FEMuS-Platform: a numerical platform for multiscale and multiphysics code coupling

Andrea Chierici, Giacomo Barbi, Giorgio Bornia, Daniele Cerroni, Leonardo Chirco, Roberto Da Vià, Valentina Giovacchini, Sandro Manservisi, Ruben Scardovelli, Antonio Cervone

INFORMATION

Keywords:
multiphysics
multiscale
code coupling

DOI: 10.23967/coupled.2021.027

Published: 12/07/2021

IX International Conference on Computational Methods for Coupled Problems in Science and Engineering
COUPLED PROBLEMS 2021
E. Oñate, M. Papadrakakis and B. Schrefler (Eds)

FEMUS-PLATFORM: A NUMERICAL PLATFORM FOR MULTISCALE AND MULTIPHYSICS CODE COUPLING

G. BARBI[1], **G. BORNIA**[2], **D. CERRONI**[3], **A. CERVONE**[4], **A. CHIERICI**[1]*, **L. CHIRCO**[5], **R. DA VIÁ**[4], **V. GIOVACCHINI**[1], **S. MANSERVISI**[1] **AND R. SCARDOVELLI**[1]

[1] DIN, Lab. of Montecuccolino, University of Bologna, Via dei Colli 16, 40136 Bologna, IT

[2] Department of Mathematics and Statistics, Texas Tech University, Lubbock, TX, USA

[3] Politecnico di Milano, P.za Leonardo Da Vinci 32, 20133 Milano, IT

[4] ENEA, Via Martiri di Monte Sole 4, Bologna, 40129, Italy

[5] Sorbonne Université, Institut Jean Le Rond d'Alembert, 4 Place Jussieu, Paris, F

*e-mail: andrea.chierici4@unibo.it

Key words: Multiscale problems, Multiphysics platform, Code coupling

Abstract. Nowadays, many open-source numerical codes are available to solve physical problems in structural mechanics, fluid flow, heat transfer, and neutron diffusion. However, even if these codes are often highly specialized in the numerical simulation of a particular type of physics, none of them allows simulating complex systems involving all the physical problems mentioned above. In this work we present a numerical framework, based on the SALOME platform, developed to perform multiscale and multiphysics simulations involving all the mentioned physical problems. In particular, the developed numerical platform includes the multigrid finite element in-house code FEMuS for heat transfer, fluid flow, turbulence and fluid-structure modeling; the open-source finite volume CFD software OpenFOAM; the multiscale neutronic code DONJON-DRAGON; and a system-scale code used for thermal-hydraulic simulations. Efficient data exchange among these codes is performed within computer memory by using the MED libraries, provided by the SALOME platform.

1 INTRODUCTION

In this work, we present the multiscale and multiphysics platform FemusPlatform [1] developed at the Department of Industrial Engineering of the University of Bologna. The FemusPlatform has been developed as an environment where several open, research, and commercial numerical codes can be run together and allows modeling complex physical phenomena on different physical scales. The platform includes the multigrid finite element code FEMuS, which is based on a C++ main program that handles several external open-source libraries, such as the Libmesh and PETSc libraries. Libmesh is a C++ finite element library used in the code FEMuS to generate and handle numerical meshes with multiple level refinements [2]. PETSc is a C++ library for linear and non-linear algebra developed using LASPack codes written in Fortran and other solvers [3]. The FEMuS code contains solvers for many different physical problems.

The developed numerical platform is based on the open-source SALOME platform [4], using an approach similar to the NURESAFE platform, developed by the CEA [5]. We use the open-software SALOME platform to add new codes and develop coupling interfaces compatible with open and closed source codes. The SALOME platform brings several tools: KERNEL, GUI, GEOM, SMESH, MED, and PARAVIS module. The KERNEL module provides a common shell for all components, which can be integrated into the SALOME platform. The GUI module provides visual representation with basic widgets and the GEOM module draws and optimizes geometrical models. The SMESH module generates meshes on geometrical models previously created or imported by the GEOM component, PARAVIS performs data visualization and post-processing and finally MED allows to work with highly compressed files.

Several codes can be integrated into the platform, e.g. the open-source finite volume CFD software OpenFOAM [6], the multiscale neutronic code DONJON-DRAGON, and the system-scale code CATHARE used for thermal-hydraulic simulations have been successfully integrated into the platform. The platform has been tested on various physical problems. Some of the more significant applications have been implemented for the numerical simulation of Lead-cooled Fast Reactors (LFR), in a joint effort between ENEA and UNIBO, to simulate the thermal-hydraulics and neutronic behavior of a full reactor through the coupling between the codes integrated into the numerical platform.

The work is organized as follows. After a brief introduction on the SALOME platform, the coupling strategy between all the involved codes is introduced. Then, an example of numerical code coupling between FEMuS and OpenFOAM is discussed, providing guidelines for efficient and optimal data exchange.

2 FEMuS FEM LIBRARY

The numerical platform introduced in this work derives from the multigrid finite element library FEMuS. Such a library has been developed at the Department of Industrial Engineering of the University of Bologna, and contains several solvers for incompressible Navier-Stokes equations, heat transfer, turbulence models, Fluid-Structure Interaction problems, multi-phase flows and optimal control with adjoint method.

The FEMuS FEM library has been integrated with the SALOME platform by writing a MEDMem C++ interface to be able to couple the three-dimensional computation obtained with the finite element library with system codes (e.g. CATHARE, RELAP, etc.) or other CFD codes. The interface between the MEDMem and the FEMuS libraries consists basically of three classes: `FEMuS`, `EquationSystemsExtendedM` and `MeshExtendedM`.

All these classes are located in the `src` and `include` directories of the FEMuS library. The `FEMuS` class is the interface between the library FEMuS and the SALOME platform. The FEMuS interface allows passing commands from the MEDMem library to the `EquationSystemsExtendedM` class which is the problem solver core. To interact with the FEMuS library two existing FEMuS classes have been extended: the `MGEquationsSystem` and the `MGMesh`. The extensions are simply named as `EquationsSystemExtendedM` and `MeshExtendedM`, respectively.

The `EquationSystemsExtendedM` class, which uses only MEDMem functions, inherits the `EquationSystem` which uses only FEMuS functions. The `EquationSystem` class contains all the assembly and solver of the FEMuS code. The data from the `FEMuS` class can be transferred into the assembly routine by a dynamic cast operator, that allows to use *child* class functions from the *parent* class. Data can also be transferred in the opposite direction from the `EquationSystem` to the MEDMem interface by standard C++ inheritance rules.

The `MGMesh` class contains the multilevel mesh in FEMuS format. To interface FEMuS with MEDMem library a new mesh format should be introduced and the mesh class should be extended. The `MeshExtendedM` class, which uses only MEDMem functions, inherits the `MGMesh` which uses only libMesh functions [2]. In the FEMuS library, differently from LibMesh library, the mesh is known inside the `EquationSystemsExtendedM` class. Therefore, the interface `FEMuS` class does not need to communicate directly to the `MeshExtendedM` class. As mentioned above, the data from the `FEMuS` class can be transferred by using a dynamic cast operator into the assembly routine which is user-accessible.

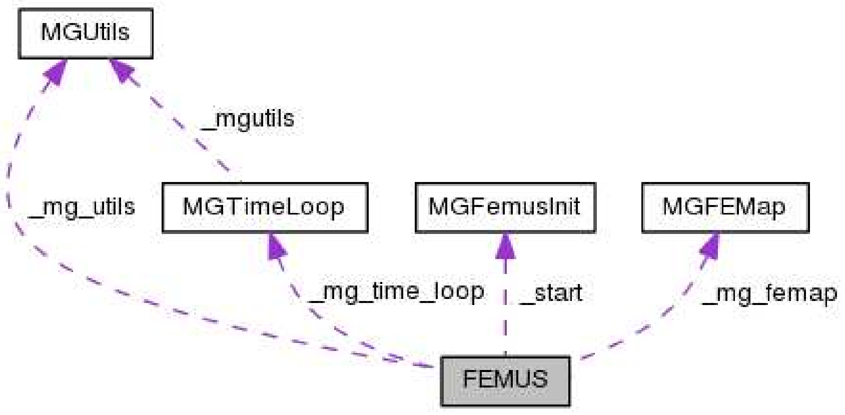

Figure 1: `FEMuS` class data to communicate with the FEMuS library.

The `FEMuS` class is the unique interface class. It contains data from both FEMuS and MEDMem libraries to transfer data from one mesh to another (e.g. from FEMuS mesh format, MGMesh, to the MED format). Thus, inside this class there are both pointers `_mg_mesh` and `_med_mesh` to their respective MGMesh and MED formats. The `FEMuS` class needs information to extract data from these different mesh structures. In particular, as shown in Figure 1, the classes `MGUtils`, `MGFEMap` and `MGFEMuSinit` are available inside FEMuS to transfer basic information such as file names, parameters, multigrid level, fem elements, etc. The `MGUtils` contains all the file names and input parameter values. The `MGFEMap` contains information about the finite element mesh structure and `MGFEMuSinit` class is the manager class for MPI and multigrid solvers.

3 THE NUMERICAL PLATFORM

The numerical platform SALOME introduced above has been developed by Commissariat à lénergie atomique (CEA) and Électricité de France (EDF) to provide an advanced open-source platform for Computer-Aided Engineering (CAE) purposes. The platform is composed of several modules, such as KERNEL, GEOM, MESH, PARAVIS, and MED, that can be used for integrating external codes. These modules have been previously described.

We focus now on the MED module, which provides a standard for storing and recovering computer data associated with numerical meshes and fields and facilitates the exchange between codes and solvers. This module is used to read a numerical field from file, and store it in the memory by using the MEDMem library. Moreover, the library provides a complete set of functions, used to access, modify, create meshes and perform operations on fields. Note that with the MED memory library data exchange between numerical codes can be performed at the memory level, avoiding read/write operations and data access on the hard disk.

3.1 SALOME interface

A numerical interface has been developed to couple all the involved codes inside the computational platform and to control their execution. In this section, we describe the main classes and functions involved in the developed SALOME interface. The interface can be divided into different classes. A *driver* class, belonging to the top level of the interface, has been implemented to communicate directly with the SALOME platform. The *problem* class contains three basic functions: `setType`, `setMesh` and `solve`. The `setType` command sets the physical problem to be solved for each involved code (e.g. Navier-Stokes, energy, turbulence, etc.). The `setMesh` command sets and prepares the mesh that should be available in both MED and code-specific formats for data exchange. The `solve` command controls the solution of the discrete system. The *equations* class inherits the system's particular class which contains the assembly and solver of the generic code. In this class, the solution is accessible, and a field can be extracted or set by using *methods* class with `setField` and `getField` functions. Finally, the *grids* class is an extension of the mesh class and allows the exchange of data between the code-specific format and the MED library. In Figure 2 we show the numerical scheme implemented to couple the two codes `Code 1` and `Code 2` in the developed platform. In particular, the two codes are communicating in a process where a numerical field is extracted from

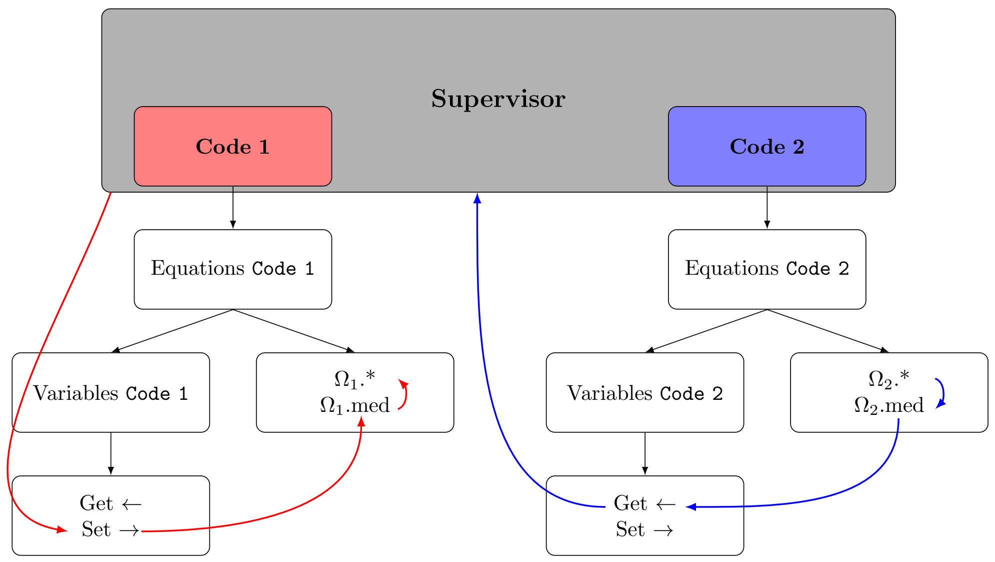

Figure 2: Example of coupling between two different codes (`Code 1` and `Code 2`) controlled by a supervisor.

`Code 2` and projected and used into `Code 1`. The solution is read from the mesh $\Omega_2.*$ of the `Code 2` and duplicated in MED format, then it is passed to the supervisor through the function `Get`. The numerical field is now available in computer memory as a MED memory object and can be passed to the `Code 1` using the function `Set`. The field is projected on a MED copy of the target mesh $\Omega_1.*$, i.e. Ω_1.med.

The presented functions `Get` and `Set` are based on code interfaces based on the MED format. In particular, an interface is created on a volume or boundary portion of the computational grid $\Omega.*$. A MED grid duplicate Ω.med of the considered mesh portion is created together with maps ω that allow associating nodes and cells numbering of $\Omega.*$ to nodes and cells numbering of Ω.med. For more information on the developed SALOME interface, the interested reader can consult [7].

3.2 Data manipulation and projection

The coupling between different codes can be implemented by using interpolation methods to project the field between different meshes. The used MED library offers several functions for data manipulation and interpolation. In particular, interpolation functions are provided for the piece-wise $P0$ and node-wise $P1$ fields. Thus, many algorithms are provided by the MED library to interpolate between $P0$ and $P0$, $P0$ and $P1$, $P1$ and $P1$. Moreover, we added a class for the node-wise interpolation $P2$. Therefore, it is now possible to interpolate between $P2$ and $P2$ fields. Moreover, a transformation of $P2$ fields into $P0$ fields has been implemented. We underline that the advantage of using MED data structures is that all the developed classes of the interface are not built for a specific code, but can generally be used wherever the MED library is used.

Node-wise field projection We illustrate now the node-wise field projection technique implemented to project fields between different codes. Let Ω_s and Ω_t be the source and the target mesh divided into $\mathcal{N}_s^e$ and $\mathcal{N}_t^e$ finite elements, respectively. Also, let ψ_s and ψ_t be the source and the target fields. The projection relies on a point search algorithm, in order to find the element Ω_s^e in the source mesh containing every node x_j^t of the target mesh. Then a reconstruction step is needed, aimed at locating the x_j^t node into the canonical element associated with Ω_s^e.

The interpolation takes place on the canonical element, so the value of ψ_t on node x_j^t can be determined through the value of ψ_s on the nodes of Ω_s and the corresponding interpolation weights. Therefore, if we define Ψ_s as the array containing the ψ_s values, and Ψ_t as the array containing the ψ_t values, we have

$$\Psi_t = |\boldsymbol{P}|\Psi_s\,, \tag{1}$$

where $|\boldsymbol{P}|$ is the projection operator. We define now the linear transformation that transforms the local canonical element into the original element as $\pi = \sum_{j=1}^{n_e} \boldsymbol{x_j^e}\varphi_j^e(\boldsymbol{\xi}_j^e)$, where $\boldsymbol{\xi}_j^e$ are the coordinates of the points in the local reference frame, $\varphi_j^e(\boldsymbol{\xi})$ are the approximation functions and n_e is the number of nodes that composes the e-th element. Thus, under the presented hypotheses, the projection operator can be defined as

$$\Psi_t = |\boldsymbol{P}|\Psi_s = \sum_{e=1}^{\mathcal{N}_s^e}\sum_{j=1}^{n_e} \Psi_{s,j}^e \varphi_j^e(\pi^{-1}(\boldsymbol{x}_j^{t,e}))\,. \tag{2}$$

Note that the building of the projection operator requires a searching algorithm to find the cell in the source mesh containing each node of the target one. We implemented a numerical search algorithm based on [8]. We use an iterative procedure to calculate the coordinates $\boldsymbol{\xi}$ of the selected node in the canonical element [9]. This allows assembling the projection operator $|\boldsymbol{P}|$, and then performing the projection (1). The described algorithm can be used to project numerical fields with both $P1$ or $P2$ approximations.

Galerkin field projection We also implemented the Galerkin node-wise field projection technique [10]. In this work, we report only a brief sketch of the Galerkin field projection implementation. For more information on the implemented algorithm in the numerical platform one can refer to [7].

The Galerkin technique relies on the fact that the interpolated field Ψ_t is the best approximation in L^2-norm, as

$$\|\Psi_s - \Psi_t\| = \min_{\Psi\in\text{span}\{\varphi_t\}} \|\Psi_s - \Psi\|_{L^2}\,.$$

With Galerkin projection method the target field is approximated through the weighted interpolation

$$\Psi_t(\boldsymbol{x}) = \sum_{j=1}^{\mathcal{N}_t} w_j(\boldsymbol{x}) \sum_{k=1}^{n_e^s} \Psi_{s,k}^e(\boldsymbol{x})\varphi_k^e(\boldsymbol{x})\,, \tag{3}$$

where $w_j(\boldsymbol{x})$ are the used weights. The source mesh element containing the j-th target mesh node has been labeled with superscript e, and it contains n_e^s nodes upon which the basis functions $\varphi_k^e(\boldsymbol{x})$ are defined.

In general, the pointwise technique is useful to project fields from a coarse mesh to a finer one. Conversely, the Galerkin technique is used to project fields from a fine mesh to a coarser one.

4 RESULTS

In this section, some simple numerical results are reported to show the effectiveness of the implemented coupling platform. In particular, the integration technique used to couple CFD and system codes is tested, as well as the projection algorithm to couple two CFD codes. Finally, a simple coupling between the FEMuS code and OpenFOAM is presented.

4.1 Coupling between system and CFD codes

The coupling between a system code and the FEMuS finite element library has been presented in past works, and the interested reader can consult [11, 12]. In particular, the FEMuS library has been coupled with the system code CATHARE to have multiscale simulations of complex systems. The data are transferred between the two codes while the program is running, with an efficient algorithm that does not involve reading and writing fields on external files.

The coupling between the two codes can be performed with many techniques. In most of them, the CFD code uses the system code variables as inlet values. To have a two-way coupling, the average value of the solution at the outlet of the CFD code's domain is used either as an imposed value in the system code (decomposed domain technique), or to calculate a fictitious source term to be imposed in the system domain to make the system code solution equal to the CFD one (overlapping domain technique, see [11]). Thus, we implemented a function able to build a MED interface on a desired portion of the domain, containing all the solution fields on it. Then, a function able to integrate all the variables on a given MED interface based on a classical Gauss integration has been developed as

```
Integrate( const MEDCoupling::MEDCouplingFieldDouble* Field, int order,
           int n_cmp, int first_cmp, int method,
           const MEDCoupling::MEDCouplingFieldDouble* VelField ) ,
```

where `Field` is the MED solution field on the specified interface and `method` can be `method = mean` for a classical integration divided by the value of the area, `method = aximean` for axisimmetric integrations, `method = bulk` for integrals weighed to the velocity field (with velocity `VelField`), etc. The user can integrate `n_cmp` solution components, starting from the component `first_cmp`, with interpolation order given by `order`.

To test this function, we consider a quadrangular domain $\Omega = \{x \in [0,1], y \in [0,1]\}$, and initialize an interface on the lower boundary of the domain (I_Γ) and an interface on the whole domain I_Ω. We impose some simple polynomial fields $\mathcal{F}(x,y)$ on Ω and compare the results of the `Integrate` function calculated both on I_Γ and I_Ω with the analytical results.

Table 1: Comparison between the numerical results obtained with the function `Integrate` and the analytical results.

I_Γ			I_Ω		
$\mathcal{F}(x,y)$	$\int_{I_\Gamma} \mathcal{F}(x,y)d\gamma$	A_{res}	$\mathcal{F}(x,y)$	$\int_{I_\Omega} \mathcal{F}(x,y)d\gamma$	A_{res}
x	0.5	0.5	$0.5(x+y)$	0.5	0.5
x^2	0.33333	0.33333	$0.5(x^2+y^2)$	0.33333	0.33333
x^3	0.25	0.25	$0.5(x^3+y^3)$	0.25	0.25
x^4	0.20052	0.20000	$0.5(x^4+y^4)$	0.20052	0.20000
x^5	0.16797	0.16667	$0.5(x^5+y^5)$	0.16796	0.16667

In Table 4.1 we report the numerical results of the integral carried out on both I_Γ and I_Ω, for various functions $\mathcal{F}(x,y)$. We also compare the numerical result with the analytical one, A_{res}. The numerical integrating algorithm shows the expected results since the second-order Gauss integration matches the analytical results for polynomials of order $n \leq 3$. In general, all the results are consistent with the expectations, thus the implemented function can be considered a useful tool to develop the integration of fields for all the CFD codes included in the numerical coupling platform.

This approach can be reproduced to pass information from the CFD code to the system one, and it allows an efficient code coupling. At the same time, the values can be passed from the system code to the CFD one by a developed function that allows imposing a fixed value on a certain MED interface.

4.2 Coupling between different CFD codes

The developed numerical platform provides the coupling between different CFD codes, as stated above. Since the information between the codes passes through the MED format, we can couple different numerical schemes (e.g. finite element, finite volume, etc.) or different element orders (e.g. linear and quadratic finite elements). To do so, we implemented various functions to extract a numerical field from a mesh, and project it into a copy of the source mesh in the MED format. Then, as reported in Section 3, we project the field into a copy of the target mesh in the MED format. Lastly, we project the field from the MED format to the target format. In particular, the projection of the field `srcField` between the two MED meshes is performed through the function

```
setFieldSource(int interface_name, int n_cmp,
               const MEDCoupling::MEDCouplingFieldDouble* srcField) ,
```

where `n_cmp` components are projected from the source MED mesh to the MED interface `interface_name`. Then, the values are projected from the MED format to the target mesh format with the function

```
write_Boundary_value(int interface_name, std::string mgsystem_name,
                     int n_cmp, int first_cmp) ,
```

where `mgsystem_name` is a string which identifies the physical system to be solved.

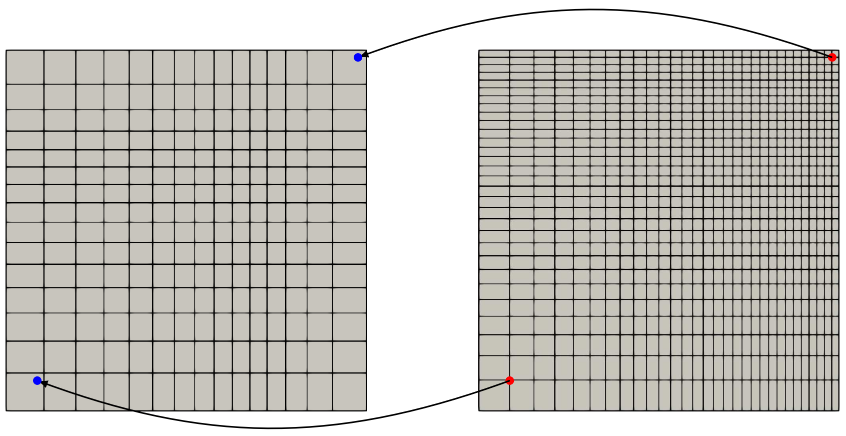

Figure 3: Source (left) and target (right) meshes used to the projection test. The mapping of the target nodes in the source meshes is also highlighted.

We test these functions by developing the projection of some simple fields between two different meshes. In Figure 3 we report the source $\mathcal{T}_s$ and the target $\mathcal{T}_t$ meshes used to test the coupling algorithm, defined on the domain Ω. We also report the reverse mapping of the nodes of the target mesh. In particular, we find the source cell containing every target node (in red in Figure). This search algorithm is the basis of the data projection described in section 3.2.

We impose different fields $\mathcal{F}(x, y)$ on $\mathcal{T}_s$, and we project them on $\mathcal{T}_s$. In particular, we consider a quadratic case $\mathcal{F}_1(x, y) = (2x - 1)^2 + (2y - 1)^2 + 0.5$, a quartic case $\mathcal{F}_2(x, y) = (2x - 1)^4 + (2y - 1)^4 + 0.5$, and a sinusoidal function $\mathcal{F}_3(x, y) = 2 - 2\cos(\pi x)$.

In Figure 4 we report the projection between the two presented meshes in the sinusoidal case. Note that the projection algorithm projects the field correctly. To analyze the correct field projection, we report in Table 4.2 the L^2-norm of the solution calculated over Ω in all the tested cases. We also report the percentage error between the source and the target fields. Note that the L^2-norm of the target fields matches the L^2-norm of the tested source fields, and all the errors are smaller than 1%.

Therefore, the function implemented for the CFD code coupling works properly for the field projection in all the simple cases tested. The implemented functions are designed to project fields between different finite-element and/or finite-volume codes.

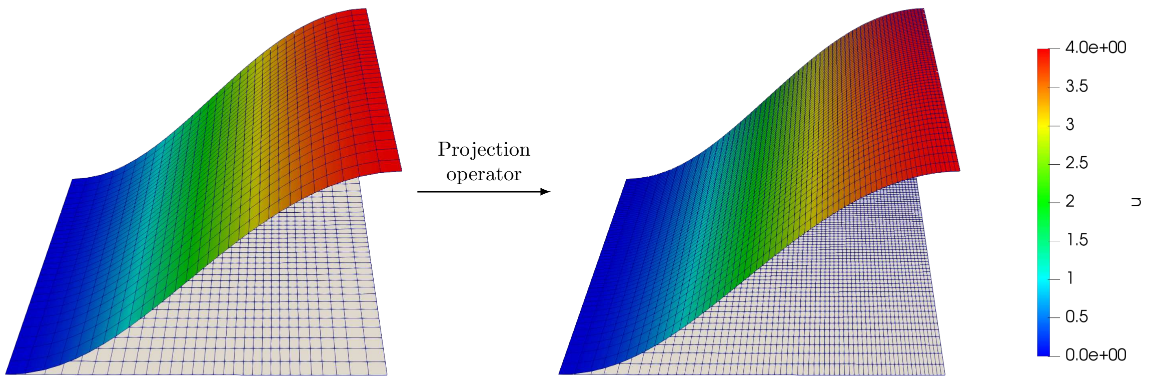

Figure 4: Sinusoidal projected field $\mathcal{F}_3(x,y)$ between $\mathcal{T}_s$ and $\mathcal{T}_t$.

Table 2: L^2-norm of the source and target solution in all the tested cases. The percentage error is also reported.

$\mathcal{F}(x,y)$	$\|u\|_s$	$\|u\|_t$	$\varepsilon_\%$
$\mathcal{F}_1(x,y)$	0.9856	0.9795	0.62%
$\mathcal{F}_2(x,y)$	1.2437	1.2417	0.16%
$\mathcal{F}_3(x,y)$	2.4497	2.4492	0.02%

4.3 Coupling FEMuS-OpenFOAM: natural convection in a squared cavity

We now briefly report the numerical results presented in [7, 13] to illustrate the numerical coupling between the finite element platform FEMuS and the finite volume code OpenFOAM. We consider the laminar Navier Stokes system of equations, with Boussinesq approximation for the buoyancy term, composed by the continuity equation $\nabla \cdot \boldsymbol{u} = 0$, the momentum equation, given by

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = -\nabla P + \nabla\cdot\left[\mu\left(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^T\right)\right] + \underbrace{\rho\boldsymbol{g}\beta(\tilde{T} - T_{ref})}_{coupling\, term}, \tag{4}$$

and the temperature equation $\rho c_p\left(\frac{\partial T}{\partial t} + \boldsymbol{u}\cdot\nabla T\right) = k\Delta T$. We consider the domain $\Omega = \{x \in [0,L], y \in [0,L]\}$, and we impose no-slip velocity boundary conditions on the bottom, right and left walls, and tangent velocity condition on the top wall. For the temperature field, we impose insulation boundary conditions on the top and the bottom walls, $T = T_c$ Dirichlet boundary condition on the left wall, and $T = T_h$ Dirichlet boundary condition on the right wall.

The temperature field is calculated with OpenFOAM, then it is projected to FEMuS for the buoyancy force term calculation. The results are presented in terms of non-dimensional variables, namely $x^+ = x/L$, $y^+ = y/L$, $u^+ = \frac{\rho c_p}{k}uL$, $v^+ = \frac{\rho c_p}{k}vL$, and

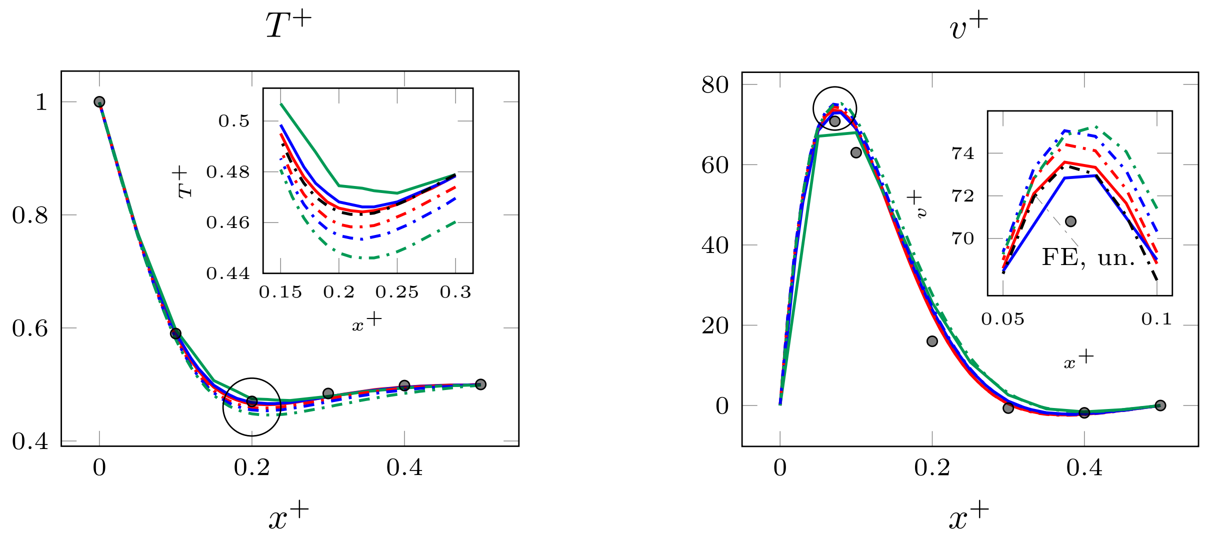

Figure 5: Non-dimensional temperature T^+ (left) and velocity v^+ (right) profiles along $y^* = 0.5$.

$T^+ = (T - T_c)/(T_h - T_c)$.

In Figure 5 we report the non-dimensional temperature T^+ and velocity v^+ profiles along the line $y^* = 0.5$. Solid lines stand for OpenFOAM solutions,while dash-dotted lines stand for the solution of the coupling between FEMuS and OpenFOAM. The different used meshes can be identified by the line color: green for the 20×20 coarse mesh, blue for the 40×40 mesh, and red for the 80×80 fine mesh. Dots represent reference values taken from [14].

The presented results show that the coupling between the two codes works properly, and the temperature field is successfully passed from OpenFOAM to FEMuS. The obtained numerical fields match the standalone OpenFOAM results and the reference results.

5 CONCLUSIONS

In this work, the numerical platform FemusPlatform has been presented. The platform allows the coupling of several different codes, ranging from one-dimensional system codes to finite element and finite volume CFD codes, in a unified framework. The SALOME platform has been used as a central tool to develop all the routines and classes for the projection and integration of fields through the MED format. The platform can be used for complex multiscale and multiphysics numerical simulations, combining different codes.

The presented preliminary results show that the implemented routines for the integration and projection of numerical fields work properly. Finally, a simple example of numerical coupling between the finite element code FEMuS and the finite volume code OpenFOAM has been reported, showing the proper functioning of the coupling process.

REFERENCES

[1] FemusPlatform `https://github.com/FEMuSPlatform`

[2] Kirk B. S., et al. *libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations.* Engineering with Computers 22.3-4 (2006): 237-254.

[3] Balay, Satish, et al. *PETSc users manual.* (2019).

[4] Bergeaud V. and Lefebvre V. *SALOME. A software integration platform for multiphysics, pre-processing and visualisation.* (2010).

[5] Chanaron B. *Overview of the NURESAFE European Project.* Nuclear Engineering and Design 321 (2017): 1-7.

[6] OpenFOAM project. `URL:https://www.openfoam.com/`

[7] Da Vià R. *Development of a computational platform for the simulation of low Prandtl number turbulent flows.* (2019).

[8] Löhner R. *Robust, vectorized search algorithms for interpolation on unstructured grids.* Journal of computational Physics 118.2 (1995): 380-387.

[9] Silva, G.H.C., et al. *Exact and efficient interpolation using finite elements shape functions.* European Journal of Computational Mechanics 18.3-4 (2009): 307-331.

[10] Farrell P. E. and Maddison J. R. *Conservative interpolation between volume meshes by local Galerkin projection.* Computer Methods in Applied Mechanics and Engineering 200.1-4 (2011): 89-100.

[11] Chierici A., et al. *A multiscale numerical algorithm for heat transfer simulation between multidimensional CFD and monodimensional system codes.* Journal of Physics: Conference Series. Vol. 923. No. 1. IOP Publishing, 2017.

[12] Cervone A., et al. *Validation of a multiscale coupling algorithm by experimental tests in tall-3D facility.* 6th ECCM and 7th ECFD ECCOMAS 2018 conference, 2020.

[13] Da Vià R, et al. *Natural convection in a squared cavity via a numerical coupling between a FEM code and OpenFOAM.* AIP Conference Proceedings. Vol. 1978. No. 1. AIP Publishing LLC, 2018.

[14] Wan C., et al. *A new benchmark quality solution for the buoyancy-driven cavity by discrete singular convolution.* Numerical Heat Transfer: Part B: Fundamentals 40.3 (2001): 199-228.