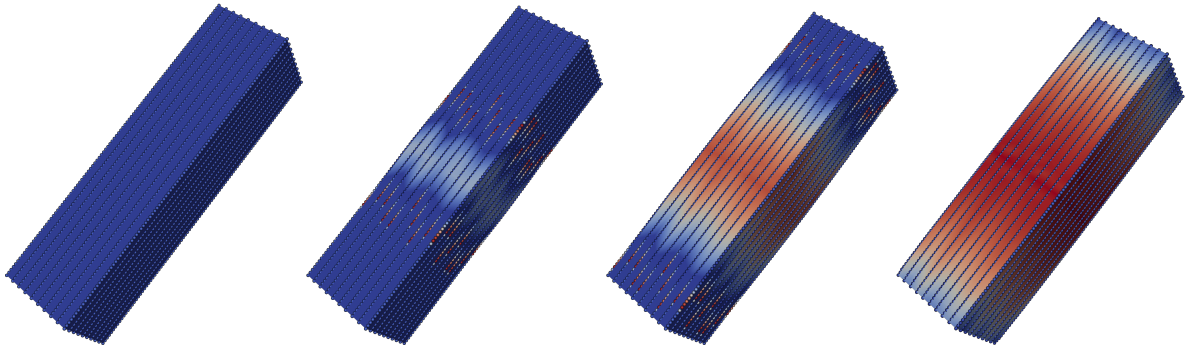


Comparison and coupling of the neuromuscular simulation software OpenDiHu with the biomechanics solver FEBio

Paul Arlt, Luis Morgenstern, Silas Natterer, Jan Stein

July 16, 2023

Abstract

This report presents a project focused on the modeling of skeletal muscles, specifically exploring the comparative analysis between two simulation frameworks, FEBio and OpenDiHu. The goal is to use the well tested mechanics solver of FEBio, while leveraging the more realistic modelling of the muscle fibers in OpenDiHu. To achieve this, we use the coupling frame preCICE to create a combined FEBio and OpenDiHu simulation of a skeletal muscle. We also provide a simple FEBio plugin, which includes an archaic preCICE adapter and a custom material. This plugin allows us to create a FEBio simulation case which is comparable to the implementation in OpenDiHu. We show that both the combined FEBio and OpenDiHu case and the OpenDiHu implementation produce similar results, with the combined case being significantly faster. These insights into the strengths and limitations of FEBio and OpenDiHu, along with the custom plugin, may provide valuable knowledge for future research in skeletal muscle modeling and simulation. By leveraging the respective capabilities of these frameworks, further advancements in biomechanical modeling can be realized.

1 Introduction

In recent years, the field of biomechanical modeling has witnessed significant advancements, enabling researchers to delve deeper into the complex behavior of skeletal muscles. The ability to accurately simulate the mechanical properties and active fiber contractions of muscles has broad implications, ranging from understanding musculoskeletal disorders to guiding the development of novel rehabilitation strategies. In this report, we present a comparative analysis of two simulation frameworks, FEBio and OpenDiHu, with the aim of assessing their respective capabilities in modeling muscle behavior. The motivation behind this study stems from the need to identify the strengths and limitations of existing simulation tools for muscle modeling.

1.1 Muscle model

The muscle model we utilize for this report, uses the monodomain equation to simulate the behaviour of the fibers: On each point of the fibers, a zero dimensional Hodgkin-Huxley is employed to simulate reaction term of the monodomain equation. In addition, a one dimensional diffusion term is solved, to accomodate for the action-potential propagation along each fiber. This produces the lumped activation parameter γ.

A continuum mechanical model is then used to simulate the macroscopic deformations and stresses in the muscle tissue. In particular a transversely isotropic Mooney-Rivlin material is used, accounting for the specific behaviour in the direction of the muscle fibers. In addition, the activation γ is used to compute the active stress caused by the activation of the muscle fibers. For a more detailed description of this model we refer to [BEE+18].

1.2 Software

FEBio is a well-established software for muscle modelling developed in collaboration with the University of Utah and Columbia University. It is written in C++ and simulation cases can be created using a XML configuration file. Alternatively, FEBioStudio can be used, offering a graphical user interface to create those configuration files. FEBio also allows the creation of plugins, which are written in C++ and compiled as shared object files [MEAW12].

OpenDiHu, developed at the University of Stuttgart, is an emerging software specializing in skeletal muscle simulation. While OpenDiHu is still in active development and may not have the same level of maturity as FEBio, it is believed that its fiber model is more realistic. OpenDiHu is able to read CellML files, which are, for example, used for specifying the monodomain equation [MMR+10]. OpenDiHu is written in C++ and configured using Python. This allows the user to procedurally create meshes, or load them from a external file and convert them with Python. OpenDiHu also comes with a working preCICE adapter, which allows it to be coupled with other software [MGH+].

In our project, we seek to combine the strengths of FEBio and OpenDiHu by coupling their respective solvers to develop a hybrid simulation approach. To facilitate this coupling, we utilize the preCICE library, developed at the Technical University of Munich and the University of Stuttgart. To use preCICE, each participating solver has to implement a preCICE adapter. The user can then configure the coupling via a XML configuration file, specifying parameters such as the time step width, which variables to transfer between the solvers and how to map them between the different meshes. Each solver is then started as a separate process, exchanging data over a specified directory [CDR+22].

In the subsequent sections of this report we will first provide a comparative analysis of FEBio's and OpenDiHu's mechanics solvers. After that we will provide a detailed description of our simulation case in OpenDiHu and the methods used to couple the mechanics and monodomain solvers. Our development of a custom plugin for FEBio, allowing us to couple OpenDiHu with FEBio, will be covered before we summarize the obtained results and discuss their implications. We use ParaView to visualize our results. Furthermore we will highlight the limitations of our study and propose avenues for future research and development. The GitHub repository of this project can be found at [GIT].

Parameter	SI	OpenDiHu
L x W x H	0.06m x 0.02m x 0.02m	6cm x 2cm x 2cm
Force	0-5 N	0-5 N
Time step size	0.0001 s	0.1 ms
Density	1000 kg/m^3	10 dg/cm^3
C1	3.176e^{-6} N/m^2	3.176e^{-10} N/cm^2
C2	18130 N/m^2	1.813 N/cm^2

Figure 1: Used Parameters in SI units and OpenDiHu units

2 Comparing OpenDiHu's and FEBio's mechanics solver

Part of our student project is to compare the respective mechanics solvers of OpenDiHu and FEBio. To do this, we simulate the same problem case in both softwares and assess if the resulting data is comparable. In order to look at the mechanics solver only, we model a case of a passive muscle, that is influenced by an external force.

We use a Mooney-Rivlin material to model a cuboid muscle, that measures 2cm in width and height and 6cm in length. The muscle is divided into 2x2x6 quadratic elements, where every element is a cube with an edge length of 1cm. The backside of the muscle is fixed in place with a Dirichlet boundary condition. The front of the muscle is pulled by a force increasing from 0N to 5N over the time of the simulation. We run the simulation for 20ms, with a time step size of 0.1ms, resulting in 200 time steps.

It is not trivial to set up equivalent cases in OpenDiHu and FEBio. Firstly, FEBio does not support incompressible materials. In our problem case, we use a nearly incompressible material. This can be achieved by setting the bulk modulus k to a high value. The bulk modulus represents the counterforce of material compression.

Secondly, OpenDiHu uses rather unconventional units. [ms] is used to measure time, [cm] is used for distance and [N] is used for force. With this in mind, the measurement for mass can be derived as $[10^{-4}\text{kg}] = [\text{dg}]$. As follows, the units of pressure and density are measured in [N/cm^2] and [dg/cm^3] respectively. The parameters we use can be seen in figure 1, in the SI unit system, as well as in the OpenDiHu unit system.

FEBio on the other hand does not assume a specific unit system. Any unit system is supported, as long as the units are consistent with each other. This can also be seen in figure 2. The blue and orange plot show our case modeled in FEBio using the OpenDiHu and the SI unit system respectively. Since the unit system can be choosen freely in FEBio, the simulation of the models yield the same result.

The green plot in figure 2 shows the results of the simulation in OpenDiHu. The mean squared error of the FEBio simulation using OpenDiHu units and the OpenDiHu simulation amounts to $4.169 \cdot 10^{-6}$cm. This result confirms that the mechanics solver of FEBio is a valid replacement for OpenDiHu's mechanics solver.

FEBio needed 23s to simulate the case, while OpenDiHu needed 127s for the cases with either unit system. For our case, FEBio ran the simulation 5.5 times faster, which further motivates the use of FEBio's mechanics solver.

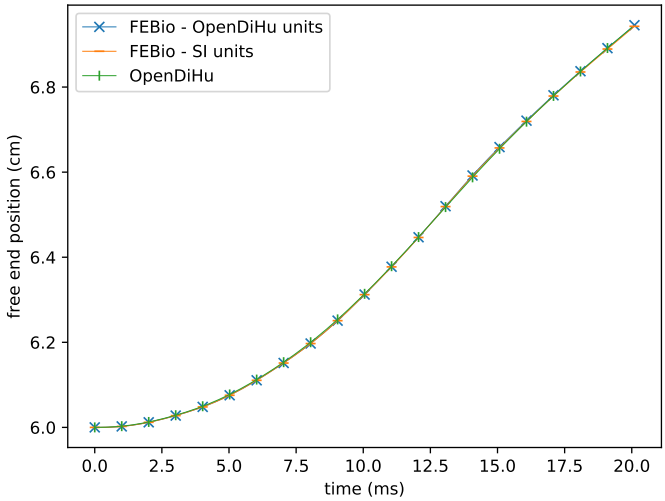


Figure 2: Position of the free end of a passive muscle, that is pulled over time. Simulated in FEBio using OpenDiHu units (cyan), using SI units (orange) and simulated in OpenDiHu (green)

3 OpenDiHu's multi-scale muscle model

Our next step was implementing a muscle contraction case in OpenDiHu, using OpenDiHu's internal coupling. The purpose of this case is to be later used as a reference for the more complicated cases using preCICE coupling.

3.1 OpenDiHu's MuscleContractionSolver

We use OpenDiHu's *MuscleContractionSolver* for the mechanics simulation. This solver is a simple wrapper for the *DynamicHyperelasticitySolver*, OpenDiHu's main mechanics solver. It uses a transversely isotropic Mooney-Rivlin material by default. Additionally, it computes the fiber stretch λ and provides the activation parameter γ as a connector slot, which is then used for coupling. It computes the active stress as

$$\frac{1}{\lambda} P_{\max} f\left(\frac{\lambda}{\lambda_{\text{opt}}}\right) \gamma \tag{1}$$

where $P_{\max}$ is the maximum active stress. The function f evaluates to

$$f\left(\frac{\lambda}{\lambda_{\text{opt}}}\right) = -\frac{25}{4}\left(\frac{\lambda}{\lambda_{\text{opt}}}\right)^2 + \frac{25}{2}\frac{\lambda}{\lambda_{\text{opt}}} - 5.25 \tag{2}$$

if force length relation is enabled and $0.6 \leq \frac{\lambda}{\lambda_{\text{opt}}} \leq 1.4$, and $f\left(\frac{\lambda}{\lambda_{\text{opt}}}\right) = 1$ otherwise [MGH+]. We use $P_{\max} = 7.3$ and enable force length relation for all our results. For the material parameters we use $\rho = 10, c_1 = 3.176 \cdot 10^{-10}$, $c_2 = 1.813$, $b = 1.075 \cdot 10^{-2}$ and $d = 1$.

3.2 OpenDiHu's FastMonodomainSolver

For the fiber simulation we use OpenDiHu's *FastMonodomainSolver*. This class solves the monodomain equation and contains two nested solvers, which are coupled using Strang operator splitting: Along each of the fibers a one dimensional diffusion process is solved using

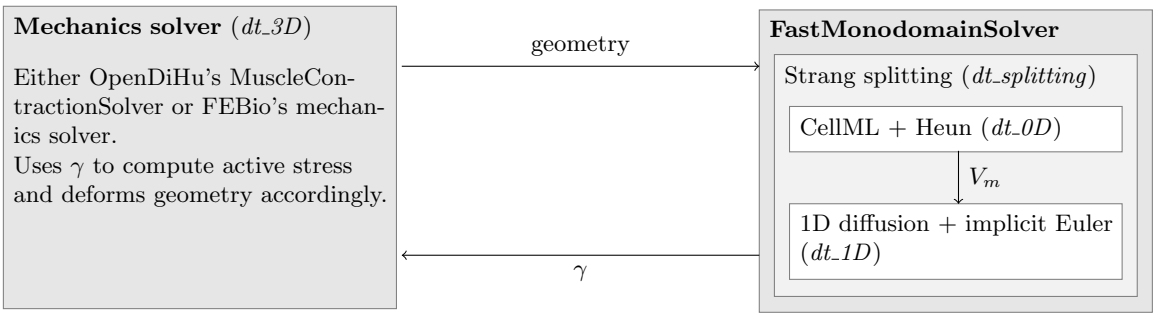
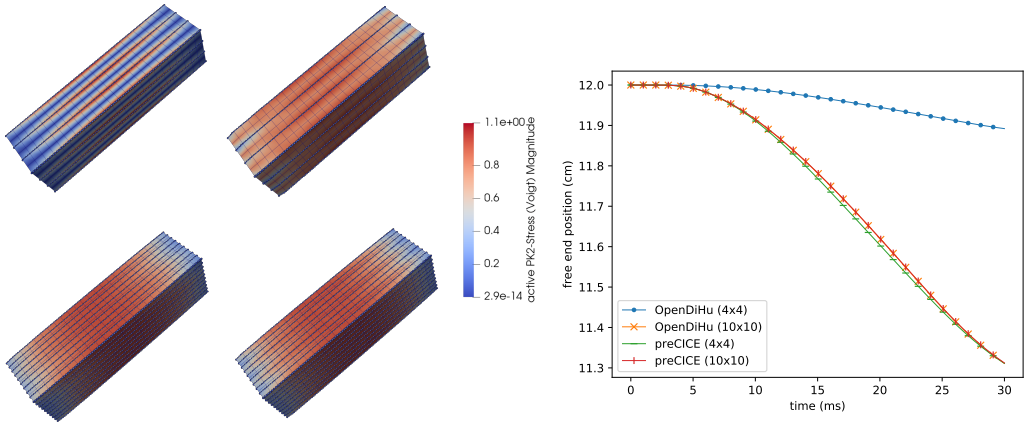


Figure 3: Illustration of the muscle contraction coupling scheme



(a) Muscle at the end of the simulation using OpenDiHu (left) and preCICE (right) with 4x4 fibers (top) and 10x10 fibers (bottom)

(b) Position of the free end of the muscle for the different mapping methods and fiber resolutions

Figure 4: Results of the two different mapping methods

finite elements and implicit Euler time stepping. Also, a zero dimensional CellML problem is solved on each fiber point using Heun time stepping. This calculates the transmembrane potential V_m [BEE+18], which is transfered to the diffusion solver. Finally, the diffusion solver computes the activation parameter γ and provides it on a connector slot for coupling. The *FastMonodomainSolver* increases the performance of these nested solvers roughly by a factor of 10. It achieves this by utilizing Thomas' algorithms for the diffusion process [MGH+]. We use the Hodgkin-Huxley-Razumova CellML model for our simulation.

The coupling then transfers the geometry from the *MuscleContractionSolver* to the *FastMonodomainSolver* and γ in the reverse direction. An overview of the entire coupling scheme can be seen in figure 3. Implementing this using OpenDiHu's internal coupling is straightforward. However, for our project we also need to understand preCICE coupling, as it will be used for the combined FEBio and OpenDiHu case later. Thus, we also couple the two solvers using preCICE.

We use a 4x4 fiber mesh with 100 points per fiber in conjunction with a 3x3x12 quadratic elements mechanics mesh of size 3cm x 3cm x 12cm to test the two coupling schemes. The time steps widths are chosen as $dt_3D = 0.1$ms, $dt_splitting = dt_1D = 2 \cdot 10^{-3}$ms and $dt_0D = 10^{-3}$ms. The simulation runs for 30ms. The results can be seen in figure 4b. Surprisingly there is a distinct visible difference between them. In fact, we have a mean

squared error of 0.0921. To understand why, a closer look at how mappings between the meshes are handled differently in preCICE and OpenDiHu is necessary.

3.3 Mesh mapping in OpenDiHu

OpenDiHu's mapping method maps data from source points s_i to target points t_j. The target points are associated with a mesh, in particular with their elements, while there is no structural information for the source points. First, it is determined in which target element each source point s_i lies. Then the corresponding element coordinates $\xi_i \in [0,1]^3$ are computed. The coupled value v_{t_j} at the target point t_j is then computed using barycentric interpolation:

$$v_{t_j} = \sum_{E \in \text{adj}(t_j)} \frac{\sum_{s_i \in E} v_{s_i} \phi_E(\xi_i)}{\sum_{s_i \in E} \phi_E(\xi_i)} \tag{3}$$

where $\text{adj}(t_j)$ is the set of elements adjacent to the target point t_j and ϕ_E is the ansatz function for element E [MGH+]. A result of this approach is, that if the fiber resolution is chosen to low, some target points may not be mapped to at all. This can be observed in figure 4a, where a 4x4 fiber resolution is not sufficient to cover the entire mechanics mesh, resulting in target points not getting activated, and therefore, less contraction.

3.4 Mesh mapping in preCICE

In preCICE several mapping methods such as nearest neighbor, nearest projection and radial basis functions are supported. For our simulations we use radial basis functions, as they require no topological data. This mapping uses radially symmetric basis functions centered at the source points s_i, to define a global interpolation function

$$S(x) = \sum_{i=1}^{N} \lambda_i \phi\left(\frac{||x - s_i||}{r}\right) + \beta_0 + \sum_{k=1}^{d} \beta_k x_k \tag{4}$$

where d is the number of dimensions, N the number of source points, and r is the chosen support radius. The coefficients $\lambda_1, \ldots, \lambda_N, \beta_0, \ldots, \beta_d$ are chosen such that $S(s_i) = v_{s_i}$, $\sum_{i=1}^{N} \lambda_i s_i = 0$ and $\sum_{i=1}^{N} \lambda_i = 0$. Finally one has $v_{t_j} = S(t_j)$ [LMU17]. We use $r = 0.5$ and the compact polynomial C6 basis function, which is defined as

$$\phi(\xi) = (1-\xi)^8(32\xi^3 + 25\xi^2 + 8\xi + 1) \tag{5}$$

for $\xi \in [0,1]$ and $\phi(\xi) = 0$ otherwise [BLG+16]. Note that for $||x - s_i|| > r$ we get $\phi\left(\frac{||x-s_i||}{r}\right) = 0$. Therefore, each source point s_i affects a target points t_j only if $||t_j - s_i|| \leq r$.

A visual illustration of the two mapping techniques can be seen in figure 5. We have found that in OpenDiHu it is important to always use a sufficiently high fiber resolution. By increasing our fiber resolution from 4x4 to 10x10, we can fix our original problem, achieving a mean squared error of $5.29 \cdot 10^{-7}$.

4 Coupling Challenges and Implementation Details

To enable the coupling of OpenDiHu's FastMonodomainSolver with FEBio's mechanics solver, we implement a preCICE adapter in for FEBio using FEBio's plugin system. In this section we describe how our plugin works and how it is structured.

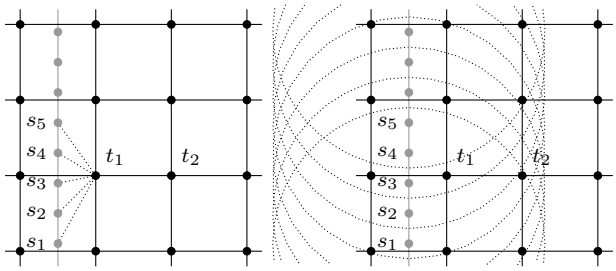


Figure 5: Illustration of OpenDiHu mapping (left) and preCICE mapping (right). The mesh point t_2 is not mapped to using OpenDiHu's mapping, while it still lies inside the radial basis functions of the fiber points $s_1, \ldots, s_5$ when using preCICE.

4.1 Implementing the preCICE adapter for FEBio

Our preCICE adapter is based on an existing experimental FEBio-preCICE adapter [FEB]. This adapter was primarily designed to couple a simulation of hepatic tissue in FEBio with metabolic processes calculated using libRoadRunner, as described in the master's thesis of Fritz Otlinghaus [Otl22].

The adapter was originally written for FEBio 3 and although there is a branch for FEBio 4, we ran into compilation issues. We could solve them with only minor modifications, but the communication between FEBio and OpenDiHu was only working unidirectional from OpenDiHu to FEBio and we were not able to set data in FEBio and retrieve it in OpenDiHu. To resolve this issue, we focused on removing extraneous components, such as the reflection approach for code generation and runtime configuration code. By eliminating these elements, we were able to create a more concise and understandable codebase. However, it is important to acknowledge that our custom FEBio plugin lacks the ability to be configured at runtime, so the user must write the necessary logic and configuration in C++ and compile it accordingly. As a result, our modified adapter is much faster to compile and easier to understand, but may not be suitable for broader applications or scenarios beyond our project scope.

The adapter works by reading the activation parameter γ from the preCICE mesh and writing it to the corresponding *FEMaterialPoint*. On the other hand, the current position *m_rt* is read from the *FEMaterialPoint* and written to the preCICE mesh for coupling.

4.2 Implementing a OpenDiHu contraction material

Using our modified preCICE adapter for FEBio we are able to transfer values between FEBio and OpenDiHu. However, FEBio does not support OpenDiHu's active contraction model described in section 3.1. This means that the coupled γ value remained unused on the FEBio side. To resolve this problem, we had to implement our own custom *DiHuMaterial* in FEBio, emulating OpenDiHu's active contraction model. This material uses our own *DiHuMaterialPoint*, which inherits from FEBio's *FETransIsoMooneyRivlin* material but uses our own *DiHuMaterialPoint* instead of the usual *FEElasticMaterialPoint*. The *DiHuMaterialPoint* in turn inherits from FEBio's *FEElasticMaterialPoint* and additionally exposes the member variable *m_gamma*. The idea behind this is that we want our material to behave the same as FEBio's *FETransIsoMooneyRivlin* material, but additionally have access to the activation parameter γ for the active stress calculation later.

The active contraction behaviour of the *FETransIsoMooneyRivlin* material can be config-

ured via its member *m_ac*, which is of type *FEActiveContractionMaterial*. This class exposes two abstract functions: *ActiveStress* and *ActiveStiffness*, which are used for active stress and active stiffness calculations respectively. As none of the existing implementations of this abstract class suit our purposes, we also offer our own implementation with the *DiHuContraction* class. This class uses the *m_gamma* parameter of our material point to calculate the active stress using equation 1. The active stiffness, on the other hand, is always zero, like in OpenDiHu.

4.3 Choosing the material parameters

Unfortunately the transversely isotropic Mooney-Rivlin formulation is slightly different in OpenDiHu and FEBio. In OpenDiHu the strain energy density function is computed as

$$\Psi_{\mathrm{DH}} = c_1(\bar{I}_1 - 3) + c_2(\bar{I}_2 - 3) + \frac{b}{d}(\lambda^d - 1) - b\ln\lambda \tag{6}$$

where λ is the deviatoric part of the stretch along the fiber direction and c_1, c_2, b, d are material parameters [MGH+]. In FEBio, on the other hand, the strain energy density is computed as

$$\Psi_{\mathrm{FE}} = c_1(\bar{I}_1 -) + c_2(\bar{I}_2 - 3) + F(\lambda) + \frac{k}{2}(\ln J)^2 \tag{7}$$

where k is the bulk modulus, J the Jacobian of the deformation and

$$F(\lambda) = \begin{cases} 0 & \lambda \leq 1 \\ c_3\left(e^{-c_4}\left(\mathrm{Ei}(c_4\lambda) - \mathrm{Ei}(c_4)\right) - \ln\lambda\right) & 1 < \lambda < \lambda_m \\ c_5(\lambda - 1) + \left(c_3\left(e^{c_4(\lambda_m - 1)} - 1\right) - c_5\lambda_m\right)\ln\lambda & \lambda_m \leq \lambda \end{cases} \tag{8}$$

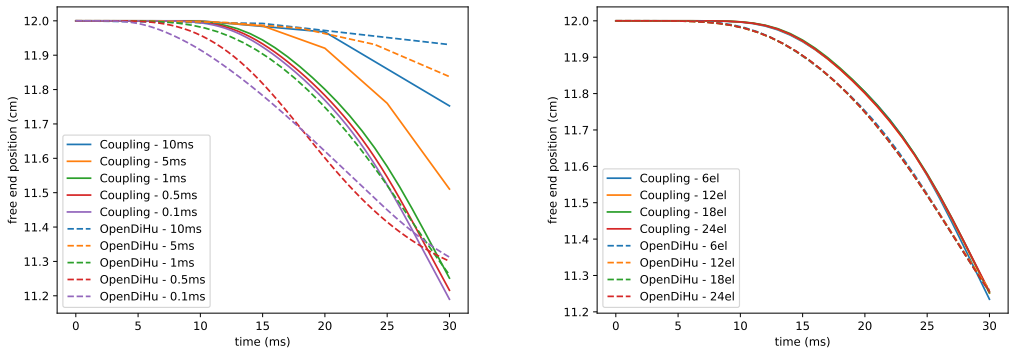
where Ei is the exponential integral function, λ_m is the stretch at which the fibers are straightened and c_1, c_2, c_3, c_4, c_5 are material parameters [MEAW12]. However, if we choose $c_3 = 0, c_4 = 1, c_5 = b$ and $\lambda_m = 1$ we get $F(\lambda) = b(\lambda - 1) - b\ln\lambda$. Therefore, if we additionally use $d = 1$ in OpenDiHu, we get $\Psi_{\mathrm{FE}} = \Psi_{\mathrm{DH}} + \frac{k}{2}(\ln J)^2$. For the bulk modulus we use $k = 1000$ in order to get the behaviour of an incompressible material.

5 Results

In this section, we present the results achieved with our software coupling. First, we verify the correctness of our coupling by comparing the resulting simulation to a simulation performed by OpenDiHu. The second result we verify, is the achieved improvement in the runtime of simulations. Coupling the software is motivated due to the superior performance of the mechanics solver of FEBio. Since we exchange the mechanics solver of OpenDiHu to that of FEBio, our overall runtime should improve accordingly.

To test our results, we parameterize the case described in section 3. The FEBio material parameters are chosen as in section 4.3. The active muscle has a width and height of 3cm and a length of 12cm and is comprised of 3x3xN quadratic elements, where N is one of 6, 12 18 or 24. A 10x10 fiber mesh is used, to avoid the issues described in section 3. The simulation is run over 30ms, while the time step size is one of 10ms, 5ms, 1ms, 0.5ms or 0.1 ms.

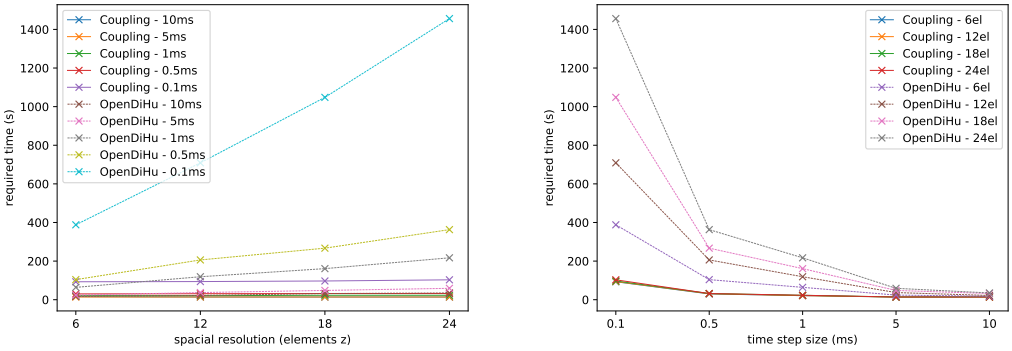
In figure 6a we can see the results of our coupling compared to that of OpenDiHu. Although the results deviate noticeably, the plot have a similar curvature and magnitude. We attribute the deviation to the complexity of the active muscle contraction model, and minor



(a) Position of the free end of the muscle with 12 elements in z direction, for different time step sizes, simulated with our coupling and OpenDiHu

(b) Position of the free end of the muscle with a time step size of 1ms, for different numbers of elements, simulated with our coupling and OpenDiHu

Figure 6: Results of our coupling compared to OpenDiHu



(a) Comparison of execution time of simulations using our Coupling and OpenDiHu for different time step sizes and varying spacial resolution

(b) Comparison of execution time of simulations using our Coupling and OpenDiHu for different spacial resolutions and a varying time step size

Figure 7: Execution times of our coupling compared to OpenDiHu

problems in our implementation. A phenomenon that catches the eye is how the contraction of the muscle slows down in the case of OpenDiHu when a time step size of less than 1ms is chosen. However, we do not have an explanation for this behavior.

Changing the spacial resolution of our model does not influence the results of the simulation in a significant way. This can be seen in figure 6b, where multiple spacial resolutions are plotted. Neither for our coupling nor for OpenDiHu, the plots change visibly.

We executed each simulation on the same machine and measured the runtime. The underlying machine possesses an Intel(R) Core(TM) i7-8565U CPU with 4 cores and 1.80GHz, as well as 16GB RAM. The achieved times can be seen in figure 7a and 7b.

By utilizing the mechanics coupling of FEBio, our coupling achieves a significant speedup compared to OpenDiHu. Especially impressive is, how little a high spacial resolution tolls

the execution time. For a time step size of 0.1ms and a spacial resolution of 24 elements in z direction, OpenDiHu requires 1457s, while our coupling requires only 103s, a speedup by a factor of 14.

6 Conclusion and Outlook

In this study, we conducted a comparative analysis of two simulation frameworks, FEBio and OpenDiHu, in modeling skeletal muscle behavior. We successfully verified that FEBio's mechanics solver has superior performance compared to that of OpenDiHu, while producing the same results.

Regarding the simulation of active muscle contraction, we successfully implemented a custom plugin for FEBio, utilizing the preCICE coupling framework. The coupled simulation demonstrated considerably faster performance compared to running the simulation in OpenDiHu alone. This hybrid approach allowed us to leverage the speed of FEBio's mechanical solver while benefiting from the realistic fiber simulation provided by OpenDiHu. However, it is important to acknowledge the limitations of our custom plugin for FEBio. It was specifically tailored to our specific simulation case, and its applicability to other scenarios may be limited. Further development and optimization would be necessary to ensure its wider applicability and to overcome these limitations.

We also demonstrated that both the OpenDiHu case and the coupled FEBio case produce similar results. Nevertheless, we have found that depending on the chosen time step size, there is still a considerable difference between the two simulations. Further research may be able to close that gap and achieve the same results using both approaches.

Moving forward, future research could also focus on enhancing the flexibility and adaptability of the custom plugin for FEBio, enabling its use in a wider range of scenarios. Additionally, exploring alternative coupling strategies and further optimizing the hybrid approach could lead to even greater improvements in accuracy and efficiency. Ultimately, our project contributes to the ongoing development of biomechanical modeling techniques, and may be useful for more advanced simulations and a deeper understanding of skeletal muscle behavior.

References

[BEE+18] Chris P. Bradley, Nehzat Emamy, Thomas Ertl, Dominik Göddeke, Andreas Hessenthaler, Thomas Klotz, Aaron Krämer, Michael Krone, Benjamin Maier, Miriam Mehl, Tobias Rau, and Oliver Röhrle. Enabling detailed, biophysics-based skeletal muscle models on hpc systems. *Frontiers in Physiology*, 9, 2018.

[BLG+16] Hans-Joachim Bungartz, Florian Lindner, Bernhard Gatzhammer, Miriam Mehl, Klaudius Scheufele, Alexander Shukaev, and Benjamin Uekermann. precice – a fully parallel library for multi-physics surface coupling. *Computers Fluids*, 141:250–258, 2016. Advances in Fluid-Structure Interaction.

[CDR+22] G Chourdakis, K Davis, B Rodenberg, M Schulte, F Simonis, B Uekermann, G Abrams, HJ Bungartz, L Cheung Yau, I Desai, K Eder, R Hertrich, F Lindner, A Rusch, D Sashko, D Schneider, A Totounferoush, D Volland, P Vollmer, and

OZ Koseomur. preCICE v2: A sustainable and user-friendly coupling library [version 2; peer review: 2 approved]. *Open Research Europe*, 2(51), 2022.

[FEB] Experimental precice adapter for febio. https://github.com/precice/febio-adapter. Accessed: 2023-07-16.

[GIT] Github repository of this project. https://github.com/silasnatterer/bfp. Accessed: 2023-07-16.

[LMU17] Florian Lindner, Miriam Mehl, and Benjamin Uekermann. *Radial Basis Function Interpolation for Black-Box Multi-Physics Simulations*. 05 2017.

[MEAW12] Steve A Maas, Benjamin J Ellis, Gerard A Ateshian, and Jeffrey A Weiss. Febio: finite elements for biomechanics. *Journal of Biomechanical Engineering*, 2012.

[MGH+] Benjamin Maier, Dominik Göddeke, Felix Huber, Thomas Klotz, Oliver Röhrle, and Miriam Schulte. Opendihu: An efficient and scalable framework for biophysical simulations of the neuromuscular system.

[MMR+10] Andrew Miller, Justin Marsh, Adam Reeve, Alan Garny, Randall Britten, Matt Halstead, Jonathan Cooper, David Nickerson, and Poul Nielsen. An overview of the cellml api and its implementation. *BMC bioinformatics*, 11:178, 04 2010.

[Otl22] Fritz Otlinghaus. Coupling of macro and micro scale in a continuum-biomechanical model of the human liver using precice, Aug 2022.