Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Coupling of macro and micro scale in a continuum-biomechanical model of the human liver using preCICE

Fritz Otlinghaus.

Course of Study:	M.Sc. Simulation Technology
Examiner:	Jun.-Prof. Dr. Benjamin Uekermann
Supervisor:	Ishaan Desai, Steffen Gerhäusser
Commenced:	Febuary 1, 2022
Completed:	August 1, 2022

Abstract

As the human liver is one of the most important organs in the human body, the scientific community aims to improve our understanding of its internal processes to enable better planning for individual liver surgeries. This includes the complex relationship between hepatic tissue and metabolic cell processes.

This work is based on the modeling approach proposed by [RWH+14] where the hepatic tissue is modeled by partial differential equations (PDEs) in a homogenization approach based on the extended Theory of Porous Media (eTPM). These equations are then coupled with ordinary differential equations (ODEs), representing the metabolic processes in the liver cells.
These models are solved by the FEM-solver FEBio for the hepatic tissue, and the ODEs on the cell scale are solved by the biochemical software library libRoadRunner.

This work introduces a new two scale coupling using the coupling library preCICE and the Micro Manager. A new FEBio-preCICE adapter is implemented and compared to an existing coupling. To evaluate the differences in the couplings, three test cases are defined and simulation time, quality and memory usage for each test case is compared. This is followed up by discussing and comparing advantages and disadvantages for each coupling.

Contents

List of Figures

1 Introduction

The human liver is the largest solid organ in the body. It removes toxins from the body's blood supply, maintains healthy blood sugar levels, regulates blood clotting, and is responsible for metabolizing medication. A solid understanding of all effects that occur in the human liver is required to predict possible outcomes of new medication on a patient. Human trials that determine effects of medication and treatment strategies on a human liver are dangerous and expensive in early stages. Simulations are also a suitable, fast and inexpensive approach to gain insights on the biochemical processes in the human liver. If the designed simulations are sufficiently realistic, they would allow individualized functional prediction when planning liver surgeries [Com20].

To fully simulate liver lobule behavior, a two scale coupling between cell metabolism and blood perfusion is required. [RWH+14] introduces such a modeling approach, where isolated biological reactions are modeled on a micro scale and fluid simulations on a macro scale. Using this modeling approach, this work aims to allow closer studies of inter-scale effects in the human liver, by introducing a new coupling approach between both scales using the preCICE [BLG+16] library.

In Chapter 2, the human liver anatomy is described, along with the theory of porous media and the system biological markup language (SBML) [HFSB01]. Chapter 3 presents the liver model from Ricken [RL19] and the cell model from Koenig [Com20].
In Chapter 4, the software components are described, followed by the development strategy for building a prototype coupling. The implementation details and challenges of a preCICE adapter for FEBio are discussed in Chapter 5, as well as the software configuration options. This is followed up by Chapter 6, which details the benchmark cases and outlines interesting features for a comparison between the two coupling approaches.

Chapter 7 then presents the results of the coupling and Chapter 8 discusses the advantages of our preCICE coupling approach in comparison to the existing coupling, and outlines improvements to the development process.

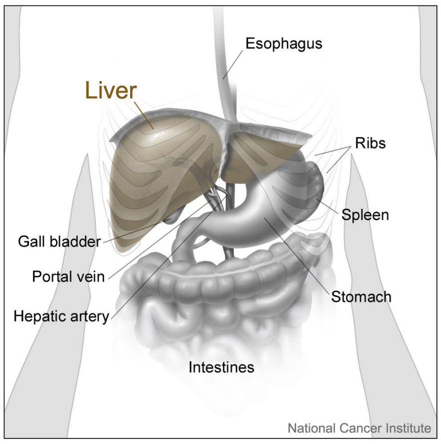
2 Fundamentals of the Human Liver

To understand the big picture idea of this thesis, it is essential to acquire some basic knowledge of the human liver anatomy and its functions, therefore we recap the liver anatomy and explain how to simulate a human liver. This is followed by introductions to the theory of porous media and to the System Biological Markup Language (SBML).

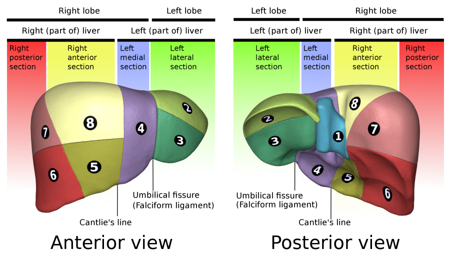
2.1 Liver Anatomy

On average, the human liver weighs one to two kilos and is located in the upper right stomach region, see Figure 2.1a. It is responsible for producing bile, breaking down fats during digestion, converting glucose into glycogen for storage and much more. Physically, the liver can be split into four liver lobes, namely the left lobe, the right lobe, the quadrate lobe and the caudate lobe or into eight parts as described by Couinaud [RK15], which can be seen in Figure 2.1b. In a broad understanding, all parts perform the same functions and are only differentiated through their geographic location.

Each liver lobe consists of so-called liver lobules, which are approximately 1.5 mm wide hexagonal components. Each lobule is supplied by multiple portal triads, one on each corner. The layout can be seen in Figure 2.3. The portal triads supply the lobule with nutrients via a hepatic artery and oxygen rich blood via the portal vein. This oxygen and nutrient rich blood mixture then flows through tiny tunnels, so-called sinosoids (c.f. Figure 2.2) to the central vein. While passing through the sinosoids, the hepatocyte cells, which are located on the tunnel wall, interact with the nutrients in the blood. The third component of a portal triad is the bile duct that gathers bile produced in the lobules and sends the bile towards the intestine, where it is needed for digestion.



(a) The liver and nearby organs and structures [Wik22b]



(b) Segments of liver lobes as classified by Couinaud. [Wik22a]

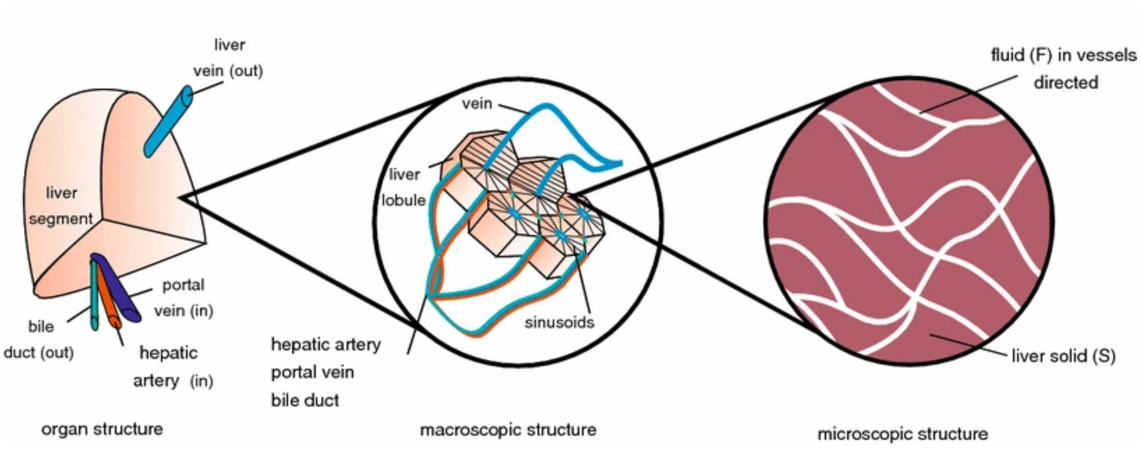


Figure 2.2: Liver Marco meso micro overview. [RWH+14]

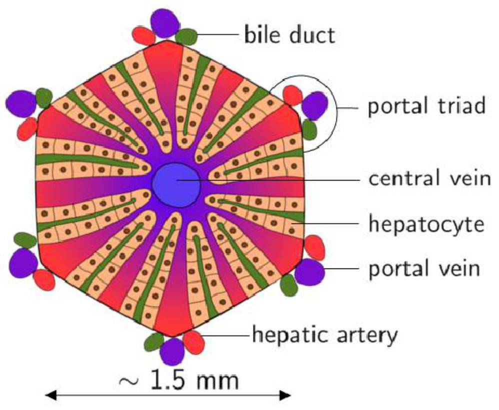


Figure 2.3: Structure of one liver lobules [RWH+14]

2.2 Simulating a Human Liver

As most of the work is performed on the lobule level and the lobules do not interact with each other, the simulation of a single lobule is a first step necessary for simulating a full human liver. To simulate such a lobule, all relevant parts need to be addressed. This is accomplished by [RWH+14], where the lobule is split into a simulation for the blood flow using the theory of porous media, and the hepatocytes are simulated through an SBML model from König.
The theory of porous media, or short TPM, is used for materials where a traditional approach of separating all involved constituents is not feasible. Instead of considering separate constituents, the theory introduces one averaged multiphase volume segment. All balance equations are formulated as a sum containing the volume fractions of the different phases [Ehl96]. The approach of describing a multiphase volume fraction fits ideally to describe blood with different solutes that flows through a solid sponge (sinusoids).

2.3 System Biological Markup Language

The System Biological Markup Language, or SBML for short, is an XML-based format for representing biochemical reaction networks [HFSB01]. Such networks can be used to describe the cell metabolism reaction to certain molecules or the decay of a molecule within the whole body. The standardized language allows for using a variety of solvers that support SBML and can solve the containing equations. In this work, SBML is used for describing how liver cells (hepatocytes) transform the paracetamol molecule apap to napqi, while using the solver, libRoadRunner.

3 Models

This chapter describes the mathematical models used in this work, on both the macro- and micro scale.

3.1 Liver Model

The model we are using in this work is based on the work of Ricken et al [RWH+14] and a work in progress model by the same authors. It models the liver tissue as homogenization structure by using the theory of porous media. There, liver tissue consists out of φ^α constituents, which are averaged over the whole body. Using this theory, the model describes the whole body with Equation (3.1), where φ^S is the tissue(solid) phase and $varphi^F$ the blood(fluid) phase.

$$\varphi = \sum_{\alpha=1}^{K} \varphi^\alpha = \varphi^S + \varphi^F \tag{3.1}$$

To allow for the modeling of solutes in the blood stream that transfer substances to and from the hepatocytes, the model extends Equation (3.1) to the extended theory of porous media (eTPM) Equation (3.2). φ^α denotes the solvent and $\varphi^{\alpha\beta}$ the solute resolved in solvent α.

$$\varphi = \sum_{\alpha=1}^{K} \left[\sum_{\beta=1}^{v-1} (\varphi^{\alpha\beta}) + \varphi^\alpha \right] \tag{3.2}$$

The introduction of the eTPM yields the saturation condition, where n^α stands for the volume fractions (Solid, Fluid).

$$\sum_{\alpha=1}^{K} n^\alpha = 1 \tag{3.3}$$

The model describes the seepage velocity of external components with Equation (3.4). n^F is the volume fraction of the blood , λ the blood pressure and the permeability parameter Equation (3.5).

$$n^F \mathbf{w}_{FS} = \frac{(n^F)^2}{2\gamma^F_{\mathbf{w}_{FS}}} [-grad\lambda] \tag{3.4}$$

$$\gamma^F_{\mathbf{w}_{FS}} = \frac{(n^F)^2}{2k_D} \tag{3.5}$$

A Neo Hookean solid with Helmholtz free energy is used to characterize the liver tissue, as can be seen in Equation (3.6).

$$\Psi^S = \frac{1}{\rho_{0S}^S}\left[\lambda^S \frac{1}{2}(ln\mathbf{J}_S)^2 - \mu^S(ln\mathbf{J}_S) + \frac{1}{2}\mu^S(tr\mathbf{C}_s - 3)\right] \tag{3.6}$$

The interested reader is referred to Ricken et. al [RWH+14], for a motivation and complete introduction of the model. This includes the balance equations for the model.

3.2 Paracetamol Metabolism Cell Model

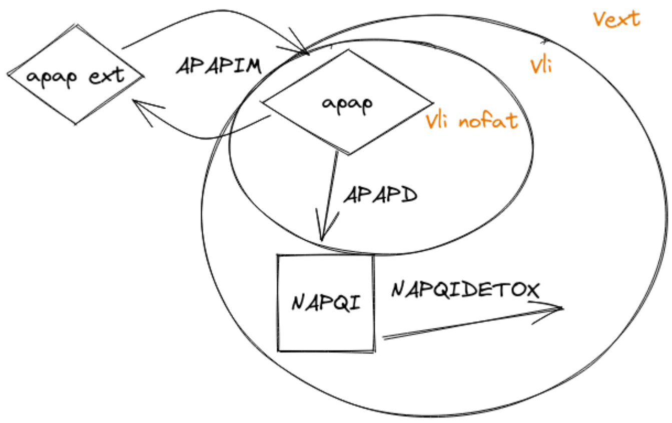


Figure 3.1: Sketch of cell simulation model

The model for hepatic APAP metabolism in the liver used in this work is developed by Koenig [Com20]. It consists of three compartments, as can be seen in Figure 3.1, separating the cell into fatty tissue (Vli_{fat}), no fatty tissue (Vli_{nofat}) and a membrane. Those compartments are surrounded by plasma (V_{ext}). With those cell compartments, the cell model allows for adding fatty tissue to simulate a so-called fatty liver, where fat deposits build up in the liver cells. Three reactions describe the interaction of the cell with its surrounding tissue.

In the model, the speed of the APAP import (Equation (3.7)) into the cell depends on the apap concentrations in the cell ($apap$) and the concentration in the surrounding plasma ($apap_{ext}$). The change in concentration is limited by the apap import speed ($APAPIM_{Vmax}$) and the Michaelis constant[1] $APAPIM\_Km\_apap$.

$$\dot{apap} = \frac{\frac{(1-necrosis)APAPIM_{Vmax}}{APAPIM\_KM\_apap} Vli_{nofat}(apap_{ext} - apap)}{1 + \frac{apap_{ext}}{APAPIM\_Km\_apap} + \frac{apap}{APAPIM\_Km\_apap}} \tag{3.7}$$

[1]https://en.wikipedia.org/wiki/Michaelis%E2%80%93Menten_kinetics

The $apap$ concentration inside the cell is then metabolized by the enzyme "Relative Cytochrome P450 2E1"($CYP2E1$) to toxic $NAPQI$, at the speed of digestion $APAPD_{Vmax}$ and the Michaelis constant $APAPD\_Km\_apap$ in non fatty tissue, as described in Equation (3.8)

Passing the threshold $necrosis_{threshold}$ with a high concentration of NAPQI in the cell will lead to the cell dying off Equation (3.9).

$$\dot{napqi} = (1 - necrosis) * CYP2E1 * APAPD_{Vmax} * Vli_{nofat} * \frac{apap}{apap + APAPD\_Km\_napqi} \tag{3.8}$$

$$necrosis = \begin{cases} 1 & \text{if NAPQI} \geq \text{necrosis}_{\text{threshold}} \\ 0 & \text{else} \end{cases} \tag{3.9}$$

The toxic $NAPQI$ is cleared by the $NAPQIDETOX$ reaction Equation (3.10), where the speed of clearance is limited the detox speed $NAPQIDETOX_{Vmax}$ and the Michaelis constant $NAPQIDETOX\_Km\_napqi$

$$napq\dot{i}detox = (1-necrosis)*NAPQIDETOX_{Vmax}*Vli_{nofat}*\frac{napqi}{napqi + NAPQIDETOX\_Km\_napqi} \tag{3.10}$$

These reactions form a system of ordinary differential equations which represent the cell reaction to the $apap$ concentration in the bloodstream that is simulated by the macro model presented in Section 3.1.

4 Software Tools

In this chapter, we present the software tools that will be used throughout this thesis. We give a introduction into the preCICE library, the FEBio framework and the surrounding tools, detail relevant preCICE features and the FEBio plugin system, that we use for our implementation. We also introduce the microcodes for the microsimulation, which use the simulation environment libRoadRunner.

4.1 preCICE

preCICE (Precise Code Interaction Coupling Environment) is a coupling library that allows a minimally-invasive, black-box coupling to combine single physics solvers to form a multi physics partitioned simulation [BLG+16]. preCICE handles data communication, mesh mapping and more, allowing to treat the physics solver as a black box. The advantages of this approach are that solvers can easily be replaced by a newer or better suited solver, users can continue to work in their known domain and work can be split among different groups, each only working on a part of the problem.

preCICE provides these features using so-called preCICE adapters that hook into the solver suite and manipulate the data and time steps as needed. A preCICE adapter is a piece of "glue-code", which enables preCICE to access the simulation data and control the simulation flow for a specific solver.

preCICE is written in C++ and boost and has a quite active community. It offers programming interfaces for C++, Fortran, Python and Matlab, allowing easy coupling of proprietary simulations. There are also already finished adapters for OpenFOAM, deal.II, CalculiX, SU2, FeniCS, code_aster and Nutils.

With such a large feature set and the already provided adapters, preCICE reduces the time to solution drastically by avoiding monolithic couplings that tailor the whole system to a specific problem and cannot easily be reused.

For further details on preCICE, please refer to the preCICE documentation[1].

[1] https://precice.org

4.1.1 Micro Manager

The preCICE Micro Manager is a tool to facilitate two-scale coupling in multi-physics simulations using preCICE [Des22]. It handles the instantiation of all micro simulations, controls them for the entire simulation time, and handles the data exchange with preCICE. The API and the micro manager is written in python and publicly hosted on GitHub [2].

4.2 MPI

The Message Passing Interface (MPI) [GLDS96] is a communication protocol for high performance computing. It supports one-to-one and one-to-many communication for distributed processes. MPI ïs a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation.-[GLDS96]. It focuses on scalability, portability and high performance.

4.3 FEBio

Finite elements for biomechanics (FEBio) [MEAW12] is a software tool for nonlinear finite element analysis in biomechanics and biophysics and is specifically focused on solving nonlinear large deformation problems in biomechanics and biophysics. Aside from structural mechanics, it can also solve problems in mixture mechanics (i.e. biphasic or multiphasic materials), fluid mechanics, reaction-diffusion, and heat transfer. As a true multi-physics code, it can also solve coupled physics problems, including fluid-solid interactions [1]. It is written in C++ and the code is hosted on GitHub under a MIT License.[2]

4.3.1 FEBio Studio

FEBio Studio is the main software tool for designing, running, and analyzing FEBio models. It offers a graphical user interface for interacting with the FEBio software. Since FEBio is a command-line application, it uses file-based communication. FEBio Studio is the easiest way for creating FEBio XML input files, and for visualizing and analyzing the FEBio output files [3].

[2] https://github.com/precice/micro-manager

[1] https://github.com/febiosoftware/FEBio

[2] https://github.com/febiosoftware/FEBio

[3] https://febio.org

4.3.2 Plugin System

FEBio provides a powerful plugin system to extend FEBio's functionality [MLAW18]. This is achieved by providing dynamically linked libraries that are loaded by FEBio at run-time, which allows for plugins to be separated from the FEBio code base and to be maintained by different developers. There are seven different plugin types, namely material plugins, plot data plugins, callback plugins, task plugins, solver plugins and nonlinear constraint plugins. A callback plugin is used for the FEBioPrecice Adapter in Chapter 5. This type of plugin allows to register callback functions for certain events. The other plugin types are described in the FEBio documentation [4].

4.4 libRoadRunner

The libRoadRunner Simulation Engine is a C++ library for simulating and analyzing systems of differential equations. libRoadRunner was designed with performance as a priority and is an exceptionally fast SBML solver [SBG+15]. Roadrunner takes SBML files (Systems Biology Markup Language) as input model and runs the included models. It provides C++ and python interfaces to set input variables, run the simulation and extract the result data. In this work, libRoadRunner is used to run a cell model that transforms paracetamol (apap) to the byproduct (napqi), but this can easily be replaced with a different cell model.

[4]https://help.febio.org/doxygen/html/plugins.html

5 Implementation FEBio-preCICE adapter

In this chapter, we describe the goals for the FEBio-preCICE adapter[1], that is being developed alongside this work and detail the relevant internal structures of FEBio. Further, we describe the challenges we faced during the development and how these challenges were solved. The chapter closes with an introduction to the FEBio-preCICE adapter configuration.

5.1 Objectives

The first objective for this prototype is that we want to avoid changing the FEBio code, as this would require the user to recompile FEBio or use a separate release. We also wanted to keep the adapter as general as possible to allow later extension to a more general FEBio-preCICE adapter that can be used by the community. This meant that we wanted a way for the user to easily configure which data should be exchanged.

The adapter is supposed to be a prototype to demonstrate a possible implementation option. At the time of writing FEBio has been released at version 3.6, but as to allow for a correct benchmark against the existing coupling, which is based on FEBio 3.2, we only tested and developed the adapter against that version. The upstream code changes introduced since FEBio 3.2 did not modify the plugin framework, therefore this adapter should work with every version from 3.2 onward.

5.2 FEBio Internals

The object that contains all relevant information in FEBio is the `FEModel`. From this object, all relevant simulation information can be reached and it is responsible for running the simulation steps. A simulation step is contained in a `FEAnalysis` object. A normal simulation may contain multiple steps as can be seen in Figure 5.1 Usually, a simulation contains an initialisation step and a step in which the actual finite element method is run. These steps are also responsible for maintaining the simulation time.

Another important object is the `FEMesh`, which contains the simulation mesh with all nodes, elements and data points. Nodes (`FENode`) are points located in the simulation domain that have a global 3D position, that is provided by the user. These Nodes are then used to form elements (`FEElement`), which can be either shell elements or solid elements with different shapes for each type (see Figure 5.2).

[1]https://github.com/precice/febio-adapter

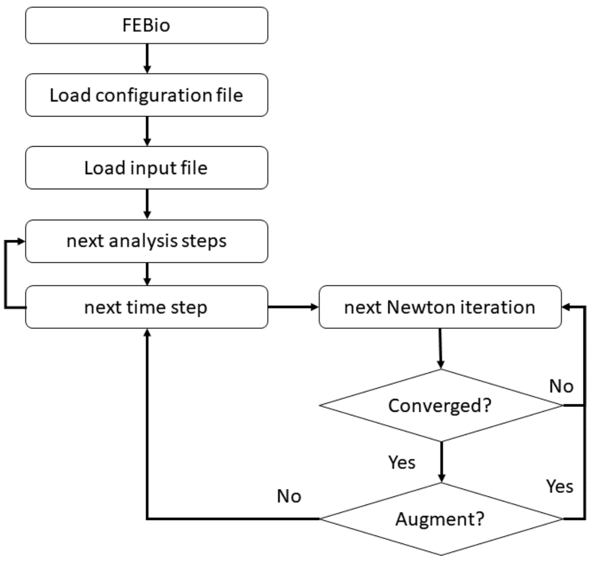
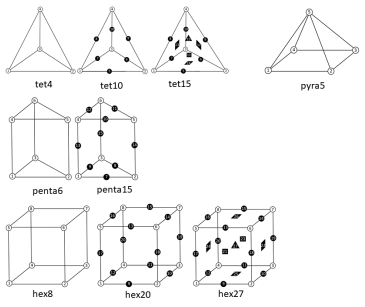
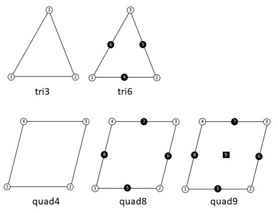


Figure 5.1: FEBio flow [MEAW22]

Each element contains a number of `FEMaterialPoints` which are used as integration points and contain the relevant data. The number of points is different per element type and can be found in the FEBio Theory Manual [1]. These elements can be grouped to element sets via FEBioStudio to create an easy handle to refer to them. To visualize the relationship between the `FEMesh`, `FEElements` and `FEMaterialPoints` please refer to Figure 5.3.



(a) Node numbering for solid elements [MEAW22]



(b) Node numbering for shell elements [MEAW22]

Figure 5.2: MaterialPoint locations on FEElement

[1] https://help.febio.org/FEBioTheory/FEBio_tm_3-4-Section-4.1.html

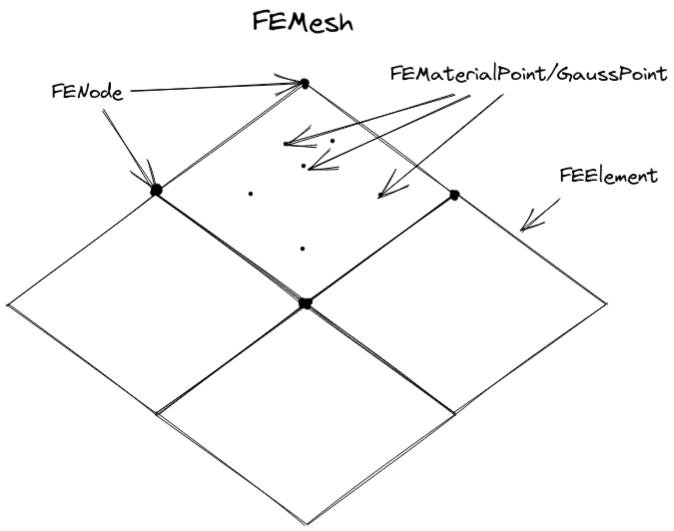


Figure 5.3: FEBio Simulation Mesh Structure

5.3 Proposed solution and challenges

The solution we propose is an adapter in the form of an FEBio callback plugin. As mentioned in Section 4.3.2, callback plugins are loaded by FEBio on startup and register callback functions that will be executed at certain points in the FEBio simulation. These points are shown in Figure 5.4 and are the labels that start with `CB`. The user can load the adapter via the `febio.xml` configuration file, which can be provided on the command line and only requires the user to add a short plugin snippet to the simulation model to couple the simulation using preCICE:

```
<Code>
<callback name="precice_callback"/>
</Code>
```

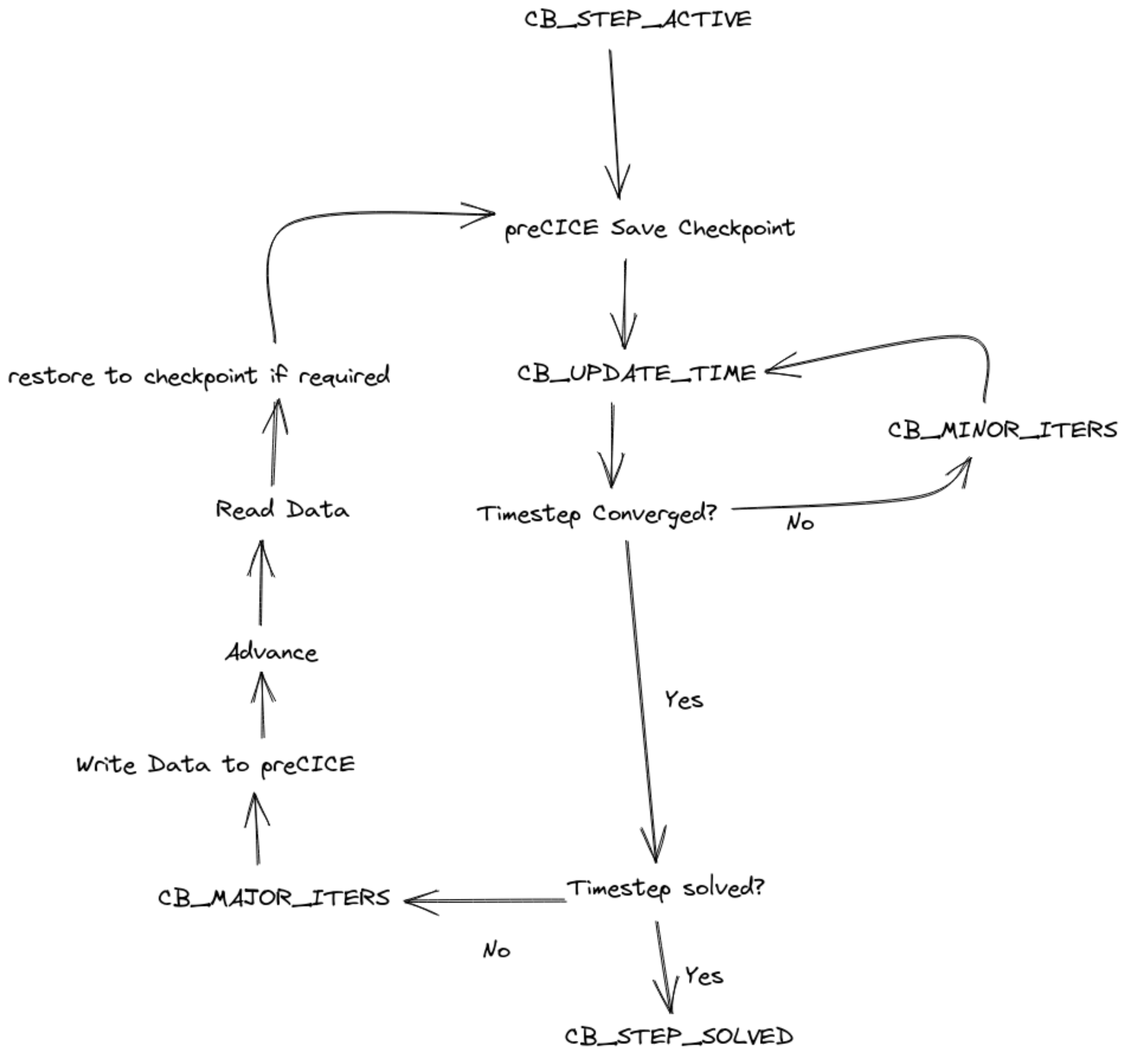
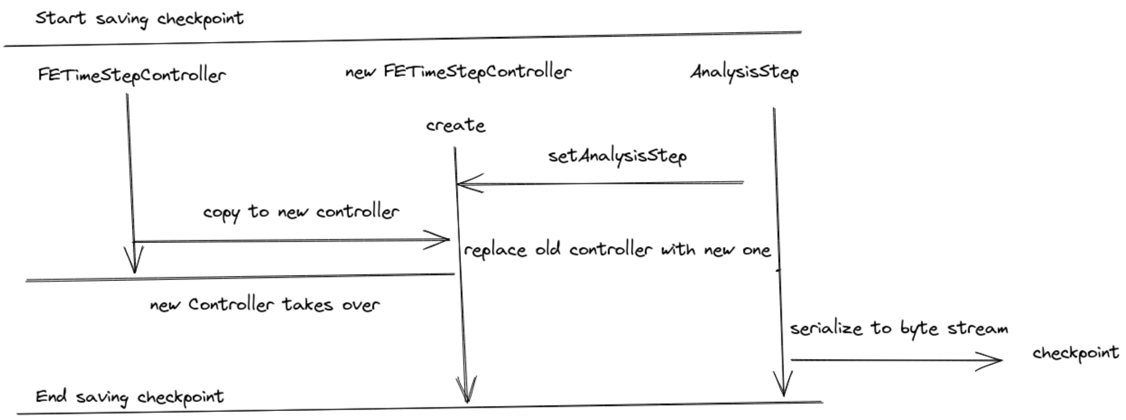


Figure 5.4: Control flow for FEAnalysis step in FEBio
This flow chart displays how a simulation step in FEBio is solved and at what point which preCICE action is executed.

With this plugin approach we have successfully created a system where the user does not have to adapt any FEBio code, but as Figure 5.4 shows, we still have to propose a solution on how to write/read data from preCICE and how we are going to save/reload a checkpoint. preCICE tells the solver whether an iteration has converged or not, and if it has not converged, the solver state has to be fully reverted. This means that the time and quantities which define the state of the solver need to be reversed. To allow for such a revert, the prior state needs to be saved. The adapter does that by replacing the running `FETimeStepController` with a new controller instance on checkpoint creation, allowing to use the newly created controller to advance, or to copy the old state again if we need to reset. Afterwards, the `FEAnalysisStep` is serialized to a binary archive, which is kept in memory for a reset. The whole process is also detailed in Figure 5.5



EB

Figure 5.5: FEBio-preCICE adapter Checkpoint process

If the solver state needs to be reset, the binary archive is extracted and a new time step controller is generated to continue the simulation. The time step controller is treated separately here, by not being serialized, as FEBio raises an error otherwise. This behavior was not further investigated, but is probably a software bug.

To fulfill the objective of being a general adapter, we still need to show a way on how to read/write data from preCICE that can be configured by the user during runtime.
We do this by reading a `febio-config.json` on startup, that is easily configurable by the user. This file provides all required information on how to handle and exchange the simulation data. More information on the configuration options can be found in Section 5.4.

As the simulation data is saved in the `FEMaterialPoints` of each element, we are using these MaterialPoints as vertices in preCICE and read/write data from/to these points. We identify which points to transfer by a named element set, the name being provided in the configuration file. This allows the user to easily pick which vertices are transfered via preCICE.

To allow for custom material behaviors, a `FEMaterialPoint` can be an instance of multiple subclasses, which have different variables and functions. To access these functions and variables and the data contained within, FEBio requires the developer to cast the MaterialPoint reference to the correct type. This creates an interesting challenge, as it requires the adapter to cast an object to a type which is unknown during compile time, as the adapter read the configuration at startup. To deal with this, we added the reflection framework rttr [RTT22] to the adapter project, preregister all possible material points with all variables and functions with rttr, and autogenerated a code snippet, which creates the correct cast operation in the code.

With all these approaches in place, we were able to select which elements with what variables to transfer, but as preCICE and FEBio have different datatypes, we still needed to add code that converts the data between both environments. preCICE offers scalar and vector options for data transfer, while FEBio offers scalar(double, float, int), matrices (mat2d, mat3d, mat3da, mat3dd, mat3ds, matrix), quaternions (quatd), tensors (tens3drs, tens3ds) and vectors (vec2d, vec3d) as data types. As preCICE has the smaller set, we only support scalar vector data types. Further options could be implemented later-on.

5.4 Configuration

```
{
        "coupling_params": {
                "element_set_to_couple": "Part1",
                "participant_name": "FEBio",
                "config_file_name": "../precice-config.xml",
                "read_mesh_name": "FEBioMesh",
                "read_data_name": [
                {"type": "function", "febio_class_name": "FESolutesMaterialPointTPM", "name":
↪   "set_concentrations", "mapping_name": "apap_ext'", "febio_type": "double", "precice_type": "scalar"}
                ],
                "write_mesh_name": "FEBioMesh",
                "write_data_name": [
                {"type": "function", "class_type": "materialPoint", "febio_class_name":
↪   "FESolutesMaterialPointTPM", "name": "Vext", "mapping_name": "Vext", "febio_type": "double",
↪   "precice_type": "scalar"}
                ]
        }
}
```

The configuration file needs to be named `febio-config.json` and has to be located in the folder where the simulation is launched. It contains all relevant simulation options for the adapter. The keys read_data_name and write_data_name contain the attributes that are supposed to be read/written to/from FEBio. Each entry consists out of multiple options that are described in Section 5.4.

Key	Possible Values	Description
type	function variable	Type of attribute for the FEMaterialPoint that is read or written.
class_type	materialPoint	This option sets what class type is accessed in FEBio, in future the possibility of a FEMaterial or something similar could be added.
febio_class_name		Class name of the object that is being manipulated in FEBio e.g FEFluidMaterialPoint
name		Name of the attribute that is written/read
febio_type	vec3d vec2d double int float vector<dobule>	Data type of the attribute being write/read
mapping_name		Name of data point in preCICE
precice_type	scalar vector	Type of data point in preCICE

Table 5.1: Configuration options for read/write_data_name

For more configuration options, refer to the adapter README [1].

[1] https://github.com/precice/febio-adapter

6 Methods

Our objectives are to compare two coupling approaches and investigate improvements to simulation results and the development process. In direct response, in this section we will first discuss both coupling approaches and parameters that are relevant for the simulation performance, followed by a description of the benchmark cases. We then discuss possible improvements to the simulation.

6.1 Coupling approaches

This section introduces the different simulation setups used in this work.

6.1.1 FEBio proprietary coupling

The existing approach from the Ricken [LWWR17; RDD10; RWH+14] allows to directly couple the FEBio code with the roadrunner micro simulations. The coupling is done by adding libRoadRunner as a dependency to the FEBio project and using the libRoadRunner C++ bindings. The bindings are used to create a libRoadRunner micro simulation for each node. A list of active simulations is managed in FEBio[1].

6.1.2 FEBio preCICE coupling with Micro Manager

This work approaches the coupling thats different that the proprietary coupling. Instead of extending the FEBio code base, a FEBio plugin is developed, which allows the FEBio simulation to interface with the preCICE library. To add the libRoadRunner simulations to the coupling, a micro manager is used. The micro manager handles instantiating and data transfer for the micro simulations. On this basis, only a minimal Python based simulation for a single cell had to be developed in order to add the required libRoadRunner functionality.

[1]https://github.tik.uni-stuttgart.de/isd/FEBio3.2

6.2 Analysing the performance of both couplings

To compare the two couplings, we first need to define measurements for this comparison. The first measurement we choose to validate the new simulation approach is simulation accuracy, which is measured through the Equation (6.1). This is done by selecting all elements, creating a graph and exporting the raw graph data, which can then be compared between the different simulation runs. These steps are necessary because FEBio lacks a standardized data output format like `.vtk`[2].

$$\sum_{t}^{T} \sum_{i}^{I} |A_t^i - B_t^i| \tag{6.1}$$

with T being the set of all timesteps, I the length of the vectors A_t and B_t, and A,B being the matrix holding the effective apap concentration for each timestep and node.

The second measurement is the execution time, as this is the most interesting metric when using different simulation setups that produce similar results. As FEBio provides the simulation run time on completion, this is chosen as a measurement, where shorter times are better. Another choice is the memory usage of the simulation setup. As the actually used memory can not be easily measured by the operating system, heaptrack[3] was used to track memory allocation for the coupling setups. In cases of multiple concurrent processes, each process memory usage was tracked and summed up, where less resource usage is better. Another interesting metric would be the processor utilization, but as we are already tracking execution times, monitoring CPU usage would just introduce another measurement that does not provide further insights.

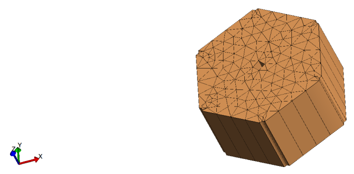
We also compare different acceleration schemes for the preCICE coupling for the benchmark case in Section 6.3.2. For the same case, we also investigate the difference between an explicit and implicit coupling.

6.3 Benchmark cases

In this section we describe the four benchmark cases used for the macro micro coupling as well as their macro topology. The test cases were developed at the Institute of Mechanics, Structural Analysis and Dynamics of Aerospace Structures[4] and are scheduled to be published in the near future as a reproducible dataset.

[2]https://en.wikipedia.org/wiki/VTK
[3]https://github.com/KDE/heaptrack
[4]https://www.isd.uni-stuttgart.de/

6.3.1 Benchmark liver lobule


(a) side view

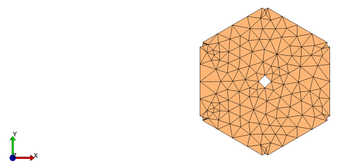
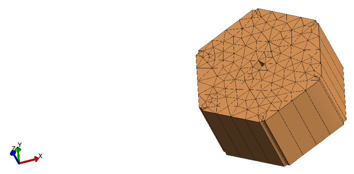

(b) top view

Figure 6.1: Mesh of the liver lobule case

The liver lobule case simulates a single liver lobule with 1584 nodes over 8 seconds. The mesh is shown in Figure 6.1. Boundary conditions provide an initial fixed concentration of apap, which then gets processed in the liver lobule.

6.3.2 Benchmark liver lobule inflow


(a) side view

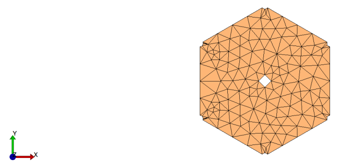
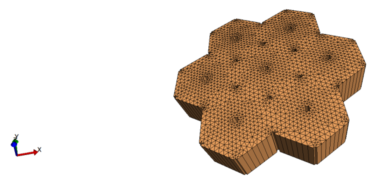

(b) top view

Figure 6.2: Mesh of the liver lobule inflow case

The liver lobule inflow case simulates a single liver lobule with 1584 nodes over 10 seconds. The mesh is shown in Figure 6.2. It uses different inflow conditions than the liver lobule case by providing a linear decreasing apap supply to the lobule.

6.3.3 benchmark group tpm


(a) side view

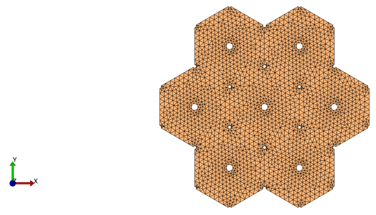

(b) top view

Figure 6.3: Mesh of the group tpm case

The group benchmark case is the largest test case with 24006 nodes. It groups seven liver lobules together, as can be seen in Figure 6.3. Boundary conditions provide an initial fixed concentration of apap, which then gets processed in the liver lobules. The test case is simulated for 2 seconds.

6.4 Coupling

As we are using the preCICE library, we want to test the possible simulation acceleration schemes, expecting that these schemes can provide real time improvements. We used the benchmark case in Section 6.3.2 with a constant under-relaxation, a dynamic Aitken under-relaxation and a Quasi-Newton scheme.

For the constant under-relaxation, we chose `0.5` as a relaxation value, as the preCICE manual recommends that value.[5]

The dynamic Aitken under-relaxation was run with a `0.1` initial under relaxation value, as this should lead to a robust simulation according to the preCICE manual[6].

For the Quasi Newton scheme, the following config section was used.

```xml
<acceleration:IQN-ILS>
        <data name="apap_ext'" mesh="FEBioMesh"/>
        <preconditioner type="residual-sum"/>
        <filter type="QR2" limit="1e-3"/>
        <initial-relaxation value="0.1"/>
        <max-used-iterations value="100"/>
        <time-windows-reused value="20"/>
</acceleration:IQN-ILS>
```

In order to see whether an explicit test case is a viable solution, we ran the benchmark case in Section 6.3.2 and compared it to the implicit test case.

[5]https://precice.org/configuration-acceleration.html#constant-under-relaxation
[6]https://precice.org/configuration-acceleration.html#dynamic-aitken-under-relaxation

7 Results

All tests were performed on a 32 core AMD EPYC 7502P 2.5 GHZ server with 128 GB of DDR4 memory. FEBio version 3.2 with extensions from the ISD and libRoadRunner 2.2.0 were used.

7.1 Benchmarks

In this section, we look at the performance of both the proprietary and the preCICE coupling approach for each benchmark case, followed by a summary and comparison of the results. In order to see the influence of adding libRoadRunner to the simulation pipeline, FEBio was also run standalone with just the macro model and its simulation times were included in the results.

7.1.1 Simulation accuracy

We compared the simulation results for each simulation between the preCICE coupling, the proprietary coupling and the standalone simulation. We extracted the result data by opening the simulation results in FEBioStudio, selecting all elements and exporting the data. This export is done by creating a graph, selecting the `effective solute concentration TPM(apap)` and then saving the graph. We then calculated the distance between both exported data sets with Equation (6.1). The calculation resulted in the value 0 for all benchmark pairs. Our investigation showed that the FEBio graph export facility only outputs numbers with a precision of 10^{-9}, which is not precise enough for our simulation results. Thus, a conclusive comparison was not possible as FEBio lacks an output format like `vtk`. Under these circumstances, the test cases only deal with simulation solving times.

7.1.2 Benchmark liver lobule

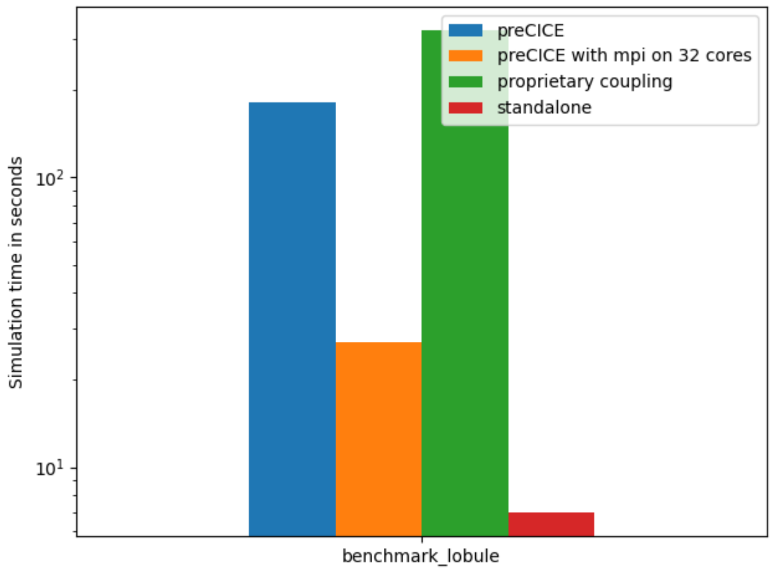


Figure 7.1: Simulation time for each scenario

Running the test case resulted in the following solution times.

test case	solving time in seconds
standalone	7
proprietary	321
preCICE	182
preCICE with MPI and 32 cores	27

7.1.3 Benchmark liver lobule inflow

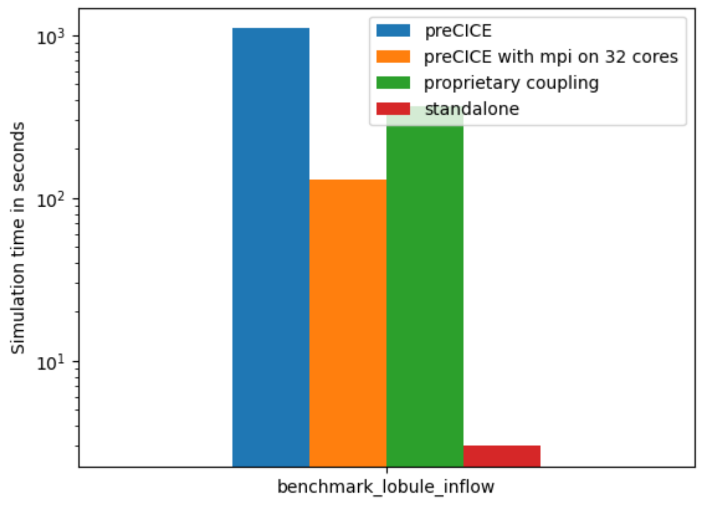


Figure 7.2: Simulation time for each scenario

Running the test case resulted in the following solution times.

test case	**solving time in seconds**
standalone	3
proprietary	368
preCICE	1212
preCICE with MPI and 32 cores	130

7.1.4 Benchmark group tpm

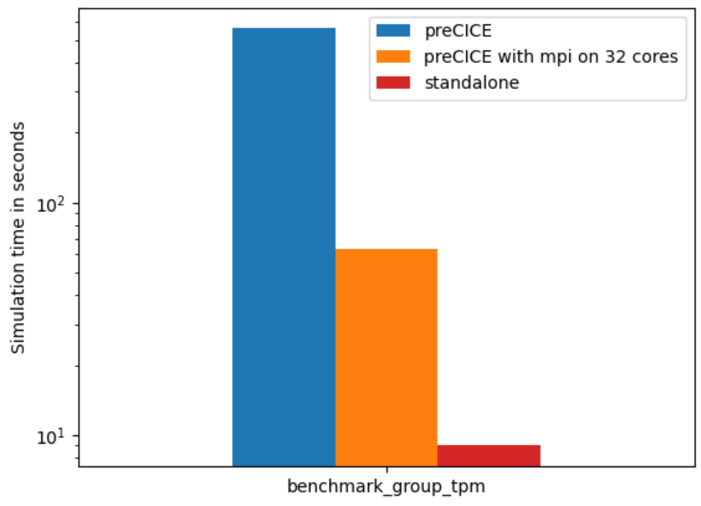


Figure 7.3: Simulation time for each scenario

test case	solving time in seconds
standalone	9
proprietary	failed
preCICE	562
preCICE with MPI and 32 cores	63

7.1.5 MPI

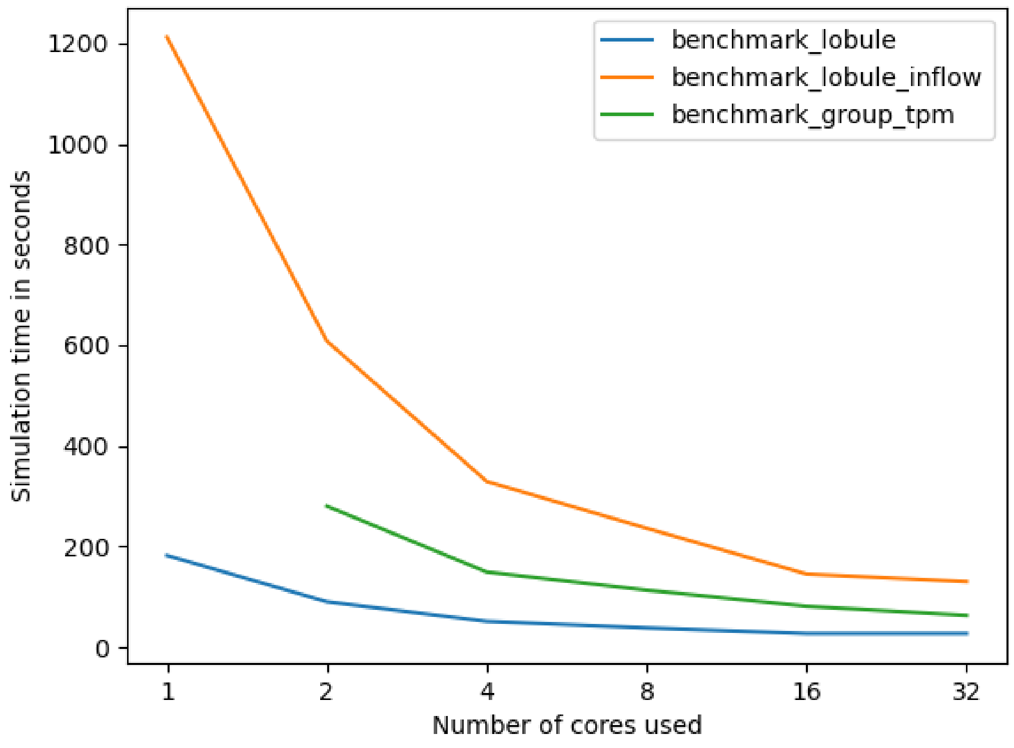


Figure 7.4: Simulation time with a varying number of cores for MPI
The benchmark case group tpm failed when only 1 core was used.

To analyze the performance improvements of adding MPI capabilities, to the preCICE coupling approach we measured the execution times for different numbers of cpu cores. As can be seen in Figure 7.4, the simulation time decreases for each benchmark case with the increasing number of cores. The benchmark_group_tpm case with just one core produced stack traces with memory errors, seemingly originating from the libRoadRunner framework. As the same happened with the proprietary coupling, the only way to solve this benchmark case was with the preCICE MPI setup. The memory errors seemed to be originating from the libRoadRunner framework.

7.1.6 Summary

Our experiments show similar results for both coupling approaches, but faster simulation times for the preCICE coupling. The test case Section 7.1.3 is slower with the preCICE setup on a single core, but this is caused by the convergence measurement, which runs a step multiple times if necessary. The group tpm benchmark in Section 7.1.4 only shows results for the preCICE coupling, as the proprietary coupling execution fails with a memory error.

This demonstrated that the preCICE coupling using the FEBio-preCICE adapter is faster and more reliable.

7.2 Acceleration schemes

All acceleration schemes produced the same results and similar solving times than cases without an acceleration scheme. Presumably, this is because the cell contribution to the overall simulation is too small ($< 10^{12}$).

7.3 explicit and implicit coupling

The explicit coupling produces the same results as an implicit test case, but does so in a considerably shorter run time of only around 15 seconds, compared to the 130 seconds needed to run the implicit test case.

7.4 Memory usage and performance

Analyzing the memory usage displayed in Figure 7.5 showed that the proprietary coupling performed a lot better than the preCICE approach (2,3GB vs 9,8GB). Further research showed an increasing memory footprint over time for the preCICE micro manager which hints at a memory leakage somewhere either in the micro manager, micro simulation or in the libRoadRunner Python bindings.

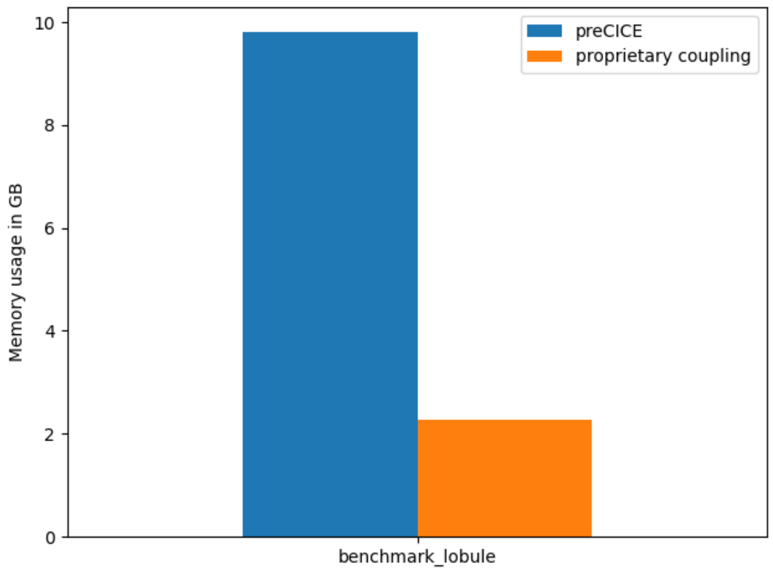


Figure 7.5: Memory usage for the benchmark lobule case

While looking at the memory usage, the resulting flamegraph (c.f. Figure 7.6) also shows that the cell simulations spent most of their time in the libRoadRunner bindings. This shows that the micro manager and preCICE overhead for the simulation is negligible.

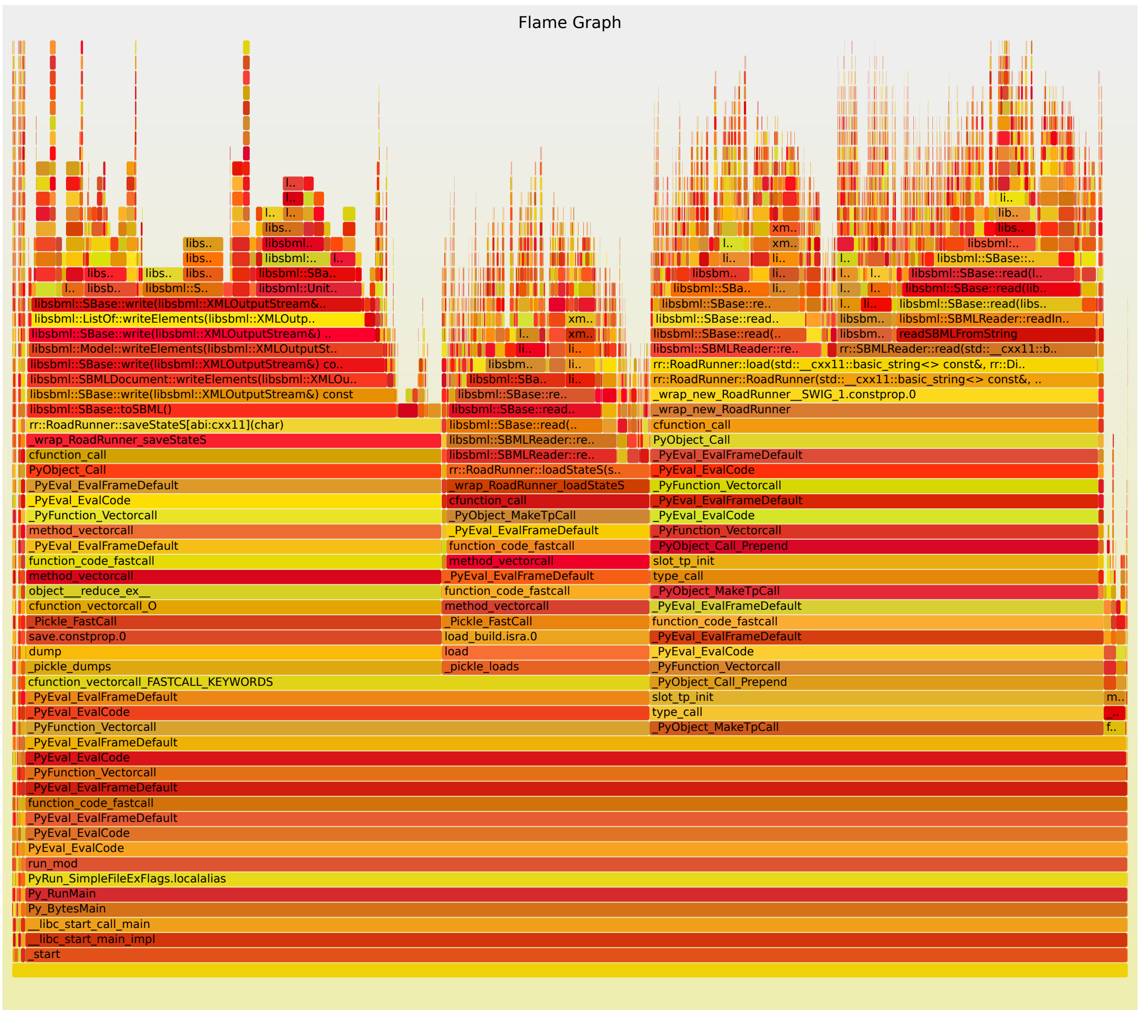


Figure 7.6: Call graph of a micro manager instance during a simulation run. The Y axis represents the call stack at a specific time X.

8 Conclusion and Outlook

We set out to develop a FEBio-preCICE adapter to investigate whether such a coupling would prove to be superior to the existing proprietary coupling. We achieved that goal by demonstrating that both coupling produce similar results, that solving times using the adapter are faster and that with the preCICE library we allow for parallelization of the micro scale simulation.

In this chapter, we discuss in detail the advantages of the preCICE coupling approach and outline future work. The chapter is split into three sections, the first considers simulation improvements, where we discuss advantages gained for the simulation process and pipeline. The second section then deals with advantages that directly affect the development process. Lastly, we finish this chapter by discussing future work.

8.1 Simulation Improvements

8.1.1 Full Body Model

It is a necessity and important step to further extend the liver simulation model to include the effects of other organs in the human body. This could be done by adding a sbml model that bundles all contributions from other organs and interfaces with the liver macro scale simulation. Using the preCICE approach, adding such a model would be an easy task, as it would only be another coupling partner that needs to be added in the xml configuration file. In order to use the proprietary coupling for this, it would be required to write additional C++ code in a huge work effort, compared to the preCICE solution.

8.1.2 Distributed resources

Another direct advantage of the preCICE coupling are the parallelization capabilities, which we already demonstrated in this work. To implement the same feature set into the proprietary coupling would require months of work and even then would not be as flexible and tested as the preCICE solution.

8.2 Development Process Improvements

To estimate the improvements and work hours saved for the development process, one has to focus on the advantages and disadvantages that each approach supplies. The proprietary approach is really flexible and new features can modify every aspect of the simulation as one is not bound to plugin system capabilities or the limits of the preCICE library. These advantages are negligible compared to the disadvantages of this approach. As the proprietary coupling approach is split from the FEBio toolkit on version 3.2 and the git history was not kept, correctly integrating upstream updates will lead to massive amounts of work. Further, this approach is completely implemented in C++ and is deeply integrated with the FEBio toolkit. As such, new developers need to be familiar with the FEBio architecture and have prior experience on how to extend existing C++ frameworks, which is a difficult skill to find. On top of that, this approach leads to more lines of code being written that need to be maintained by a small PhD team that wants to focus on improving the liver simulation model, instead of fixing memory leaks.

The preCICE coupling approach is not without disadvantages, as it adds another framework to the simulation stack that could cause problems and that developers need to be familiar with to diagnose errors. But the advantages outweigh that drawback in that most users would only need to be familiar with the preCICE framework and their excellent documentation and could depend on the active community for support. Further, the preCICE coupling allows for easily adding new coupling partners. Updates to the FEBio toolkit would just require installing the new version, instead of painfully managing the whole compilation process that the proprietary coupling requires. Should the plugin interface of FEBio change, adaptation to the FEBio-preCICE adapter would be needed, but as this is an open source project, one could benefit from potential external contributors.

Outlook

As the FEBio-preCICE adapter has only been tested against the existing proprietary coupling, the first order of business should be extending the test cases to ensure the adapter works reliably. The adapter could also be extended to allow for more configuration options, e.g passing constant values, instead of always extracting data via reflected function calls or variable access.

Another interesting test case for the adapter would be to use multiple liver lobule simulations and couple these via preCICE to allow for a distributed FEBio simulation setup.

Officially releasing the FEBio-preCICE adapter as open source software would benefit this work immensely, as the support of the adapter plugin would shift to the preCICE project.

Bibliography

[BLG+16] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann. "preCICE – A fully parallel library for multi-physics surface coupling". In: *Computers and Fluids* 141 (2016). Advances in Fluid-Structure Interaction, pp. 250–258. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2016.04.003. URL: http://www.sciencedirect.com/science/article/pii/S0045793016300974 (cit. on pp. 9, 19).

[Com20] T. G. A. for Computational Mechanic. *GACM-Report Qualiperf 2020*. 2020. URL: https://livermetabolism.com/paper/GACM-Report_Qualiperf2020.pdf (cit. on pp. 9, 16).

[Des22] I. Desai. *preCICE Micro Manager*. https://github.com/precice/micro-manager. 2022 (cit. on p. 20).

[Ehl96] Ehlers. "Grundlegende Konzepte in der Theorie Poröser Medien". In: *Technische Mechanik* (1996) (cit. on p. 12).

[GLDS96] W. Gropp, E. Lusk, N. Doss, A. Skjellum. "A high-performance, portable implementation of the MPI message passing interface standard". In: *Parallel Computing* 22.6 (1996), pp. 789–828. ISSN: 0167-8191. DOI: https://doi.org/10.1016/0167-8191(96)00024-5. URL: https://www.sciencedirect.com/science/article/pii/0167819196000245 (cit. on p. 20).

[HFSB01] M. Hucka, A. Finney, H. Sauro, H. Bolouri. "Systems Biology Markup Language (SBML) Level 1: Structures and Facilities for Basic Model Definitions". In: (Mar. 2001) (cit. on pp. 9, 13).

[LWWR17] L. Lambers, N. Waschinsky, D. Werner, T. Ricken. "A Multi-scale and Multi-phase Model for the Description of Toxicity caused by Paracetamol in Biological Tissue using the Example of the Human Liver". In: *PAMM* 17.1 (2017), pp. 199–200. DOI: https://doi.org/10.1002/pamm.201710069. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pamm.201710069. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pamm.201710069 (cit. on p. 31).

[MEAW12] S. Maas, B. Ellis, G. Ateshian, J. Weiss. "FEBio: Finite Elements for Biomechanics". In: *Journal of biomechanical engineering* 134 (Jan. 2012), p. 011005. DOI: 10.1115/1.4005694 (cit. on p. 20).

[MEAW22] S. Maas, B. Ellis, G. Ateshian, J. Weiss. *FEBio Documentation*. https://help.febio.org/. 2022 (cit. on p. 24).

[MLAW18] S. A. Maas, S. A. LaBelle, G. A. Ateshian, J. A. Weiss. "A Plugin Framework for Extending the Simulation Capabilities of FEBio". In: *Biophysical Journal* 115.9 (2018), pp. 1630–1637. ISSN: 0006-3495. DOI: https://doi.org/10.1016/j.bpj.2018.09.016. URL: https://www.sciencedirect.com/science/article/pii/S0006349518310695 (cit. on p. 21).

[RDD10] T. Ricken, U. Dahmen, O. Dirsch. “A biphasic model for sinusoidal liver perfusion remodeling after outflow obstruction”. In: *Biomechanics and modeling in mechanobiology* (Jan. 2010). DOI: 10.1007/s10237-009-0186-x (cit. on p. 31).

[RK15] J. Renz, M. Kinkhabwala. “Surgical Anatomy of the Liver”. English (US). In: *Transplantation of the Liver: Third Edition*. Elsevier Inc., Jan. 2015, pp. 23–39. ISBN: 9780323396936. DOI: 10.1016/B978-1-4557-0268-8.00002-6 (cit. on p. 11).

[RL19] T. Ricken, L. Lambers. “On computational approaches of liver lobule function and perfusion simulation”. In: *GAMM-Mitteilungen* 42.4 (2019), e201900016. DOI: https://doi.org/10.1002/gamm.201900016. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/gamm.201900016. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/gamm.201900016 (cit. on p. 9).

[RTT22] RTTR. *rttr*. [Online; accessed July 21, 2022]. 2022. URL: https://github.com/rttrorg/rttr (cit. on p. 27).

[RWH+14] T. Ricken, D. Werner, H. Holzhütter, M. König, U. Dahmen, O. Dirsch. “Modeling function–perfusion behavior in liver lobules including tissue, blood, glucose, lactate and glycogen by use of a coupled two-scale PDE–ODE approach”. In: *Biomechanics and modeling in mechanobiology* (Sept. 2014). DOI: 10.1007/s10237-014-0619-z (cit. on pp. 3, 9, 12, 15, 16, 31).

[SBG+15] E. T. Somogyi, J.-M. Bouteiller, J. A. Glazier, M. König, J. K. Medley, M. H. Swat, H. M. Sauro. “libRoadRunner: a high performance SBML simulation and analysis library”. In: *Bioinformatics* 31.20 (June 2015), pp. 3315–3321. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btv363. eprint: https://academic.oup.com/bioinformatics/article-pdf/31/20/3315/17087875/btv363.pdf. URL: https://doi.org/10.1093/bioinformatics/btv363 (cit. on p. 21).

[Wik22a] Wikipedia, the free encyclopedia. *Liver loves*. [Online; accessed July 21, 2022]. 2022. URL: https://en.wikipedia.org/wiki/File:Liver_04_Couinaud_classification.svg (cit. on p. 11).

[Wik22b] Wikipedia, the free encyclopedia. *Liver structure*. [Online; accessed July 21, 2022]. 2022. URL: https://en.wikipedia.org/wiki/Lobes_of_liver#/media/File:Liver_and_nearby_organs.jpg (cit. on p. 11).

All links were last followed on July 30, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature