SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Interdisciplinary project report in Informatics

Design, implementation, and validation of a volume coupling extension for the OpenFOAM-preCICE adapter

Tina Vladimirova

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Interdisciplinary project report in Informatics

Design, implementation, and validation of a volume coupling extension for the OpenFOAM-preCICE adapter

Design, Implementierung und Validierung einer Erweiterung für Volumenkopplung für den OpenFOAM-preCICE Adapter

Author:	Tina Vladimirova
Supervisor:	Prof. Dr. Hans-Joachim Bungartz
Advisor:	Gerasimos Chourdakis, M. Sc.
Submission Date:	15.08.2023

I confirm that this interdisciplinary project report is my own work and I have documented all sources and material used.

Munich, 15.08.2023 Tina Vladimirova

Acknowledgments

I am sincerely thankful to my advisor Gerasimos Chourdakis for his valuable insights and consistent support.

Abstract

The OpenFOAM-preCICE adapter is a plug-in for the CFD software OpenFOAM which allows it to connect to the open-source coupling library preCICE. Initially, the adapter only supported coupling over a surface at the interface between two subdomains. Since there are various cases in practice where the subdomains can overlap, this project adds the feature of volume coupling, i. e. coupling over cells of the internal field of the domain.

We review contributions from the community which have implemented volume coupling for specific applications and unify them into a comprehensive long-term component of the adapter. We discuss how to best incorporate the new feature into the adapter and we integrate elements of previous work, resulting in a volume coupling extension with a configurable coupling region. Finally, we validate our implementation with a simulation scenario for each coupling direction (data transfer to and from OpenFOAM).

Contents

1 Introduction

Multi-physics simulation is an increasingly important tool in science and engineering, especially when we consider the growth of computational power over time. [1] [2]

preCICE is a coupling library for partitioned multi-physics simulations, namely simulations where a different solver is responsible for each subpart of the physics. The goal of preCICE is to allow for the coupling of an arbitrary number of different solvers in a flexible and convenient way. [3] preCICE provides adapters for software such as OpenFOAM, CalculiX and FEniCS. [4]

This project focuses on preCICE's adapter for the fluid dynamics simulation library OpenFOAM.[1] Currently, the adapter only supports surface coupling, i. e. the solvers can only exchange values over the boundary between each pair of subdomains. The goal of this project is to implement volume coupling, thereby allowing the adapter to also handle overlapping subdomains. In other words, the participants should be able to exchange and enforce source terms in addition to boundary conditions.

This report describes the step-by-step process of the development of a volume coupling extension for the OpenFOAM-preCICE adapter. Chapter 2 guides the reader through relevant background regarding OpenFOAM and the OpenFOAM adapter, and Chapter 3 provides a review of related work. In Chapter 4, I present the design of the extension and the factors which have influenced it, as well as an overview of the implementation. Finally, Chapter 5 describes validation experiments and evaluates the correctness of the new extension.

[1] https://www.openfoam.com/

2 Background

In this chapter, I'm introducing some relevant background and context which can aid the reader in understanding the rest of this work. First, I present the OpenFOAM-preCICE adapter, which is at the core of this project. Additionally, I describe some key characteristics of OpenFOAM's solvers and their relevant source terms.

2.1 OpenFOAM-preCICE adapter

The OpenFOAM-preCICE adapter is a plug-in for OpenFOAM which allows the user to connect an OpenFOAM solver to the preCICE coupling library and thus couple it to various other solvers. The adapter approach is advantageous because it does not require the user to modify OpenFOAM's code, it's independent of the particular OpenFOAM solver used, and it supports all recent versions of OpenFOAM. The adapter can be used for conjugate heat transfer, fluid-structure interaction and fluid-fluid coupling, and it can also be extended according to each project's needs. [2] [5]

The general structure of the adapter can be seen on Figure 2.1. The `Adapter` class reads the configuration and creates an `Interface` object for each interface between two participants' subdomains. It also handles the control flow of the simulation loop by calling preCICE methods underneath the hood. The `preciceAdapterFunctionObject` is a wrapper for the `Adapter` object which abstracts the interfacing between OpenFOAM and preCICE from the core adapter functionality.
Each `Interface` object configures the coupling mesh, which can be built on either face centers or face nodes at the interface.
Finally, the modules FF, CHT and FSI contain the logic for the variables relevant to each respective application. This includes individual implementations for reading and writing and possible additional operations to be done on the variable. Almost all of these variables are instances of the `CouplingDataUser` class, which defines some basic functionalities needed for coupled variables. [2] [6] [7]

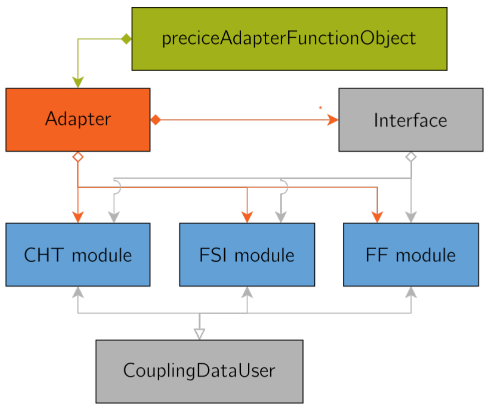


Figure 2.1: Architecture of the OpenFOAM-preCICE adapter [6]

2.2 OpenFOAM solvers

In this work we focus on the OpenFOAM solvers `pimpleFoam` [8] and `buoyantPimpleFoam` [9], for incompressible and compressible cases respectively. They both employ the PIMPLE algorithm - an iterative method which uses the momentum equation and the continuity equation and computes a new state for `p` (pressure) and `U` (velocity) in every iteration. [10] [11] [12]

The core structure of `pimpleFoam` can be seen in Listing 2.1. In each PIMPLE iteration it solves the momentum equation as specified in `UEqn.H` and then iterates through a pressure corrector loop. In `pEqn.H` the pressure is corrected and then the velocity is updated according to the new pressure values. [13] [11] [12]

```
while (pimple.loop())
{
  ...
    #include "UEqn.H"

    // --- Pressure corrector loop
    while (pimple.correct())
    {
        #include "pEqn.H"
    }
  ...
}
```

Listing 2.1: Simplified code snippet from `pimpleFoam.C` [13]

The `buoyantPimpleFoam` solver has a similar structure, but it additionally solves an energy equation before entering the pressure corrector loop. Apart from defining and solving the energy equation, `EEqn.H` also propagates the updated energy-related variables through the thermophysical model to update the temperature `T` and other fields. [14]

2.3 Source terms in OpenFOAM

Another relevant OpenFOAM concept are the finite volume options. The `fvOptions` structure allows the user to modify the source term of the equation and apply additional constraints without explicitly modifying the source code of the solvers. The user specifies the relevant options in the case files, more specifically in `constant/fvOptions`. [15]

The three main categories of options are sources, constraints and corrections, and we observe a common pattern in how they are applied in our chosen solvers. Firstly, each equation is discretized as a matrix and the source term on the right side is fetched from `fvOptions`. Then, the constraints are applied before the equation is solved, while the corrections are applied afterwards. [15]

```
// Solve the Momentum equation

MRF.correctBoundaryVelocity(U);

tmp<fvVectorMatrix> tUEqn
(
    fvm::ddt(U) + fvm::div(phi, U)
  + MRF.DDt(U)
  + turbulence->divDevReff(U)
 ==
    fvOptions(U)
);
fvVectorMatrix& UEqn = tUEqn.ref();

UEqn.relax();

fvOptions.constrain(UEqn);

if (pimple.momentumPredictor())
{
    solve(UEqn == -fvc::grad(p));

    fvOptions.correct(U);
}
```

Listing 2.2: UEqn.H of `pimpleFoam` [13]

Listing 2.2 is the implementation of the velocity equation for `pimpleFoam`. On line 11 in the definition of `tUEqn` we see the source term. The constraints are applied in line 17, then the equation is solved in line 21 and finally `U` is corrected in line 23. Since `U` is modified again in `pEqn.H`, the corrections are applied again afterwards. [13]

OpenFOAM provides several types of predefined sources meant for specific applications and variables. They are grouped into momentum, energy and atmospheric boundary layer applications. [16]
There are also two types of general sources, which allow for finer control and more flexibility. The semi-implicit source breaks down the source term into an explicit and implicit part: $S(x) = S_u + S_p x$ and the user can consequently choose S_u and S_p for each variable using the keyword `injectionRateSuSp`. [17]
On the other hand, the `codedSource` lets the user provide custom C++ implementations for the source (`codeAddSup`), constraints (`codeConstrain`) and corrections (`codeCorrect`). Each `codedSource` is bound to one or several variables. The user can access and modify the variables and equation matrices directly. OpenFOAM dynamically generates a class, where the class methods are generated from the three corresponding user-defined functions in `codedSource` and then plugged into the solver. [18]

3 Related work

This project was inspired by several community contributions which implement different variants of volume coupling for specific applications.

I focused on the following three contributions as the basis for our approach to volume coupling. See Figure 3.1 for a general timeline of these projects.
In 2019, Marta Camps Santasmasas from the University of Manchester created a fluid-fluid module with volume coupling for cell regions in the context of urban wind flow simulation. [19]
Later on in 2021, Stefan Scheiblhofer, Stephan Jäger and Amir M. Horr from AIT LKR [1] implemented volume coupling for temperature to be used in continuous casting process simulation. [20]
Based on that project, Prasad Adhav, Xavier Besseron, Alban Rousset and Bernhard Peters from LuXDEM [2] set out to develop a general-purpose volume coupling module. [21]

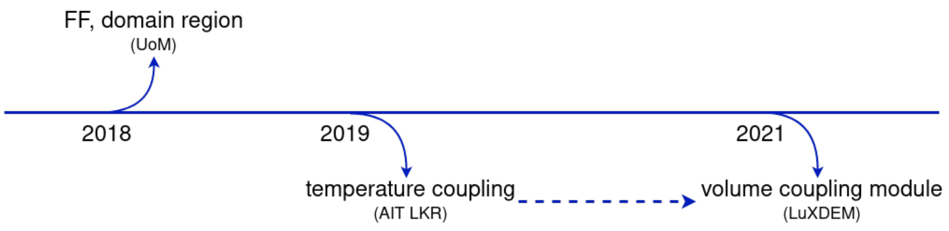


Figure 3.1: Timeline of community contributions

Each of these contributions was made by different teams independently of the preCICE project, each with their own individual goals. They were not integrated into the adapter and are largely unmaintained at this point in time. The aim of this project is to unify these existing approaches and integrate the new feature in a way that aligns with the long-term goals for preCICE and for the adapter.

[1]Austrian Institute of Technology, LKR Leichtmetallkompetenzzentrum Ranshofen
[2]Luxembourg XDEM Research Centre

3.1 Fluid-fluid coupling for a cell region

Santasmasas' approach is distinct in that it implements volume coupling for one or multiple regions of the domain, as opposed to the other approaches which would always couple the full domain. The cell regions are defined by OpenFOAM's `cellSets` - a collection of cells which can be created in many different ways, e. g. by using a box to bound a section of the domain. [22] [23]

Her case simulates urban windflow and involves two subdomains which overlap - one simulated by the Lattice-Boltzmann (LB) solver GASCANS and another by a Navier-Stokes (NS) solver, namely OpenFOAM's `pisoFoam`. Figure 3.2 depicts an exam-

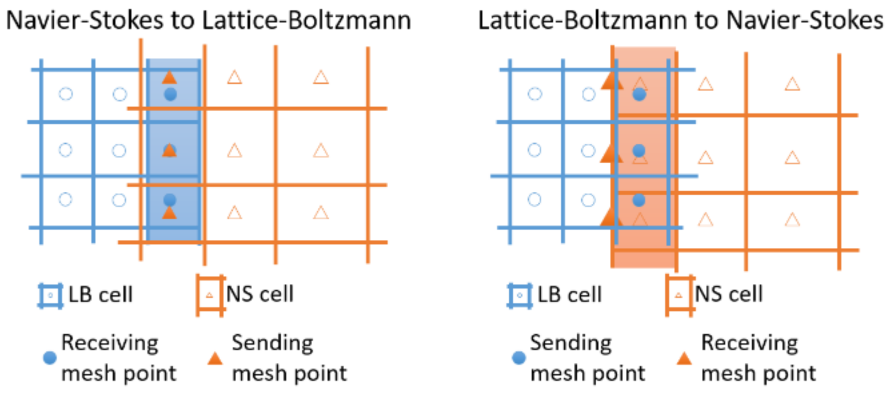


Figure 3.2: Sending and receiving meshes at the overlap between the NS and LB domain for each coupling direction [19]

ple of sending and receiving meshes for two-way coupling on these subdomains. Each subdomain has its own boundary conditions and uses different units for the velocity. The LB to NS boundary cells (right, orange) are in the NS domain and they receive velocity interpolated from the LB domain. A correction of the velocity can be done here to prevent a feedback loop of unstable results. The velocity is later interpolated from the NS domain to the NS to LB boundary (left, blue) and hence back to the LB domain. [19]
The code in the corresponding PR#88 implements volume coupling with `cellSets` for pressure and velocity in the `write()` direction.

3.2 Temperature volume coupling for casting simulation

The work of the AIT LKR team couples OpenFOAM, which is finite volume-based, with the finite element-based structural solver LS-DYNA. The goal is to simulate the casting process, which involves a mix of liquid metal and an ingot which is gradually solidifying. Initially, this was done with surface coupling only: more specifically, `laplacianFoam` was used to solve the heat equation and compute the temperature, then preCICE was used to write the temperature on the patches to LS-DYNA. In the opposite coupling direction, OpenFOAM was only reading a dummy value ($T_{sink} = 0$). [20]
At a later stage, volume coupling for temperature was implemented in PR#97.

3.3 General volume coupling module

The work of the LuXDEM team partly builds on the temperature volume coupling implementation from the previous section. However, this approach involves adding a separate volume coupling module independent of the existing modules and variables, as can be seen on Figure 3.3. The new module includes generic data handlers, which

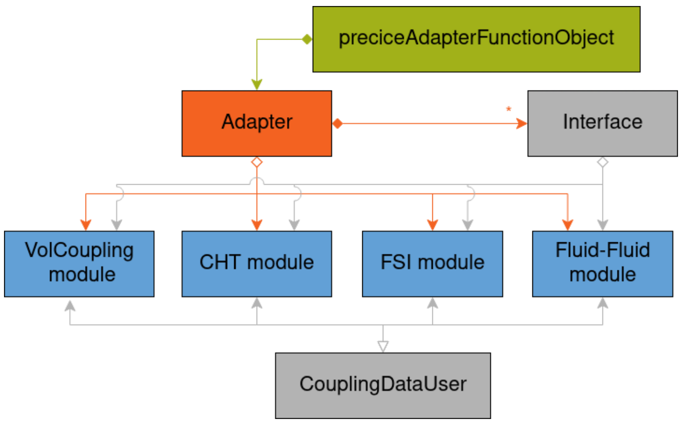


Figure 3.3: Class diagram for volume coupling module approach. Adapted from [6]

aim to provide a general implementation of `read()` and `write()` for any variable. Only the type of the field is taken into account, distinguishing between `volScalarField` and `volVectorField`. The corresponding code can be found in PR#183. The limitations of this approach are discussed in depth in Subsection 4.1.3.
The volume coupling was implemented in the context of a 6-way coupling scenario for an Abrasive WaterJet Cutting Nozzle. More specifically, three solvers (CFD, finite elements and discrete elements) are coupled bi-directionally. The coupling involves

variables such as force, displacement, fluid velocity, and porosity. [21]

3.4 Other contributions

My exploration of related work also featured other project which use the adapter and implement volume coupling for particular applications.
FERMI [24] is a framework developed in the context of fusion reactor simulation by researchers from Oak Ridge National Laboratory, Massachusetts Institute of Technology and Lawrence Livermore National Laboratory. This work helped me understand some specific practical aspects of volume coupling in OpenFOAM.
Another notable work is a crystal growth simulation framework [25] developed by the Leibniz Institute for Crystal Growth.

4 Design and implementation

4.1 Design

In this section I discuss different factors which have influenced the design of our volume coupling extension, and the decisions we have taken to reflect that. I point out our design goals and analyse in detail where and when it makes sense to apply volume coupling. Then, I explain our approach for the extension and finally the addition of configurable coupling regions.

4.1.1 Design goals

We defined objectives for this extension which align with the long-term goals for the adapter and for the preCICE library. They have guided us through the decision-making in the design phase.
Our aim was to tightly integrate the extension with the existing features and structure of the adapter. Consequently, this improves the long-term maintainability of the system. Another important point was configurability, especially in regards to the mesh region to be coupled. Finally, we wanted to keep the non-invasive approach of the adapter, i. e. the ability to use its full functionality without needing to modify OpenFOAM.

4.1.2 Scope of applications

An important point in our design was to consider which modules and variables would be compatible with volume coupling. In the related work we have seen examples for `Temperature` (CHT) [20], as well as `Pressure` and `Velocity` (Fluid-Fluid) [19]. Thus, we focused on these two modules, while also developing an extensible design to allow further implementations for future FSI applications or others.

Apart from `Pressure` and `Velocity`, the FF module includes variables for the respective gradients. Since these gradients are taken on the boundary patches by definition, we didn't consider these variables suitable for volume coupling. A possible analogue in volume could be a type of flux variable, but further discussion is needed regarding the type of flux (e. g. volumetric flux, mass flux, etc.) and the coupling locations.

The CHT module is based on equations that describe heat transfer on a fluid-solid interface [26], hence most of these variables would not be directly translatable to volume.

```
const scalarField gradientPatch(
      (T_->boundaryField()[patchID])
          .snGrad());
...
// For every cell of the patch
  forAll(gradientPatch, i)
  {
      // Copy the heat flux into the buffer
      // Q = - k * gradient(T)
      //TODO: Interpolate kappa in case of a turbulent calculation
      buffer[bufferIndex++] =
          -getKappaEffAt(i) * gradientPatch[i];
  }
```

Listing 4.1: Modified code snippet from `CHT/HeatFlux.C` [7]

For instance, `HeatFlux` is computed from the temperature gradient on the boundary patches (see code snippet in Listing 4.1) and `SinkTemperature` is already the temperature on specific internal fields (the ones neighboring the boundary patches). [7] These variables are specifically defined in relation to boundary patches, therefore they wouldn't make sense in a volume coupling context. Since the related work [20] demonstrates a practical application of volume-coupled `Temperature`, we have included it as the only CHT variable to support volume coupling.

4.1.3 Core idea

Our approach is based on PR#183 from the LuXDEM team [21] as described in Section 3.3. At the core of our implementation is the data transfer: the idea is to iterate over all inner cell centers of the domain and respectively read or write the values to preCICE's buffer. Additionally, the user can still specify patch names with volume coupling - the boundary patches will then be coupled on face centers in addition to the volume. Even though we kept the fundamental aspects of LuXDEM's approach, we chose a different design for the extension in order to optimally fulfil our design goals.

In terms of architecture, we moved away from the idea of a separate volume coupling module. We consider the centers of the volume as a type of coupling location, similarly to the existing `faceCenters` and `faceNodes`. Since the related work [20] [21] already adds volume as a type of location, we continue that idea and incorporate it with existing variables and modules (see Figure 4.1). This approach allows us to integrate the new feature more smoothly with the rest of the adapter's functionality.

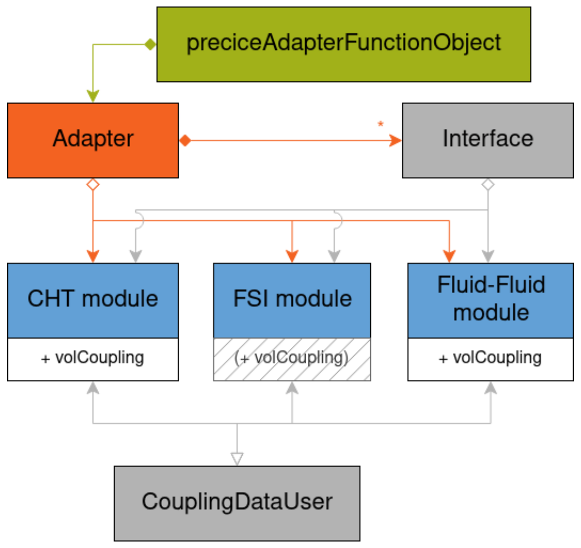


Figure 4.1: Class diagram for our volume coupling approach. Adapted from [6]

Another significant characteristic of LuXDEM's approach are the generic data handlers. While they allow the flexible addition of new variables, in their current state they constrain how the variables can be treated. The generic data handlers can read and write variables directly, but cannot deal with variables that need extra computation on top of the data transfer. For instance, the adapter's `HeatFlux` is computed from the temperature gradient and the thermal conductivity before it can be written, while reading it involves additionally updating the temperature gradient. [7]
Integrating the generic data handlers with each variable and the individual operations it requires would create extra complexity. On the other hand, only using it with some variables would be inconsistent. Thus, this feature could be implemented in the future, but requires further discussion and is outside the scope of this project.

4.1.4 Restricting to a domain region

We adopted Santasmasas' [19] idea from PR#88 to use `cellSets` for volume coupling regions. In summary, the user can specify one or multiple `cellSets` to couple. The data transfer is consequently done not for all internal fields, but only for the cells of the desired `cellSets`.

In order to integrate Santasmasas' work with the base volume coupling implementation, we make the following case distinction (Figure 4.2): if the user chooses the volume `locationType` and doesn't specify any `cellSet` names, we use the base case and couple the full domain; otherwise, we only couple the specified regions.

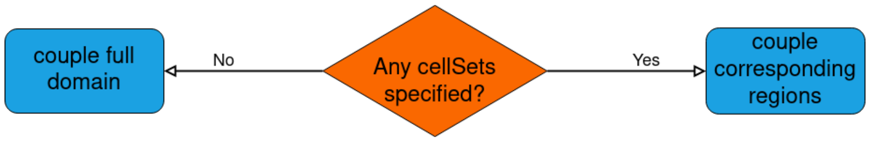


Figure 4.2: Case distrinction to determine which part of the domain is coupled

4.2 Implementation

This section describes how I implemented the volume coupling extension based on the design from the previous section. First, I explain the implementation of the basic case, then I move on to the addition of a coupling region and how the user can configure it.

4.2.1 Base implementation

The core of my implementation is the addition of a new `locationType` `volumeCenters` next to the existing `faceCenters` and `faceNodes` from the surface coupling. The code discussed in this subsection and demonstrated in the snippets can be found on PR#255[1].

On the configuration side, the new `locationType` needs to be integrated in the configuration processing and the new mesh has to be set up and passed to the relevant variables. First, I add `volumeCenters` to the `LocationType` enums and include it as a permitted value in the parsing of `preciceDict`.
Next, I integrate previous work from the AIT LKR team [20] (PR#97) and from the LuXDEM team [21] (PR#183) to implement the new mesh in `Interface.C`. The number of coupling locations is counted by summing up the number of cell centers in the volume and the number of face centers on each patch (if applicable).

```
const vectorField& CellCenters = mesh.C();

for (int i = 0; i < CellCenters.size(); i++)
{
    vertices[verticesIndex++] = CellCenters[i].x();
    vertices[verticesIndex++] = CellCenters[i].y();
```

[1]The code snippets, descriptions, and results reflect the state of the PR as of 14.08.2023.

```
    if (dim_ == 3)
    {
        vertices[verticesIndex++] = CellCenters[i].z();
    }
}
```

Listing 4.2: Code snippet from `Interface.C` where cell centers are added as coupling locations

Then, the 2D or 3D coordinates of each mesh point (all cell centers and relevant face centers) is added to the list of vertices and passed to preCICE. Part of this process can be seen in Listing 4.2.

Furthermore, we propagate the new `locationType` to the variables. This involves implementing data handling for the volume-coupled `Pressure`, `Velocity` and `Temperature`, but also making sure `volumeCenters` is excluded from the permitted `locationTypes` for the rest of the variables. For reading and writing of a volume-coupled variable, I reuse the generic data handlers from LuXDEM's work [21] (PR#183) and apply the code to the individual variables. Essentially, we iterate over each cell of the internal field and copy it to or from the adapter's buffer. Listing 4.3 and Listing 4.4 demonstrate how writing and reading respectively are implemented for `Temperature` (a `volScalarField`).

```
for (const auto& cell : T_->internalField())
{
  buffer[bufferIndex++] = cell;
}
```

Listing 4.3: Writing volume-coupled `Temperature` (`CHT/Temperature.C`)

```
for (auto& cell : T_->ref())
{
  cell = buffer[bufferIndex++];
}
```

Listing 4.4: Reading volume-coupled `Temperature` (`CHT/Temperature.C`)

I use `ref()` [27] to reference the internal field when writing, and the const-reference equivalent `internalField()` [28] when reading into the buffer.
Last but not least, I modify the method `isLocationTypeSupported` for both variables that do and do not support volume coupling. Apart from raising an error when `volumeCenters` is used with an incompatible variable, I also prevent it from being used with `meshConnectivity`. Mesh connectivity is a feature of the adapter used for nearest projection mapping between meshes, but at this stage I have not implemented it for volume coupling.

4.2.2 Restrict to domain region

In order to restrict volume coupling to a region instead of the full domain, the user can specify one or multiple `cellSets` in `system/topoSetDict` and execute the command `topoSet` in the run-script. [29] An example for the content of `topoSetDict` can be seen in Listing 4.5. The user also has to reference the names of the desired `cellSets` to be coupled in `preciceDict` - they can be a subset of all of the `cellSets` defined in `topoSetDict`.

```
actions
(
    {
        name    box1;
        type    cellSet;
        action  new;
        source  boxToCell;
        box     (3.0 1.0 0.0) (3.5 1.5 1.0);
    }
);
```

Listing 4.5: Creating a coupling region in a box of cells defined by two points. Source: topoSetDict as of 14.08.2023.

The main aspects of the implementation are again extending the configuration and adapting the variables. I adopt Santasmasas' previous work and update it to fit with the adapter's current state. The code described in this subsection, including Listing 4.6, can be found on PR#270[2].

In the configuration, the `cellSet` names are added to a list and passed to the respective `Interface` object. There they are used for the mesh setup: similarly to the full domain case, for each `cellSet` name we find the corresponding cells on the mesh and they are reflected in the number of coupling locations, and respectively in the list of vertices for preCICE.

Additionally, I extend the volume coupling implementation for `Pressure`, `Velocity` and `Temperature`. If there are any `cellSetNames` specified, for each name we find the respective region of overlapping cells and read from or write the values to the buffer. Listing 4.6 shows how this is implemented for `Temperature`, more specifically the `cellSets` iteration is covered in lines 10-20.

```
1  if (cellSetNames_.empty())
2  {
3    for (const auto& cell : T_->internalField())
```

[2]The code snippets, descriptions, and results reflect the state of the PR as of 14.08.2023.

```
  {
    buffer[bufferIndex++] = cell;
  }
}
else
{
  for (const auto& cellSetName : cellSetNames_)
  {
    cellSet overlapRegion(T_->mesh(), cellSetName);
    const labelList& cells = overlapRegion.toc();

    for (const auto& currentCell : cells)
    {
      // Copy temperature into the buffer
      buffer[bufferIndex++] = T_->internalField()[currentCell];
    }
  }
}
```

Listing 4.6: Volume-coupled writing for `Temperature` with case distinction (`CHT/Temperature.C`)

Finally, both in the configuration and in data handling, we fall back on the implementation from the previous section if the list of `cellSet` names is empty - we assume that means the user wants to couple the full domain.

4.3 Limitations of reading fields in OpenFOAM

In this section, I examine issues with reading internal fields in OpenFOAM. After describing the limitations and pointing out examples, I suggest how they can be mitigated. I use OpenFOAM's variable names `p`, `U` and `T` for pressure, velocity, and temperature respectively.

4.3.1 Issues with reading internal field cells in OpenFOAM

When reading cells of the internal field in OpenFOAM, we expect that the value at the end of OpenFOAM's iteration is the same value that was read during the coupling. However, it is necessary to utilize OpenFOAM's `fvOptions` [15] in order to enforce that value. Without this additional configuration step, OpenFOAM overwrites the value it has received from the coupling, and this is demonstrated by the following experiment. I used a dummy solver to write constant values to a region of the domain for different variables. The scenario coupled the dummy solver to OpenFOAM's `buoyantPimpleFoam` uni-directionally. An example of this setup for `U.x()` is depicted on Figure 4.3 and

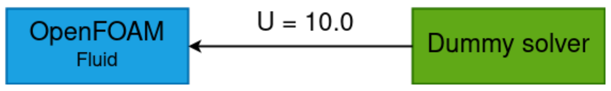


Figure 4.3: Setup for reading experiment on OpenFOAM's internal field

Figure 4.5 shows how this looks on the OpenFOAM side. Although the scenario is only concerned with the x coordinate of `U`, I use `U` for simplicity.

The results show different behaviors depending on the coupled variable. Reading `p` and `U` shifts OpenFOAM's values in the right direction, but not necessarily to the exact value written by the dummy participant. For example, Figure 4.4 shows the average values and ranges of `U.x()` on the coupled region (in OpenFOAM) over time. The dummy solver writes $U.x() = 10.0$ to a domain initialized with $U.x() = 0.1$. The values do increase compared to the initial value, but they are in the range between 7.0 and 10.0, whereas we expect a constant at $U.x() = 10.0$.

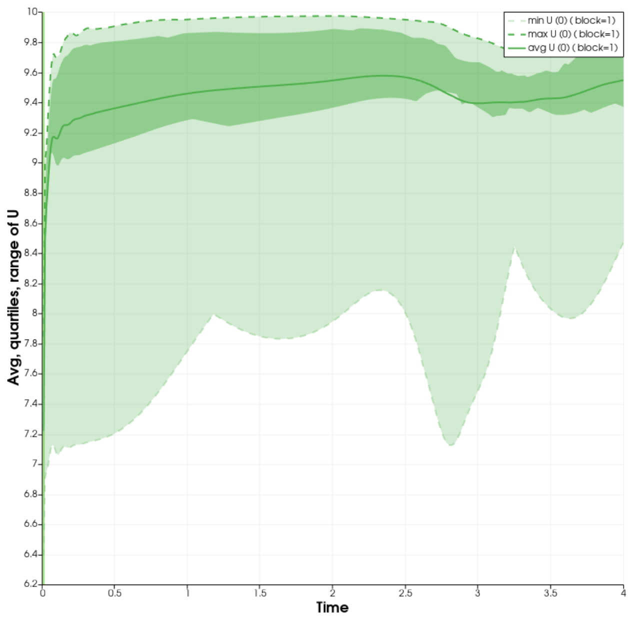


Figure 4.4: Average value and ranges of `U.x()` on the coupled cell region over time. The initial value was $U.x() = 0.1$, the value read in OpenFOAM was $U.x() = 10.0$

This behavior can be explained by examining inner workings of the solver. The

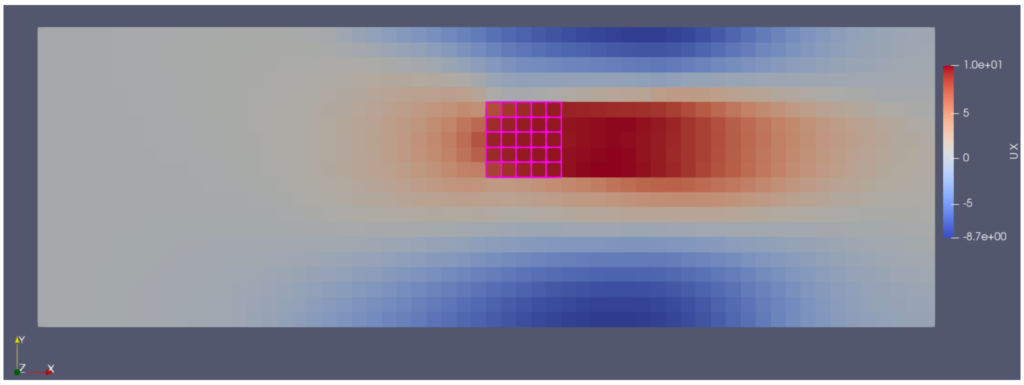


Figure 4.5: Surface plot of `U.x()` in OpenFOAM. The cells of the coupled region are highlighted.

equations are solved through an iterative method (see Section 2.2 and Listing 2.1), which means the value from the previous iteration v_{old} influences the current value v_{new}. When the reading is done in OpenFOAM, this essentially overwrites v_{old} before the equation is solved, instead of overwriting v_{new} at the end. A visualization of this workflow is depicted on Figure 4.6.

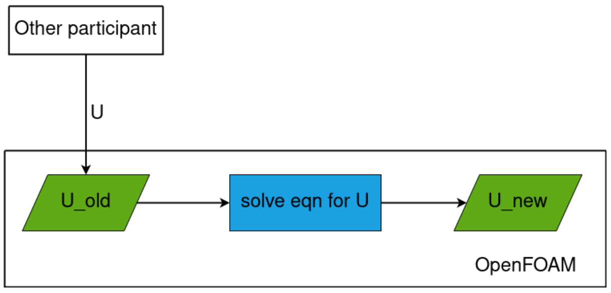


Figure 4.6: Flow of information when `U` is read in OpenFOAM without using `fvOptions`

I also performed this experiment for `T`, but this time the values on the OpenFOAM side were completely unaffected by the values received in the coupling. On Figure 4.7 we see that `T` is constant at the initial value of $T = 300.0$. The temperature is a special case, because it is involved with a thermophysical model. Namely, its value depends on multiple other variables, such as enthalpy (`he`) (e. g. see [30]). Since we are not updating those variables, the model enforces that the temperature also stays the same. These results motivate the usage of `fvOptions`(see Section 2.3), which would provide finer control and more precision over where we want to "inject" our code and how we

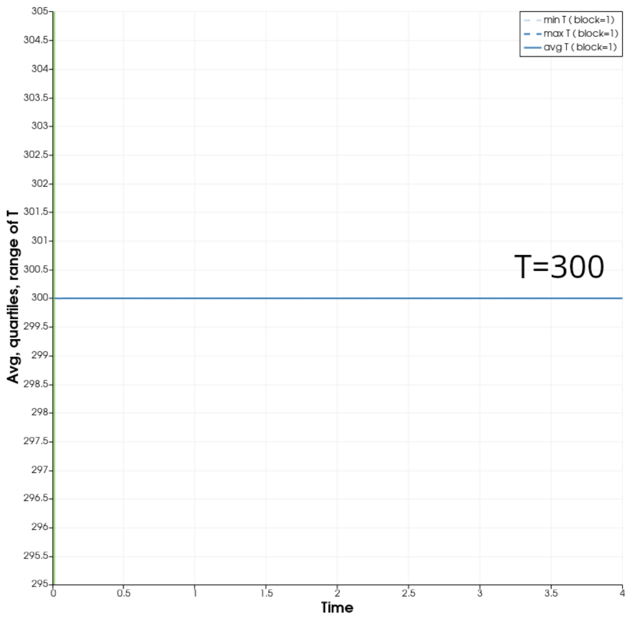


Figure 4.7: Average value and ranges of U.x() on the coupled cell region over time. The initial value was $T = 300$, the value read in OpenFOAM was $T = 600$. The result is a constant at $T = 300$.

can enforce source terms.

4.3.2 Reading fields with finite volume options

Out of the possible specific fvOptions sources referenced in Section 2.3, none seems to fit our application. Thus, I implemented a codedSource [18] and focused specifically on the codeCorrect function. The correction is done after solving the equation and this guarantees that the desired value will not be overwritten.

It is worth noting that the codedSource is not always activated - it depends on the combination of solver and bound variables. For example, I found that p and T (and additionally p_rgh) cannot be updated through a codedSource, while I succeeded with U (tested with buoyantPimpleFoam and pimpleFoam).
Additionally, the preset constraint fixedTemperatureConstraint also succeeded in enforcing the correct value for T, even though it is a constraint and not a correction. In constrast to a T-bound codedSource, this constraint updates the enthalpy he in order to enforce the new T value through the thermophysical model (see [31]).

I succeeded in enforcing values for `U` through `fvOptions`. In order to carry over the value through the adapter to OpenFOAM, it is necessary to create a "mirror" variable of `U` and use it in the `fvOptions`. This allows us to keep the value buffered and apply it at the right time to modify the "real" `U`. In order for this to work, we require the user to specify an alternative name for the coupled velocity in the OpenFOAM case files - since OpenFOAM uses `U`, we need an alternative name for our buffer variable. Essentially, OpenFOAM sees two velocity variables - one to receive the coupled velocity, and one for its own velocity. Figure 4.8 demonstrates a graphical explanation of this workflow, and the case itself is introduced in PR#350[3] under `volume-coupled-flow`.

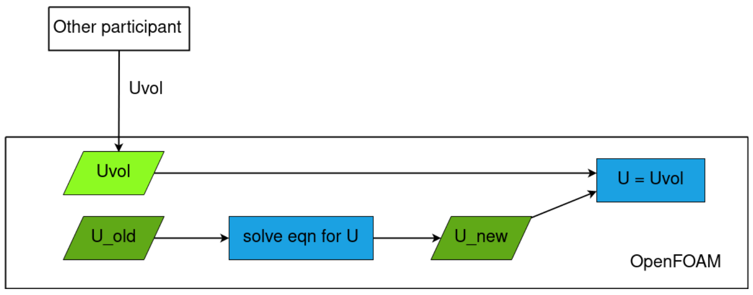


Figure 4.8: Flow of information when `U` is read in OpenFOAM using a `codedSource`

[3]The code snippets, descriptions, and results reflect the state of the PR as of 14.08.2023.

5 Validation

In this chapter, I describe two experiments and how they validate different aspects of the volume coupling extension. First, the *channel transport* case is used to validate a basic volume coupling scenario where the full domain is coupled and OpenFOAM is writing the source terms. In contrast, in *volume-coupled flow* only a region of the domain is coupled, and the source terms are read and enforced in the OpenFOAM domain. Additionally, I present and interpret the results of those experiments.

5.1 Channel transport

To validate the basic volume coupling implementation, I adapted the existing preCICE tutorial *channel transport* [32]. The case originally coupled two Nutils participants. I replaced one of them with an OpenFOAM participant and compared the new result to the reference result.

5.1.1 Setup

The *channel transport* tutorial scenario involves a blob travelling through a channel with an obstacle. Two participants are involved: one fluid participant, which controls the fluid flowing in the channel, and a transport participant responsible for the movement of the blob. The coupling is uni-directional, wherein the fluid participant writes velocity `U` to the transport participant. [32] [33] I extended the tutorial by adding another variant

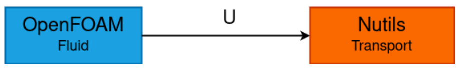


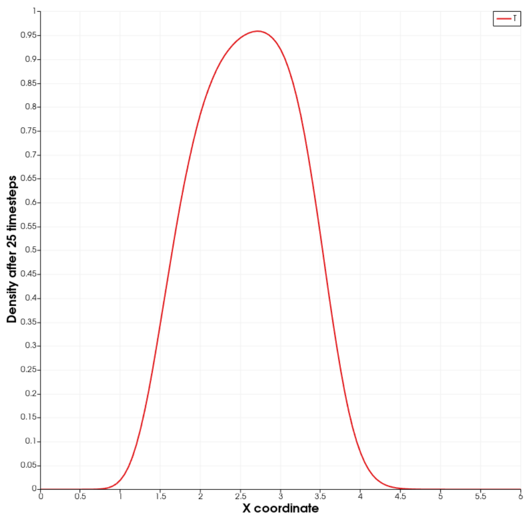
Figure 5.1: Setup for channel transport tutorial with an OpenFOAM fluid participant

for the fluid participant, namely OpenFOAM's `pimpleFoam`. Thus, the version with the Nutils fluid participant can be used as a reference, while the OpenFOAM version serves to validate the volume coupling extension of the adapter. This setup is depicted on Figure 5.1. The extended tutorial can be found in PR#315[1].

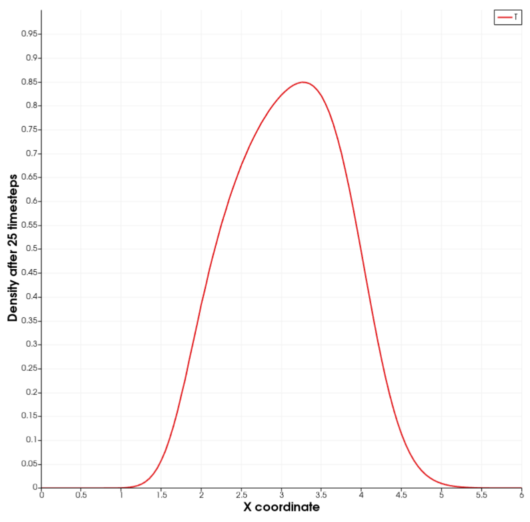
[1]The code snippets, descriptions, and results reflect the state of the PR as of 14.08.2023.

5.1.2 Results

The line plots in Figure 5.2 and Figure 5.3 depict the density of the transported species over a line at $y = 1.5$ (parallel to x). The figures are a comparison of the reference output and the new output side by side. In Figure 5.2 at timestep 25 the results look similar, but the peak is a bit sharper on the reference graph. The density reaches 0.95 in the reference and only about 0.85 in the new result.



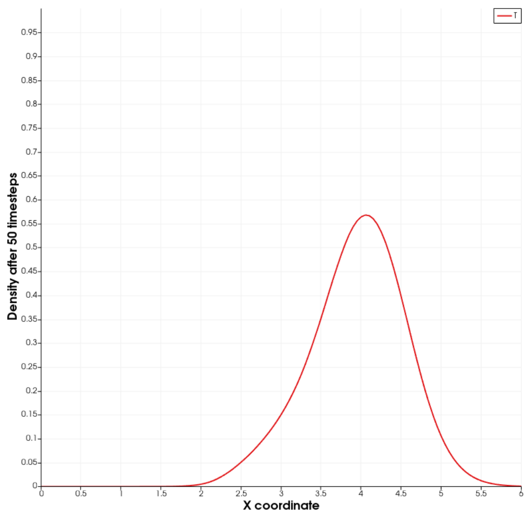
(a) Density of species over x ($y = 1.5$) at timestep 25: reference output (Nutils fluid)



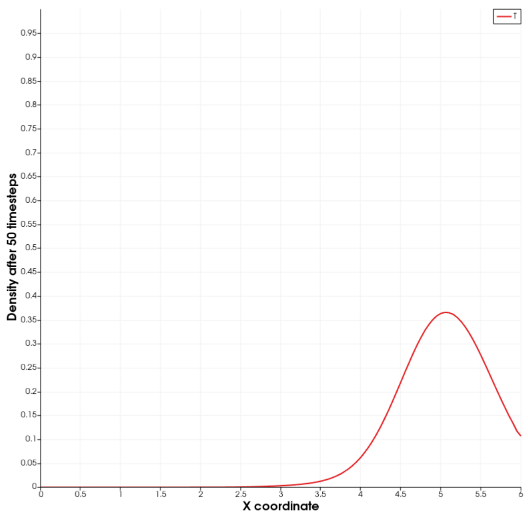
(b) Density of species over x ($y = 1.5$) at timestep 25: new output (OpenFOAM fluid)

Figure 5.2: Density of species over x ($y = 1.5$) at timestep 25: comparison of reference output (Nutils fluid) and new output (OpenFOAM fluid)

A similar observation can be made about Figure 5.3 at timestep 50, but the differences become more noticeable. The peak of the reference results reaches just over 0.55, whereas for the OpenFOAM participant it's between 0.35 and 0.4. Here we see that the peak also seems to be at slightly different positions: around $x = 4$ for the reference output and $x = 5$ for the new output.
From these results it seems that the blob in the OpenFOAM variant is moving slightly faster than the one in the Nutils variant. A plausible explanation for this discrepancy is that OpenFOAM uses finite-volume methods, while Nutils is a finite-element solver. [34] Nevertheless, the results do show enough similarity to conclude that the experiment is successful and the volume coupling extension is functioning correctly.



(a) Density of species over x ($y = 1.5$) at timestep 50: reference output (Nutils fluid)



(b) Density of species over x ($y = 1.5$) at timestep 50: new output (OpenFOAM fluid)

Figure 5.3: Density of species over x ($y = 1.5$) at timestep 50: comparison of reference output (Nutils fluid) and new output (OpenFOAM fluid)

5.2 Volume-coupled flow

This case couples a "dummy solver" with OpenFOAM (`buoyantPimpleFoam`), this time putting OpenFOAM in the reading position. The experiment serves to validate the volume coupling over a domain region, as well as the correctness of reading source terms in OpenFOAM with the help of `fvOptions`.

5.2.1 Setup

The *volume-coupled flow* tutorial case is derived from the reading experimentation as described in Section 4.3. A dummy solver, which only writes a constant value of $U.x() = 10.0$ to the domain, is coupled with OpenFOAM over a square `cellSet`. The coupled `cellSet` is created from a bounding box as defined in `topoSetDict`. Figure 5.4 depicts the OpenFOAM domain and the coupled `cellSet` is highlighted. Additionally, I utilize `fvOptions` as described in Subsection 4.3.2 to enforce the desired values on the coupled region. This new case can be found on PR#350[2] under `volume-coupled-flow`.

[2]The code snippets, descriptions, and results reflect the state of the PR as of 14.08.2023.

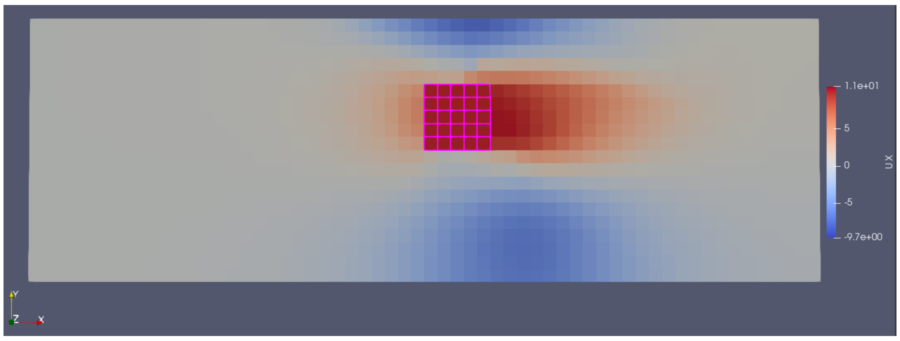


Figure 5.4: Surface plot of U.x() at $t = 0.5$, *volume-coupled flow* tutorial.

5.2.2 Results

For this experiment, we once again look at the averages and ranges of $U.x()$ on the coupled region over time. In contrast to the experiments in Subsection 4.3.1, Figure 5.5 clearly shows a constant value at $U.x() = 10.0$, which indicates the values are being read and applied correctly in OpenFOAM. Additionally, looking back on Figure 5.4

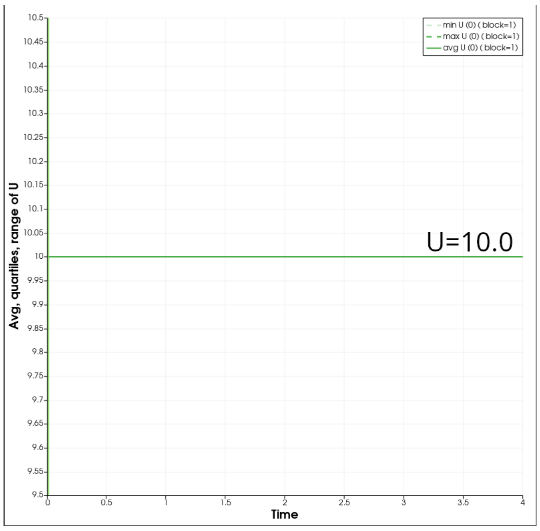


Figure 5.5: Average value and ranges of U.x() on the coupled cell region over time. The initial value was $U.x() = 0.1$, the value read in OpenFOAM was $U.x() = 10.0$. The result is a constant at $U.x() = 10.0$.

we can see that the coupling is restricted to the specified region and the values evolve over time ouside of that region. These results confirm that the extension supports reading and enforcing of source terms, and it also successfully allows for coupling over a domain region.

6 Conclusion

In this report, I have described the development of a volume coupling extension for the OpenFOAM-preCICE adapter. I have highlighted related work and its role in our design, as well as other considerations relating to our design goals and the architectural decisions we took to adhere to them. Then, I have discussed the most important points of the implementation of the volume coupling base, and the addition of a configurable mesh coupling region. I have presented problems encountered in relation to source terms in OpenFOAM, and some ways they can be mitigated. Finally, I have demonstrated two validation experiments. The results show that the volume coupling extension works as intended in both coupling directions (reading and writing), and for the coupling of domain regions.

Throughout this project, a few potential ideas for future work have emerged. For instance, the idea for generic data handlers from the related work is promising and could be implemented for the adapter in the future.
Additionally, the volume coupling can be expanded to other existing variables in the adapter by creating volumetric equivalents of them. For example, I discussed the possibility to add flux variables which would serve as the volume-coupled equivalent to gradients.
Another possible task for the future would be to integrate the adapter's mesh connectivity feature with the volume coupling.
Finally, it would be interesting to experiment further with the reading of internal fields in OpenFOAM. It is worth examining other aspects of the finite volume options (e. g. comparing the capabilities of contraints as opposed to corrections), or exploring the capabilities of other solvers and variables.

Bibliography

[1] D. E. Keyes, L. C. McInnes, C. Woodward, W. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors, E. Constantinescu, D. Estep, K. Evans, C. Farhat, A. Hakim, G. Hammond, G. Hansen, J. Hill, T. Isaac, X. Jiao, K. Jordan, D. Kaushik, E. Kaxiras, A. Koniges, K. Lee, A. Lott, Q. Lu, J. Magerlein, R. Maxwell, M. McCourt, M. Mehl, R. Pawlowski, A. P. Randles, D. Reynolds, B. Rivière, U. Rüde, T. Scheibe, J. Shadid, B. Sheehan, M. Shephard, A. Siegel, B. Smith, X. Tang, C. Wilson, and B. Wohlmuth, "Multiphysics simulations: Challenges and opportunities," *The International Journal of High Performance Computing Applications*, vol. 27, no. 1, pp. 4–83, 2013.

[2] G. Chourdakis, D. Schneider, and B. Uekermann, "OpenFOAM-preCICE: Coupling OpenFOAM with External Solvers for Multi-Physics Simulations," *OpenFOAM® Journal*, vol. 3, p. 1–25, Feb. 2023.

[3] "preCICE homepage." `https://precice.org/`. Accessed: 14.08.2023.

[4] "preCICE docs v2.5.0: Overview of adapters." `https://precice.org/adapters-overview.html`. Accessed: 14.08.2023.

[5] "preCICE docs v2.5.0: The OpenFOAM adapter." `https://precice.org/adapter-openfoam-overview.html`. Accessed: 14.08.2023.

[6] "preCICE docs v2.5.0: Extend the OpenFOAM adapter: Architecture." `https://precice.org/adapter-openfoam-extend.html#architecture`. Accessed: 14.08.2023.

[7] "OpenFOAM-preCICE adapter." `https://github.com/precice/openfoam-adapter/tree/develop`. Accessed: 14.08.2023.

[8] "OpenFOAM User Guide v2112: pimpleFoam." `https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-incompressible-pimpleFoam.html`. Accessed: 14.08.2023.

[9] “OpenFOAM User Guide v2112: buoyantPimpleFoam.” https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-heat-transfer-buoyantPimpleFoam.html. Accessed: 14.08.2023.

[10] “OpenFOAM User Guide v2112: Pressure-velocity algorithms.” https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-pressure-velocity-intro.html. Accessed: 14.08.2023.

[11] S. Ye, Y. Lin, L. Xu, and J. Wu, “Improving Initial Guess for the Iterative Solution of Linear Equation Systems in Incompressible Flow,” *Mathematics*, vol. 8, p. 119, 01 2020.

[12] G. Holzinger, *OpenFoam - a little user manual*. 03 2020.

[13] “pimpleFoam source code.” https://develop.openfoam.com/Development/openfoam/-/tree/master/applications/solvers/incompressible/pimpleFoam. Accessed: 14.08.2023.

[14] “buoyantPimpleFoam source code.” https://develop.openfoam.com/Development/openfoam/-/tree/master/applications/solvers/heatTransfer/buoyantPimpleFoam. Accessed: 14.08.2023.

[15] “OpenFOAM User Guide v2112: Finite volume options.” https://www.openfoam.com/documentation/guides/latest/doc/guide-fvoptions.html. Accessed: 14.08.2023.

[16] “OpenFOAM User Guide v2112: Sources for Finite volume options.” https://www.openfoam.com/documentation/guides/latest/doc/guide-fvoptions-sources.html. Accessed: 14.08.2023.

[17] “OpenFOAM User Guide v2112: Semi-implicit sources for Finite volume options.” https://www.openfoam.com/documentation/guides/latest/doc/guide-fvoptions-sources-semi-implicit.html. Accessed: 14.08.2023.

[18] “OpenFOAM User Guide v2112: Coded source for Finite volume options.” https://www.openfoam.com/documentation/guides/latest/doc/guide-fvoptions-sources-coded.html. Accessed: 14.08.2023.

[19] M. Camps Santasmasas, *Hybrid GPU / CPU Navier-Stokes lattice Boltzmann method for urban wind flow*. PhD thesis, 06 2021.

[20] S. Scheiblhofer, S. Jäger, and A. Horr, "Coupling FEM and CFD solvers for continuous casting process simulation using preCICE," in *ECCOMAS Coupled Problems 2019*, pp. 23–32, 2019. VIII International Conference on Computational Methods for Coupled Problems in Science and Engineering ; Conference date: 03-06-2019 Through 05-06-2019.

[21] X. Besseron, A. Rousset, A. Peyraut, and B. Peters, "6-way coupling of DEM+CFD+FEM with preCICE," 02 2020.

[22] "OpenFOAM User Guide v2112: cellSet." `https://www.openfoam.com/documentation/guides/latest/api/classFoam_1_1cellSet.html`. Accessed: 14.08.2023.

[23] "OpenFOAM User Guide v2112: cellSet: boxToCell." `https://www.openfoam.com/documentation/guides/latest/doc/guide-cellSet-boxToCell.html`. Accessed: 14.08.2023.

[24] A. Sircar, J. W. Bae, E. Peterson, J. Solberg, and V. Badalasi, "FERMI: A multiphysics simulation environment for fusion reactor blanket," tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2022.

[25] A. Wintzer, J. Pal, and K. Dadzis, "Development of a python-based crystal growth simulation framework," 02 2021.

[26] L. C. Yau, "Conjugate Heat Transfer with the Multiphysics Coupling Library preCICE," 2016.

[27] "OpenFOAM API Guide v2112: Foam::GeometricField::ref." `https://www.openfoam.com/documentation/guides/latest/api/classFoam_1_1GeometricField.html#a77a3ea1ce7e2adc04d27301292b095ae`. Accessed: 14.08.2023.

[28] "OpenFOAM API Guide v2112: Foam::GeometricField::internalField." `https://www.openfoam.com/documentation/guides/latest/api/classFoam_1_1GeometricField.html#ad05db1e8059011ab19b10d3e38a25ba6`. Accessed: 14.08.2023.

[29] "OpenFOAM User Guide v2112: topoSet." `https://www.openfoam.com/documentation/guides/latest/doc/guide-meshing-topoSet.html`. Accessed: 14.08.2023.

[30] "heRhoThermo.C." `https://develop.openfoam.com/Development/openfoam/-/blob/master/src/thermophysicalModels/basic/rhoThermo/heRhoThermo.C`. Accessed: 14.08.2023.

[31] "fixedTemperatureConstraint source code." https://develop.openfoam.com/Development/openfoam/-/tree/master/src/fvOptions/constraints/derived/fixedTemperatureConstraint. Accessed: 14.08.2023.

[32] "preCICE Tutorials: Channel transport." https://precice.org/tutorials-channel-transport.html. Accessed: 14.08.2023.

[33] "Channel transport tutorial case files." https://github.com/precice/tutorials/tree/master/channel-transport. Accessed: 14.08.2023.

[34] "About Nutils." https://nutils.org/index.html. Accessed: 14.08.2023.