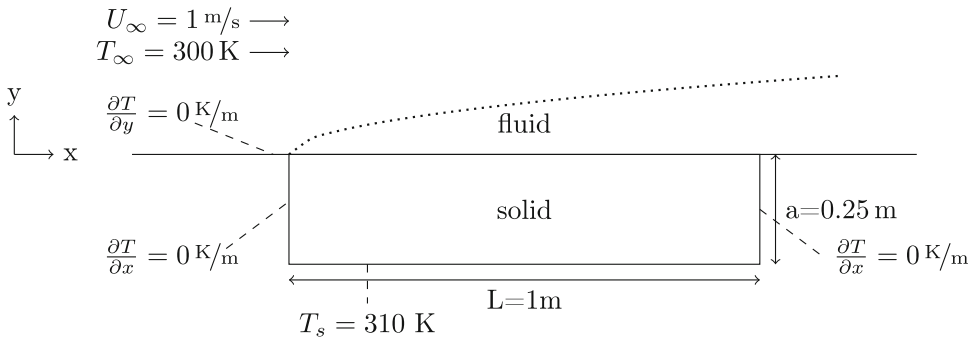


Fig. 7 Computational domain and boundary conditions for the flow over a heated plate

solid side. The generic jump and flux boundary conditions (Sect. 5.4) are used for partitioned coupling while the region couple patch field `monolithicTemperature` is used for monolithic coupling. The rest of the boundary conditions are summarized in Table 2. The authors of [71] investigated several factors affecting heat transfer, including the aspect ratio of the plate $\lambda = a/L$, the Reynolds number, Re, the Prandtl number, Pr, and the thermal conductivity ratio, $k = k_s/k_f$, between the plate and the fluid. In this study, the aspect ratio is fixed at $\lambda = 0.25$, while different combinations of the

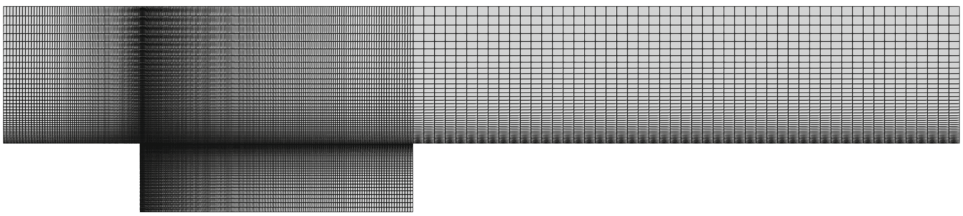


Fig. 8 Meshes for the flow over a heated plate simulation

Table 6 Numerical schemes for the flow over a heated plate simulation

	Scheme	Setting
Time scheme	ddtScheme	backward
Finite volume schemes	gradScheme	leastSquares
	divScheme div(phi,U)	Gauss upwind
	divScheme div(phi,T)	Gauss linearUpwind Gauss linear
	laplacianScheme	Gauss linear corrected
	interpolationScheme	linear
	snGradScheme	corrected

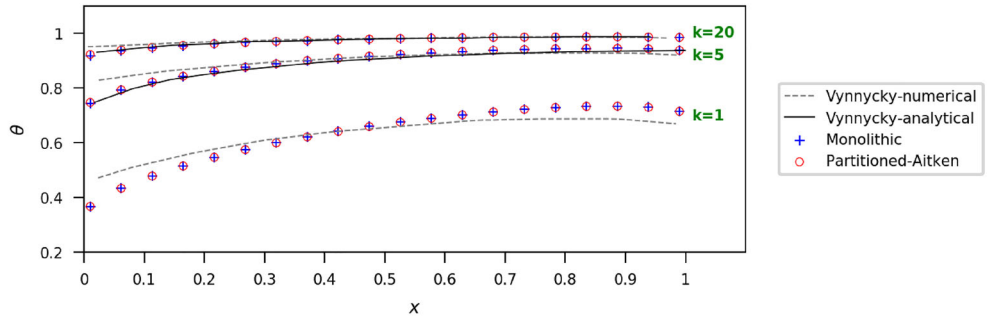
Listing 15 `genCases.py` script to generate the parameter study with `CaseFOAM`

```
# Specify the base case name and a directory name for the tests
baseCase = 'baseCase'
baseCaseDir = 'tests'

# Specify the names of the test
cases caseStructure = [# main cases
                       ['monolithic',' partitioned_Aitken'],
                       # subcases for each of the main cases
                       ['Pr_0.01_Re_500_k_1', 'Pr_0.01_Re_500_k_5',
                        'Pr_0.01_Re_500_k_20', 'Pr_0.01_Re_10000_k_1',
                        'Pr_0.01_Re_10000_k_5', 'Pr_0.01_Re_10000_k_20',
                        'Pr_100_Re_500_k_1', 'Pr_100_Re_500_k_5',
                        'Pr_100_Re_500_k_20']]

# Partitioned coupling with acceleration type and init. relaxation factor
def p_coupled(accType, relaxValue):
    return {
        '0/fluid/orig/partitioned/T': {'boundaryField':
                                        {'interface':
                                          {'accType': accType,
                                           'relax' : relaxValue}}},
        '#!bash': 'cp baseCase/AllrunP tests/baseCase/Allrun'
    }

# Monolithic coupling
m_coupled = {'#!bash': 'cp baseCase/AllrunM  tests/baseCase/Allrun'}

# Parameters
def update_params(Pr, Re, k):
    L = 1.0
    rhof = 1.0
    ks = 100.0
    Uinf = 1.0
    mu = rhof*Uinf*L/int(Re)
    kf = ks/k
    cp = kf*Pr/mu

    return {
        'constant/fluid/transport Properties': {
                'mu': 'mu [ 1 -1 -1 0 0 0 0 ] %s' %mu,
                'cp': 'cp [ 0 2 -2 -1 0 0 0 ] %s' %cp,
                'k': 'k [1 1 -3 -1 0 0 0] %s' %kf},
        '0/fluid/orig/monolithic/k': {'internalField': 'uniform %s' %kf}
    }

# Define input parameters for each of the cases listed in "caseStructure"
caseData = {
    'monolithic' : m_coupled,
    'partitioned_Aitken_0.75': p_coupled('aitken', '0.75'),
    'Pr_0.01_Re_500_k_1': update_params(0.01, 500, 1),
    'Pr_0.01_Re_500_k_5':update_params(0.01, 500, 5),
    'Pr_0.01_Re_500_k_20': update_params(0.01, 500, 20),
    'Pr_0.01_Re_10000_k_1': update_params(0.01, 10000, 1),
    'Pr_0.01_Re_10000_k_5': update_params(0.01, 10000, 5),
    'Pr_0.01_Re_10000_k_20': update_params(0.01, 10000, 20),
    'Pr_100_Re_500_k_1': update_params(100, 500, 1),
    'Pr_100_Re_500_k_5': update_params(100, 500, 5),
    'Pr_100_Re_500_k_20': update_params(100, 500, 20)
}

casefoam.mkCases(baseCase, caseStructure, caseData,
                 hierarchy='tree', writeDir=baseCaseDir)
```

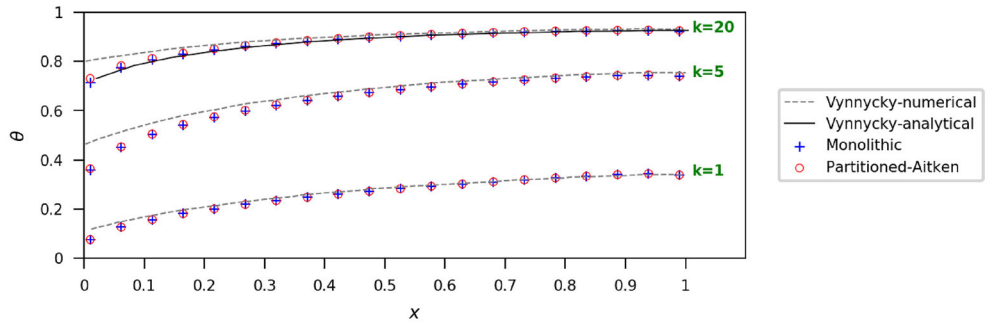
other parameters are considered as specified in Table 4; the thermophysical properties of the fluid and solid are given in Table 5. The simulations are run for 10 s using a time step size of $\Delta t = 0.01$ s. The numerical schemes used are listed in Table 6 according to OpenFOAM equivalent terms [72].

The results are validated by computing the dimensionless conjugate boundary temperature, θ, defined as

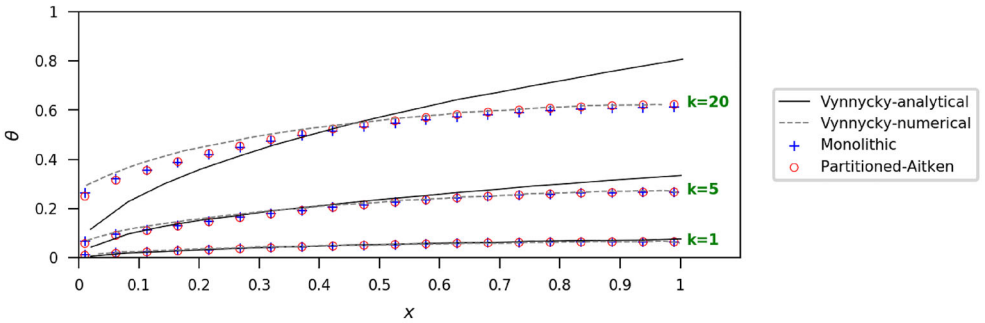
$$\theta = \frac{T - T_\infty}{T_s - T_\infty},$$



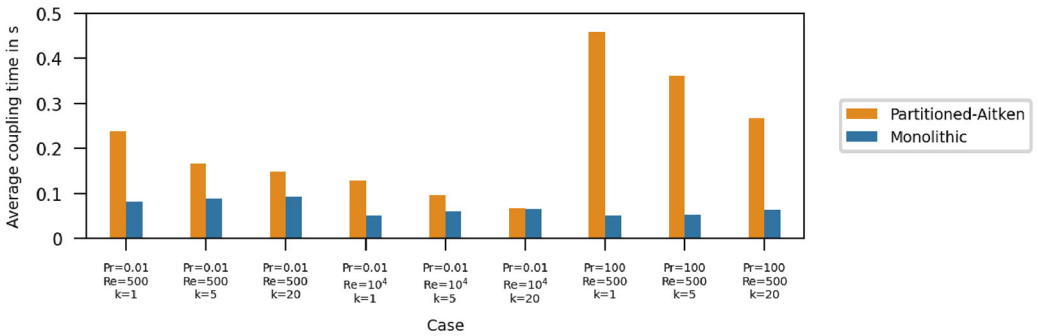
(a) Dimensionless conjugate boundary temperature θ over non-dimensional distance x for Pr = 0.01 and Re = 500



(b) Dimensionless conjugate boundary temperature θ over non-dimensional distance x for Pr = 0.01 and Re = 10^4



(c) Dimensionless conjugate boundary temperature θ over non-dimensional distance x for Pr = 100 and Re = 500



(d) Average coupling time for partitioned and monolithic coupling

Fig. 9 Simulation results for different Pr, Re, and k values

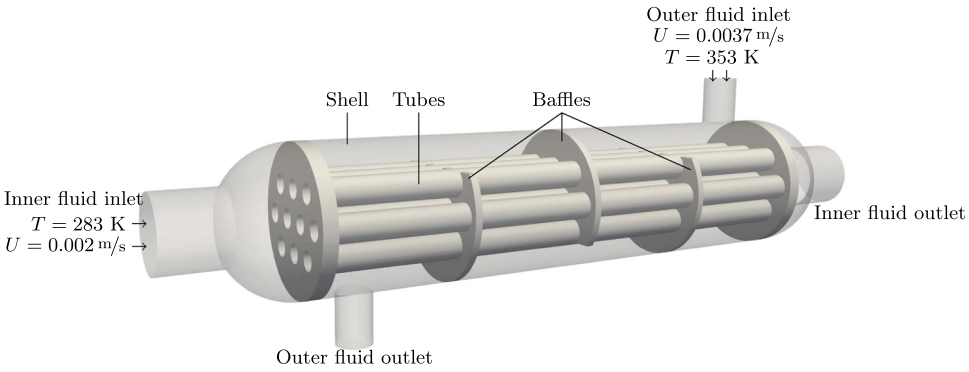


Fig. 10 Geometry for the heat exchanger

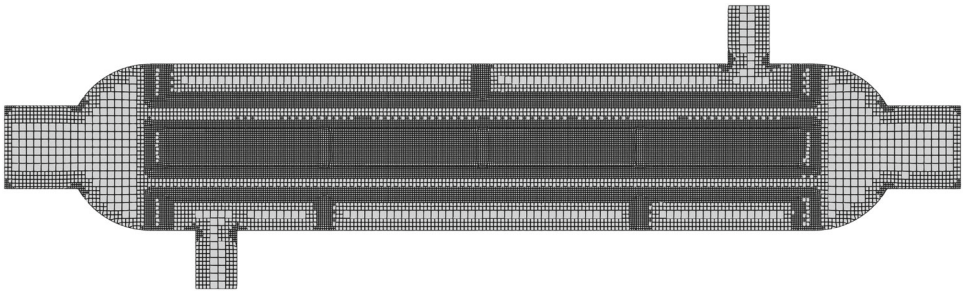


Fig. 11 Meshes for the heat exchanger simulation

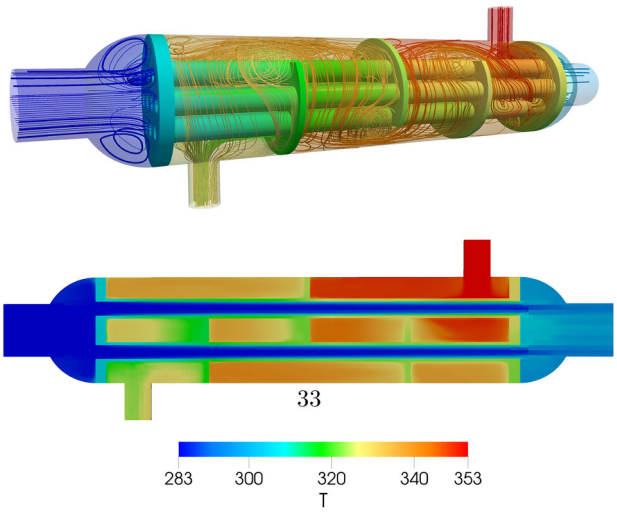


Fig. 12 Final temperature distribution for the heat exchanger simulation

where T is the temperature along the fluid–solid interface. Fig. 9 summarizes the results of `multiRegionFoam` using monolithic and partitioned coupling. The latter was performed with Aitken's relaxation procedure to accelerate the convergence. Both approaches result in good agreement with the numerical and analytical results from Vynnycky et al. [71]. The deviations from the reference solutions in Fig. 9c are due to the fact that the solutions for the case $Pr \gg 1$ were derived under the constant-flux approximation assumption, which does not provide an accurate description of the flow as k increases. Fig. 9d reports the average time spent on the coupling during one time step using partitioned and monolithic approaches. In comparison to partitioned coupling, monolithic coupling exhibits either the same or reduced average coupling time across all cases. However, a notable distinction is observed when Pr is equal to 100. Notably, for monolithic coupling, the time remains almost constant regardless of the simulated parameters.

7.2 Shell-and-tube heat exchanger

The next example demonstrates an industrial application where conjugate heat transfer takes place. Fig. 10 shows a shell-and-tube heat exchanger. This particular design includes a shell, tubes, and baffles. Heat transfer occurs between two fluids; an inner fluid, flowing at lower temperature inside the tubes, and an outer fluid, flowing within the shell, but outside the tubes. The solid walls of the tubes ensure that the two fluids do not mix while the baffles help directing the flow on the shell side.

The case setup is based on a case prepared by SimScale GmbH that is publicly available at [73]. The computational mesh is shown in Fig. 11 and the boundary conditions are summarized in Table 7. The coupled boundary conditions are as in the previous case in Table 3. The material properties of the solid and fluid regions are shown in Table 8 where the inner and the outer fluids both have the same properties.

The simulations are run for 500 s using a time step size of $\Delta t = 1$ s. The numerical schemes used are listed in Table 9. Fig. 12 shows the results of the heat exchanger simulation using monolithic coupling. It depicts the distri-

Table 7 Boundary conditions for the heat exchanger

Boundary	Thermal	Velocity
Inner fluid		
inlet	283 K	$(0.002\ 0\ 0)^\top$ m/s
inner_to_solid	coupled	$(0\ 0\ 0)^\top$ m/s
outlet, walls	zeroGradient	zeroGradient
Outer fluid		
inlet	353 K	$(0\ 0.0037\ 0)^\top$ m/s
outer_to_solid	coupled	$(0\ 0\ 0)^\top$ m/s
outlet, walls	zeroGradient	zeroGradient
Solid		
solid_to_inner	coupled	–
solid_to_outer	coupled	–
walls	zeroGradient	–

Table 8 Thermophysical properties of the fluid and solid for the heat exchanger case

Property	Symbol	Unit	Fluid	Solid
Density	ρ	kg/m^3	1027	8960
Thermal conductivity	k	W/m · K	0.668	401
Dynamic viscosity	μ	kg/ms	3.645e^{-4}	–
Specific heat capacity	c_p	J/kg · K	4195	385

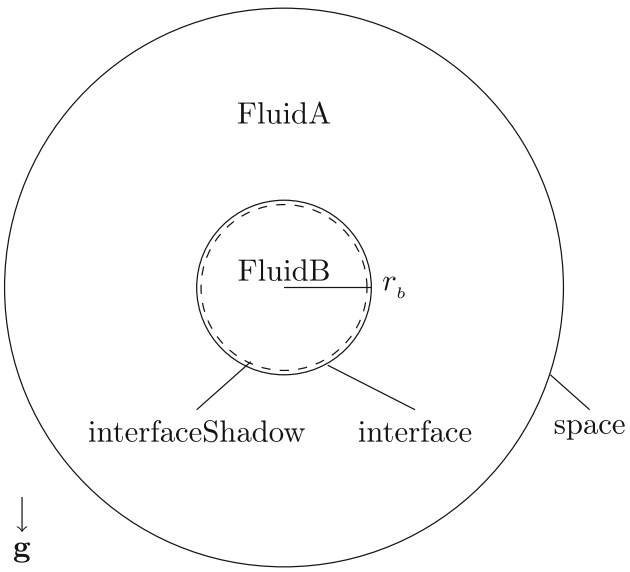


Fig. 13 Sketch of the computational domain for the rising bubble simulation

bution of the temperature at t = 30 s where no significant change in the temperature is observed afterwards. The computed field values using the partitioned approach are not remarkably different, but it takes longer to reach a steady state (around t = 500 s).

7.3 Air bubble rising in water

This case demonstrates the usage of `multiRegionFoam` to implement a moving mesh ALE interface tracking method, originally developed by Hirt et al. [74] and originally implemented into OpenFOAM by Tuković and Jasak [75]. A single air bubble rising in still pure water is considered, following the setup of Duineveld [76] and using his experimental data for validation. The bubble assumes an initial spherical shape with radius r_b and accelerates from zero velocity at release to its terminal rise velocity. As shown in Fig. 13, the computational domain has two regions comprising the bubble and the outer medium which is represented by a sphere of radius $20r_b$. The meshes consist of polyhedral cells for the bubble and prismatic cells with a polyhedral base for the water, as shown in Fig. 14. The bubble mesh is bounded by the "interfaceShadow" patch which coincides with the "interface" patch from the liquid side forming the bubble-liquid interface where the coupled boundary conditions are imposed according to Table 10.

The study considered bubbles with equivalent initial radii of $r_b = 0.5, 0.6, 0.7, 0.8,$ and 0.9 mm. The physical properties of the bubble and the surrounding liquid are reported in Table 11. The simulations run with a time step size of $\Delta t = 10^{-5}$ s until the terminal rise velocities are observed. The numerical schemes used are listed in Table 12. The results are compared with the experimental data obtained by Duineveld [76], as well as the simulation results from

Table 9 Numerical schemes for the heat exchanger

	Scheme	Setting
Time scheme	ddtScheme	steadyState
Finite volume schemes	gradScheme	Gauss linear
	gradScheme grad(U)	cellLimited Gauss linear 1
	divScheme div(phi,U)	Gauss upwind
	divScheme div(phi,T)	Gauss upwind
	laplacianScheme	Gauss linear corrected
	interpolationScheme	linear
	snGradScheme	corrected

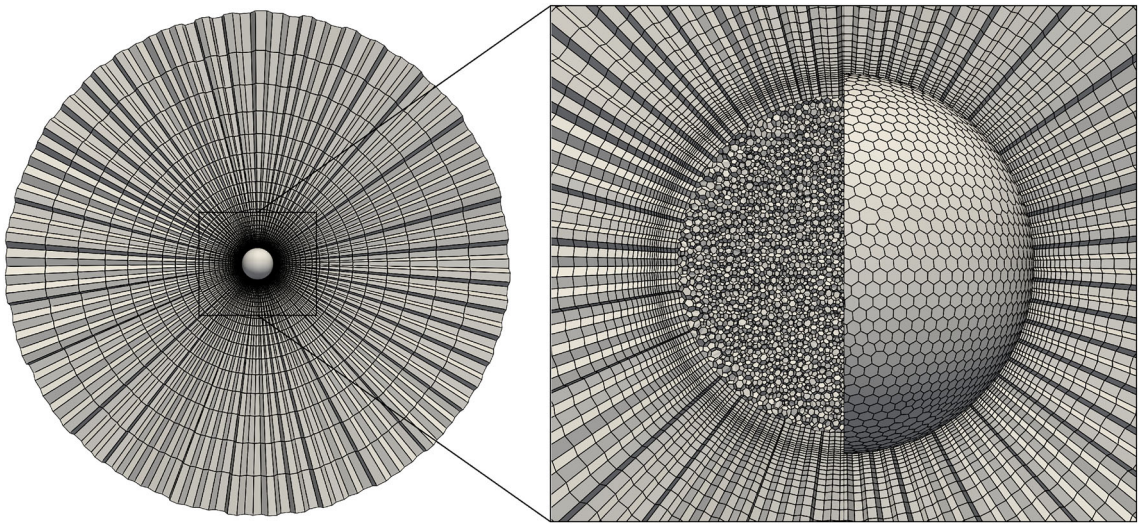


Fig. 14 Mesh for the rising bubble simulation

Table 10 Boundary conditions for the rising bubble simulation

Boundary	Pressure	Velocity
FluidA		
interface	regionCoupledPressureValue	regionCoupledVelocityFlux
space	zeroGradient	inletOutlet
FluidB		
interfaceShadow	regionCoupledPressureFlux	regionCoupledVelocityValue

Tuković and Jasak [75]. Fig. 15 depicts the rise velocity of the bubble with radius $r_b = 0.5$ mm over time until the expected terminal rise velocity value is attained. Fig. 16 displays the terminal rise velocity for a set of larger bubbles. The high level of agreement between the simulation results and the available data from the literature indicates that the present work has successfully addressed any potential robustness issues related to significant mesh deformation. Fig. 17 depicts the terminal state of the rising bubble with $r_b = 0.9$ mm.

Table 11 Physical properties for the rising bubble simulation

Property	FluidB (air)	FluidA (water)
Density	1.205 kg/m^3	998.3 kg/m^3
Dynamic viscosity	$1.82 \cdot 10^{-5}$ kg/ms	10^{-3} kg/ms
Surface tension coefficient	0.0727 N/m	

Table 12 Numerical schemes for the rising bubble simulation

	Scheme	Setting
Time scheme	ddtScheme	backward
Finite volume schemes	gradScheme	Gauss linear
	divScheme div(phi,U)	Gauss GammaVDC 0.5
	divScheme div(phi,T)	Gauss linearUpwind Gauss linear
	laplacianScheme	Gauss linear corrected
	interpolationScheme	linear
	snGradScheme	corrected
Finite area schemes	gradScheme	Gauss linear
	divScheme	Gauss linear
	interpolationScheme	linear

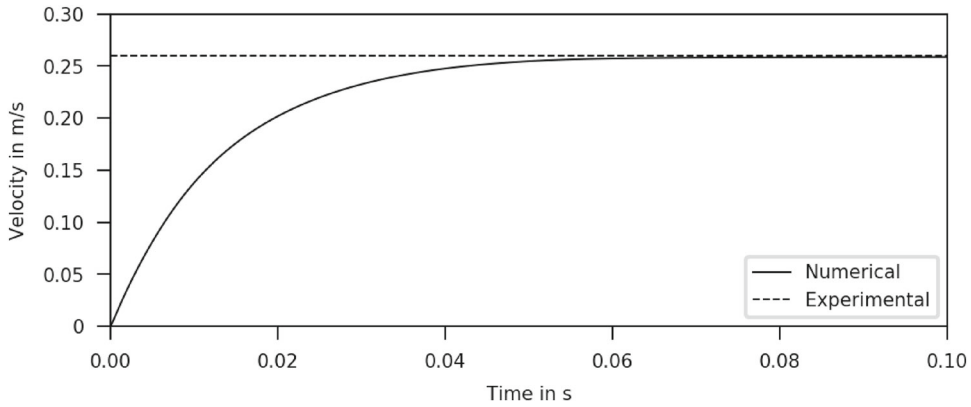


Fig. 15 The rise velocity over time for a bubble with radius $r_b = 0.5$ mm compared with the experiment results

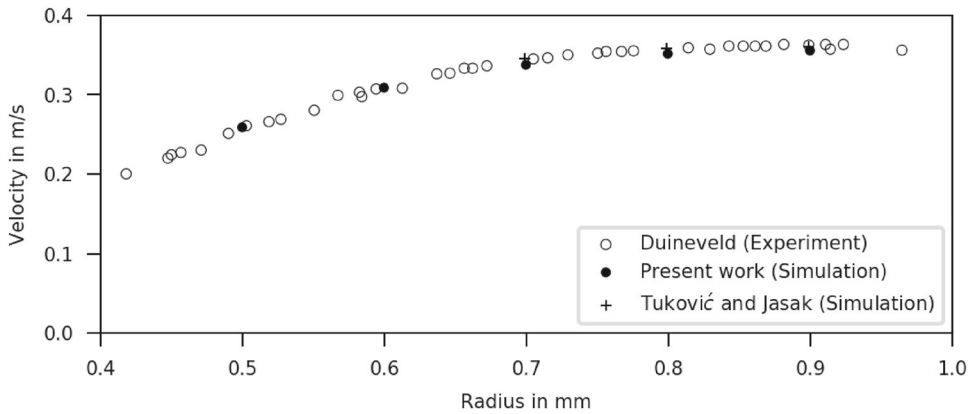


Fig. 16 Comparison of the terminal rise velocity for varying equivalent bubble radii

7.4 Channel flow around an elastic object

The moving mesh ALE interface tracking method for multiphase flow is modified to extend the applicability for fluid–structure interaction (FSI). A frequently examined validation case involves laminar incompressible flow around a cylinder with an elastic plate, as detailed by Hron and Turek [77]. The configuration, illustrated in Fig. 18, comprises a horizontal channel containing a cylinder with an elastic plate attached to its right side.

From the left side of the channel, the fluid flows with a parabolic velocity profile with a maximum inflow velocity of 1.5 $\bar{U}$, where $\bar{U}$ is the mean inflow velocity. The tests typically encompass three cases: FSI1, FSI2, and FSI3, each varying in the inflow speed, resulting in two outcomes—steady for FSI1 and unsteady periodic solutions for FSI2 and FSI3. The latter is considered in this discussion. The mean inflow velocity along with other fluid and solid properties are listed in Table 13. The boundary conditions are summarized in Table 14.

Three cases of mesh refinement levels are considered. The one depicted in Fig. 18 is refered to as the coarse mesh. It has a total of 630 cells for the solid structure against 5336 cells for the fluid. Two more variations are consid-

Table 13 Fluid and solid properties for FSI3 case

Property	Symbol	Unit	Solid	Fluid
Kinematic viscosity	ν^f	m^2/s	–	10^{-3}
Mean inflow velocity	$\bar{U}$	m/s	–	2
Density	ρ	kg/m^3	10^3	10^3
Young's modulus	E^s	kg/ms^2	$5.6 \cdot 10^6$	–
Poisson ratio	ν^s	–	0.4	–

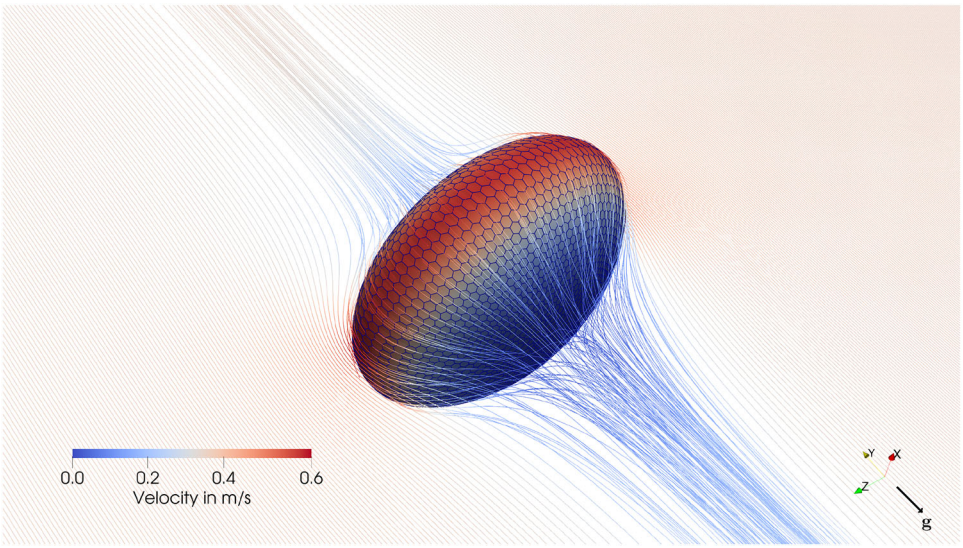
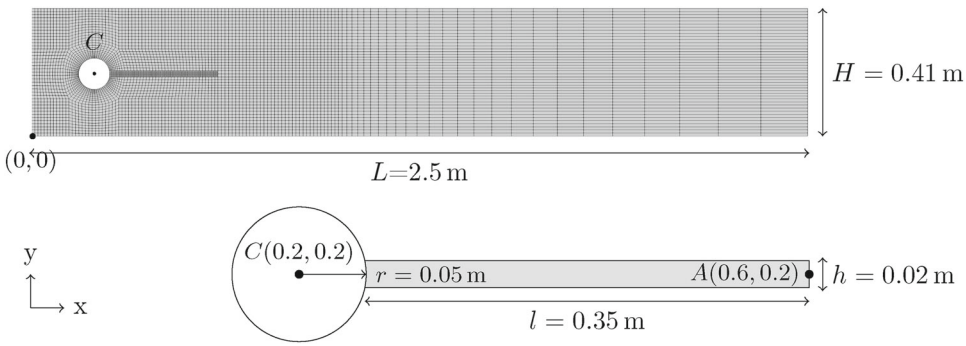


Fig. 17 Velocity field visualisation for the rising bubble with $r_b = 0.9$



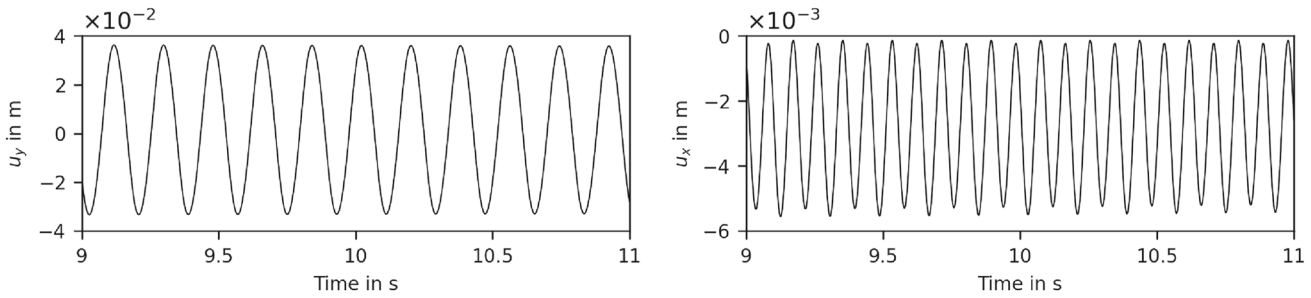


Fig. 19 x and y displacement of point A obtained from the finest mesh

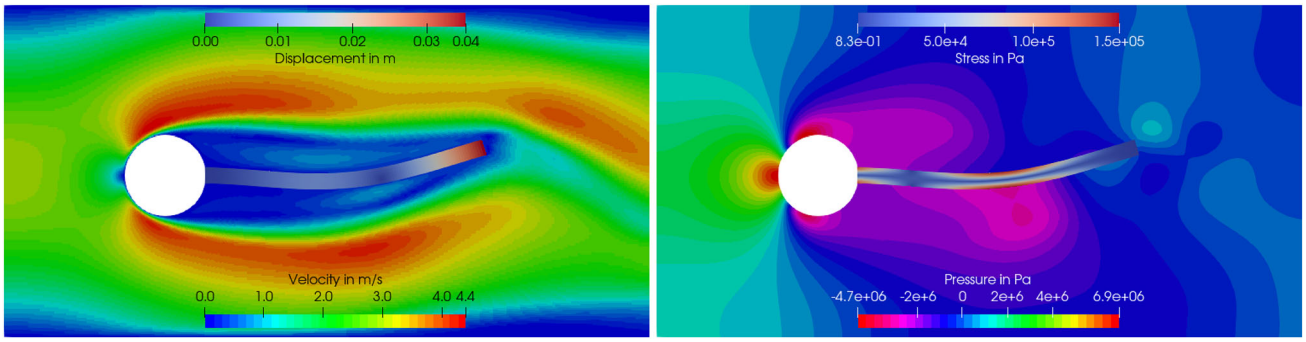


Fig. 20 The velocity and the pressure fields in the fluid along with the displacement and the stress fields in the structure domain

Table 14 Boundary conditions for the FSI3 case

Boundary	Velocity	Kinematic Pressure	Traction
Fluid			
interfaceShadow	movingWallVelocity	zeroGradient	–
inlet	parabolicVelocity	zeroGradient	–
outlet	zeroGradient	$0\,\mathrm{m}^2/\mathrm{s}^2$	–
cylinder, bottom, top	$(0\ 0\ 0)^\top$ m/s	zeroGradient	–
front & back planes	empty	empty	–
Solid			
Interface	–	–	regionCoupledTraction

Table 15 Comparison of the Displacements of point A in the form `mean ± amplitude [frequency]`

Case	u_y	u_x
Coarse	1.67 ± 28.55[5.9]	-1.99 ± 1.84[11.5]
Medium	1.53 ± 33.41[5.3]	-2.66 ± 2.45[10.9]
Fine	1.47 ± 34.37[5.3]	-2.67 ± 2.53[10.9]
Benchmark	1.48 ± 34.35[5.3]	-2.69 ± 2.53[10.9]

ered which affect the fluid domain only, namely medium and fine meshes with 21,344, and 85,376 cells, respectively. The coarse mesh is generated using the `blockMesh` utility [78] in OpenFOAM. The medium and fine meshes are obtained using the `refineMesh` utility [79] and employing the `extrudeMesh` utility [78] to preserve the two-dimensional mesh. Different time steps, i.e., $\Delta t = 1 \cdot 10^{-3}$ s, $7.5 \cdot 10^{-4}$ s, $5 \cdot 10^{-4}$ s are used for the different levels of refinement, coarse, medium, and fine, respectively. The simulations are

Table 16 Governing equations for the single-phase PEM fuel cell

Equation type	**Fluid regions**
Continuity	$\nabla \cdot (\rho \mathbf{U}) = R$
Momentum	$\nabla \cdot (\rho \mathbf{U}\mathbf{U}) = -\nabla p + \nabla \cdot (\mu \nabla \mathbf{U}) + \rho \mathbf{g} + S_{\mathrm{Darcy}}$
Species	$\nabla \cdot (\rho \mathbf{U} Y_{\mathrm{i}}) = \nabla \cdot (\rho D_{\mathrm{i}}^{\mathrm{eff}} \nabla Y_{\mathrm{i}}) + R_{\mathrm{i}}$
Nernst potential	$E_{\mathrm{Nernst,ae}} = \sum_{\mathrm{i}} \frac{-(H_{\mathrm{i}} - T\,S_{\mathrm{i}})\omega - R_g T \omega \ln p_{\mathrm{i}}}{zF}$
Overpotential	$j_{\mathrm{ae}} = i_{\mathrm{ae},0}\Big[\prod_{\mathrm{reac,i}} \left(\frac{C_{\mathrm{i}}}{C_{\mathrm{ref}}}\right)^{\xi} \exp\left(\frac{-\alpha_{\mathrm{ae}} z F \eta_{\mathrm{ae}}}{R_g T}\right) - \prod_{\mathrm{prod,i}} \left(\frac{C_{\mathrm{i}}}{C_{\mathrm{ref}}}\right)^{\xi} \exp\left(\frac{(1-\alpha_{\mathrm{ae}}) z F \eta_{\mathrm{ae}}}{R_g T}\right) \Big]$
	Electric/ionic regions
Electrical potential	$\nabla \cdot (\sigma_{\mathrm{E}} \nabla \Phi_{\mathrm{E}}) = J_{\mathrm{E}}$
Ionic potential	$\nabla \cdot (\sigma_{\mathrm{I}} \nabla \Phi_{\mathrm{I}}) = J_{\mathrm{I}}$
Dissolved water	$\nabla\left(\frac{\mathbf{i}}{F} n_d\right) = \frac{\rho_{\mathrm{m}}}{\mathrm{EW}} \nabla \cdot \left(D_{\mathrm{m}}^{\mathrm{eff}} \nabla \lambda\right) + R_\lambda$
	Global region
Energy equation	$\rho c_p \mathbf{U} \cdot \nabla T = \nabla \cdot \left(k^{eff} \nabla T\right) + Q$

For details refer to [82]

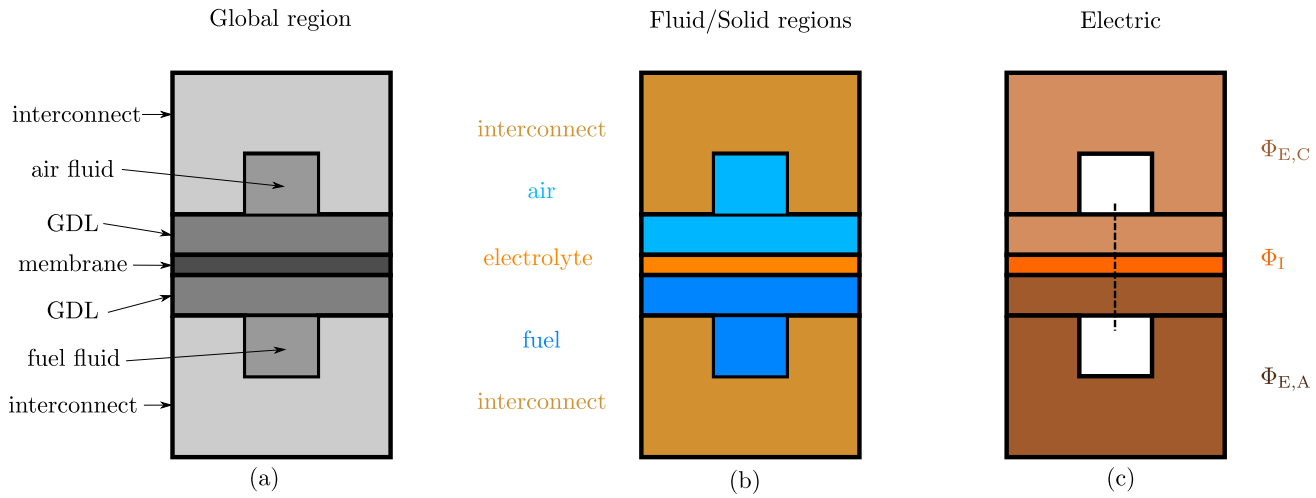


Fig. 21 General modeling approach of the PEM fuel cell

carried out until a periodic solution is reached. From the last period of oscillations, the mean value is calculated as the average of the minimum and maximum values, while the amplitude is their difference from the mean. The frequency is computed using fast Fourier transforms. Fig. 19 shows the x and y displacement of point A, denoted u_x and u_y, while Fig. 20 captures the simulation at the instant in time when point A reaches the maximum position. For quantitative comparison, the displacements are given in Table 15 in the same format of the benchmark results, i.e., `mean ± amplitude [frequency]`. The mesh sensitivity analysis showed that as the resolution of the fluid mesh increases, the computed displacements remarkably converge towards and get very close to the benchmark values.

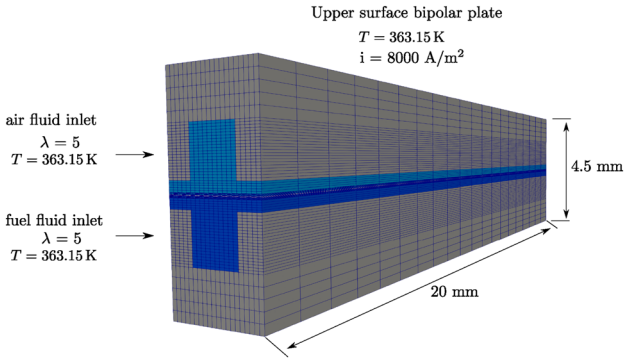


Fig. 22 Mesh and simulation domain

7.5 Polymer electrolyte fuel cell

The possibility of simulating complex multi-physical systems within the `multiRegionFoam` framework will be demonstrated in the following example, namely a single polymer electrolyte fuel cell (PEMFC) channel assembly. A fuel cell is a flow-through device that converts chemical energy into electricity and heat. The general structure and the physical components of a typical PEMFC are depicted in Fig. 21a.

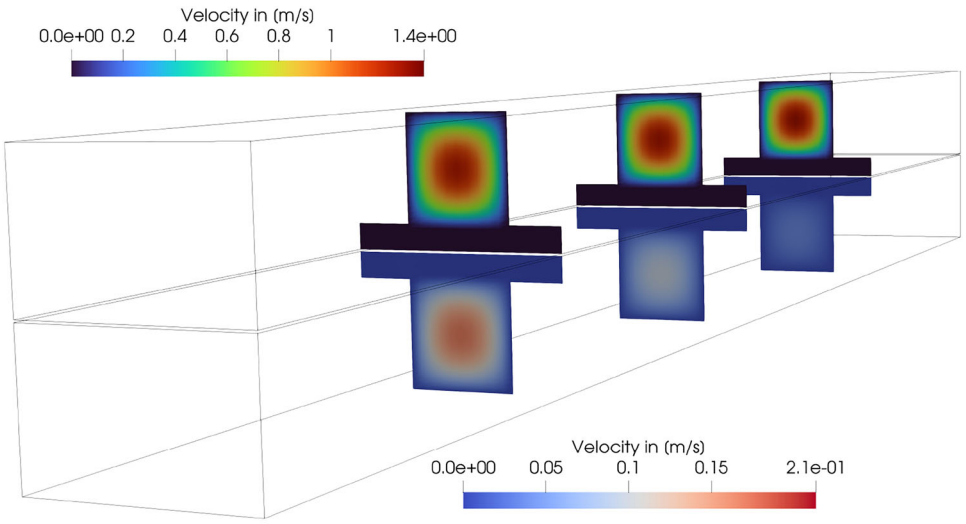
Beginning with the outer components, the cell consists of two end plates, or interconnects, to which the external load is connected. The reactants/products are conveyed through the air and fuel channels and transported via porous gas diffusion layers (GDLs) to the catalyst layers (CLs), which are represented here as infinitely thin surfaces. Besides effecting gas transport to the catalyst layers, the electrically conducting GDLs also facilitate electrical contact between the membrane electrode assembly and the end plates (`interconnects`). The CLs are located between the ionically conducting membrane and the GDLs. On the cathode (CL at the air side) oxygen is reduced and water is produced, whereas on the anode side (CL at the fuel side) hydrogen is oxidized and protons are transported through the ionic conductive membrane to the cathode. Fig. 21b shows the general structure and decomposition of the modeling and computational domains. The design of the single-phase PEMFC implemented here in `multiRegionFoam` is based on developments described in [80–83] which have been further extended to incorporate interface coupling. Since the present study of a PEM fuel cell serves primarily as a showcase for the capabilities of `multiRegionFoam`, a detailed explanation of the underlying physics is not provided here; Only the fundamentals and the differences from the previous work (to which the reader is referred) associated with surface coupling of the electric potential field will be expanded in detail below.

Table 16 lists the governing equations for each region, depicted in Fig. 21, in a compact form. For a comprehensive overview on how the equations are used and solved, the interested reader is referred to the original publication of the `openFuelCell2` software suit [82]. The novice reader can find an elementary introduction to modeling a

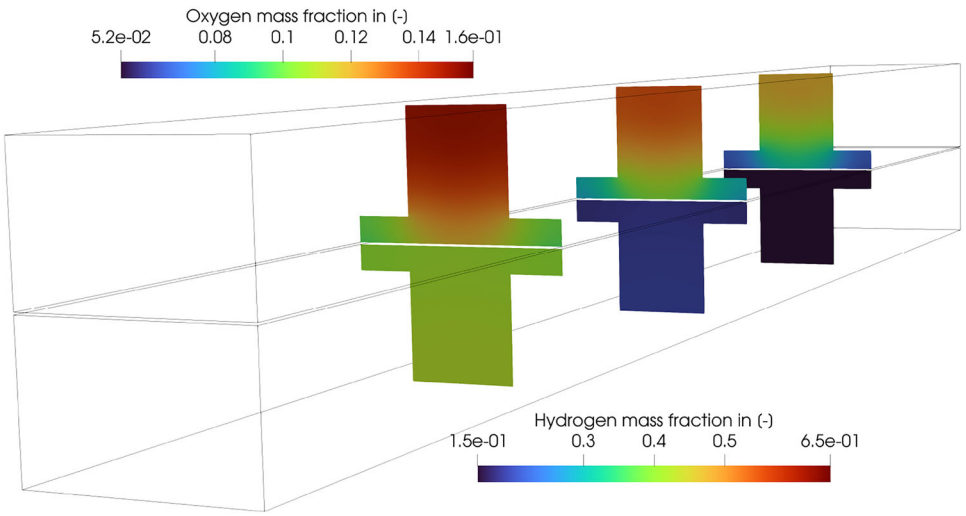
Table 17 Boundary conditions for the PEM fuel cell

Boundary	Thermal	Velocity	Species mass fraction	Potential
Air				
airInlet	–	$(0.655\,0\,0)^{\top}$ m/s	$Y_{O2} = 0.156, Y_{H2O} = 0.35$	–
airOutlet	–	zeroGrad.	zeroGrad.	–
airToInterconnect	–	noSlip	zeroGrad.	–
airToElectrolyte	–	noSlip	zeroGrad.	–
airSides	–	noSlip	zeroGrad.	–
Fuel				
fuelInlet	–	$(0.161\,0\,0)^{\top}$ m/s	$Y_{H2} = 0.65$, $Y_{H2O} = 0.35$	–
fuelOutlet	–	zeroGrad.	zeroGrad.	–
fuelToInterconnect	–	noSlip	zeroGrad.	–
fuelToElectrolyte	–	noSlip	zeroGrad.	–
fuelSides	–	noSlip	zeroGrad.	–
$\boldsymbol{\Phi}_{I}$				
electrolyteSides	–	–	–	zeroGrad.
electrolyteToAir	–	–	–	coupled flux
electrolyteToFuel	–	-	–	coupled jump
$\boldsymbol{\Phi}_{E,A}$				
anodeSides	–	–	–	zeroGrad.
interconnectDown	–	–	–	0 V
$\Phi_{E,A}$ToElectrolyte	–	–	–	coupled flux
$\boldsymbol{\Phi}_{E,C}$				
cathodeSides	–	–	–	zeroGrad.
interconnectUp	–	–	–	adjustedPotential
$^{\boldsymbol{\Phi}}{}_{E,C}$ToElectrolyte	–	–	–	coupled jump
Global region				
airInlet	363.15 K	–	–	–
airOutlet	zeroGrad	–	–	–
fuelInlet	363.15 K	–	–	–
fuelOutlet	zeroGrad.	–	–	–
interconnectSides	zeroGrad.	–	–	–
interconnectUp	363.15 K	–	–	–
interconnectDown	363.15 K	–	–	–

simple electrode with OpenFOAM in [84]. Following the methodology of `openFuelCell2`, the subdomains defined as different region types are solid, fluid and electric/ionic region. In the fluid (`air` and `fuel`) regions, both the modified Navier-Stokes equations, which take into account the porous GDLs via Darcy's law, and the species mass fraction equations are solved. Oxygen is consumed and water vapor is produced at the cathode whereas hydrogen is consumed at the anode in accordance with Faraday's law of electrolysis. These are accounted-for by corresponding source terms on the right-hand side of the species and continuity equations, see Table 16. The gas mixtures are assumed to be ideal gases. Diffusion is characterized by Fick's law, taking into account the presence of the porous layers; for further details the reader is referred to [85]. On the cathode side, the inflowing gas mixture is composed of oxygen, nitrogen and water vapor, whereas on the anode side, the influent is made up of hydrogen and water vapor. The constraint that the sum of all mass fractions equals unity must be fulfilled at all times. Humidification of the gas mixtures is necessary to ensure that the ionic conductivity of the membrane, which is strongly dependent on water content, is sufficient. At the same time, flooding must be avoided. Within the electric/ionic conducting regions, Fig. 21c, the equations for the electronic potential at the cathode and anode, ($\Phi_{E,C}$, $\Phi_{E,A}$) and the ionic potential (Φ_{I}) are solved using the partitioned coupling approach. The open cell voltage is described by a Nernst equation and the activation overpotential, i.e., the change in voltage result-



(a) Velocity distribution



(b) Species mass fractions

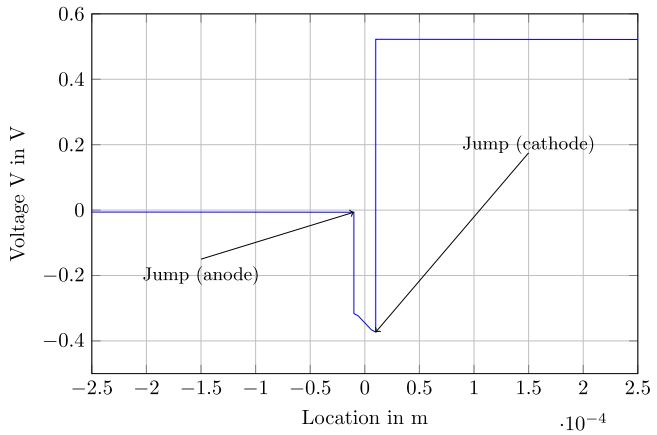
Fig. 23 Velocity and species mass fraction distribution along the channels for $i = 8000\ \mathrm{A/m^2}$

ing from the finite-rate reaction, is expressed in terms of a Butler-Volmer equation, Table 16, [86, 87].

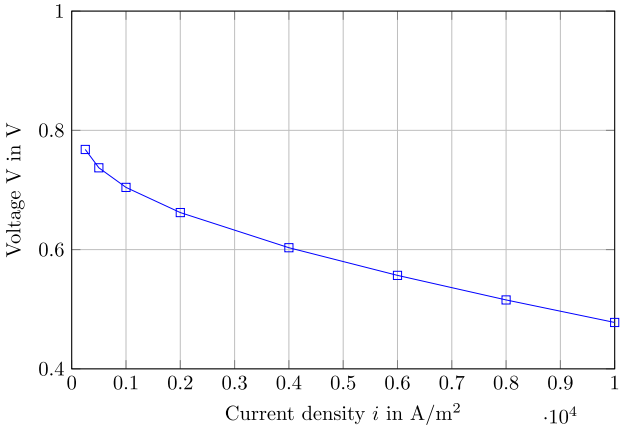
The cell operates in galvanostatic (as opposed to potentiostatic) mode, i.e., a constant current is applied. Therefore, the potential field specified at the upper surface of $\Phi_{\mathrm{E,C}}$ is adjusted at every iteration until the prescribed mean current density is reached. This is indicated by the entry `adjustedPotential` for the boundary interconnectUp in Table 17. The calculation of the temperature within each component of the PEM fuel cell takes place on the global region, Fig. 21a, by mapping the required quantities such as densities and specific heat capacities etc. onto this mesh. A further adaptation of the source code, whereby the energy equation on all the individual regions are solved using the coupled boundary conditions as shown in the previous sections, is also possible in the framework here. Fig. 22 shows the dimensions of the single channel and selected boundary conditions taken from Table 17.

The cell under consideration consists of a single channel with a length of 20 mm, where the fluid channels have an area of 1×1 mm. The simulation itself is steady state and the velocity used for the boundary condition at the fluid inlets is given by a constant stoichiometry of $\lambda = 5$ calculated via Faraday's law.

The temperature of the incoming gases is set to $T = 363.15$ K and a mean current density of $i = 8000\ \mathrm{A/m^2}$ is applied. Fig. 23 shows (a) the corresponding velocity and (b) species mass fraction contours at different positions along



(a) Potential distribution in the middle of the cell from the top of the cathode to the buttom of the anode GDL (dotted line in Figure 21 (c)) for $i = 6000$ A/m^2



(b) Polarization curve

Fig. 24 **a** Potential distribution along the middle of the cell perpendicular to the flow, and **b** polarization curve

the channel inside the anodic and cathodic fluid regions. At the air side (cathode) the velocity increases due to the reaction and decreased density of the gas mixture and added water. For the fuel side (anode), the velocity is decreasing, mainly due to the removal of water and hydrogen/protons. This is consistent with the displayed mass fraction contours, Fig. 23(b), where the oxygen and hydrogen mass fractions both decrease in the main flow direction.

The main focus of the present example is the surface coupling of the electrical field potential. The electrochemical reactions, characterized by the Nernst (ideal) potential and the losses/overpotentials given in Table 16, are presumed to take place on infinitesimally thin CL interfaces, where the coupled jump/flux boundary conditions for the potential are prescribed (see Table 17). Fig. 24a shows the potential variation in the middle of the cell traversing from the upper surface of the GDL at the cathode to the lower surface of the GDL at the anode. Due to the high electric conductivity of the porous media, the potential drop in the porous regions is negligibly small. At the two CLs, however, the proximity of two dissimilar materials results in overall potential jumps at open ciruit. The overpotentials arising from the reactions (flowing current) induce a change in these jumps. A gradient in the potential can also be noticed in the membrane region. This is because of ohmic losses associated with the low ionic conductivity of the membrane. Fig. 24b shows the characteristic 'polarization curve' (voltage versus current density) obtained by conducting a series of calculations for different current densities at constant λ. The polarization curve of a PEM fuel cell typically exhibits a characteristic form featuring a non-linear region at low current densities, which is primarily influenced by the activation overpotential, and a subsequent linear ohmic region, predominantly governed by the internal resistances of the fuel cell.

8 Summary and outlook

The present contribution sets out `multiRegionFoam`, a unified framework for multiphysics problems of multi-region coupling type within OpenFOAM (FOAM-extend). `multiRegionFoam` offers a flexible design that allows for the assembly of multiphysics problems region-by-region and the specification of coupling conditions interface-by-interface. For this, the framework enables to formulate region-specific physics in form of sets of partial differential equations in a modular fashion and incorporates mathematical jump/transmission conditions in their most general form—accommodating tensors of any rank. This advancement allows for a unified treatment of coupled transport processes across regions. Moreover, users have the freedom to choose between monolithic and partitioned coupling for each coupled transport equation separately. To address fluid flow problems, the code implements various pressure–velocity algorithms, including SIMPLE, PISO, and PIMPLE, incorporating loops of predictor and corrector steps across regions. The framework's maturity and versatility is demonstrated through its successful deployment in various multi-region coupling cases, including conjugate heat transfer, multiphase flows, fluid–structure interaction, and fuel cells.

The authors anticipate that the creation and release of `multiRegionFoam` will empower numerous domain experts and developers to derive significant advantages from this framework. Specifically, they aspire for this contribution to facilitate thorough investigations in diverse domains of

multiphysics problems and across various application areas. This, in turn, is expected to foster synergistic collaborations among previously separate disciplines, enabling mutual benefits to be realized.

Acknowledgements Computations for this work were partly conducted on the Lichtenberg II high performance computer of the Technical University of Darmstadt.

Author contributions **Heba Alkafri:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data Curation, Writing—Original Draft, Writing—Review and Editing, Visualization. **Constantin Habes:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data Curation, Writing—Original Draft, Writing—Review and Editing, Visualization. **Mohammed Elwardi Fadeli:** Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing—Original Draft, Writing—Review and Editing. **Steffen Hess:** Software, Writing—Original Draft, Writing—Review and Editing. **Steven B. Beale:** Writing—Review and Editing. **Shidong Zhang:** Software, Writing—Review and Editing. **Hrvoje Jasak:** Software, Methodology. **Holger Marschall:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing—Original Draft, Writing—Review and Editing, Visualization, Supervision, Project administration, Funding acquisition.

Funding Open Access funding enabled and organized by Projekt DEAL. The authors H.A. M.E.F. and H.M. are grateful for the funding by the Hessian Ministry of Higher Education, Research, Science and the Arts, and the National High Performance Computing Center for Computational Engineering Science (NHR4CES). S.H. is funded by the AI Data Analytics and Scalable Simulations (AIDAS) project, the financial support of which is highly appreciated.

Data availability Data will be made available on request.

Code availability multiRegionFoam v1.1 [1], repository: https://bitbucket.org/hmarschall/multiregionfoam/

Declarations

Conflict of interest The authors have no Conflict of interest to declare that are relevant to the content of this article.

References

1. multiRegionFoam. https://doi.org/10.5281/zenodo.8025525
2. Weller HG, Tabor G, Jasak H, Fureby C (1998) A tensorial approach to computational continuum mechanics using object-oriented techniques. Comput Phys 12(6):620–631. https://doi.org/10.1063/1.168744
3. Kataoka S, Minami S, Kawai H, Yamada T, Yoshimura S (2014) A parallel iterative partitioned coupling analysis system for large-scale acoustic fluid-structure interactions. Comput Mech 53(6):1299–1310. https://doi.org/10.1007/s00466-013-0973-1. (**Publisher: Springer Science and Business Media LLC**)
4. MpCCI—interfaces for co-simulation and mapping. https://www.mpcci.de/en/mpcci-software.html. Accessed 02 Dec 2022
5. Joppich W, Kürschner M (2005) MpCCI—a tool for the simulation of coupled applications. Concurr Comput Pract Exp 18(2):183–192. https://doi.org/10.1002/cpe.913. (**Publisher: Wiley**)
6. Buis S, Piacentini A, Déclat D (2005) PALM: a computational framework for assembling high-performance computing applications. Concurr Comput Pract Exp 18(2):231–245. https://doi.org/10.1002/cpe.914. (**Publisher: Wiley**)
7. Duchaine F, Jauré S, Poitou D, Quémerais E, Staffelbach G, Morel T, Gicquel L (2015) Analysis of high performance conjugate heat transfer with the OpenPALM coupler. Comput Sci Discov 8(1):015003. https://doi.org/10.1088/1749-4699/8/1/015003. (**Publisher: IOP Publishing**)
8. Valcke S (2013) The OASIS3 coupler: a European climate modelling community software. Geosci Model Dev 6(2):373–388. https://doi.org/10.5194/gmd-6-373-2013. (**Publisher: Copernicus GmbH**)
9. Craig A, Valcke S, Coquart L (2017) Development and performance of a new version of the OASIS coupler, OASIS3-MCT_3.0. Geosci Model Dev 10(9):3297–3308. https://doi.org/10.5194/gmd-10-3297-2017. (**Publisher: Copernicus GmbH**)
10. Pawlowski R (2014) Physics integration KErnels (PIKE), Version 00. https://www.osti.gov/biblio/1253615
11. The Trilinos Project Website. https://trilinos.github.io/ Accessed 19 June 2023
12. Bungartz H-J, Lindner F, Gatzhammer B, Mehl M, Scheufele K, Shukaev A, Uekermann B (2016) preCICE—a fully parallel library for multi-physics surface coupling. Comput Fluids 141:250–258. https://doi.org/10.1016/j.compfluid.2016.04.003. (**Publisher: Elsevier BV**)
13. preCICE coupling library for partitioned multi-physics simulations. https://precice.org/index.html. Accessed 02 Dec 2022
14. FEniCS project. https://fenicsproject.org/. Accessed 02 Dec 2022
15. FEniCS FEA Solver. https://www.featool.com/doc/fenics. Accessed 02 Dec 2022
16. Budisa A, Hu X, Kuchta M, Mardal K-A, Zikatanov L (2022) HAZniCS—software components for multiphysics problems. https://doi.org/10.48550/ARXIV.2210.13274
17. Casoni E, Jérusalem A, Samaniego C, Eguzkitza B, Lafortune P, Tjahjanto D, Sáez X, Houzeaux G, Vázquez M (2015) Alya: computational solid mechanics for supercomputers. Arch Comput Methods Eng 22(4):557–576. https://doi.org/10.1007/s11831-014-9126-8. (**Publisher: Springer**)
18. Alya—high performance computational mechanics. https://www.bsc.es/research-development/research-areas/engineering-simulations/alya-high-performance-computational. Accessed 02 Dec 2022
19. Cajas JC, Houzeaux G, Vázquez M, García M, Casoni E, Calmet H, Artigues A, Borrell R, Lehmkuhl O, Pastrana D et al (2018) Fluid–structure interaction based on HPC multicode coupling. SIAM J Sci Comput 40(6):677–703. https://doi.org/10.1137/17M1138868. (**Publisher: SIAM**)
20. code_saturne. https://www.code-saturne.org/cms/web/. Accessed 02 Dec 2022

21. PLE (Parallel Location and Exchange) documentation. https://www.code-saturne.org/documentation/ple-2.0/html/index.html. Accessed 02 Dec 2022
22. SYRTHES. https://www.edf.fr/en/the-edf-group/inventing-the-future-of-energy/r-d-global-expertise/our-offers/simulation-softwares/syrthes?logiciel=10818. Accessed 02 Dec 2022
23. Zavala-Aké M, Mira D, Vázquez M, Houzeaux G (2017) A partitioned methodology for conjugate heat transfer on dynamic structures. In: Di Napoli E, Hermanns M-A, Iliev H, Lintermann A, Peyser A (eds) High-Perform Sci Comput, vol 10164. Springer, Cham, pp 37–47. https://doi.org/10.1007/978-3-319-53862-4_4
24. Houbar S, Gerschenfeld A, Allaire G (2021) Simulation of the fluid–structure interaction involving two-phase flow and hexagonal structures in a nuclear reactor core. In: 14th WCCM-ECCOMAS congress 2020, vol 1500. https://doi.org/10.23967/wccm-eccomas.2020.232
25. deal.II—an open source finite element library. https://www.dealii.org/. Accessed 02 Dec 2022
26. Wick T (2013) Solving monolithic fluid–structure interaction problems in arbitrary Lagrangian Eulerian coordinates with the deal.II library. Arch Numer Softw 1(1):1–19. https://doi.org/10.11588/ans.2013.1.10305
27. deal.II publications. https://www.dealii.org/publications.html. Accessed 02 Dec 2022
28. MOOSE multiphysics object-oriented simulation environment. https://mooseframework.inl.gov/index.html. Accessed 02 Dec 2022
29. Permann CJ, Gaston DR, Andrš D, Carlsen RW, Kong F, Lindsay AD, Miller JM, Peterson JW, Slaughter AE, Stogner RH et al (2020) MOOSE: enabling massively parallel multiphysics simulation. SoftwareX 11:100430. https://doi.org/10.1016/j.softx.2020.100430. (**Publisher: Elsevier**)
30. Dhulipala SL, Bolisetti C, Munday LB, Hoffman WM, Yu C-C, Mir FUH, Kong F, Lindsay AD, Whittaker AS (2022) Development, verification, and validation of comprehensive acoustic fluid–structure interaction capabilities in an open-source computational platform. Earthq Eng Struct Dyn. https://doi.org/10.1002/eqe.3659. (**Publisher: Wiley Online Library**)
31. Cardiff P, Karač A, De Jaeger P, Jasak H, Nagy J, Ivanković A, Tuković Z (2018) An open-source finite volume toolbox for solid mechanics and fluid–solid interaction simulations. arXiv preprint . https://doi.org/10.48550/arXiv.1808.10736
32. Beale SB, Choi HW, Pharoah JG, Roth HK, Jasak H, Jeon DH (2016) Open-source computational model of a solid oxide fuel cell. Comput Phys Commun 200:15–26. https://doi.org/10.1016/j.cpc.2015.10.007
33. Renze P, Akermann K (2019) Simulation of conjugate heat transfer in thermal processes with open source CFD. ChemEngineering 3(2):59. https://doi.org/10.3390/chemengineering3020059. (**Publisher: MDPI AG**)
34. Micale D, Ferroni C, Uglietti R, Bracconi M, Maestri M (2022) Computational fluid dynamics of reacting flows at surfaces: methodologies and applications. Chem Ing Tech 94(5):634–651. https://doi.org/10.1002/cite.202100196
35. New modular solver framework for single- and multi-region simulations. https://github.com/OpenFOAM/OpenFOAM-dev/commit/968e60148ab31ec017f275673496d6193713d7e5. Accessed 24 May 2023
36. YALES2 public page. https://www.coria-cfd.fr/index.php/YALES2. Accessed 02 Dec 2022
37. Sarkar P, Ghigliotti G, Franc J-P, Fivel M (2021) Mechanism of material deformation during cavitation bubble collapse. J Fluids Struct 105:103327. https://doi.org/10.1016/j.jfluidstructs.2021.103327. (**Publisher: Elsevier**)
38. Boulet L, Bénard P, Lartigue G, Moureau V, Didorally S, Chauvet N, Duchaine F (2018) Modeling of conjugate heat transfer in a kerosene/air spray flame used for aeronautical fire resistance tests. Flow Turbul Combust 101(2):579–602. https://doi.org/10.1007/s10494-018-9965-8. (**Publisher: Springer**)
39. OpenPALM, a dynamic parallel code coupler. https://www.cerfacs.fr/globc/PALM_WEB/. Accessed 02 Dec 2022
40. COMSOL multiphysics. https://www.comsol.com/. Accessed 02 Dec 2022
41. FEATool multiphysics. https://www.featool.com/ Accessed 2022-12-02
42. Ansys fluent. https://www.ansys.com/products/fluids/ansys-fluent#tab1-2. Accessed 02 Dec 2022
43. Ansys LS-DYNA multiphysics solver. https://www.ansys.com/products/structures/ansys-ls-dyna. Accessed 02 Dec 2022
44. Gosman AD, Pun WM, Runchal AK, Spalding DB, Wolfshtein M (1969) Heat and mass transfer in recirculating flows. Academic Press, London
45. Bothe D (2022) Sharp-interface continuum thermodynamics of multicomponent fluid systems with interfacial mass. Int J Eng Sci 179:103731. https://doi.org/10.1016/j.ijengsci.2022.103731
46. Slattery JC, Sagis LM, Oh E-S (2007) Interfacial transport phenomena, 2nd edn. Springer, New York
47. Ishii M, Hibiki T (2011) Thermo-fluid dynamics of two-phase flow, 2nd edn. Springer, New York (**OCLC: ocn690084123**)
48. Kjelstrup S, Bedeaux D (2008) Non-equilibrium thermodynamics of heterogeneous systems, vol 16. Series on advances in statistical mechanics. WORLD SCIENTIFIC, Norway. https://doi.org/10.1142/6672
49. Casey J (2011) On the derivation of jump conditions in continuum mechanics. Int J Struct Changes Solids Mach Appl 3(2):61–84
50. Cermelli P, Fried E, Gurtin ME (2005) Transport relations for surface integrals arising in the formulation of balance laws for evolving fluid interfaces. J Fluid Mech 544(1):339. https://doi.org/10.1017/S0022112005006695
51. Habes C (2023) Towards a unified multiphysics framework applied to reactive bubbly flows. https://doi.org/10.26083/TUPRINTS-00023030
52. Uekermann B, Gatzhammer B, Mehl M (2014) Coupling algorithms for partitioned multi-physics simulations. In: Plödereder E, Grunske L, Schneider E, Ull D (eds) Informatik 2014. Gesellschaft für Informatik e.V., Bonn, pp 113–124
53. Toselli A, Widlund OB (2005) Domain decomposition methods—algorithms and theory, vol 34. Springer series in computational mathematics. Springer, Berlin
54. Smith BF, Bjørstad PE, Gropp W, Gropp WD (2004) Domain decomposition: parallel multilevel methods for elliptic partial differential equations, 1. paperback edn. Cambridge University Press, Cambridge
55. Gatzhammer B (2015) Efficient and flexible partitioned simulation of fluid-structure interactions, 1, aufl. Informatik. Verl. Dr. Hut, München
56. Irons BM, Tuck RC (1969) A version of the Aitken accelerator for computer iteration. Int J Numer Methods Eng 1(3):275–277. https://doi.org/10.1002/nme.1620010306
57. Degroote J, Haelterman R, Annerel S, Bruggeman P, Vierendeels J (2010) Performance of partitioned procedures in fluid–structure interaction. Comput Struct 88(7–8):446–457. https://doi.org/10.1016/j.compstruc.2009.12.006
58. run Time Selection mechanism Heat exchanger—openfoamwiki. https://openfoamwiki.net/index.php/OpenFOAM_guide/runTimeSelection_mechanism. Accessed 24 May 2023
59. Stevens P, Pooley R (1999) Using UML: software engineering with objects and components. Addison-Wesley Longman Publishing Co., Inc., Reading
60. coupledFvMatrix. https://sourceforge.net/p/foam-extend/foam-extend-4.1/ci/master/tree/src/coupledMatrix/coupledFvMatrices/coupledFvMatrix. Accessed 28 Feb 2023

61. Issa RI (1986) Solution of the implicitly discretised fluid flow equations by operator-splitting. J Comput Phys 62(1):40–65. https://doi.org/10.1016/0021-9991(86)90099-9
62. Patankar SV, Spalding DB (1972) A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. Int J Heat Mass Transf 15(10):1787–1806. https://doi.org/10.1016/0017-9310(72)90054-3
63. Davis K, Schulte M, Uekermann B (2022) Enhancing quasi-Newton acceleration for fluid–structure interaction. Math Comput Appl 27(3):40. https://doi.org/10.3390/mca27030040
64. Gropp W, Lusk E, Doss N, Skjellum A (1996) A high-performance, portable implementation of the MPI message passing interface standard. Parallel Comput 22(6):789–828. https://doi.org/10.1016/0167-8191(96)00024-5
65. Tuković Z, Karač A, Cardiff P, Jasak H, Ivanković A (2018) OpenFOAM finite volume solver for fluid–solid interaction. Trans FAMENA 42(3):1–31. https://doi.org/10.21278/TOF.42301
66. oftest. https://oftest.readthedocs.io/en/latest/readme.html. Accessed 06 March 2023
67. pytest. https://pypi.org/project/pytest/. Accessed 25 Aug 2023
68. pytest-xdist. https://pytest-xdist.readthedocs.io/en/latest/distribution.html. Accessed 25 Aug 2023
69. PyFoam. https://pypi.org/project/PyFoam/. Accessed 25 Aug 2023
70. CaseFOAM. https://casefoam.readthedocs.io/en/latest/?badge=latest. Accessed 25 Aug 2023
71. Vynnycky M, Kimura S, Kanev K, Pop I (1998) Forced convection heat transfer from a flat plate: the conjugate problem. Int J Heat Mass Transf 41(1):45–59. https://doi.org/10.1016/S0017-9310(97)00113-0
72. Chris Greenshields: OpenFOAM v10 user guide—4.5 numerical schemes (2022). https://doc.cfd.direct/openfoam/user-guide-v10/fvschemes. Accessed 07 Feb 2023
73. Heat exchanger - CHT simulation. https://www.simscale.com/projects/cheunglucia/heat_exchanger_-_cht_simulation/ Accessed 2023-05-24
74. Hirt CW, Amsden AA, Cook JL (1974) An arbitrary Lagrangian–Eulerian computing method for all flow speeds. J Comput Phys 14(3):227–253. https://doi.org/10.1016/0021-9991(74)90051-5
75. Tuković Z, Jasak H (2012) A moving mesh finite volume interface tracking method for surface tension dominated interfacial fluid flow. Comput Fluids 55:70–84. https://doi.org/10.1016/j.compfluid.2011.11.003. (**Publisher: Elsevier BV**)
76. Duineveld PC (1995) The rise velocity and shape of bubbles in pure water at high Reynolds number. J Fluid Mech 292:325–332. https://doi.org/10.1017/S0022112095001546
77. Turek S, Hron J (2006) Proposal for numerical benchmarking of fluid–structure interaction between an elastic object and laminar incompressible flow. In: Bungartz H-J, Schäfer M (eds) Fluid–structure interaction. Springer, Berlin, pp 371–385
78. Mesh generation in OpenFOAM. https://www.openfoam.com/documentation/guides/latest/api/group__grpMeshGenerationUtilities.html. Accessed 20 Jan 2024
79. refineMesh utility. https://openfoamwiki.net/index.php/RefineMesh. Accessed 20 Jan 2024
80. openFuelCell. https://openfuelcell.sourceforge.io/. Accessed 20 Feb 2023
81. Zhang S (2020) Modeling and simulation of polymer electrolyte fuel cells. Springer, Forschungszentrum Jülich
82. Zhang S, Hess S, Marschall H, Reimer U, Beale S, Lehnert W (2023) openFuelCell2: a new computational tool for fuel cells, electrolyzers, and other electrochemical devices and processes. Rochester. https://doi.org/10.2139/ssrn.4540105
83. Beale S, Lehnert W (eds) (2022) Electrochemical cell calculations with OpenFOAM. Corrected publication 2022 edn. Lecture notes in energy, vol 42. Springer, Cham
84. Steven B, Werner L (2023) A simple electrochemical cell model. In: Electrochemical cell calculation with OpenFOAM. Lecture Notes in Energy, vol. 42, pp 21–57. Springer, Cham. https://doi.org/10.1007/978-3-030-92178-1
85. Beale SB, Choi, H-W, Pharoah JG, Roth HK, Jasak H, Jeon DH (2015) Open-source computational model of a solid oxide fuel cell. Comput Phys Commun 200:15–26. https://doi.org/10.1016/j.cpc.2015.10.007
86. Bard AJ, Faulkner LR (2001) Electrochemical methods: fundamentals and applications, 2nd edn. Wiley, New York
87. Bockris JO, Reddy AKN, Gamboa-Aldeco ME (2001) Modern electrochemistry 2A: fundamentals of electrodics. Modern electrochemistry, Springer, New York