Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Elmer Adapter for Precice

Hisham Saeed

Computational Science and Engineering (International Master's Program)

Technische Universität München

Master's Thesis

Elmer Adapter for Precice

Author:	Hisham Saeed
1st examiner:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Assistant advisor:	M.Sc. Benjamin Rodenberg
Submission Date:	October 15th, 2021

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

October 15th, 2021 Hisham Saeed

Acknowledgments

Sincere gratitude to all people who helped me and who was necessary to make this work possible.
First of all i would like to thank my supervisor Benjamin Rodenberg who was very supportive and patient and always encouraging to help through obstacles, not only on the professional level but on apersonal level as well, i really appreciate his support to me through my work.
Secondly, i feel very grateful for my supervisor at my student job Wolfram Klein, he is very helpful person and put the student interest before the interest of the job, he always wanted me to learn.
Also deepest gratitude to my beloved mother, who supported me always through my education.
Last but not least, I would like to thank my family and friends, especially my colleague at CSE Amr ElSharkawy, without him it would not be possible to finish my master's.

Abstract

Multiphysics simulation is a widely spread field, and it has many important applications in engineering and real science. it allows scientists and engineers to predict certain phenomena and their effect in a relatively precise way, such as to help in designing sophisticated products, or analyzing such natural phenomena to understand its events(physics) and take precaution against it; for example wheather simulation as well stress analysis simulation can both be used to predict system failure in the automotive and aerospace industry. One of the powerful tools designed for these is Elmer Multiphysics simulation software.It was mainly developed by CSC IT Center for Science in Finland for Glacier simulation. it has other powerful areas ranging from heat conduction to acoustics simulation. One motivation for simulation engineers and scientists to use it is the crystal growth simulation by Leibniz Institute for crystal growth. But as any software, it has limitations. Competitors software could be more efficient in other applications, so the advantage is to couple different solvers together; that is where the power of preCICE coupling library comes in. preCICE is a coupling which is an API that allows users to couple different solvers solving the same problem but in different parts of the domain. Furthermore, it allows the user to steer the simulation and exchange data between the solvers, but in order to use any solver with preCICE, an adapter must be developed for the intended solver. The aim of this thesis is to develop an Elmer adapter for the preCICE coupling library.

Contents

Part I.

Introduction and Background

1. Introduction

In the current field of research and development,product development and engineering, simulation had become an indispensable tool. Since the cost of expermients became very high as well as the challenges that arise for study physical phenomenas made some expermients unfeasible. Such a demand requires improvement in mathematical and software solutions to be able to face the challenges in different industries. Recently,simulation has played a vital role in many applications ranging from Electronic Engineering to molecular dynamics.

To be able to utilize all the available resources, coupled simulation is a great solution to use different numerical solvers hence taking the benefit of each solver in its field. For example, one of the major applications for coupled simulation is fluid structure interaction, so it requires the study of fluid flow as well as linear elasticity models. Each model can be tackled differently with different mathematical tools. Also coupled simulation gives the chance to use legacy code, in the sense of making use of old powerful softwares without having the need to use for the whole problm in hand while it can only be used to solve one part of the problem. This wher comes the power of preCICE, a coupling library that allows performing coupled simulation using different solvers, by means of a black box coupling. But in order to be able to use preCICE with any solver extra code has to be developed which called an adapter to allow using preCICE. preCICE has existing adapters for several open source and comercial softwares [2]. One of these software lacking an adpater to use preCICE is Elmer multiphysics simulation. It was mainly developed by CSC IT center for science in Finland for Glacier simulation. The main motivation for developing an adapter for Elmer is crystal growth simulation by Leibniz Institute for crystal growth [4]. Crystal growth is a very important process for crystalline materials since it is included in several modern industries, where the silicon enters the semiconductor technology. The crystal growth process contain an important phase which is the melt of the raw material to reach crystalline phase[4]. Such process can be modeled by fluid flow simulation, it also includes phase change simulation. The research team for the crystal growth used openFoam and aims to use Elmer and couple it with different solver whether it is open source or comercial. The chose coupling tool is preCICE, and in order to be able to use it an adapter has to be developed which is the aim of this thesis; Elmer adapter for preCICE development. The thesis structure is as follows; first a quick overview of the softwares to be used, its feature,usability and limitations. Then a description of the development process, the exsisting challenges for the development and the structure of the adapter as well as its usage. At the end, the study problems to test the adapter, the results and the limitations of the adapter. And finally an outlook of future work.

2. preCICE Overview

As mentioned in the introduction multiphysics simulation is a widely spread field that has a great impact on many applications. In order to perform such simulations several softwares are developed which called solvers to perform several mathematical computations to reach desired result. Such simulations could be monolithic or partitioned. Monolithic means that all the domain including parameters, governing equations and mesh are included in one program, while the partitioned simulation, the desired domain is splitted into 2 or more parts and a program operates on each part(could be the same program but with different parameters). This is were preCICE comes in. preCICE (Precise Code Interaction Coupling Environment) is an open-source library for surface coupled partitioned multiphysics simulations. Similar to any industry, different softwares provide different features, or may be emphasised on specific domains, so preCICE gives the user the ability to utilize the benefits of different softwares to perform a multiphysics simulation. In addition, it does not require modification of the software performing computation on different part of the domain, as well it requires from the user minimal knowldge on internal dynamics of the used softwares. All it is required is to define a coupling interface and make the simulation parameter consisting in different solvers(simulation softwares) and preCICE will take care of the task of communication and data transfer. Such a coupling is referred as a black-box coupling in the literature. preCICE is a coupling library, not a software, so it provides users, developers several function calls to be able to handle the required task for coupling. Such necessary tasks are

- communication
- data transfer
- coupling schemes

To be able to perform this task, additional code has to be written to handle such function calls, this is exactly the role of the adapter, which the aim of this thesis is to develop. Several solvers can be coupled together as long as an adpater developed for these solvers to perform the necessary tasks for coupling.

preCICE itself is run by a config file in xml format, it has to be provided at initialization phase, explained more in detail in chapter 3. This overview will focus on the features used in this thesis, for other advanced topics it is recommended to refer to preCICE documentation and tutorials. The main topics to be focused on are the coupling scheme, communication and important API calls.This is important because the developed adapter is not complete and does not support all the features provided by preCICE.

- **Coupling Scheme:** The coupling schemes are very important ascpects, because it affects to some extent the design of the adpater as will be later shown in chapter 3. there are 2 important parts of coupling, one concerned about the sequence of running the solvers and other part is concerned about the quality of the coupled data at the coupling interface. The first part regarding the sequence of solvers has 2 coupling schemes serial coupling and parallel coupling.
  - serial coupling: This type of coupling includes of alternating roles in making computation, in the sense that one solver starts while all other solvers having coupling interface with this solver are waiting the running one to finish. Then the newly calculated data is transfered to the waiting solvers and then the solver that has its turn starts computation until all solvers finishes. Then the time step is advanced. Such a scheme is shown in Fig. 2.1a. In this case only the first participant uses data from previous iteration while second participant uses data computed in the current iteration.
  - parallel coupling: Here the coupling scheme is similar to parallel programs, such as both participant runs in the same time and perform computation in parallel, no particpant wait for the other.then when the computation is done a sync point is reached to advance to next time step or repeat current time step. this will be explained in the other part of coupling scheme.parallel coupling is shown in Fig. 2.1b

  And with respect to coupling quality, there are 2 other schemes; namely explicit and implicit coupling. This part of coupling is concerned about the continuty of the solution across the coupling interface.
  - explicit coupling: In explicit coupling data is exchange across the coupling interface without putting into account the solution convergence, so the solution can be iterated over it a prescribed number of times per time step. The method used is conventional staggered scheme, the method is shown in alg. 1, it can be noticed that the iteration is stopped after certain number of iteration, even if the data is not continuous enough across the coupling interface.

Algorithm 1 conventional staggerred

for $n = 0$ to n_{end}
  solve $F^n(s^n) = f^{n+1}$
  solve $S^n(f^{n+1} = s^{n+1})$

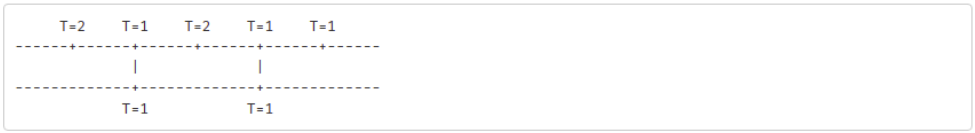
  - implicit coupling: The implicit coupling is different from the explicit coupling as it takes the continuty across the interface seriously, and try to reach acceptable data continuty across the coupling interface iteratively base on a prescribed tolerance. So the implicit coupling tend to get a solution close to the monolithic case where the problem is solved without coupling, but it requires more com-

putation, also it could lead to certain instability, therfore additional techniques are used to reach solution stability. Such techniques will not be disscussed here since it has no use in our study which will be shown in later chapters. Just to mention these techniques are Newton's method and Schwarz procedures both are data post-processing techniques.

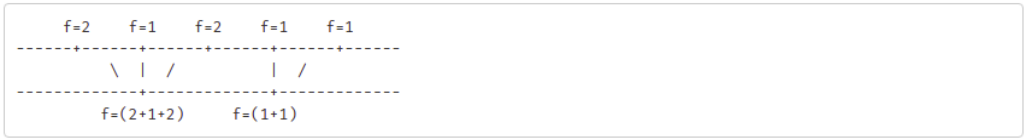
(a) Serial Coupling (b) Parallel Coupling

Figure 2.1.: Difference between serial and parallel coupling, image from [1]

Data Mapping: One of the major features of preCICE is that it gives the user low coupling which allow the user to have a totaly different simulation conditions in the coupled solvers. This low coupling allow the user to have different granularity levels in different part of the domain, it even allows to have different type of meshes(e.g. one domain could have rectangular mesh while the other has triangular mesh), and during data communication preCICE deal with this difference by mapping data from one mesh to another. There exist mapping types and mapping methods, concerning mapping types, there exist 2 types of mapping; consistent and conservative. Consistent mapping means that data at a certain point is transmitted to the same node without any contribution of the neighbouring nodes as shown in Fig. 2.2a, image from [1].Conservative mapping takes into account the neighbouring nodes and gives it a weight while restricting from fine to coarse mesh as seen in Fig. 2.2b, image from [2].



(a) consistent mapping



(b) conservative mapping

Figure 2.2.: Difference between consistent and conservative mapping

Regarding the methods for data mapping, preCICE provides three methods; nearest-neighbor,nearest-projection and radial-basis function. nearest-neighbor is the simplest

[1] https://precice.org/configuration-mapping.html
[2] https://precice.org/configuration-mapping.html

```
turnOnSolver(); //e.g. setup and partition mesh

double dt; // solver timestep size

while (not simulationDone()){ // time loop
          dt = beginTimeStep(); // e.g. compute adaptive dt
          solveTimeStep(dt);
          endTimeStep(); // e.g. update variables, increment time
}
turnOffSolver();
```

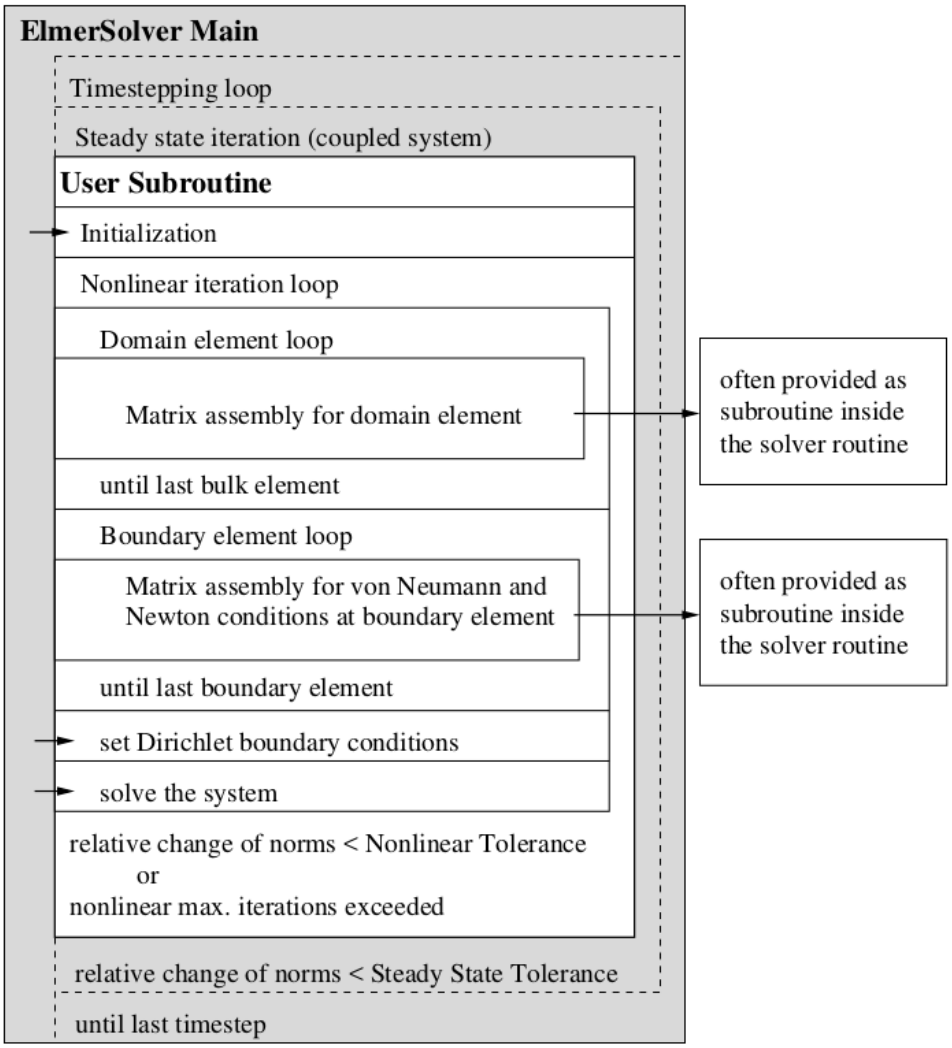
Source Code 2.1.: Solver Structure

way for data mapping as it takes the value of the nearest node and transfer that data.Such a method is simple but cause numerical instability or inaccuracy.Nearest-projection makes some modification to the previous method as it computes the value of the node to be transfered through linear interpolation, it still simple and fast and gives better numerical result, but it requires knowldge of the mesh where the data is transfered to. Last method is radial-basis mapping, this method generate an inerpolated function for the coupling interface and evaluate the required nodes to be transfered, this is much more accurate method but requires more calculation. There are several techniques to compute the interpolant, which will not be discussed here since it is not used in our study, for more details refer to precice reference [3].

API Calls: The internal structure of most solvers contain a time loop and partial differential equation solver which might contain another loop for solving linear or non-linear system of equations. This structure is depicted in source code 2.1 code from [3] where it explains the general idea of the solver, it is also realizd in Elmer as shown in Fig. 2.3. To be able to couple different solvers additional calls has to be introduced to the solver in hand to provide communication and data transfer as well as controlling the time loop. Controlling the time loop in case of serial coupling means prohibiting one solver to advance in time to the next time step unless the other participant has finished computation and did send the newest data. In implicit coupling controlling time loop goes even further by forcing to repeat the current time step computation with newly calculated data but this requires more tricky ways to interfer with the solver, in this case it depends on solver and what it porvides.

Source code 2.5 is snippet code of a tutorial called solver dummies provided by Elmer developers, for the complete code check [4], but for just introducing the required preCICE API calls, it is enough, there exist tutorials like this written in c and c++ but here fortran

[3] https://precice.org/couple-your-code-preparing-your-solver.html
[4] https://github.com/precice/precice/tree/develop/examples/solverdummies/fortran



(a) Elmer internal loops

Figure 2.3.: Elmer solver structure, image from [9]

tutorial is presented since Elmer is written in FORTRAN and adapter is developed using FORTRAN as well, that is why also the preCICE call will be presented in FORTAN syntax provided by FORTRAN bindings offered by preCICE team [5]. In source code 2.5 the while loop represents the time loop inside the finite elment solver(Elmer in our case), before the time loop there are some setup and initalization to be done. First, preCICE creation, acquiring data about couping interface, then check if data needs to be initialized and initialize if

[5] https://github.com/precice/precice/tree/v2.0.2/extras/bindings/fortran

```
precicef_create(CHARACTER participantName(*),CHARACTER configFileName(*),
                INTEGER   solverProcessIndex,INTEGER   solverProcessSize )

precicef_initialize( DOUBLE PRECISION timstepLengthLimit )

precicef_get_dims( INTEGER dimensions )
precicef_get_mesh_id(CHARACTER meshName(*),INTEGER   meshID )

precicef_set_vertices(INTEGER meshID,INTEGER size,
                DOUBLE PRECISION positions(dim*size),INTEGER  positionIDs(size) )

precicef_intialize_data()
```

Source Code 2.2.: preCICE calls required for initialization

```
precicef_finalize();
precicef_ongoing( INTEGER isOngoing )
```

Source Code 2.3.: preCICE calls required for controlling simulation

required, the required calls are presented in source code 2.2. Then during the time loop, preCICE have to check whether the simulation is finished or not and if it is finished it ends the simulation, clean memory by destroying internal data structures, this is done by preCICE calls shown in source code 2.3. And within the time loop it needs to communicate data, checking if data availble before communicating it and advance time step as noticed in 2.4, the mark action fullfiled has no usage here, it is used for implicit coupling.

```
precicef_advance( DOUBLE PRECISION timstepLengthLimit )

precicef_read_data_available( INTEGER isAvailable );

precicef_write_bvdata(INTEGER dataID,INTEGER size,
        INTEGER valueIndices,DOUBLE PRECISION values(dim*size) )

precicef_read_bvdata(INTEGER dataID,INTEGER size,INTEGER valueIndices,
         DOUBLE PRECISION values(dim*size) )

precicef_mark_action_fulfilled( CHARACTER action(*) )
```

Source Code 2.4.: preCICE calls required for data communication

```
    CALL precicef_create(participantName, config, rank, commsize)

    ! Allocate dummy mesh with only one vertex
    CALL precicef_get_dims(dimensions)
    CALL precicef_get_mesh_id(meshName, meshID)
    CALL precicef_set_vertices(meshID, numberOfVertices, vertices, vertexIDs)

    CALL precicef_get_data_id(readDataName,meshID,readDataID)

    CALL precicef_initialize(dt)
    CALL precicef_is_action_required(writeInitialData, bool)
    CALL precicef_initialize_data()

    CALL precicef_is_coupling_ongoing(ongoing)
    DO WHILE (ongoing.NE.0)

      CALL precicef_is_action_required(writeItCheckp, bool)

      IF (bool.EQ.1) THEN
        CALL precicef_mark_action_fulfilled(writeItCheckp)
      ENDIF

      CALL precicef_is_read_data_available(bool)
      IF (bool.EQ.1) THEN
        CALL precicef_read_bvdata(readDataID, numberOfVertices,
         vertexIDs, readData)
      ENDIF

        CALL precicef_write_bvdata(writeDataID, numberOfVertices,
        vertexIDs, writeData)

      CALL precicef_advance(dt)
      CALL precicef_is_coupling_ongoing(ongoing)

    ENDDO

    CALL precicef_finalize()
```

Source Code 2.5.: Solver dummies

3. Elmer Overview

This chapter aim is to give a background of the software used in this thesis, how they work and the files required for using it.As mentioned in the introduction this thesis aim to develop an Elmer adapter for the preCICE coupling library. So in this chapter Elmer solver will be introduced and explain its usage, also extra tools provided by Elmer developers are introduced.

Elmer comes with some extra tools to help prepare for the simulation. Also it supports other sveral tools problem setup as well as post processing. First a short overview of Elmer is given, how to use it and how to setup the problem for the simulation. The main three softwares provided by Elmer used in this thesis are ElmerSolver,ElmerGUI and ElmerGrid. ElmerSolver is the solver software and it needs 2 category of files, the first category is the mesh files which will be elaborated in more details and the second category is simulation conditions, the file is called system input file. Mesh Files can be genreated ElmerGrid, ElmerGUI can generate both categories, because it embbedes a mesh generator,either ElmerGrid or another built in generator.

3.1. Elmer Solver

Elmer solver is a multiphysics software developed mainly by CSC IT Center for Science in Finland for Glacier simulation.It uses Finite Elment method to perform numerical simulation. It is a pretty powerful software that provides support for performing simulation in different multiphysics domain. This is done by providing a wide range of modules that covers several physical models. These models are illustrated in Elmer Models Manual [7]. It has a lot of application with respect to eigen value problems, fluid mechanics, transport phenomena such as Navier-stokes equation, it also has models for acoustics field like Helmholtz equation for wave modeling. Regarding the computation Elmer can solve both steady-state simulations as well as transient simulations. With respect to transient simulation it provides 2 methods for discretizing time, the Crank-Nicolson or Backward Differences Formulae(BDF).The BDF scheme has an order up to 5.For more detail about how time discretization is done please refer to [9]. As mentioned, Elmer requires the mesh files and system input files. Elmer expect mesh files in a specific format. It expects 5 files with name mesh, and extensions as follows .boundary,.element, .header, .names, .nodes . the .boundary file contains information about the boundary all elements that has nodes on the boundary exist in this file, to track which elements are associated

```
e1 bndry p1 p2 type
5 4 33 0 202 1 12
6 4 41 0 202 12 13
7 4 49 0 202 13 14
8 4 57 0 202 14 15
9 4 65 0 202 15 16
```

Source Code 3.1.: mesh.boundary

with which boundaries, as well as the type of elements, example of boundary file in source code 3.1 is a snippet coresponds to mesh in Fig. 3.1. In the picture the red line represents boundary number 5, which is the second integer in the format bndry, above that boundary elements with number 33,41,49 and so on, which p1 identifier, the third integer p2 is zero because the boundary has elements only one side, if it is between 2 elements this attribute will hold the second element number. the type is the type of element, 202 describes linear element in 2D, for more detail on elements types refer to Elmer Solver Manual [9] . The first integer is just an identifier for the boundary element. The second file is .header holds information about the whole mesh. As seen in source code 3.2, it shows the number of nodes, elements and boundaries. Also the type of existing elements which 2 types elements on the boundary 202, are linear and elements in the inner domain are 303.the .node file seen in source code 3.3 consist of the meshing just gives node number and its coordinates, the second integer for parallel runs.The .element is for element meshing as depicted in source code 3.4 gives an identification number for the element, the second integer states which body it belongs to,the third integer describes the element type, then it list the nodes belonging to that element. The second type of file is the system input file with an extension (.sif), the file describe the problem and the simulation settings. it is a text file that consists of a certain syntax and keywords

Figure 3.1.: Mesh Boundary

```
nodes elements boundary-elements
nof_types
type-code nof_elements
type-code nof_elements
165    256    72
2
202    72
303    256
```

Source Code 3.2.: mesh.header

```
n1 p x y z
1 -1 0 0 0
2 -1 1 0 0
3 -1 1 0.5 0
4 -1 0 0.5 0
5 -1 -0.5 0 0
6 -1 -0.5 0.5 0
```

Source Code 3.3.: mesh.nodes

```
e1 body type n1 ... nn
1 1 303 5 9 72
2 1 303 72 9 73
3 1 303 72 73 71
4 1 303 71 73 74
5 1 303 71 74 70
```

Source Code 3.4.: mesh.element

3.2. Elmer GUI

One of the files required by ElmerSolver is the system input file (.sif) that describes the simulation, the solver, the boundary conditions,etc.. , this system input file can be generated manulay using a text editor, knowing the format and the geometry in hand, but this is slightly difficult if the user is unaware of the format, and gets more difficult with complicated geometries in 3D, so Elmer developers provide a tool called ElmerGUI which is a graphical user interface, that has the ability to import mesh files of different format, define the simulation conditions, the solvers required from linear systems solvers to nonlinear. It is also linked to mesh generators like ElmerGrid explained in the next point as well extra mesh generator like Netgen which is free software for mesh generation that operates on certain format. Also it has the ability to call ElmerSolver directly, so the user can start the simulation from this GUI interface not from the command line, it als has the ability to call automatically paraview for postprocessing and visualization. One of the features of ElmerGUI, is good display of the mesh and its boundaries, which is very helping in debuging your problem setting, this helped a lot during adapter development.This allowed to preview the mesh nodes and elements as shown in Fig. 3.2

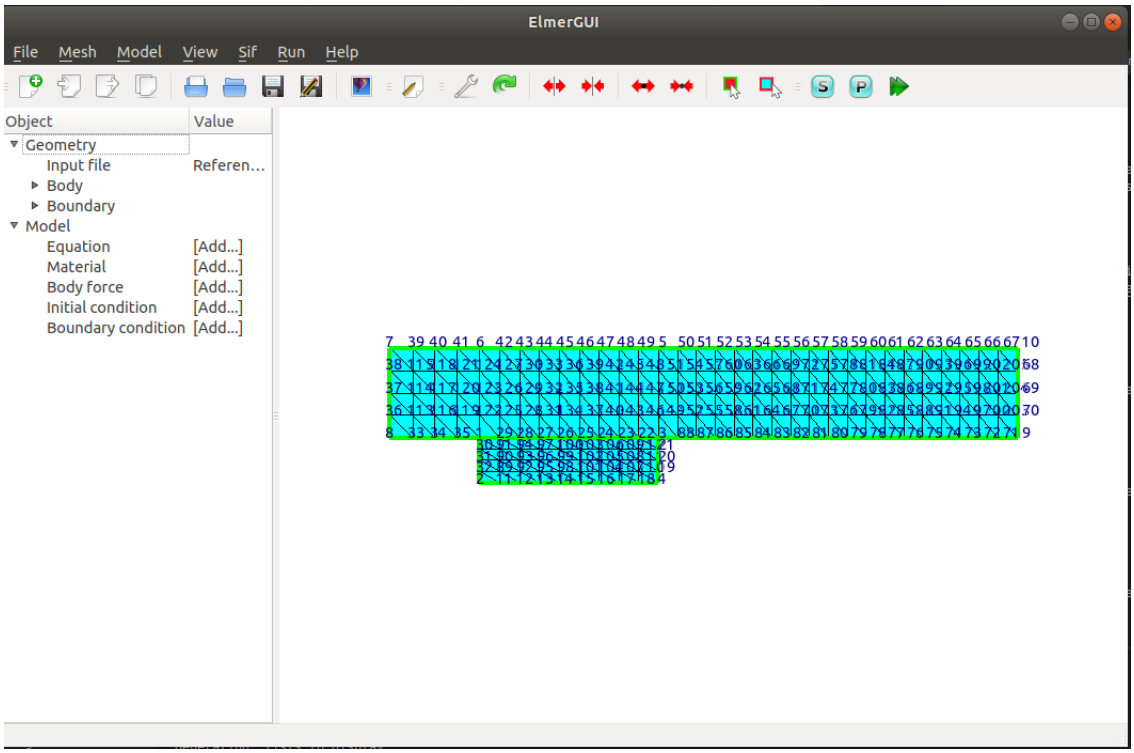


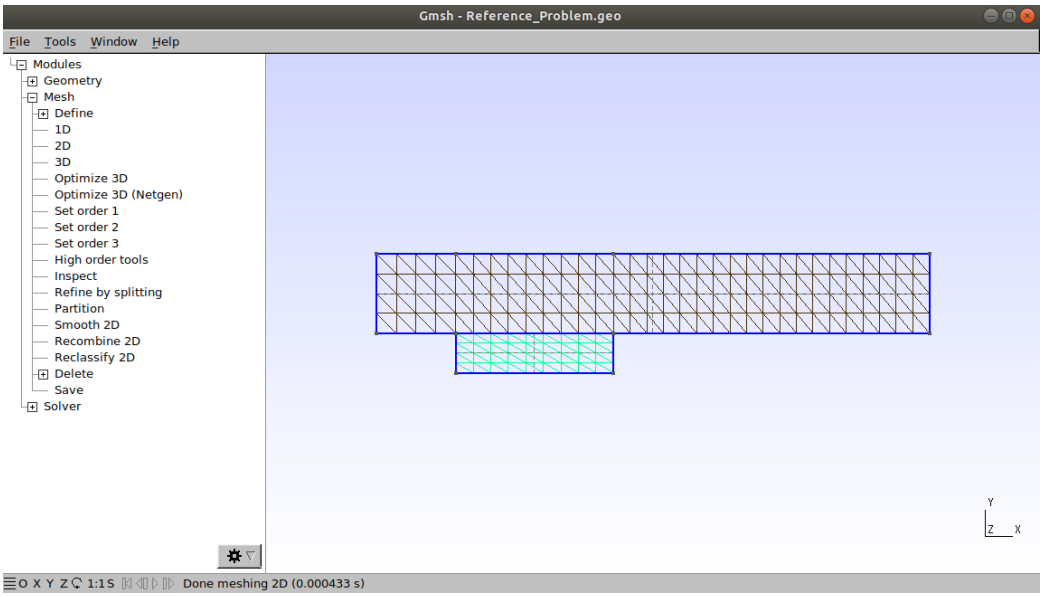
Figure 3.2.: ElmerGUI example

3.3. Elmer Grid

To be able to perform a simulation using Finite Elment tool like Elmer, it is required to define a mesh for the software to make computation for it.A mesh is simply a discretization of the continuous domain into points to compute required values at such points. In finite element method, the mesh is formed from points called nodes and elements where points are the vertices of these elements, these elements could be 2D or 3D. So to provide a mesh for Elmer solver, mesh files have to be provided in the exact format that Elmer accepts, or a different format using other solvers. Elmer developers provide an extra tool which called ElmerGrid, it is a preprocessing tool. It is a tool for generating structured meshes for 2D or 3D geometries. It also provides mesh format manipulation, so it can change the format generated by different mesh generator software to the required format by Elmer. So ElmerGrid is not a tool for defining the required geometry but only for mesh generation, but Elmer developers provides a certain ascii syntax that can be used to define geometry, the file comes with extension .grd, but this is difficult to learn and not very user friendly, so for geometry and mesh generation another tool called gmsh is used, and ElmerGrid is used for mesh manipulation to generate the format required by ElmerSolver.For more details about ElmerGrid or its .grd format please refer to ElmerGrid manual [6]. The format of the mesh generated by gmsh is .msh and the format required by ElmerSolver is .mesh.* as explained in ElmerSolver mesh files. To execute the transformation,the inline mode is used which a command line is typed in the terminal, the expected command line is 'ElmerGrid 14 2 filename', first item is calling ElmerGrid, second item is an integer refering to input format which in our case .msh, the third item is the output format which is .mesh.*, the lat item is the file fo the mesh to be transformed. For more details about the numbers that describes the format please refer to the Elmer manual [9].

3.4. GMSH

For preparing the problem of the simulation a mesh has to be generated, hence a geometry must be defined and created. As mentioned earlier in the ElmerGrid explaination, another softwore other than ElmerGrid will be used for both geometry and mesh generation which is gmsh. Gmsh is an open source software for 3D finite element mesh generation, it operates using an ASCII text file, where Gmsh scripting language is defined, gmsh has its own scripting language as well as light built in CAD engine. It is pretty good for mesh generation and for simple geomerty creation, but when it comes to complicated geometry it is recommended to use a different CAD software, although Gmsh can handle it. The mesh generation can be seen also from Fig. 3.3



(a) gmsh example

Figure 3.3.: gmsh example

3.5. Fenics And OpenFoam

As stated, the aim for this thesis is to develop an adapter for different simulation software coupling with Elmer, there are many free and comercial solver available but the chosen ones for our study is FEniCS and openFoam softwares. FEniCS is a collection of open source software components to help solve differential equations. it contains collection of libraries mainly a library called dolfin, to help give a program manipulated model of partial differential eqution and its weak form for the solving the equation using finite element method. So it does not contain modules like Elmer for different application(e.g. heat equation,Navier-Stokes equation, linear elasticity, etc..). The user has to write some code to solve the desired program. FEniCS gives the user a high-level Python and C++ interfaces to its libraries. OpenFOAM is also an open source software for computing. It has an extensive range from complex fluid flows to solid mechanics, it also can handle acoustics and cheemical reactions. But it is mainly a CFD(computational fluid dynamics) software, so for our study it will be chosen to simulate fluid flow. OpenFOAM does not use finite element method but rather finite volumes method for solving partial differential equations. it does not require from the user to write code but also has certain input files to be defined. For more details about FEniCS and openFOAM usage please refer to the original websites [1], [2] [2] [8]

[1] https://www.openfoam.com/
[2] https://fenicsproject.org/

Part II.

Adpater Development

4. Adpater Development

The aim of this chapter is to give a description of the developed adapter for Elmer FEM solver. At the begining a brief history of development approaches and of the challenges faced during the development process which may be helpful for further work. Secondly, a description of the communication flow and how data is communicated in an out of the solver. Thirdly, the adapter internal structure is demonstrated and a comparasion to the adapter development guidelines explained by preCICE developpers.At the end, how to use the adapter with Elmer is explained and some limitations of the adapter are introduced.

There exist 2 types of simulation, steady-state simulation and transient simulation as explained in the background chapter, the adpater is developed only to suit transient simulation, as steady state is not supproted by preCICE although there is a workaround to use preCICE with steady-state simulation as provided in .

4.1. Development And Challenges

In Developing an adapter to couple computational code is not a difficult process but rather a tricky one. Reasons that makes developing an adapter an achievable task are

1. There is no need to understand how preCICE works,preCICE is an API which provides the developer function calls for setup,comunication and data transfer.

2. The steps of performing a simulation are somehow similar in most solvers described below in source code 4.1 mentioned in chapter 2 in source code 2.1 , as explained in preCICE documentation [1], just repeated for sake of clearness

   - Firstly a setup phase, the solver starts to read some simulation parameters such as constants, mesh structure, simulation time etc.. these inputs differ from one solver to another but some inputs are essential as reading the mesh. Also the solver setup the system to be solved.
   - Secondly,time loop which include the computation part involivng solving system of linear or nonlinear equations each time step and update the solution data structure with the most recent calcuation for the next timestep.
   - At the end, some solvers contain post processing part like printing the final calculated values for visualization.

```
turnOnSolver(); //e.g. setup and partition mesh

double dt; // solver timestep size

while (not simulationDone()){ // time loop
  dt = beginTimeStep(); // e.g. compute adaptive dt
  solveTimeStep(dt);
  endTimeStep(); // e.g. update variables, increment time
}
turnOffSolver();
```

Source Code 4.1.: Solver Structure

Most solvers have a similar structure, but what is tricky is to identify this structure and make use of it. It is not always easy to interfere within the simulation, that is what makes the development tricky. Here are some of the chanllenges faced during the development process which might help for further work in Elmer adapter or for another adapter.

- **Philosophy of adapter development:** unfortunately the development of an adapter depends heavily on the software(solver) in hand, its structure and the features that it provides, mostly there is 2 ways to develop an adapter, either to modify the source code of the solver code which is not a recommended approach but in some cases there is no other way,also this is only possible in case of opensource solvers while this not possible with commercial sources as in that case a black box has to be dealt with.One more technical ascpect is building this modified software with recommended version of compliers in case of complie code, or using supported versions of interpreters in case of interpreted languages like python. Another way to develop the adapter is to find a way to make the solver communicate with external module and this external module does the job for communicating through preCICE as well as steering the simulation and writing communicated data to the internal datastructures of the solver, discussed in the following challenges, without modification to the source code. This module is the adapter, this the recommended approach but such an approach depends on the features provided by the solver to communicate with internal datastructures. Fortunately, Elmer provides a pretty good interface for communicating with the solver, a short overview of the provided features is described but for further details refer to chapter 18 in ElmerSolver Manual [9]. Elmer solver is very provides soutions for linking external code to the solver in case the desired problem requires further description, for example complicated boundary conditions or initial conditions. It also provides simlar approach to develop totaly a

[1] https://precice.org/couple-your-code-preparing-your-solver.html/

new module as an additional solver for custom problem other than provided module(e.g.Heat equation,Navier-Stokes,Linear Elasticity) that can be coupled with built in modules. The first solution is called User-Defined function(in case of Boundary conditions) and User-Defined solver(in case of custom module). The main approach is to use these user-defined modules to act as an adpater where the preCICE calls are handeled as well as data transfer.Both user-defined function and solver have to be written in Fortran and script is provided by Elmer to build these external libraries. Instructions on how to build external modules and how to link them to solver is described in the usage and limitations section. At first a short description interface functions and some internal data structure provided by Elmer for programming, then a description for each feature followed by explaination of the tried attempts to develop the adapter and the final used approach.

1. **Interface Function:** Elmer solver has 2 important Fortran modules, one of them is types which describes the types,classes and data structures for solver. it is not accessible from outside and also not all the types are required to understand, but understanding some classes is useful for both deveolping and usage. One important class that will be used in both the user-defined function and solve is class Model_t , it is basically a container of all required simulation data during runtime, it can give access to the variables, mesh, solvers(e.g. Heat Equation,Linear Elasticity), time, simulation parameters and many other parameters required during simulation. The second important Fortran module is called DefUtils, this is the module designed by Elmer to give user access to the simulation. It can acquire data from the system input file(.sif) like constants. It also gives access to variable values, allows the user to post messages and print to the console.

2. **User-Defined Function:** the user-defined function is pretty useful when it comes to assign a certain domain property such as boundary and initial conditions, body forces and material parameter. There is a certain signature for function presented below in source code 4.2. This is a simple example just to present the function signature, for further reading please refer to [9]. In this example we are trying to set the falue of input flow of a fluid to a recatangular pipe, the flow profile is parabolic and described as follows $v_x = 6(y-1)(2-y)$. The name of the function is GetInputStream, the calculated result is the variable velocity. The arguments that hve to be passed are model,n and y_coordinate; model is the container class described above that contains reference to required information, the n is of type integer it signifies the node number in the domain as seen from Fig. 4.1, the red line indicates the inlet side of the rectangular pipe, in this case the n will be 5,72,71,70,6. And the last argument has to be passed by the user through the system input file (.sif) as described in the snippet below in source code 4.3, as seen Coordinate 2 (which is y coordinate) is passed and GetInputStream is called as Prcedure. This user -defined function is called on a

```
Function GetInputStream(model,n, y_coordinate) RESULT(velocity)

   USE DefUtils

   IMPLICIT None

   TYPE(Model_t) :: model
   INTEGER       :: n
   REAL(KIND=dp) :: y_coordinate,velocity

   velocity = 6*(y_coordinate-1)*(2-y_coordinate)

END FUNCTION GetInputStream
```

Source Code 4.2.: user-defined function example

nodal basis such as for each node on the inlet this function is called , so in our example in 4.1, it will be called 5 times each time step before the computation, this is an important aspect that will affect our decision about which approach to b used, such a point will be further explained later on.

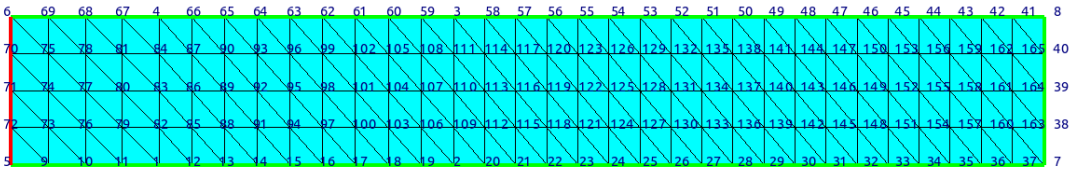


Figure 4.1.: Inlet Flow

3. **User-Defined Solver:** As described in the background chapter Elmer has several modules for different applications, so before solving the system of equations, the mass and stifness matrix have to be formulated for the finite element formulation, that is the role of the solver modules. Elmer has several built in

```
Boundary Condition 1
  Target Boundaries(1) = 1
  Name = "Inlet"
  Velocity 1 = Variable Coordinate 2
               Real Procedure "GetInputStream" "GetInputStream"
  Velocity 2 = 0.0
End
```

Source Code 4.3.: user defined function argument passing

modules ranging from Heat equation to Helmoltz equation (wave equation) that derive weak formulation, build local mass and stiffness matrix for each element then assemble the global matrix and apply the boundary conditions, then pass it to the main part of the solver to solve system of equations and get result and handle the multiphysics coupling.Our current aim is to use the user-defined solver but not as described as there is no equation to solve, but rather use it as an adpater to handle preCICE calls as well as data transfer, so no need to elaborate on how to develop a user-defined solver, for more information on how to write a user-defined solver please refer to ElmerSolver Manual [9].

4. **First Attempt with User-Defined Function:** the first idea was to use a user-defined function as the adapter, this attempt was not successful, it was chosen due to simplicity of user-defined function demonstrated by the manual compared to solver, and it has some success at the begining but there was a major drawback, so it is mentioned to avoid investing effort in that direction. The are five main parts required by the adapter in the current design which will be explained in more details in the adapter internal structure section, but just a short overview is given to show where this approach fails. The main five requirement from the adapter
   - Identify Coupling Interface And Allocate Memory for it
   - Setup preCICE interface
   - Read Data
   - Write Data
   - Finalize communication

   From the shown list the first 2 steps only done once, but the read and write operation which include data transfer and accessing internal data structures are executed each time-step. The problem with user defined function is that it is called on a nodal basis and it is used for domain parameters so the workaround to call it is by setting boundary values, this causes several challanges.

- The first challenge is that it is called each time step and in each time step it is called as much as nodes on the boundary.For example in Fig. 4.1 the function is called 5 times for vertices 5,72,71,70,6 and is called another 2 times for vertices 5 and 6 as these are on the corners so Elmer calls the function automatically when setting the upper and lower boundary. For identifying coupling interface is not a problem as it is referenced automatically in the system input file (.sif) in source code 4.3 line 5, but for the setup of preCICE this causes a problem, which is setting the mesh, all vertices at the coupling interface as well as their coordinates have to be passed in the preCICE call, and as mentioned this only called per each node, so in the first time call the required information about the whole coupling interface nodes is not available , also when it is called repeadetly the setup part should not be executed. Although there is a workaround this, tracking the first time called the setup part is executed and later on this part is skipped, this is provided due to FORTRAN syntax as it can save some data for the shared library. And for corrdinates of the coupling interface vertices, using Elmer features data about coordinates of vertices at the coupling interface could be identified but this requires knowing boundary number or adding an identifier in the system input file(.sif). the later is used and further utilized in the working attempt. Although this problem solved.
- Second challenge is calling the function for each node and calling preCICE calls for transfering data is too much work and communication for each time-step, as per node single scalar or vector data has to be written to preCICE data structure and a mapping for node number and vertex ID for preCICE has to be made, for our presented example in Fig. 4.1 this seems easy but for a finer mesh and a complicated problem this is not an efficient solution.
- Thirdly, the main challange that made this attempt fails, is the writing process, each solver has first to read the data, make some computation and then write back to preCICE the computed data, this the problem, as function calls are only controlled by boundary condition so it can only be called before the computation. An attempt to solve the problem was to create a dummy solver which will be explained in the next attempt. Another solution could be tried was to write data first and then read data in the same function call while setting the boundary condition before computation but the solution was not implemented as this approach revealed so many challenges so a decision was made to change the adapter design.

5. **Second Attempt with User-Defined Function And Solver:** As writing back the computed data to preCICE failed with user-defined function, an idea of developping a dummy user-defined solver to make some tasks of the adpater, first the setup process of preCICE, but this was useless as SolverInterface calls is ini-

tialized in the user-defined solver but there was a problem passing reference to the user-defined function of this initiated class, so when read or write data preCICE calls, it throws an error that preCICE has to be created first.the problem was both the user-defined solver and function are called by Elmer so both code cannot see each other only sees Elmer solver not external parts to it. This leads to second problem which writing back the data, the aim was to make the dummy solver call the user-defined function a second time after the computation but since it cannot pass the initialized preCICE reference to the function calls, writing back data was moved to the user-defined solver which proposed to change the design and rely only on user-defined solver.

6. **Third Attempt Solver:** the last attempt which lead to a working design was to move every required step of the adpater to user-defined solver to make the five required steps mentioned above in the first attempt which will be described in detail in the internal structure section. this will lead to some challenges in the steering but the user-defined solver helped to solve such problems ans it will be explained in the next point. also preCICE setup as well as its calls of reading and writing data is done all from the same FORTRAN subroutine.

- **Steering and Data-structures:** As explained at the begining of this section the steps of developing an adapter, there is the time loop of the computation, which is embedded inside the solver, this is pretty difficult to control, but since the user-defined solver is called each time step and its order amoung other called solver in the multphysics problem, a flexibity of calling the user-defined solver several times per each time step and control from within the adapter which part to be executed is given. Hence the adapter is called once at the begining for the setup, 2 times per time setp one before computation for reading and second time after the computation for writing and once at the end of the simulation to finalize. This also is provided due to the structure of Elmer as it will be explained in the usage section, presenting parts of the system input file to use the adapter. Also the second challange is to write data to the internal data-structure of Elmer, reading data is provided easily by the intreface provided by Elmer. Refering to the Elmer Solver Manual [9] in the basic programming section, by using Deftils module, there is a function call called

  ```
  GetScalarLocalSolution(Temporary data structure, Variable Name)
  ```

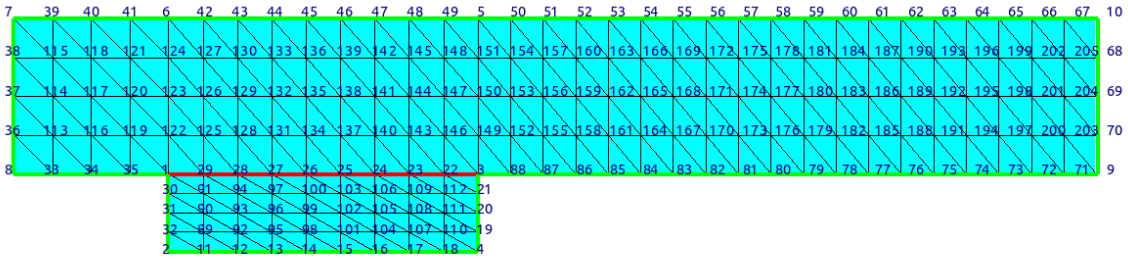
  this is a pretty useful function but only for reading data but when writing is required, it is not useful anymore, as data is copied to a temperorary data structure which does not affect the internal data-structure of Elmer. the solution is to get the variable as a pointer using

  ```
  VariableGet( mesh \% Variables, data Name as string)
  ```

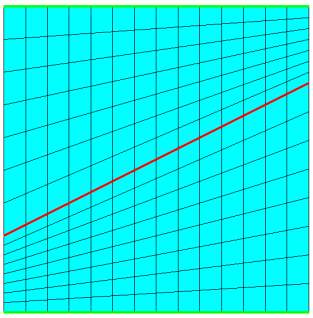
  and access its values and write to it. This is one of the problems that lead to using the user-defined function as it did not requrie good knowldge of how to write data to

the internal data strucutre, the value returned by the function is autmatically written by Elmer to variable data-structure at the correct position of the node. but later on with changing the design to use the user-defined solver.

- **Coupling Interface:** The challenge with coupling interface is a hint has to be inserted in the system input file(.sif) to be able to identify it by the adapter for allocating memory for values at its vertices as well as knowing the number of vertices, their coordinates and Ids to be able to do the mapping to preCICE mesh. The second problem is with transfering vector data, for example as depicted in Fig. 4.2a the coupling interface is horizontal, so when transfering vector data the normal is the same direction as the y-direction but in case of Fig. 4.2b the normal is perpendicular to coupling iterface, so for transfering vector data additional work has to be done for considering such a case.



(a) Coupling Interface



(b) Coupling Interface Slope

Figure 4.2.: Coupling Interface Difference

```
Boundary Condition 4
  Target Boundaries(1) = 5
  Name = "Coupling_boundary"
  Noslip wall BC = True
  Temperature = 310
  Coupler Interface = Logical True
End
```

Source Code 4.4.: coupling interface identifier

4.2. Adapter Internal Structure

This section describes the internal structure of the adapter, how it is organised, parts executed repeatedly and helper subroutines. As mentioned earlier in section Development 4.1 the adapter consists of five main parts

1. Identify Coupling Interface And Allocate Memory for it: As depicted in Fig. 4.2a, the coupling interface has to be identified, the number of vertices which is 10 nodes in this example, also the coordinates of the vertices have to be saved for later use in preCICE setup. This is done with the help of setting the problem as well as an identifier put the system input file(.sif) as shown in source code 4.4, the Coupler interface attribute, which is boolean set to true, the adapter check for this keyword and throws an error if it is not found. once found it saves the coordinates of the vertices at this boundary, as well as saving a mapping for the node number to be accessible later on for reading and writing. It utilises existing variables, and in case of a non-existent variable(user-defined) it creates one with its internal data structure. This is shown in 4.5.

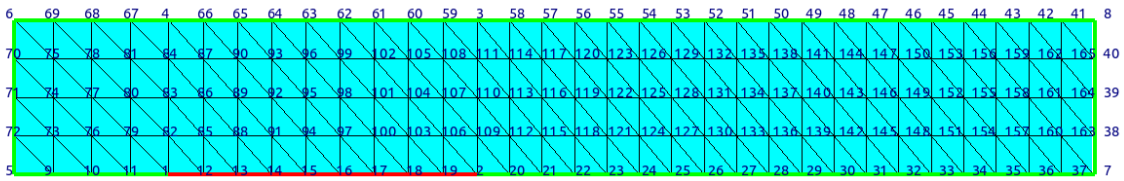
2. Setup preCICE interface: For intializing preCICE, a certain setup calls have to be executed, but first some data have to be specified to identigy which part of the solver is calling preCICE, these data is also described in the syste,m input file and has to be read by the adapter at first run.

   - participant name
   - mesh name
   - path to the xml path

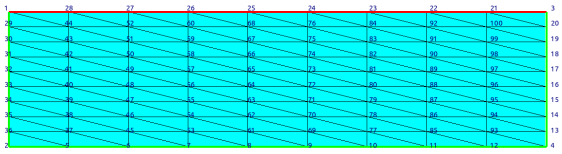
   Then preCICE instance is to be created and the vertices coordinates at the coupling interface are set to preCICE previously acquired in step one, and temporary data structutre are allocated for reading and writing data from preCICE as presented in 4.7. At the end of the first run data is initilazed before the first computation. There are 2 types of coupling as explained in preCICE overview in chapter 2, serial and

```
!-----------Identify Vertex on Coupling Interface &
Save Coordinates---------------
        NULLIFY( BCPerm )
        ALLOCATE( BCPerm( Mesh % NumberOfNodes ) )
        BCPerm = 0
        ! CALL MakePermUsingMask( Model, Solver, Mesh, MaskName, .FALSE., &
        !     BCPerm, vertexSize )
        CALL MakePermUsingMask( Model, Solver, Mesh, MaskName, .TRUE., &
            BCPerm, vertexSize )
        CALL Info('CouplerSolver','Number of nodes
        at interface:'//TRIM(I2S(vertexSize)))

        ALLOCATE( CoordVals(2*vertexSize) )
        ALLOCATE(vertexIDs(vertexSize))
        DO i=1,Mesh % NumberOfNodes
            j = BCPerm(i)
            CoordVals(2*j-1) = mesh % Nodes % x(i)
            CoordVals(2*j) = mesh % Nodes % y(i)
            ! CoordVals(3*j) = mesh % Nodes % z(i)
            ! IF(j /= 0) THEN
            !     vertexIDs(j) = j
            ! END IF
        END DO
        CALL Info('CouplerSolver','Created nodes at interface')

        !-----------Identify read Variables and
        !-Create it if it does not exist--------------------
        CALL CreateVariable(readDataName,'readDataName',
        mesh,BCPerm,Solver,solverParams)

        !-----------Print Read Values, For Debugging Purposes--------------
        CALL Info('CouplerSolver','Printing read Data')
        CALL Print(readDataName,mesh ,BCPerm,CoordVals)
        !------------------------------------------------------------------

        !-----------Identify write Variables and
        !-Create it if it does not exist--------------------
        CALL CreateVariable(writeDataName,'writeDataName',mesh
        ,BCPerm,Solver,solverParams)
```

Source Code 4.5.: coupling interface identification



(a) Upper Part



(b) Lower Part

Figure 4.3.: Domain partition example

parallel coupling, in case of serial which is adopted here only the second particpant is allowed to initialize data, while in parallel coupling both participant can initialize data. So this condition is checked(specification exist in xml file) and based on it data is initilazed as schown in 4.8

3. Read Data: Before making any computation, data is read and then updated inside internal data structure, this is done with the help of *precicef_read_bsdata(readDataID, vertexSize, vertexIDs, readData)* as block scalar data at the coupling interface is comunicated by preCICE to the participant which has the turn of computing as shown in 4.9. this part also utilizes a helper function for copying data from temporary data structure to internal data structure.

4. Write Data: After Computation, newly computed data is required for the other participant to be ready to read data and perform its share of computation at the next time step so it has to be communicated via preCICE. it is the same as reading but the copying from the temporary buffers is done before communication as shown in 4.10, then the time is advanced in both participants.

5. Finalize communication: At the end of the computation, both partcipants must finalize communication and destruct initialized class, so *CALL precicef_finalize()* is called.

```
!----------------Acquire Names for solver---------------------
        maskName = GetString( Simulation, 'maskName', Found )
        participantName = GetString( Simulation,
        'participantName', Found )
        meshName = GetString( Simulation, 'meshName', Found )
        configPath = GetString( Simulation, 'configPath', Found )
```

Source Code 4.6.: Parameter Acquistion

```
!---------------Initializing Precice---------------------------------


        CALL precicef_create(participantName, configPath, rank, commsize)

        CALL precicef_get_dims(dimensions)
        CALL precicef_get_mesh_id(meshName, meshID)
        CALL precicef_set_vertices(meshID, vertexSize, CoordVals, vertexIDs)

        CALL precicef_get_data_id(readDataName,meshID,readDataID)
        CALL precicef_get_data_id(writeDataName,meshID,writeDataID)

        ALLOCATE(readData(VertexSize))
        ALLOCATE(writeData(VertexSize))
```

Source Code 4.7.: preCICE Initialziation

```
!---------------------Initializing Data-----------------------------
        CALL precicef_initialize(dt)
        CALL precicef_action_write_initial_data(writeInitialData)
        CALL precicef_is_action_required(writeInitialData, bool)

        IF (bool.EQ.1) THEN
            CALL Info('CouplerSolver','Writing Initial Data')
            CALL CopyWriteData(writeDataName,mesh,BCPerm,writeData)
            CALL precicef_write_bsdata(writeDataID, vertexSize,
            vertexIDs, writeData)
            CALL precicef_mark_action_fulfilled(writeInitialData)
        ENDIF

        CALL precicef_initialize_data()
```

Source Code 4.8.: Data Initialziation

```
CALL precicef_action_write_iter_checkp(writeItCheckp)
        CALL precicef_is_action_required(writeItCheckp, bool)

        write(infoMessage,'(A,I2)') writeItCheckp,bool
        CALL Info('CouplerSolver',infoMessage)

        IF (bool.EQ.1) THEN
          CALL Info('CouplerSolver','Writing iteration checkpoint')
          CALL precicef_mark_action_fulfilled(writeItCheckp)
        ENDIF

        CALL Info('CouplerSolver','Reading Data')
        CALL precicef_read_bsdata(readDataID, vertexSize,
         vertexIDs, readData)

        CALL Info('CouplerSolver','Copy Read Data to Variable')
        CALL CopyReadData(readDataName,mesh,BCPerm,readData)
```

Source Code 4.9.: Read Data

```
!--------------------Copy Write values from Variable to buffer----------

        CALL Info('CouplerSolver','Copy Write Data to Variable')
        CALL CopyWriteData(writeDataName,mesh,BCPerm,writeData)

        CALL Info('CouplerSolver','Writing Data')
        CALL precicef_write_bsdata(writeDataID, vertexSize,
         vertexIDs, writeData)

        CALL Info('CouplerSolver','Printing write Data')
        CALL Print(writeDataName,mesh ,BCPerm,CoordVals)

        CALL Info('CouplerSolver','Printing read Data')
        CALL Print(readDataName,mesh ,BCPerm,CoordVals)

        !--------------------Advance time loop----------------------
        CALL precicef_advance(dt)

        CALL precicef_action_read_iter_checkp(readItCheckp)
        CALL precicef_is_action_required(readItCheckp, bool)
```

Source Code 4.10.: WriteData

```
1   SUBROUTINE CreateVariable(dataName,dataType,mesh,BCPerm,Solver,solverParams
2
3         dataName =  ListGetString(solverParams,dataType,Found )
4         dataVariable  => VariableGet( mesh % Variables, dataName)
5         IF(ASSOCIATED( dataVariable ) ) THEN
6             CALL Info('CouplerSolver','Using existing variable
7             : '//TRIM(dataName) )
8         ELSE
9             CALL Info('CouplerSolver','Creating variable
10            as it does not exist: '//TRIM(dataName))
11
12            Dofs = ListGetInteger( solverParams,'Field Dofs',Found )
13            IF(.NOT. Found ) Dofs = 1
14            CALL VariableAddVector( mesh \% Variables, mesh,
15             Solver, dataName, Dofs, &
16                Perm = BCPerm, Secondary = .TRUE. )
17            dataVariable => VariableGet( mesh % Variables, dataName )
18        END IF
19
20    END SUBROUTINE CreateVariable
```

Source Code 4.11.: CreateVaraible

The explained structure is realized with the help of several helper functions, the most important ones are

- *CreateVariable(dataName,dataType,mesh,BCPerm,Solver,solverParams)*
- *CopyReadData(dataName,mesh,BCPerm,copyData)*
- *CopyWriteData(dataName,mesh,BCPerm,copyData)*

The *CreateVariable* function is used for creating non-existent(user-defined) variable at the setup phase as shown in 4.11, while the copy functions are similar but one is copying data from temporary buffer to internal data structure and the other is copying in the opposite direction, only one of them is shown in 4.12.

The mechanism of selecting which part to be executed by the adapter is handled by a switch of an integer varaible *task* which is updated and saved at the end of each call and initialized at first run.

```
SUBROUTINE CopyReadData(dataName,mesh,BCPerm,copyData)

        dataVariable  => VariableGet( mesh % Variables, dataName)

        DO i = 1, mesh % NumberOfNodes
            j = BCPerm(i)
            IF(j == 0) CYCLE
            dataVariable % Values(dataVariable % Perm(i)) = copyData(j)
        END DO

    END SUBROUTINE CopyReadData
```

Source Code 4.12.: CopyReadData

4.3. Usage And Limitations

Using the adapter is not difficult, it only requires some data to be prescribed in system input file(.sif) as well handling the order of calling the adapter several times in the correct order. And of course, the shared library has to be built and available in the correct location. So we will start with building the adapter, Elmer provides a script to compile user-defined code to ensure that extra code is built with the same compiler version and same settings to avoid any incompatibility issues. Also the shared library has to be linked with the preCICE library in order to detect it at runtime. So to do this just runing *elmerf90 -o <output name ><file_name.F90 ><location of preCICE library>* the output name is the name of compiled shared library and preferably with the extension .so, this will be described in the system input file(.sif), the file_name.F90 is the adapter code with the name Coupler_Solver.F90, and lastly the location of the preCICE shared library so the complier should be able to compile preCICE calls and set reference to detect library at runtime. An example for compiling the adapter as follows. *elmerf90 -o Coupler_Solver Coupler_Solver.F90 /usr/lib/x86_64-linux-gnu/libprecice.so.2*. the preferred location for adapter is the same directory of the system input file, it is possible to put any desired location, but either relative or absolute path have to be provided in the system input file. After building the adapter a few setup in the system input file has to be done.As mentioned in internal structure section there are main five parts of the adpater. Although executing which part to a certain extent controlled by the adapter but this has to be mirrored in the system input file(.sif) as the calling part is controlled by the system input file(.sif). Just to remind of what happens in the solver

1. Identify Coupling Interface And Allocate Memory for it
2. Setup preCICE interface
3. Read Data

```
Solver 3
    Equation = "Initialize"
    Procedure = "../Coupler_Solver.so" "CouplerSolver"
    readDataName = String "temperature flux 1"
    writeDataName = String "Temperature"
    Exec Solver = before all
End
```

Source Code 4.13.: Adapter Initialization

4. Write Data

5. Finalize communication

The first and second part has to be called once at the begining, this is handeled by creating a dummy solver to call the adapter as shown in source code 4.13, the solver id starts to count from the number of existing real solvers, for example(which will be shown in detail in Chapter 7), trying to solve a multiphysics problem consisting of Heat eqaution and Navier-stokes equation, so the number of solver are 2, so starting to count dummy solver for calling adapter from 3 and so on. The solver must be given a name in our case it is Initialize, the Procedure is the important part as the path of built adapter, either relative or absolute. In the present example this is the absolute path, which is on level up in the directory with respect to the system input file(.sif). Then the readDataName and writeDataName, these are essential for setting up the preCICE reference, as each participant has to identify the communicated data as well as the direction, these data names are read by adapter and has to be identical to the exisiting names in the config(.xml) file for preCICE. At the end, the privilige feature of Elmer that helped making the adapter architecture works, which is steering the calling events. As mentioned the setup part has to be executed once at the begining, so Exec Solver is set to before all, which execute this solver that calls the adapter once at the begining before make any computation or time steps. The third part is about reading the data before making any computation, this is depicted in source code 4.14, similar to the setup, it has a name, relative path to the adapter, communicated data name, and at the end the attribute that makes the difference Exec Solver is set to before timestep, this executes the solver each time step but before the computation or calling any of the real solvers so the data can be communicated.

The fourth part is writing the newly computed data after computation, as presensted in source code 4.15, the Exec Solver is set to after timestep, calls the adapter after making computation. So this part combined with the third part, makes the following sequence for each time step, solver 4 is executed for reading data, then real solver are called which in the current example are solver 1 and 2, this the computation part, then after finishing the computation, solver 5 is called for writing newly computed data to preCICE.

```
Solver 4
    Equation = "ReadData"
    Procedure = "../Coupler_Solver.so" "CouplerSolver"
    readDataName = String "temperature flux 1"
    writeDataName = String "Temperature"
    Exec Solver = before timestep
End
```

Source Code 4.14.: Read Data

```
Solver 5
    Equation = "writeDataAdvance"
    Procedure = "../Coupler_Solver.so" "CouplerSolver"
    readDataName = String "temperature flux 1"
    writeDataName = String "Temperature"
    Exec Solver = after timestep
End
```

Source Code 4.15.: Write Data

At the end, after all computation are finished, finalizing preCICE has to be called to call the destructor and to end any calling of parallel code librarys like mpi. this shown in source code 4.16.

One more important insertion to the system input file (.sif) is the boundary condition. The coupling between 2 or more solvers is done via boundary condition such as the data at the coupling interface at one solver is considered the boundary condition for the other. Data might be copied to same variable used or not, so to make sure nothing is overwritten by unwanted values like default values if not specified, boundary condition has to be set at the coupling interface, as shown in the example source code 4.17, the Heat Flux is set to temperature flux 1, the problem in hand which will be described in details in chapter 4, takes the heat flux as boundary condition, and similar to Fig. 4.3a the coupling interface is horizontal, so the normal to it is the same direction of heat flux in y-direction, so the Heat Flux is set to the y-component of the heat flux,normally if this boundary condition is left blank, even after copying the y-component to the Elmer internal data-structure, Elmer set boundary condtion to defualt which is natural boundary condition where Heat flux is equal to zero.

Last thing for the adapter setup as mentioned in the internal structure of adapter section, the adpater must read some input data relevant for preCICE,

- participant name

```
Solver 6
    Equation = "Finalize"
    Procedure = "../Coupler_Solver.so" "CouplerSolver"
    readDataName = String "temperature flux 1"
    writeDataName = String "Temperature"
    Exec Solver = after all
End
```

Source Code 4.16.: Finalize

```
Boundary Condition 1
  Target Boundaries(1) = 2
  Name = "Coupling_boundary"
  Heat Flux = Equals "temperature flux 2"
  Coupler Interface = Logical True
End
```

Source Code 4.17.: boundary condition

- mesh name
- path to the xml path

this has to be set in simulation part as shown in source code 4.18, the last four lines, maskName,participantName ,meshName, configPath are essential, the user may change the last 3 but the mask name has to stay the same, otherwise, changing the mask name must change the boolean attribute name as well depicted in source code 4.17 in the last line.

At the end, mentioning limiatitions for user is pretty important as well for further development, currently the adapter only support transient simulation, steady state simulation is not supported. Also as mentioned steering is difficult, and it is hard to interfere in the time loop controlled by Elmer, so currently only explicit coupling is avalaible, implicit coupling is still under development.

```
Simulation
  Max Output Level = 5
  Coordinate System = Cartesian
  Coordinate Mapping(3) = 1 2 3
  Simulation Type = Transient
  Steady State Max Iterations = 3
  Output Intervals = 1
  Timestepping Method = BDF
  BDF Order = 2
  Timestep intervals = 10
  Timestep Sizes = 0.1
  Solver Input File = case.sif
  Post File = Fluid.vtu

  maskName = String "Coupler Interface"
  participantName = String "Fluid"
  meshName = String "Fluid-Mesh"
  configPath = String "../precice-config.xml"
End
```

Source Code 4.18.: boundary condition

Part III.

Test Problems and Results

5. Study and limitations

After introducing the necessary softwares such as Elmer and preCICE and other extra tools, it is the time to conduct some experiment with the developed adapter. In the next chapters 2 example problems are intoduced,such problems were helpful for development, also it represents real time applications. Then in the last chapter an outlook of the current status and future work to be developed including some progress made in athrid example problem. The first problem in hand is partitioned heated plate, it represents the area of applications concerning heat equations that study heat distribution across the material, such a problem has many applications that arise in engineering, to help design products and avoid failure due to thermal stress. The second class of problems is conjguate heat transfer (CHT). It is concerned about the heat transmission across a fluid flow, so it handles a multiphysics problem which include heat equation and Navier-Stokes equation. The last type of problems presented in the outlook is the fluid structure interaction. In the following section the current settings under which these simulation are conducted as well as the limitation of the existing prototype.

5.1. Simulation Settings

The target software is Elmer multiphysics simulation software, it is an open source software, the source code exist publicy on a git repository and can even be modified locally. The repository can be cloned and software can be built manually using cmake, but since there was no intention of modifiying the code, so it was installed from binary packages, the version installed is 8.4, that supports parallel computing (mpi library), also ElmerGUI as well as ElmerGrid are installed automatically with the binary packages, the user can choose not install them, but it is a recommendation to install them, since it can be helpful to use them.

The second important software is preCICE coupling library, also it is open source and can be built manulaly using cmake, but it was also installed from the binary packages, the version installed is 2.2.1 .

For the coupling solvers, OpenFoam bionic 4.1 was installed and its compatible adapter, and for FEniCS, python 3.7 was used.

The code used for the result generation in the next chapters is avaiable at [1] and the code under development exist under [2]

[1] https://github.com/precice/elmer-adapter
[2] https://github.com/HishamSaeed/elmer-adapter/tree/develop

5.2. Limitations

The aim of developing the adapter is to make as general as possible to support a wide range of multiphysics applications as well as complex geometries. But since it is still in the development phase, it still has some limitations. First of all it only supports eplicit coupling because Elmer does not give an easy way to steer the time loop from outside, so the further investigation is needed, otherwise, a modification to the source code has to be introduced. Also it still supports serial coupling, further development for parallel coupling is required

6. Partitioned Heated plate

The first example problem is a simple problem as an introduction to solving partial diferential equation(PDEs), it is widely used in numerical mathematics course as well as finite element courses. We can use an anology to programming, it is like a Hello world program to introduce the programming language. So It was the first trial to develop, debug and test the adpater. Our approach in each section of the test problem is to firstly introduce the mathematical background of the problem, then an explaination of the problem setup, the files used for the solvers. And at the end the results will be shown.

6.1. Mathematical Theory

The heat equation is a partial deferential equation that is classified as parabolic. It is interested in studying the heat destribution over a certain domain (material, either rigid or fluid). Detailed knowldege of the heat distribution as well as the heat flux could be very useful in many application, to determine products endurance under thermal stress, for exmple in the electronics industry, electronic components could fail under thermal stress, or get damaged over time which reduces its lifetime. Hence simulating such components to ensure it can hold thermal stress during its usage. The heat equation derivation depends on law of conservation of energy. The mathematical formulation which our problem is built on is shown in 6.1

$$\rho c_p \frac{\partial T}{\partial t} = \Delta T + f \tag{6.1}$$

where ρ is the density of the material, c_p is the heat capacity at constant pressure(in our case, the problem uses metal plate, so no use of discussing constant pressure). k is the heat conductivity(which refers to the ability of the material to transfer heat across space in a certain amount of time). f is an applied external heat source to the domain. T is the intended variable temperature, on the left side we see the change of temperature over time, and on the right hand side the change of temperature over space.

To give an insight of how the heat equation is implemented in Elmer, the heat equation from Elmer model manual [7] is presented hier in 6.2, it describes heat equation for incompressible fluids

$$\rho c_p \left(\frac{\partial T}{\partial t} + (\vec{u}.\nabla)T) \right) - \nabla.(k\nabla T) = \overline{\overline{\tau}} : \overline{\overline{\epsilon}} + \rho h \tag{6.2}$$

where $\vec{u}$ repersents the convection velocity, the term $\overline{\overline{\tau}} : \overline{\overline{\epsilon}}$ represents the frictional viscous heating and h is the external heat source.

Equation 6.2 used in Elmer can be simplified to our equation presented in 6.1, first by analogy, h is the same as f the external heat source and since our problem represents a solid material so both terms of convection velocity and viscous heating are omitted from equation, hence the resulting equation shown in 6.3

$$\rho c_p(\frac{\partial T}{\partial t}) - \nabla.(k\nabla T) = \rho h \tag{6.3}$$

And by moving the spatial derivative to the left hand side we obtain the same equation

$$\rho c_p(\frac{\partial T}{\partial t}) = \nabla.(k\nabla T) + \rho h \tag{6.4}$$

After including the mathematical formulation of the heat equation, a test problem has to be formulated in order to be able to debug the adpater and make sure that the computed values are correct. Such a problem is built with the help of FEinCS Tutorial guide in [5].

6.2. Problem Setup

The problem used here is very simple it consists of a 2D metal Plate of size 2 meter square with 2 meters in x-direction and 1 meter in y-direction. The domain is then divided into 2 parts along the x-direction exactly at 1 meter long as shown in Fig. 6.1. Then each part of the domain becomes a self-standing problem and a solver is responsible for computing the values of temperature at each node. In order to have such problem computed as whole but divided into 2 parts, communication must be done between the 2 domains as boundary conditions. So after domain decomposition, the left part of the domain recieves the communicated data as Dirichlet boundary condition, the communicated data is the temperature, this part of the domain is designated as *D(u)* and the boundary of that domain is the dirichlet boundary condition represented by Γ_D. While the right part of the domain recieves the communicated data as Neumann boundary condition, hence here the heat flux is to be communicated. This represents a challenge in the Neumann part of the domain in other problems which will be explained later on and why due to simplicity of the presented problem such a challenge does not appear. The right part of the domain is noted by *N(q)* and the Neumann boundary conditon is noted by Γ_N.

It can be noticed that both boundary conditions (Dirichlet and Neumann) lies on the same line at $(x, y) = (1, 0)$, hence data continuation across that line refers to high computation quality.

The problem is set to run with Elmer for both domains and also is set to run coupling Elmer with FEniCS. so the existing combinations for simulation are as follows.

- Elmer for *D(u)* and Elmer for *N(q)*. Elmer-Elmer coupling.

- Elmer for *D(u)* and FEniCS for *N(q)*. Elmer-FEniCS coupling FEniCS code taken from [1].
- FEniCS for *D(u)* and Elmer for *N(q)*. Elmer-FEniCS coupling FEniCS code taken from 1.
- FEniCS for *D(u)* and FEniCS for *N(q)*. FEniCS-FEniCS coupling. This is one of the reference results to compare to, since it is a working tutorial by preCICE, code used for this tutorial can be found in preCICE tutorials 1, the problem is taken from [11] and [5].

Also a monolithc run for the whole domain without coupling done by Elmer is tried out and used as reference as well with the analytical soultion and the FEniCS-FEniCS coupling. One possiblity of the settings as well is to partition the whole domain into 2 Dirichlet domains, in the sense that both domains recieve the communicated data as dirichlet boundary condition, but this does not work due to certain limitation, which will be explained later in the Results subsection with the data presented.

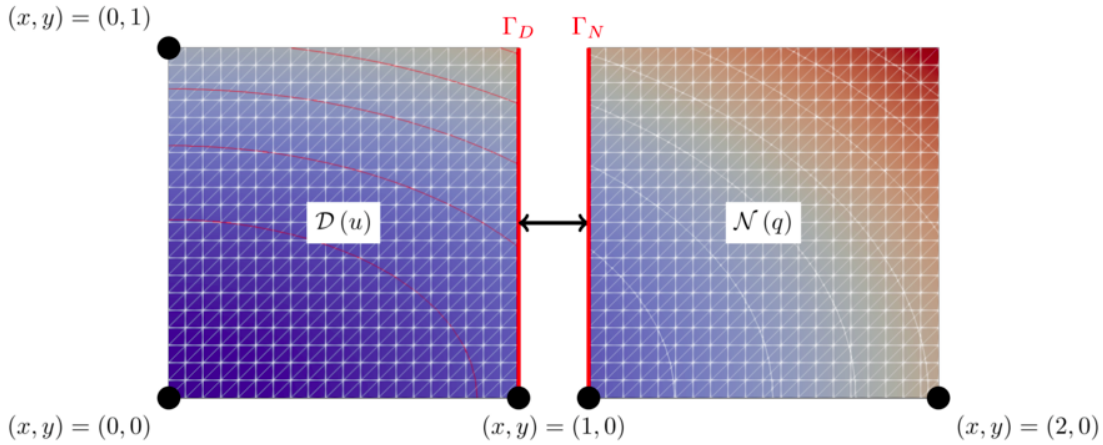


Figure 6.1.: Partitioned Heated plate, image from [11]

To prepare the problem for Elmer as mentioned in the Background section, Elmer requires both mesh files as well as the sif file. To produce the mesh files first the geometry has to be defined and generate a mesh upon it. The software used here is an open-source tool called gmsh presented earlier in the Bacground chapter. it can be handled through the GUI presented of the tool or the script file used by gmsh can be edited by any text editor. Here 3 mesh files are required to be prepared. One mesh for the monolithc run without any communication, and 2 mesh files for the partitioned case, one for the Dirichlet domain

[1] https://github.com/precice/tutorials/tree/6d90ae3eb179113beb400e472c8832e29a48db9b/partitioned-heat-conduction

```
1         //+
2  Point(1) = {0, 0, 0, 1.0};
3  //+
4  Point(2) = {1, 0, 0, 1.0};
5  //+
6  Point(3) = {2, 0, 0, 1.0};
7  //+
8  Point(4) = {0, 1, 0, 1.0};
9  //+
10 Point(5) = {1, 1, 0, 1.0};
11 //+
12 Point(6) = {2, 1, 0, 1.0};
13 //+
14 Line(1) = {1, 2};
15 //+
16 Line(2) = {2, 3};
17 //+
18 Line(3) = {3, 6};
19 //+
20 Line(4) = {6, 5};
21 //+
22 Line(5) = {5, 4};
23 //+
24 Line(6) = {4, 1};
25 //+
26 Line Loop(1) = {5, 6, 1, 2, 3, 4};
27 //+
28 Plane Surface(1) = {1};
29 //+
30 Physical Line("Plate_Boundary") = {6, 5, 4, 3, 2, 1};
31 //+
32 Physical Surface("Plate") = {1};
33 //+
34 Transfinite Line {6, 5, 4, 3, 2, 1} = 10 Using Progression 1;
35 //+
36 Transfinite Surface {1} = {1, 3, 6, 4};
```

Source Code 6.1.: Entire Domain Unpartitioned

and the other is for the Neumann domain. For debuging purpose both mesh have the same granularity.

In source code 6.1 this the script for gmsh to build the entire domain for the monolithc case run by Elmer, as shown 6 points are defined for the corners as well the coupling interface, then the surface for the plate is defined to generate the mesh later on. The scripts in source code 6.2 and 6.3 are similar only the coordinates are changed. At the end Transfinite mesh is defined, number of points on side lines are defined, more points will make the mesh finer. This definition of the mesh to have a simple mesh for debugging, gmsh can also generate a isotropic triangle mesh instead of triangular elements in square grid, such a mesh is presented in Fig. 6.4 but it is not used. Both the scripts in source code 6.2 and 6.3 are used to generate the mesh file (.msh) from gmsh then by ElmerGUI the mesh files (described in Bacground section) can be generated. The mesh with node numbers are shown in Fig. 6.3, as shown the red line is the coupling boundary in both domain, but it is an entirely different domain where each solver operate on.

As mentioned earlier, due to the decomposition made, one domain recieves communicated data as heat-flux, and this requires the flux calculation, this is achieved with the help of auxilary Elmer modules. The heat flux required in such case is a vector valued in W/m^2, it can be calculated by Elmer flux compute module, only it has to be specified in the sif file as shown in source code 6.4, in our case only the flux needed in the x-direction, since the problem setup is simple the flux in the x-direction is the same direction as the normal direction to the coupling boundary.

Also to call the adapter as explained in the adapter development chapter, dummy modules have to be introduced to the sif file for each stage with their order of calling, it can be seen from 6.5, the examples shown in source code 6.5 is for the Neumann case the Dirichlet cas is similar but the read and write data are fliped.

One last thing to be set in the sif file is the way of communication which is the boundary condition, the Dirichlet part of the domain recieves the data as temperature value in dirichlet boundary condition on the coupling boundary while the Neumann part of the domain recieves the data as heat-flux on the coupling boundary, it is the heat flux component in the x-direction which is the same as the normal direction. For the Dirichlet case it is shown in source code 6.6, the temperature variable is set to the Tempearture value,this temperature variable internal data structure in Elmer will be filled with all the correct data at the coupling boundary condition by the adapter, if this variable $Temperature = Equals"Temperature"$ is not set, Elmer puts the temperature at the boundary condition to the defualt value which is natural boundary condition where the heat-flux in the normal direction is set to zero.

$$-kdT/dn = 0 \tag{6.5}$$

And for the Neumann part it is shown in source code 6.7, the heat-flux is set to the first component of the heat flux , which is in the x-direction. Such values are filled by the adapter in the internal data-structure the same way for the Dirichlet part of the domain.

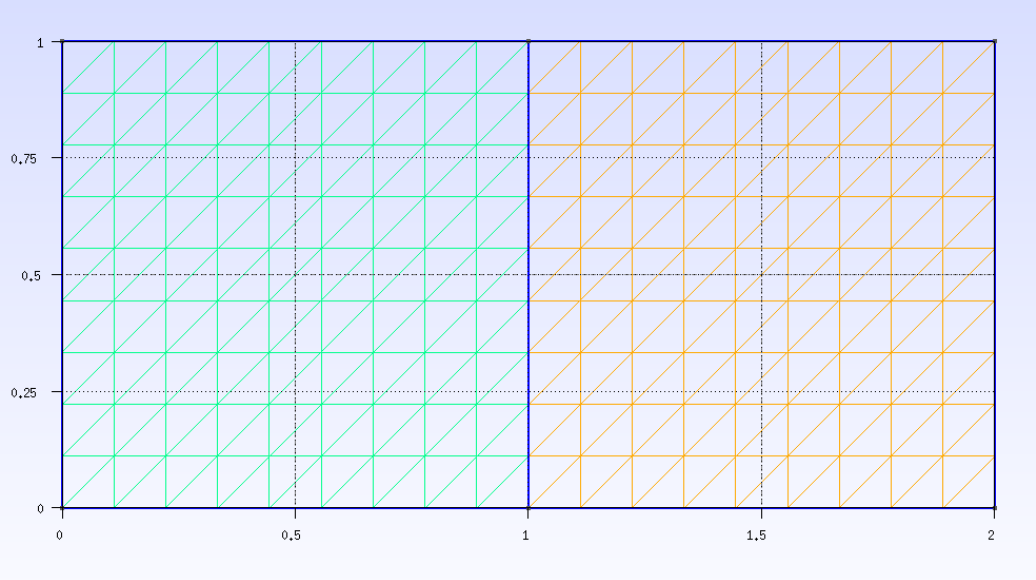
The entire sif file for both Dirichlet and Neumann part of the domain is included in the

```
1          //+
2  Point(1) = {0, 0, 0, 1.0};
3  //+
4  Point(2) = {1, 0, 0, 1.0};
5  //+
6  Point(3) = {0, 1, 0, 1.0};
7  //+
8  Point(4) = {1, 1, 0, 1.0};
9  //+
10 Line(1) = {1, 2};
11 //+
12 Line(2) = {2, 4};
13 //+
14 Line(3) = {4, 3};
15 //+
16 Line(4) = {3, 1};
17 //+
18 Line Loop(1) = {3, 4, 1, 2};
19 //+
20 Plane Surface(1) = {1};
21 //+
22 Physical Line("Plate_Boundary") = {4, 3, 1};
23 //+
24 Physical Line("Coupling_Interface") = {2};
25 //+
26 Physical Surface("Plate") = {1};
27 //+
28 Transfinite Line {4, 3, 2, 1} = 10 Using Progression 1;
29 //+
30 Transfinite Surface {1} = {1, 2, 4, 3};
```

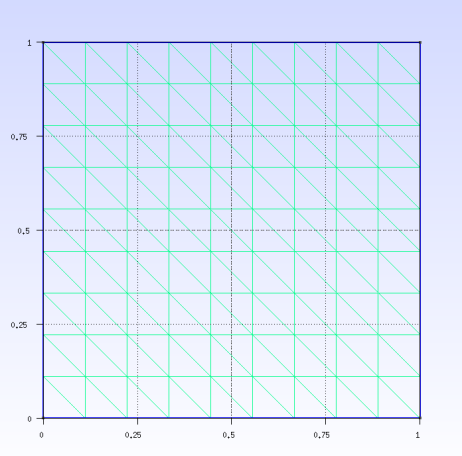
Source Code 6.2.: Dirichlet Domain

```
1  //+
2  Point(1) = {1, 0, 0, 1.0};
3  //+
4  Point(2) = {2, 0, 0, 1.0};
5  //+
6  Point(3) = {1, 1, 0, 1.0};
7  //+
8  Point(4) = {2, 1, 0, 1.0};
9  //+
10 Line(1) = {1, 2};
11 //+
12 Line(2) = {2, 4};
13 //+
14 Line(3) = {4, 3};
15 //+
16 Line(4) = {3, 1};
17 //+
18 Line Loop(1) = {4, 1, 2, 3};
19 //+
20 Plane Surface(1) = {1};
21 //+
22 Physical Line("Plate_Boundary") = {3, 2, 1};
23 //+
24 Physical Line("Coupling_Interface") = {4};
25 //+
26 Physical Surface("Plate") = {1};
27 //+
28 Transfinite Line {4, 3, 2, 1} = 10 Using Progression 1;
29 //+
30 Transfinite Surface {1};
31 //+
32 Transfinite Surface {1} = {1, 3, 4, 2};
```

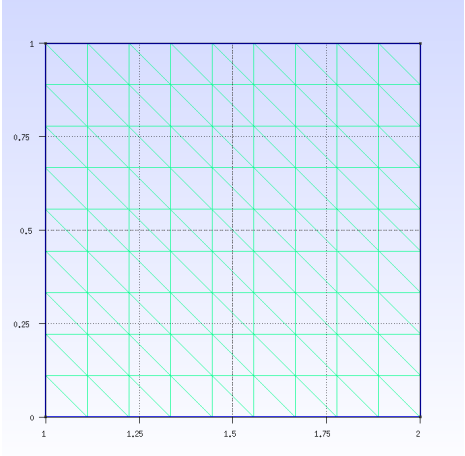
Source Code 6.3.: Neumann Domain



(a) Entire Domain gmsh

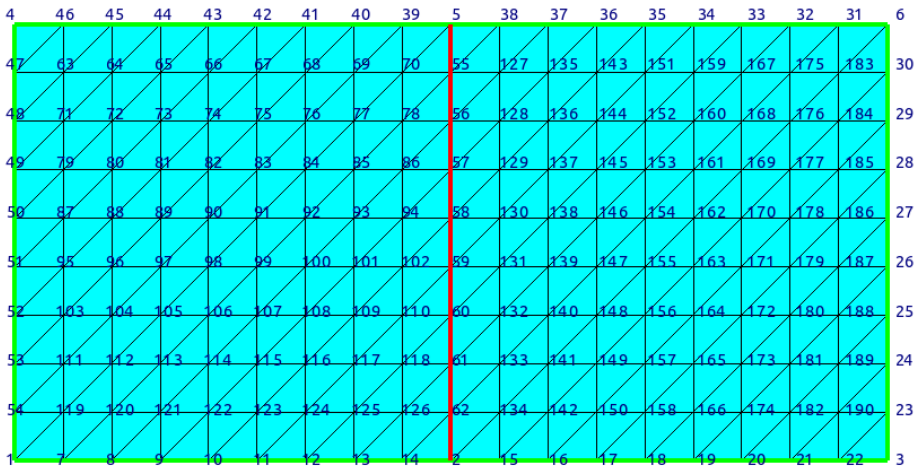


(b) Dirichlet Domain gmsh

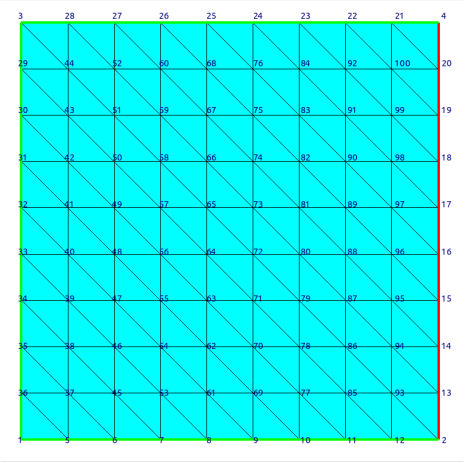


(c) Neumann Domain gmsh

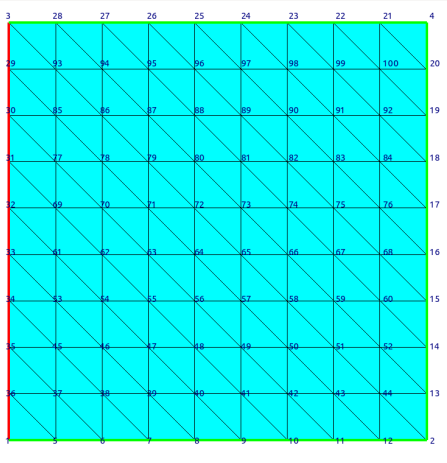
Figure 6.2.: Domain Geometry by Gmsh.



(a) Entire Domain ElmerGUI



(b) Dirichlet Domain ElmerGUI



(c) Neumann Domain ElmerGUI

Figure 6.3.: Domain Geometry by ElmerGUI.

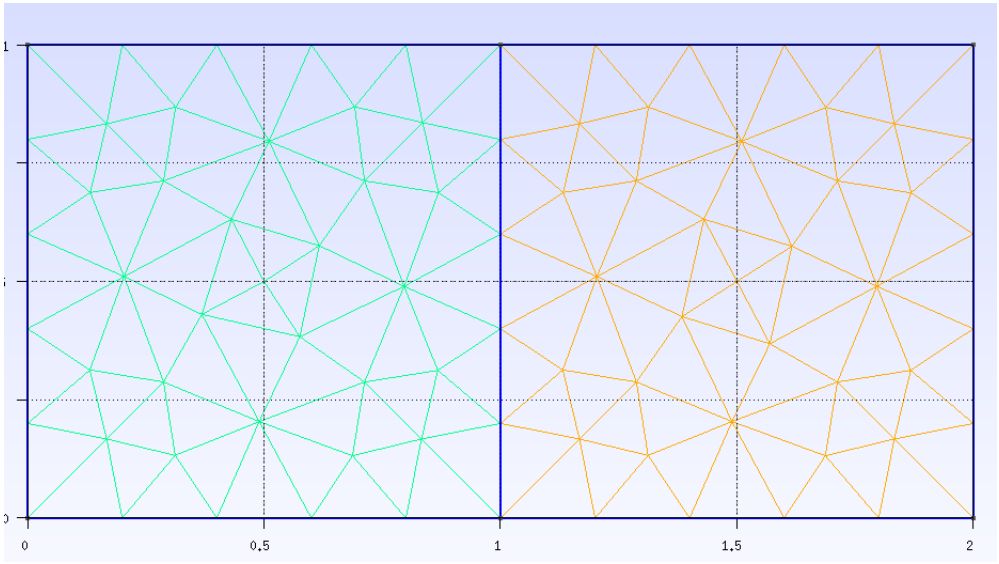


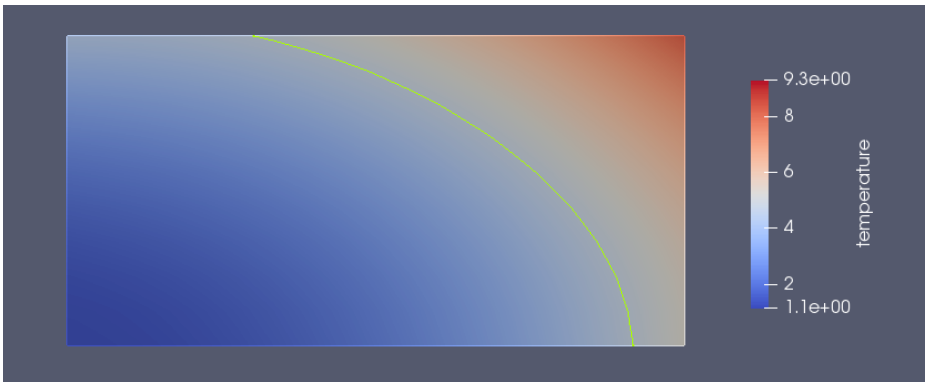
Figure 6.4.: Isotropic triangles mesh

Appendix for further details.

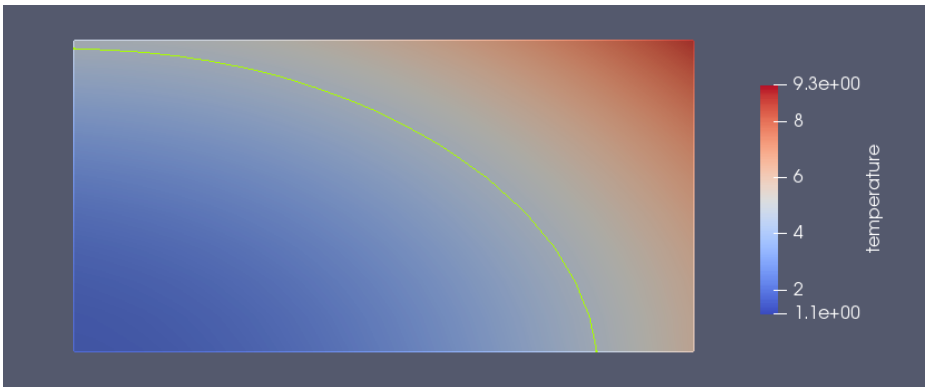
6.3. Results

In this section simulation results of the coupled partitioned heateq olate using Elmer and FEniCS as mentioned in Section 4.2 . First the result of the monolithic simulation of the heat plate as a reference that will be used later for comparision, as shown in Fig. 6.5, the problem is simulated for 1 second with timestep 0.1 second, and computed data are written to output file each time step, the data presented in 6.5 is showing computed data at 1st,5th and 8th time step, this means at $t = 0.1$,0.5 and 0.8 sec. As shown the result ranges from 1 degree celsius to 9.3 degree celsius and the contour lines are continuous across the domain since it is not a coupled simulation, this result should be close to the analytical solution presented in Section 4.1, as will be shown later. The contour line is for 4.63 degree celsius, there is no specific reason for choosing this value, it was suggested by paraview since it is a n average value at the first time step.

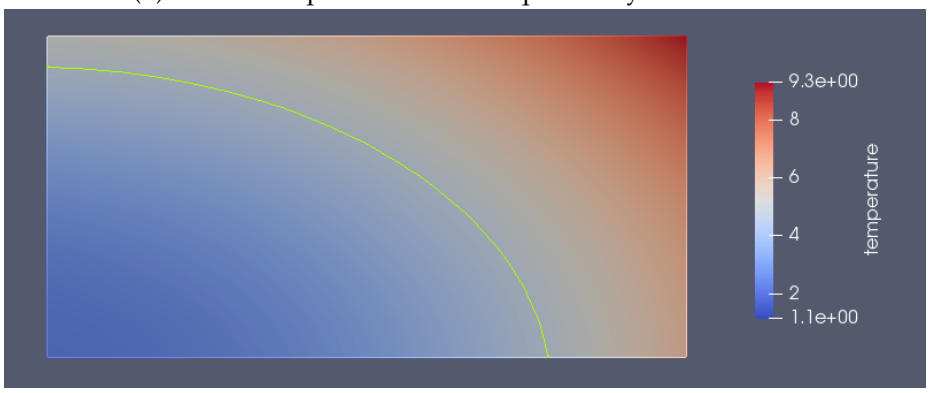
Then results of the coupled partitioned heated plate is shown in Fig. 6.6, similary to the monolithic case, 3 time steps are shown, both Dirichlet and Neumann part are simulated by Elmer, the simulation quality is good enough as in the 1st and 5th time step the contour lines are almost continuous, but the 8th time step the is a data shift at the coupling interface, the result for other time steps are alternating between good continuty and discontinuty, for further detail the whole data for the simulation can be obtained by running



(a) Monolithic partitioned heat equation by Elmer at t = 1



(b) Monolithic partitioned heat equation by Elmer at t = 5



(c) Monolithic partitioned heat equation by Elmer at t = 8

Figure 6.5.: Monolithic partitioned heat equation by Elmer, contour line(green) for temperature = 4.63.No partitioning, continuous solution across domain

```
Solver 2
 Exec Solver = after timestep
 Equation = "flux compute"
 Procedure = "FluxSolver" "FluxSolver"
 Calculate Flux = Logical True
 Flux Variable = String Temperature
 Flux Coefficient = String "Heat Conductivity"
 Linear System Solver = "Iterative"
 Linear System Iterative Method = "cg"
 Linear System Preconditioning = ILU0
 Linear System Residual Output = 10
 Linear System Max Iterations = Integer 500
 Linear System Convergence Tolerance = 1.0e-10
End
```

Source Code 6.4.: flux compute module

the tutorial from the repos [2]. Again the contour lines here are for $temperature = 4.63$.

Then results of the coupled partitioned heated plate is shown in Fig. 6.6, similary to the monolithic case, 3 time steps are shown, Dirichlet is simulated by Elmer and the Neumann part of the domain is simulated by FEniCS, the simulation quality is good enough but slighty worse than Elmer-Elmer coupled case as in the shift appears from the first time step, for further detail the whole data for the simulation can be obtained by running the tutorial from the repos 2. Again the contour lines here are for $temperature = 4.63$.

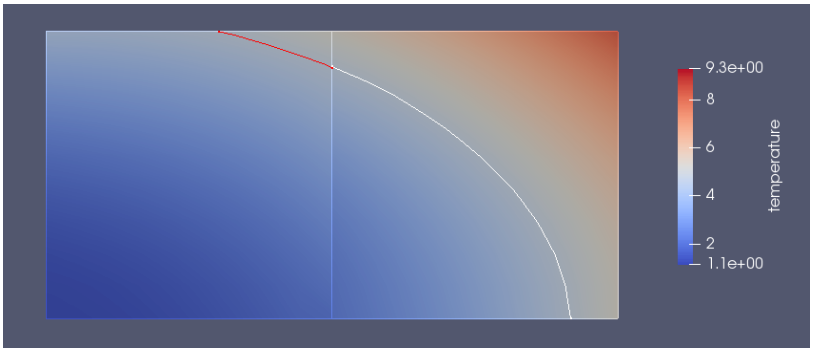
And now the comparision of the coupled simulation to a reference solution to judge the quality of the simulation. In Fig. 6.8, from the vtk colors the green and yellow(colors 2 and 4 respectively) are the analytical solution , the white and red (colors 0 and 1 respectively) are the Elmer-Elmer coupled solution and the magenta (color 5) is the FEniCS-FEniCS coupled solution at 8th time step. As shown in Fig. 6.8a the coupling qualtiy is relatively good and is pretty close to the FEniCS-FEniCS coupled solution. And if we look at Fig.6.8b, the qulaity is also good enough but it is quite clear that Elmer-Elmer coupling is better but overall both simulation gives realistic results.

Moving to Fig. 6.9, the only difference in the coloring is the replacement of the magenta color (color 5) with turkoise color(color 6) to represent monolithic solution, here also both solution looks realistic, the only curious result is how Elmer-FEniCS coupling is closer to monolithic solution in Neumann domain, but this can be judged by calculating an error norm not only by the contour line.

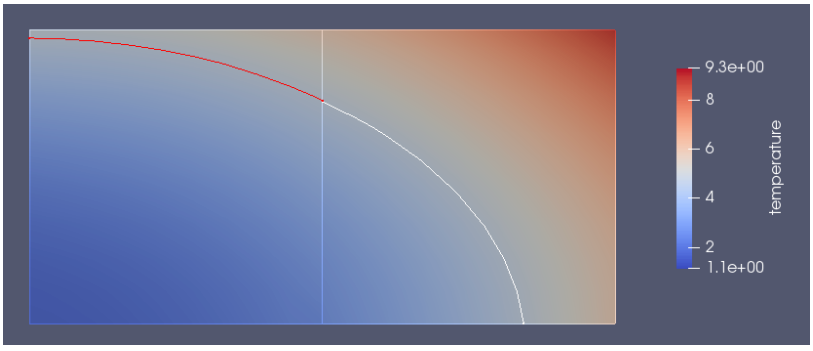
[2] https://github.com/HishamSaeed/elmer-adapter/tree/develop

```
Solver 3
    Equation = "Initialize"
    Procedure = "../Coupler_Solver.so" "CouplerSolver"
    readDataName = String "temperature flux 1"
    writeDataName = String "Temperature"
    Exec Solver = before all
End

Solver 4
    Equation = "ReadData"
    Procedure = "../Coupler_Solver.so" "CouplerSolver"
    readDataName = String "temperature flux 1"
    writeDataName = String "Temperature"
    Exec Solver = before timestep
End

Solver 5
    Equation = "writeDataAdvance"
    Procedure = "../Coupler_Solver.so" "CouplerSolver"
    readDataName = String "temperature flux 1"
    writeDataName = String "Temperature"
    Exec Solver = after timestep
End

Solver 6
    Equation = "Finalize"
    Procedure = "../Coupler_Solver.so" "CouplerSolver"
    readDataName = String "temperature flux 1"
    writeDataName = String "Temperature"
    Exec Solver = after all
End
```

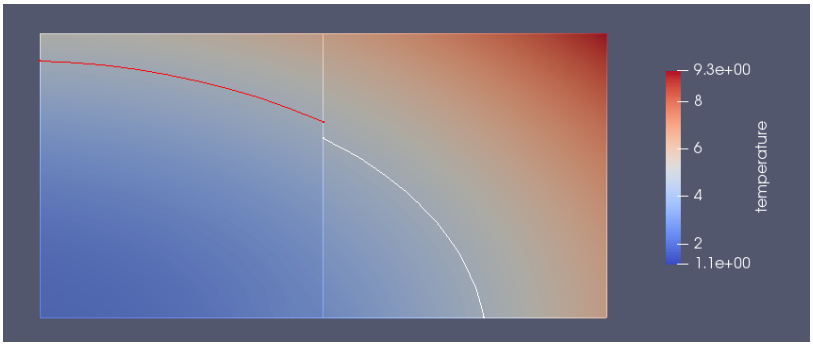
Source Code 6.5.: adapter calling modules



(a) Coupled partitioned heat equation by Elmer at t = 1

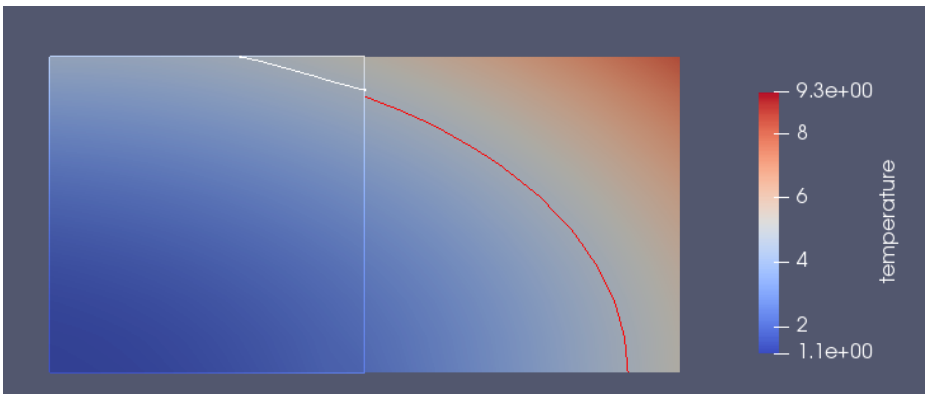


(b) Coupled partitioned heat equation by Elmer at t = 5

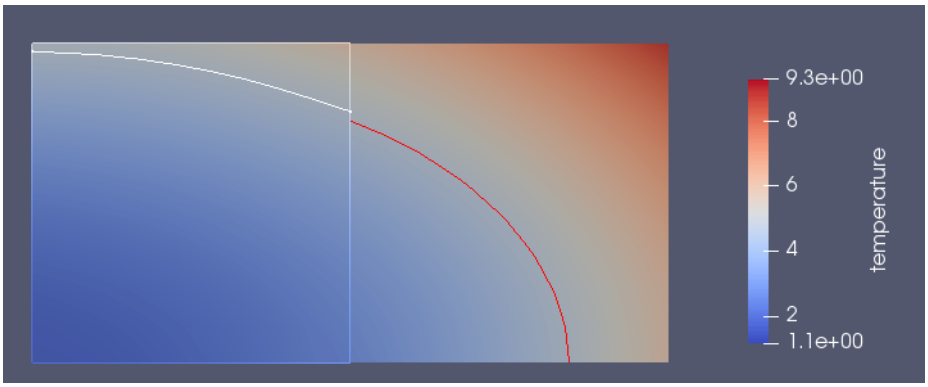


(c) Coupled partitioned heat equation by Elmer at t = 8, contour line jump at coupling interface

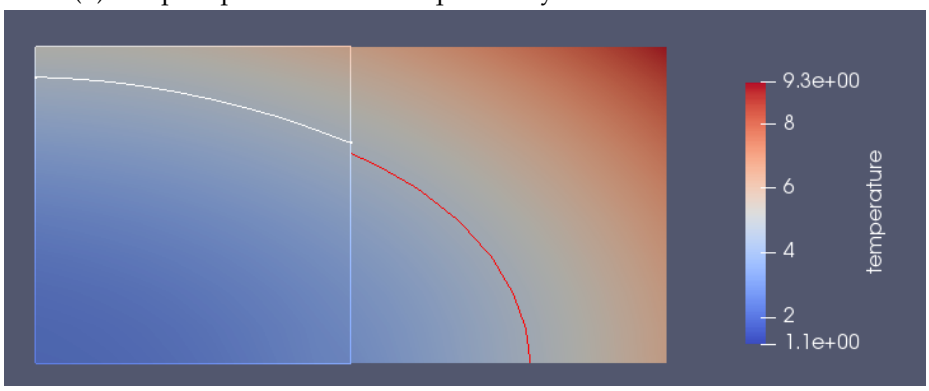
Figure 6.6.: Coupled partitioned heat equation by Elmer.Contour line temperature= 4.63, solution not continuous across coupling interface at t=8, whole solution is coupled but for t= 1 and 5, the quality was good enough so it is nearly continuous across coupling interface



(a) Coupled partitioned heat equation by Elmer and FEniCS at t = 1

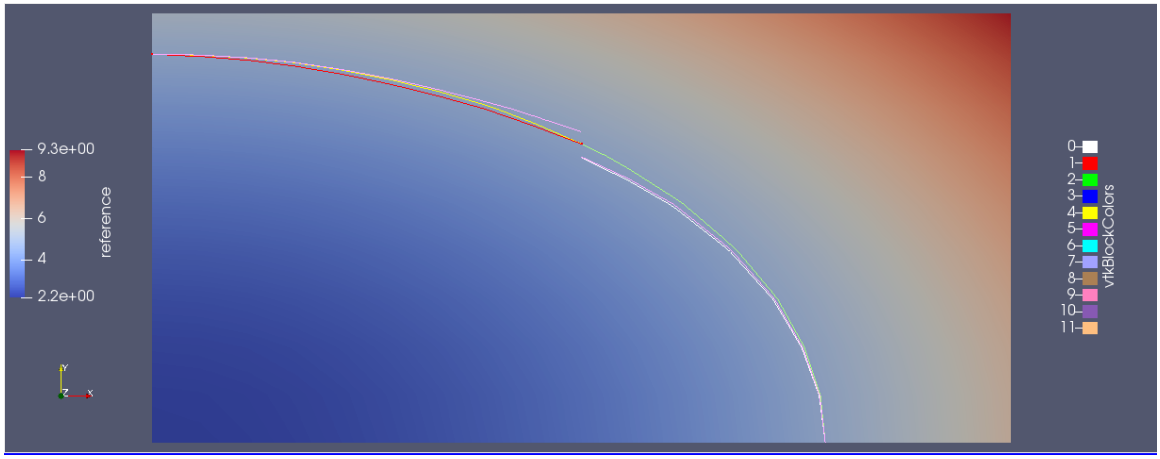


(b) Coupled partitioned heat equation by Elmer and FEniCS at t = 5

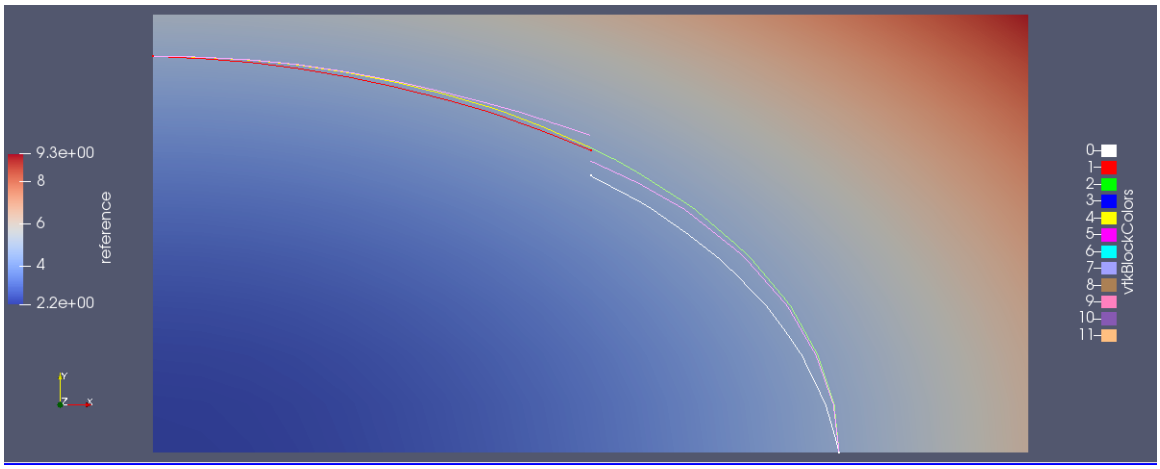


(c) Coupled partitioned heat equation by Elmer and FEniCS at t = 8

Figure 6.7.: Coupled partitioned heat equation by Elmer and FEniCS.Contour line temperature= 4.63, solution not continuous across coupling interface for all shown timesteps

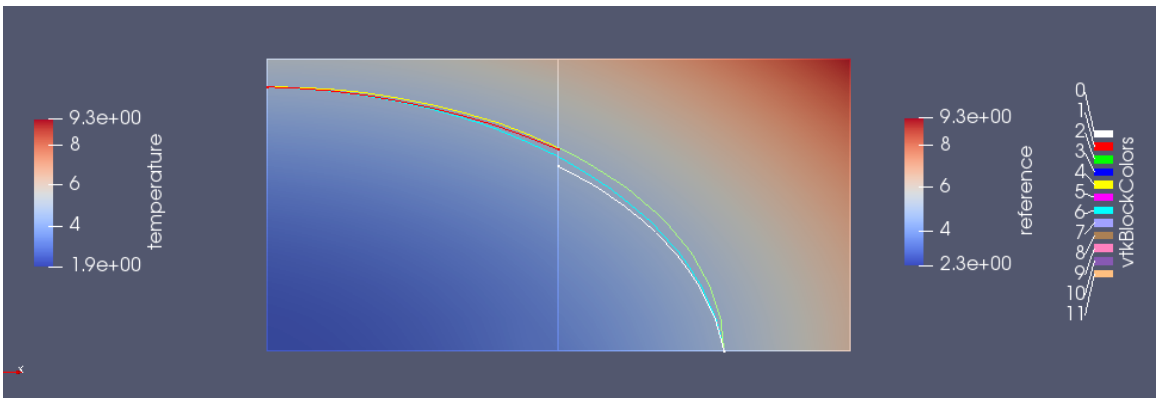


(a) Comparasion between Elmer-Elmer coupling with both analytical and FEniCS-FEniCS coupled solution at t = 8, magenta color(7) represents FEniCS-FEniCS simulation,white and red color(0 and 1) Elmer-Elmer simulation, green and yellow color(2 and 4) analytical solution

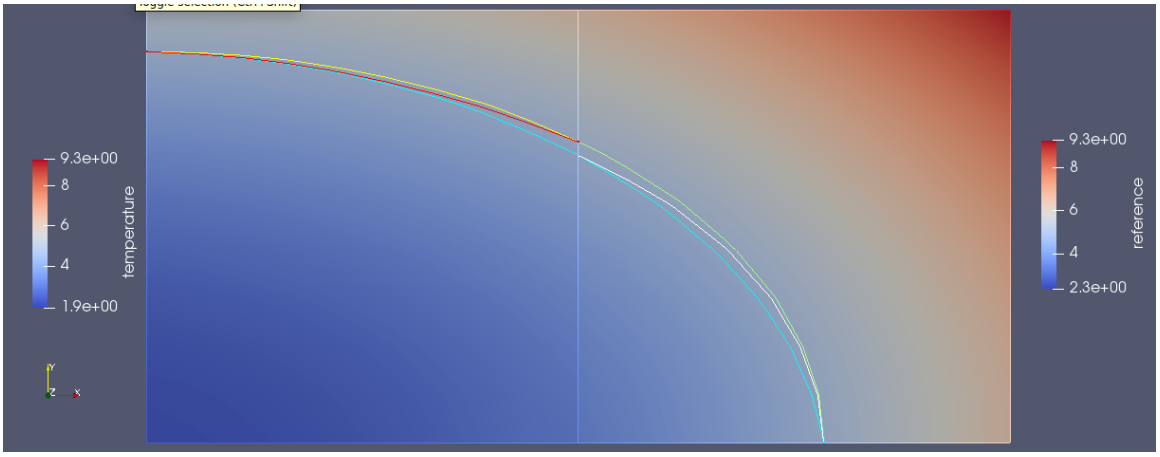


(b) Comparasion between Elmer-FEniCS coupling with both analytical and FEniCS-FEniCS coupled solution at t = 8,magenta color(7) represents FEniCS-FEniCS simulation,white and red color(0 and 1) Elmer-FEniCS simulation, green and yellow color(2 and 4) analytical solution

Figure 6.8.: Coupled partitioned heat equation by Elmer and FEniCS Comparision.



(a) Comparasion between Elmer-Elmer coupling with both analytical and monolithic solution at t = 8,turkoise color(6) represents monolithic simulation,white and red color(0 and 1) Elmer-Elmer simulation, green and yellow color(2 and 4) analytical solution



(b) Comparasion between Elmer-FEniCS coupling with both analytical and monolithic solution at t = 8,turkoise color(6) represents monolithic simulation,white and red color(0 and 1) Elmer-FEniCS simulation, green and yellow color(2 and 4) analytical solution

Figure 6.9.: Coupled partitioned heat equation by Elmer and FEniCS Comparision.

```
Boundary Condition 2
  Target Boundaries(1) = 2
  Name = "Dirichlet_Coupling"
  Temperature = Equals "Temperature"
  Coupler Interface = Logical True
End
```

Source Code 6.6.: Dirichlet Boundary condition

```
Boundary Condition 1
  Target Boundaries(1) = 2
  Name = "Neumann_Coupling"
  Heat Flux = Equals "temperature flux 1"
  Coupler Interface = Logical True
End
```

Source Code 6.7.: Neumann Boundary condition

7. Flow Over Heated Plate

This chapter deals with the second example problem that represents different application.The problem in hand is a simple demonstration for the CHT application, it is a small tutorial but different to the partitioned heat equation as it does not only involve domain partition and coupling but also include multiphysics problem, here both heat equation and Navier-Stokes equation are considered, one part of the domain is purely based on Heat equation, while the other part contains bothe the heat eqution and Navier-Stokes equation, further explaination will be introduced in the problem setup part.

7.1. Mathematical Theory

The heat equation has been presented in the partitioned heat equation problem so no need to present it again. As per Navier-Stokes equation, it describes the flow of fluids through a certain domain and contains information about the flow velocity and pressure. One nice feature about it, is that it can be coupled with the heat equation to see how the fluid flow is affected by the heat transfer across the flow and temperature change of the fluid, this will be shown after the equation represetnation.Navier-Stokes can descibe both compressible and incompressible flows but in the presented study only incompressible flow is considered. The Navier-Stokes equation depends on 2 main concepts the conservation of mass and momentum, thes physical laws are modeled by the continiuty and momentum equations respectively, such equations are not presented here as the Navier-Stokes derivation is not necessary, but it was worth mentioning.

$$\vec{\nabla}.\vec{u} = 0 \tag{7.1}$$

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u}.\vec{\nabla})\vec{u} = -\vec{\nabla}p + \nu\Delta\vec{u} + \rho\vec{g} \tag{7.2}$$

where $\vec{u}$ is the vector valued velocity in the three dimensions, and p is the pressure, $\vec{g}$ is the vector valued volume force, in our case the gravitational force and ν is the inverse of Reynolds number, which describes kinematic viscosity $\nu = \frac{\mu}{\rho}$. later the gravitational forces will play a role in coupling with heat equation. just to avoid confusion also the equation presented in Elmer models maunal is presented because it is writeen in a different form, so it is shown here to imply that both descibe the same thing.

$$\rho(\frac{\partial \vec{u}}{\partial t} + (\vec{u}.\vec{\nabla})\vec{u}) - \nabla.(2\mu\bar{\bar{\epsilon}}) + \vec{\nabla}p = \rho f \tag{7.3}$$

Equation 7.3 is the same as equation 7.2 only f replaces g, pressure moved to the left-hand side, and the diffusive term which includes ν kinematic viscosity, is replaced by $\bar{\bar{\epsilon}}$ which is lineraized tensor strain, for further details please check Elmer models manual. To include the effect of the heat transfer in flow simulation, we will use the Boussinesq approximation, which describe density of the fluid in a linear relation to the temperature, so first we have the heat equation shown in 7.4

$$\rho c_p(\frac{\partial T}{\partial t}) - \nabla.(k\nabla T) = \rho h \tag{7.4}$$

Then the desnity can be calculate through Boussinesq linear relation from the computed temperature as shown in equation 7.5

$$\rho = \rho_0 - \beta\rho_0(T - T_0) \tag{7.5}$$

where T_0 and ρ_0 are the reference temperature and density respectively, β is the thermal expansion coefficient and can be calculated through density variation with respect to temperature using relation

$$\beta = -\frac{1}{\rho_0}\frac{\partial \rho}{\partial T} \tag{7.6}$$

or it can be assumed to be constant. And hence our Navier -Stokes equation becomes dependent on temperature through density

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u}.\vec{\nabla})\vec{u} = -\vec{\nabla}p + \nu\Delta\vec{u} + \rho(T)\vec{g} \tag{7.7}$$

7.2. Problem Setup

The problem described here is a small demonstration of the conjugate heat transfer applications, it shows a fluid enters a pipe at certain temperature, and under the pipe at a certain position, a metal plate is placed, and heat is exerted on the bottom of this metal plate. By consequences, heat should travel from top to bottom and start affecting the fluid flow temperature. The domain is divided into 2 parts the metal plate which is called solid part of the domain designated by *S(u)* , and the pipe part is called fluid domain designated by *F(q)*. The solid domain only deals with the heat equation, but the fluid part deals with multiphysics problem containing the heat equation as well as the Navier-Stokes equation. Similarly to the partitioned heat equation, the data is comunicated through boundary condition; the solid part recieves the communicated data as dirichlet boundary condition which is the computed temperature at the coupling interface at the fluid part, then it computes the heat flux and transfer it to the fluid part as Neumann boundary condition, for the sake of simplicity, here the coupling interface is not inclined, so only the y-direction of the heat-flux is required.

The problem is set to run with Elmer for both fluid and solid participant as well as coupling Elmer with openFoam,where Elmer simulates the solid part and compute the

```
Boundary Condition 3
  Target Boundaries(1) = 3
  Name = "Coupling Interface"
  Temperature = Equals "Temperature"
  Coupler Interface = Logical True
End
```

Source Code 7.1.: Dirichlet Boundary condition

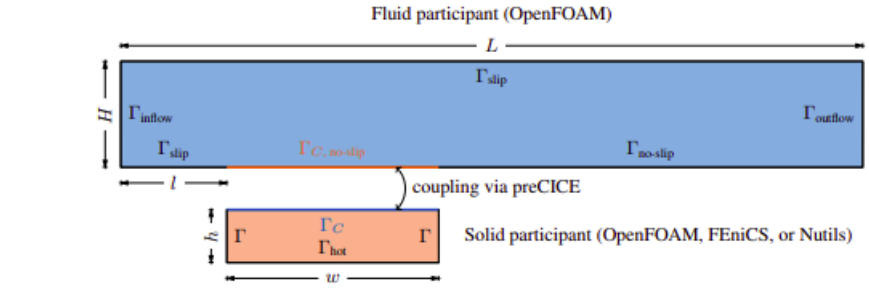
heat-flux while openFoam simulates the fluid part. so the existing combinations for the simulation are as follows.

- Elmer for *S(u)* and Elmer for *F(q)*. Elmer-Elmer coupling.
- Elmer for *S(u)* and openFoam for *F(q)*. Elmer-openFoam coupling openFOAM code taken from [1].
- FEniCS for *S(u)* and openFoam for *N(q)*. FEniCS-openFoam coupling openFOAM code taken from 1. This is one of the reference results to compare to, since it is a working tutorial by preCICE , code can be found in 1, problem taken from [3].

Also a monolithc run for the whole domain without coupling done by Elmer is tried out and used as reference. Similar to partitioned heat equation, problem preparations require mesh files as well as the system input file. For openFoam preparation please refer to preCICE tutorials, as the same fluid part is used in this problem. Also for this problem several problem parameters are required, the domain dimension as well the requred values are depicted in Fig. 7.1, the dimension of each domain is described as well as the required boundary conditions.

The whole domain construction in gmsh is shown in Fig. 7.2a and the geometry as wel as the mesh is depicte in Fig. 7.2 emphasizing the coupling interface. Similar to the partitioned heat equation, the adapter has to be called through dummy solver, also the boundary conditions has to be adjusted at the coupling interface to make sure the data is communicated correctly, this is demonstrated in source code 7.1 for the solid part of the domain as example and the fluid part of the domain is the same as Neumann part in the partitioned heat equation.

[1] https://github.com/precice/tutorials/tree/6d90ae3eb179113beb400e472c8832e29a48db9b/flow-over-heated-plate



Dimensions		
L	3.50	m
H	0.50	m
l	0.50	m
h	0.25	m
w	1.00	m
z	0.40	m

Fluid			
u_{inflow}	1	$\times 10^{-1}$	$\mathrm{m\,s^{-1}}$
T_{inflow}	3	$\times 10^{2}$	K
p_0	1.035	$\times 10^{5}$	Pa
g	9.81		$\mathrm{m\,s^{-2}}$
k_F	1	$\times 10^{2}$	$\mathrm{W\,m^{-1}\,K^{-1}}$
$c_{p,F}$	5	$\times 10^{3}$	$\mathrm{J\,kg^{-1}\,K^{-1}}$
μ	2	$\times 10^{-4}$	$\mathrm{kg\,m^{-1}\,s^{-1}}$
ρ_F	1		$\mathrm{kg\,m^{-3}}$
ν^*	2	$\times 10^{-4}$	$\mathrm{m^2\,s^{-1}}$
Pr^*	1	$\times 10^{-2}$	
M^*	24.10		$\mathrm{kg\,mol^{-1}}$

Solid			
T_{hot}	3.1	$\times 10^{2}$	K
k_S	1	$\times 10^{2}$	$\mathrm{W\,m^{-1}\,K^{-1}}$
$c_{p,F}$	1	$\times 10^{2}$	$\mathrm{J\,kg^{-1}\,K^{-1}}$
ρ_S	1		$\mathrm{kg\,m^{-3}}$
α_S^*	1		$\mathrm{m^2\,s^{-1}}$

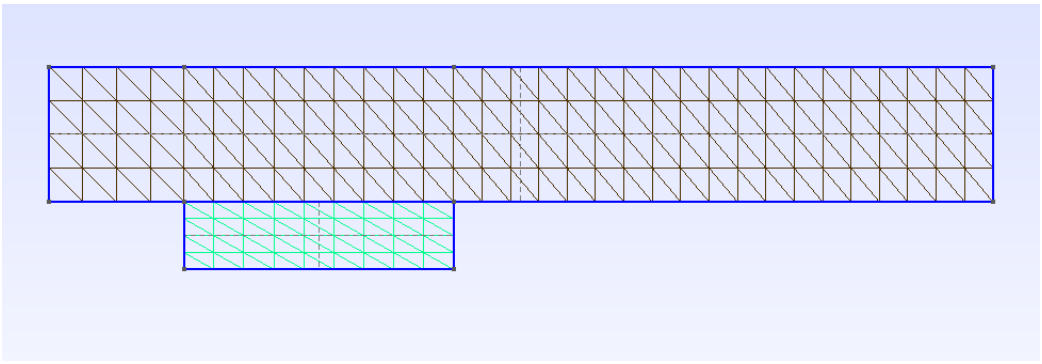
Figure 7.1.: Flow over heated plate problem parameters and configuration, image from [3]

7.3. Results

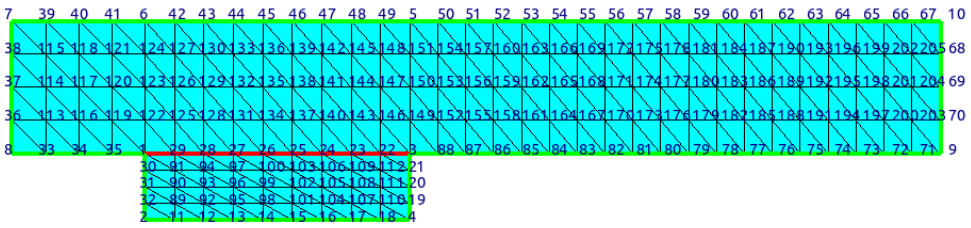
In this section the simulation results are presented and discussed. As usual we start with the monolithic run of the whole domain which will be later used as reference to compare with the coupled result.In Fig. 7.3a the heat distribution is shown, it can be noticed that heat flows from bottom to top, and starts affecting the temperature of the flow in the pipe. From the problem setup, the flow enters the pipe at temperature 300 kelvin, the contour line depicted in Fig. 7.3a is for 305 Kelvin. Also such an effect can be seen in the coupled case where Elmer simulates both the solid and fluid part, the coupled simulation results are depicted in Fig. 7.3b. Again te contour lines are for 305 Kelvin, and the fluid enters at 300 Kelvin, and the effect of the heat transfer across the fluid can be noticed.

Next is to couple Elmer with openFoam, Elmer simulates the solid part and openFoam simulates the fluid part, the results are shown in Fig. 7.4a and the temperature starts to rise across the fluid. Also similar effect can be noticed by coupling openFoam with FEniCS, where FEniCS replaces Elmer for the solid part, the results are shown in Fig. 7.4b. the contour lines in both cases is for 303 Kelvin , the effect here is not major as in previous cases as the mesh granularity is different which is shown in the comparision.

For the comparision, coupled simulation by Elmer simulating both parts is compared with monolithic case in Fig. 7.5 the white color is the monolithic results, with contour lines for 305 Kelvin, the coupled simulation is not continuous at the right part of the domain. As seen in Fig. 7.5b the grid is too coarse. Same for comparing the coupled simulation be-



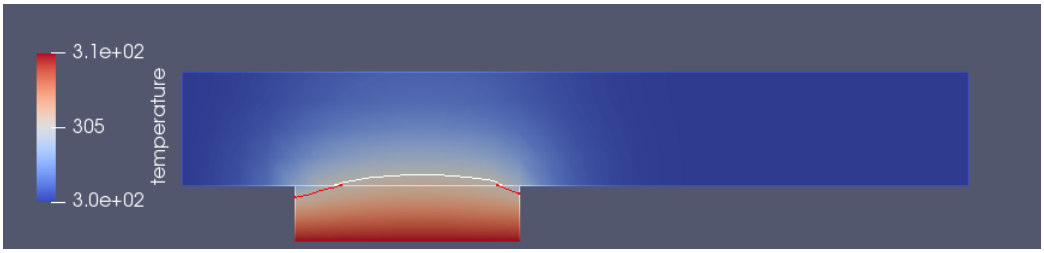
(a) Entire problem domain defined in gmsh



(b) Entire problem domain defined in gmsh

Figure 7.2.: Entire problem geometry and mesh.

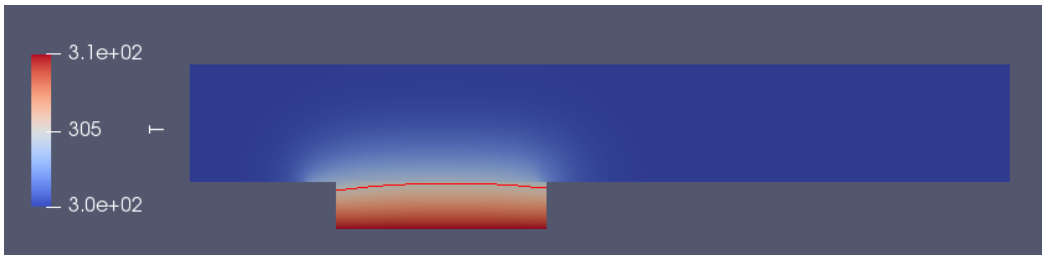
(a) monolithic flow over heated plate simulated by Elmer



(b) Coupled flow over heated plate both fluid and solid simulated by Elmer

Figure 7.3.: Monolithic and Coupled Flow Over Heated Plate.

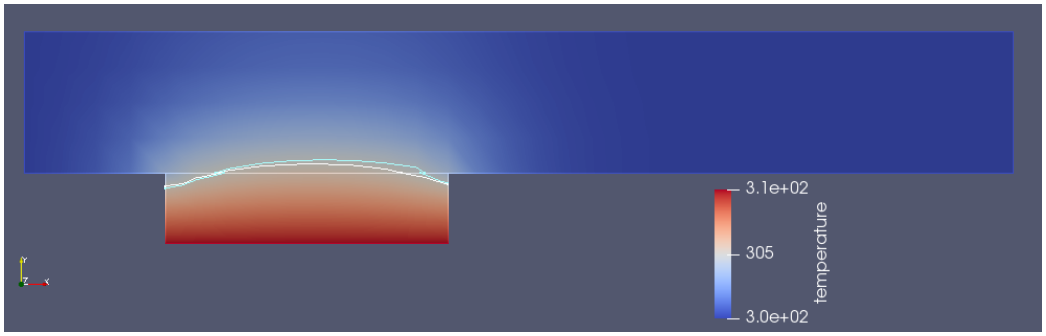
(a) Coupled flow over heated plate, fluid simulated by openFoam and solid simulated by Elmer



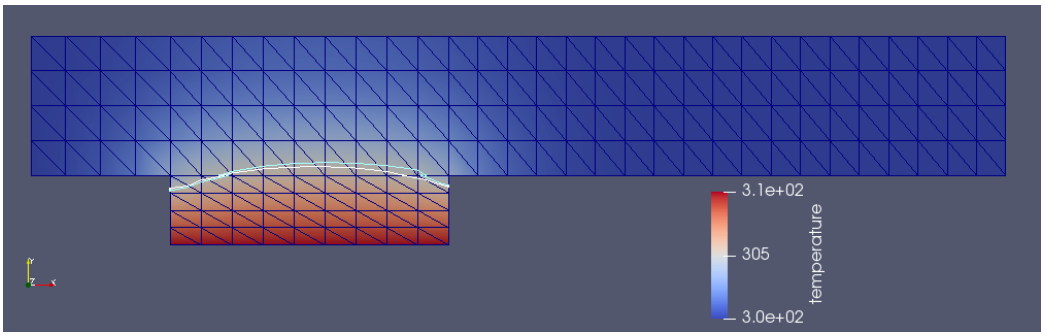
(b) Coupled flow over heated plate, fluid simulated by openFoam and solid simulated by FEniCS

Figure 7.4.: Coupled Flow Over Heated Plate.

tween Elmer and openFoam, but this time it is compared with FEniCS-openFoam coupled case.Here it can be noticed that the mesh is too fine, that is why the calculation is different from monolithic case.

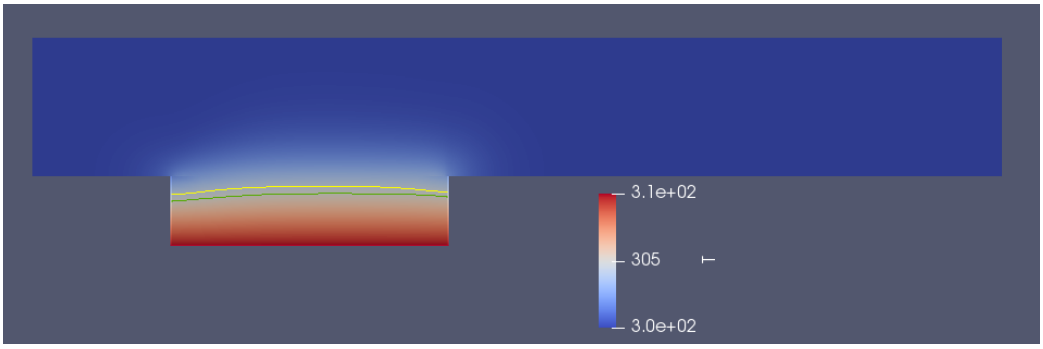


(a) Comparision between monolithic solution and coupled solution both fluid and solid simulated by Elmer

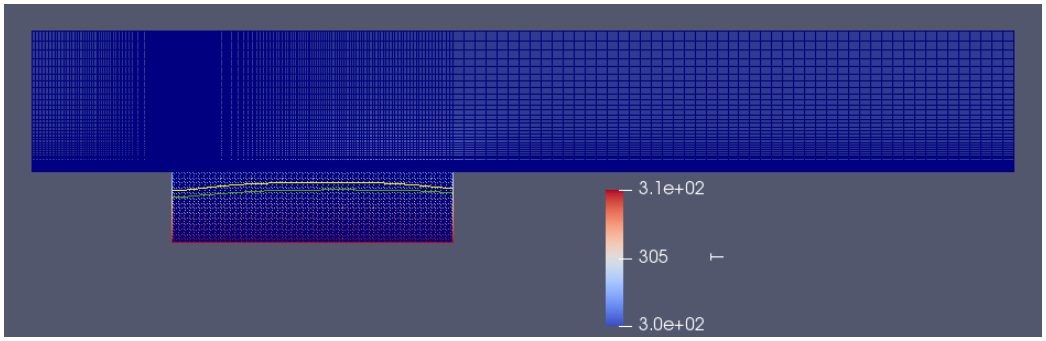


(b) Coupled and monolithic soultion grid

Figure 7.5.: Coupled Flow Over Heated Plate comparision with monolithic solution.



(a) Comparision between coupled solution Elmer-Foam and coupled solution FEniCS-Foam



(b) Coupled FEniCS-Foam grid

Figure 7.6.: Coupled Flow Over Heated Plate comparision with different solvers combination.

Part IV.

Conclusion and Outlook

8. Outlook

This las chapter is only to provide uncomplete work for further adapter development.Also provide some recomendations about how to procede and the challenges opposed in further applications.

8.1. Perpendicular flap: Preliminary results and open points

In Part III, the conducted study and its results as well as its limitations was presented. This study was done on 2 example problems that represents real applications; the heat transfer in partitioned heated plate and CHT(conjugate heat transfer) in flow over heated plate. One more important application is FSI(fluid structure interaction), this is an important application as it simulates how a solid behave when it is subjected to fluid flow, this phenomena exists in several application, the most common one is in aerospace, where the areoplane wing vibrates due to ari flow over it. Also it appears when simulating blood flows. For this a tutorial was presented by preCICE developers for fluid structure interaction which is called Perpendicular flap [1] which studies how an elastic flap behave when it is subjected to fluid flow as shown in Fig. 8.1.Also this problem is introduced in Elmer tutorials [10] tutorial 24.

The aim was to make sure that the adapter support fluid structure interaction, so a problem is developed a coupled simulation for perpendicular flap .We started with monolithic model for perpendicular flap simulated by Elmer as shown in Fig. 8.2. The result of the monolithich simulation is shown in Fig. 8.3, as noticed the elastic flap is displaced slightly with the flow. The parameters for the simulation is the same as in the tutorial 1, the result is for a slightly different parameters for the fluid and elastic flap but the dimensions are the same. It is recommended for reproducing the problem to use the same parameters as the tutorial. Such a problem contains several modules, equations; first is Navier-Stokes equation for the fluid flow simulation, then linear elasticity model for computing the displacement of the elastic beam and at last a module for computing the mesh displacement, this where the challenge lies, the coupling interface is a displaced mesh, transfering such data is tricky especially if the elastic beam contains some sides which have slope. The problem is that we start investigating a curved elastic beam similar to Elmer tutorial, then we switched to the perpendicular flap. The transfer of displacement and having a moving

[1] https://github.com/precice/tutorials/tree/6d90ae3eb179113beb400e472c8832e29a48db9b/perpendicular-flap

mesh in both domains as well as having a reference problem is what form an obstacl at the current status.

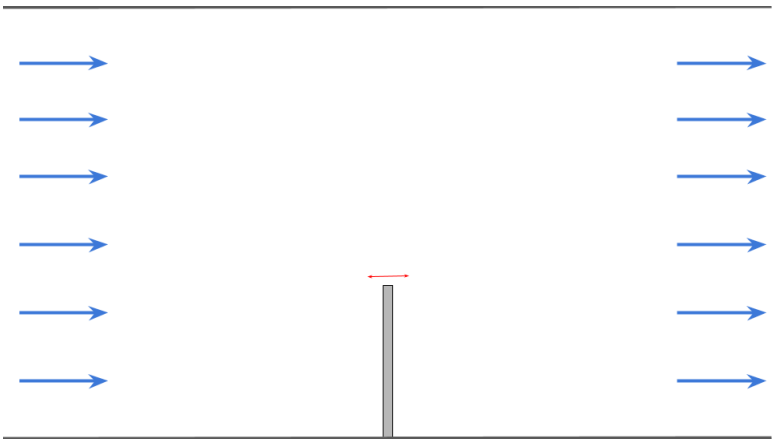


Figure 8.1.: perpendicular flap setup, from 1

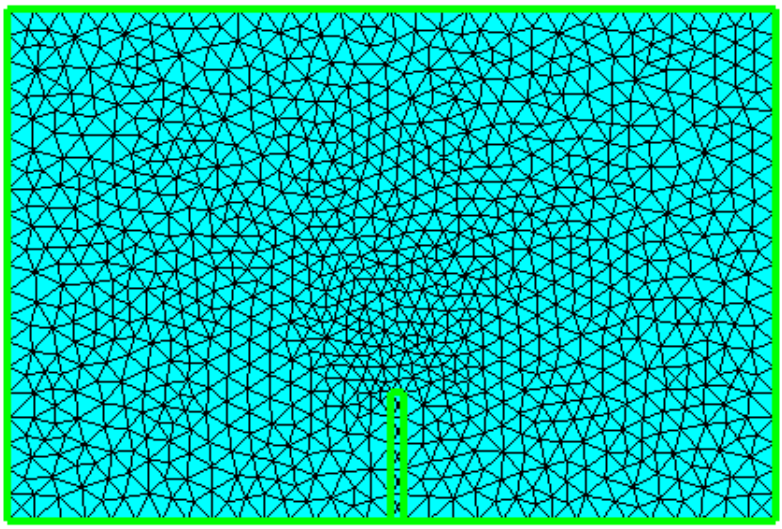
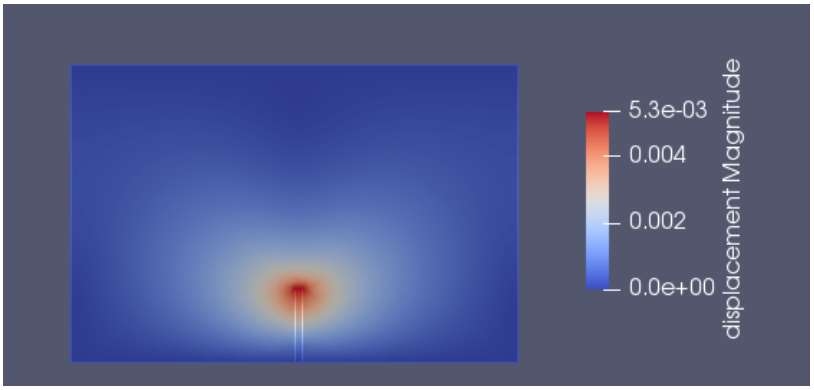
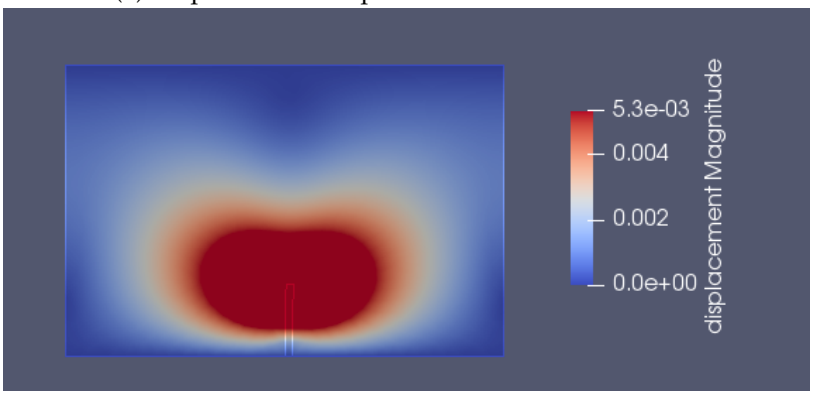


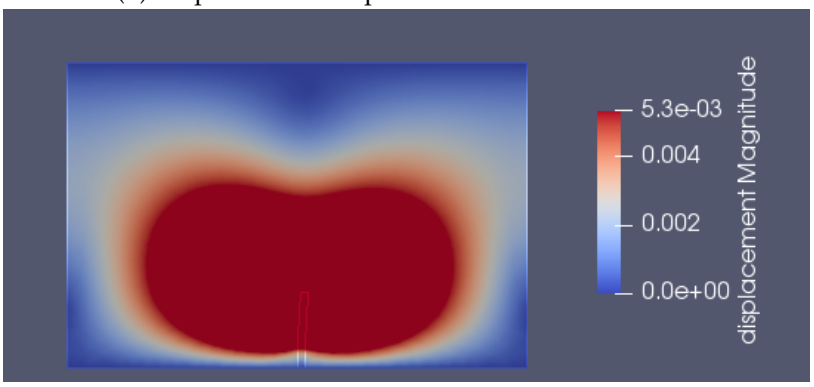
Figure 8.2.: perpendicular flap in elmerGUI



(a) Perpendicular Flap monolithic solution at t = 3



(b) Perpendicular Flap monolithic solution at t = 6



(c) Perpendicular Flap monolithic solution at t = 10

Figure 8.3.: Perpendicular Flap monolithic solution.

8.2. Conclusion

After presenting the results in chapters 6 and 7, the prototype of the adapter can achieve coupling with other open source softwares, but it has some limitations that requires further development,including implicit coupling, handling inclined coupling interface and displaced meshes at the coupling interface for fluid structure interaction applications After presenting

Bibliography

[1] Hans-Joachim Bungartz, Florian Lindner, Miriam Mehl, and Benjamin Uekermann. A plug-and-play coupling approach for parallel multi-field simulations. *Computational Mechanics*, 55(6):1119–1129, 2015.

[2] Gerasimos Chourdakis. A general openfoam adapter for the coupling library precice. 2017.

[3] Gerasimos Chourdakis, Kyle Davis, Benjamin Rodenberg, Miriam Schulte, Frédéric Simonis, Benjamin Uekermann, Georg Abrams, Hans-Joachim Bungartz, Lucia Cheung Yau, Ishaan Desai, et al. Precice v2: A sustainable and user-friendly coupling library. *arXiv preprint arXiv:2109.14470*, 2021.

[4] KASPARS DADZIS and ARVED ENDERS-SEIDLITZ. Coupled 3d calculation of fluid flow, heat and mass transport, and phase boundary shape for crystal growth processes.

[5] Hans Petter Langtangen and Anders Logg. Solving pdes in minutes–the fenics tutorial volume i. 2016.

[6] Peter Raback. Elmer grid manual. http://www.nic.funet.fi/pub/sci/physics/elmer/doc/ElmerGridManual.pdf.

[7] Peter Raback, Mika Malinen, Juha Ruokolainen, Antti Pursula, and Thomas Zwinger. Elmer models manual. http://www.nic.funet.fi/pub/sci/physics/elmer/doc/ElmerModelsManual.pdf.

[8] Benjamin Rodenberg, Ishaan Desai, Richard Hertrich, Alexander Jaust, and Benjamin Uekermann. Fenics-precice: Coupling fenics to other simulation software. *arXiv preprint arXiv:2103.11191*, 2021.

[9] Juha Ruokolainen, Mika Malinen, Peter Rback, Thomas Zwinger, Antti Pursula, and Mikko Byckling. Elmer solver manual. http://www.nic.funet.fi/pub/sci/physics/elmer/doc/ElmerSolverManual.pdf.

[10] Juha Ruokolainen, Peter Raback, and Thomas Zwinger. Elmer tutorials. http://www.nic.funet.fi/pub/sci/physics/elmer/doc/ElmerTutorials.pdf.

[11] Benjamin Rüth, Benjamin Uekermann, Miriam Mehl, Philipp Birken, Azahar Monge, and Hans-Joachim Bungartz. Quasi-newton waveform iteration for partitioned surface-coupled multiphysics applications. *International Journal for Numerical Methods in Engineering*, 2020.