Contents

1 Results and Discussion **1**
1.1 Introduction . . . 1
1.2 Solid Mechanics Solver . . . 2
1.3 Simulation of Activated Muscle Fibers . . . 2
1.4 Fiber Based Electrophysiology . . . 3
1.5 Solver for the Multidomain Model . . . 32
1.6 Electrophysiology and Coupled Solid Mechanics . . . 35
1.7 Simulations of the Neuromuscular System . . . 37
1.8 Performance Studies with OpenCMISS Iron . . . 43
1.9 Performance Studies of the Electrophysiology Solver in OpenDiHu . . . 53
1.10 Parallel Strong Scaling and Comparison with OpenCMISS Iron . . . 62
1.11 Performance Measurements on the GPU . . . 67
1.12 Parallel Scaling of the EMG model in High Performance Computing . . . 73
1.13 Performance Studies of the Solid Mechanics Solver . . . 78
1.14 Studies of Numeric Properties . . . 78
1.15 Effect of the Mesh Width on the Action Potential Propagation Velocity . . . 78
1.16 Parallel in Time master thesis . . . 81
1.17 Output file sizes . . . 81
1.18 dx-dt dependencies . . . 81
1.19 Load balancing bachelor thesis . . . 81
1.20 Application of opendihu within the field of robotics . . . 82

2 Conclusion and Future Work **83**
2.1 Future Work . . . 83

1 Results and Discussion

1.1 Introduction

Basic equations, Laplace, Poisson, Diffusion etc.

1.2 Solid Mechanics Solver

1.2.1 Linear Mechanics

1.2.2 Nonlinear mechanics, tendon material

1.2.3 FEBio adapter, validation of solid mechanics implementation

1.3 Simulation of Activated Muscle Fibers

1.3.1 Computation of Subcellular Models

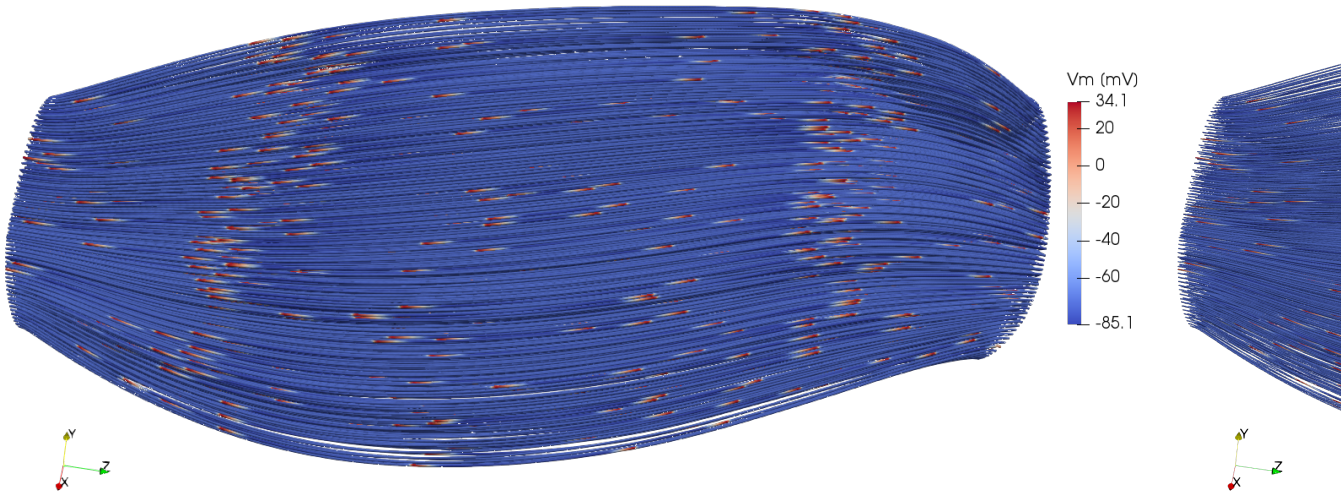


Figure 1.1: fibers mesh

1.3.2 Solver for the Monodomain Equation

1.3.3 Artificial Fiber Geometries

Rosenfalck model, artifical muscle geometry

1.4 Fiber Based Electrophysiology

In this section, we consider surface EMG signals on the upper arm by simulating the activation of the biceps brachii muscle. In Sec. 1.4.1, we introduce the setting of the simulation and present an examplary scenario to compute EMG signals. Subsequently, we simulate various scenarios to investigate the effects of different model and numerical parameters on the resulting EMG signal. Section 1.4.2 considers the effects of single motor units, Sec. 1.4.3 considers the fat layer and Sec. 1.4.4 shows effects of the mesh width. Then, Sec. 1.4.5 presents a way to simulate realistic EMG electrodes and Sec. 1.4.6 deals with the decomposition of EMG signals.

1.4.1 Overview of the EMG Simulation

Figure 1.2 shows the setting of the biceps muscle and the tendons, which attach to the skeleton near the shoulder and to the ulna bone in the forearm. For the simulation of EMG, we only consider the muscle belly of the biceps muscle. The image Fig. 1.2 shows muscle fibers inside the muscle, which run in longitudinal direction between the tendons at both ends. The image also visualizes the results of an EMG simulation. The fibers are colored according to the transmembrane potential V_m. On some fibers, action potentials can be seen.

The surface of the muscle is colored by the electric extracellular potential ϕ_e. In a reasonable approximation, the value of ϕ_e corresponds to the measured EMG signals on the skin surface. More realistic scenarios additionally consider volume conduction in a layer of adipose tissue on top of the muscle.

For the EMG simulations, we solve the multi-scale model of fiber based electrophysiology. We solve the monodomain equation **??** independently on all 1D muscle fiber meshes. After fixed numbers of timesteps, we map the membrane voltage V_m from the 1D meshes

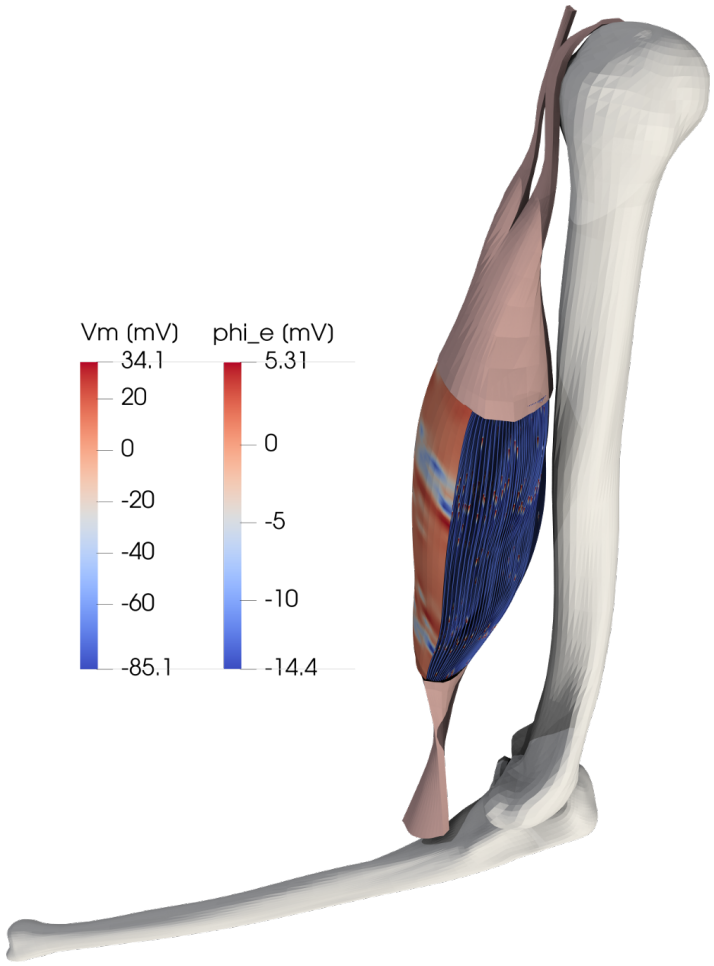


Figure 1.2: Considered setting for simulations of surface EMG on the upper arm, consisting of the biceps brachii muscle, tendons and bones. A simulation result of the membrane voltage V_m on the muscle fibers and the extracellular potential ϕ_e on the surface is shown.

to the 3D mesh. Subsequently, we solve the static bidomain equation **??** on the muscle domain and potentially the body fat domain to obtain the ϕ_e values on the skin surface.

Figure 1.3 shows a close-up view of the active muscle fibers and the resulting EMG signals on the upper surface, which are identical to Fig. 1.2. The scenario considers 961 muscle fibers, each described by a 1D mesh of 1481 nodes. It can be seen that they are approximately equally spaced as a result of the meshing algorithms described in **??**.

Figure 1.3 also shows the mesh of the muscle surface, which is colored according to the extracellular potential ϕ_e. It can be seen that the values correlate with the activation state of the underlying fibers. At the two blue colored regions at the surface near the left and right ends of the muscle, the ϕ_e value is close to its minimum, while the majority of fibers exhibits its maximum positive V_m value. Towards the center of the muscle, the value of

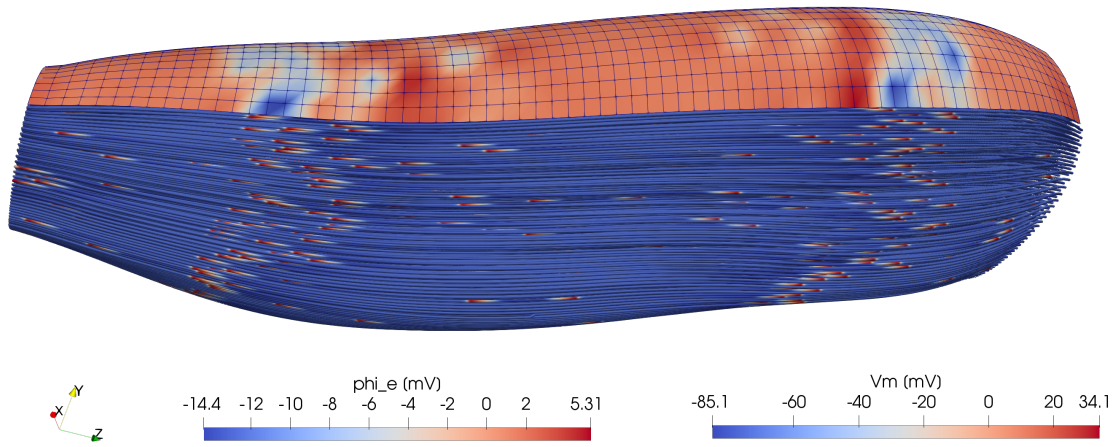


Figure 1.3: Overview of some of the meshes in an electrophysiology simulation: 1369 muscle fibers are located in the muscle belly. A 2D surface mesh on top of the muscle describes the computed EMG values. The visualized simulation is the same as in Fig. 1.2.

ϕ_e increases to its maximum, which reflects the hyperpolarization of the muscle fibers behind the propagating action potentials, i.e., the overshoot of the membrane voltage before it reapproaches the equilibrium level.

The lower left corner in Fig. 1.3 shows the coordinate frame that is used in all simulations. The z axis is approximately directed in fiber direction, the x and y axes are oriented in transverse direction and describe cross-sectional planes of the muscle.

The scenario uses the subcellular model of Hudgkin and Huxley [Hod52]. The shown image corresponds to the simulation time of $t = 200\,\text{ms}$. The simulation also considers a body fat mesh layer on top of the muscle, which is not visualized in Fig. 1.3.

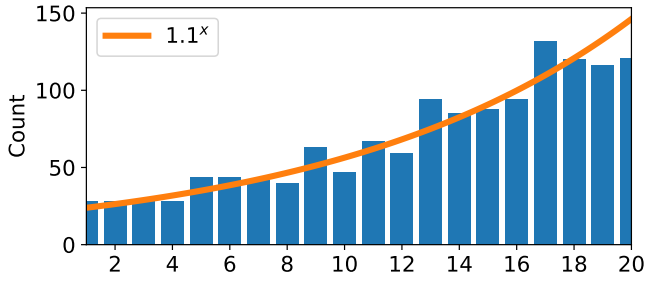
We run the simulation with 128 processes on a two-socket shared-memory node comprising two AMD EPYC 7742 64-core processors with 2.87 GHz clock frequency and 1.96 TiB RAM. The total runtime for a simulation end time of one second is 8 h 38 min.

1.4.2 Effects of Single Motor Units on the Electromyography Signal

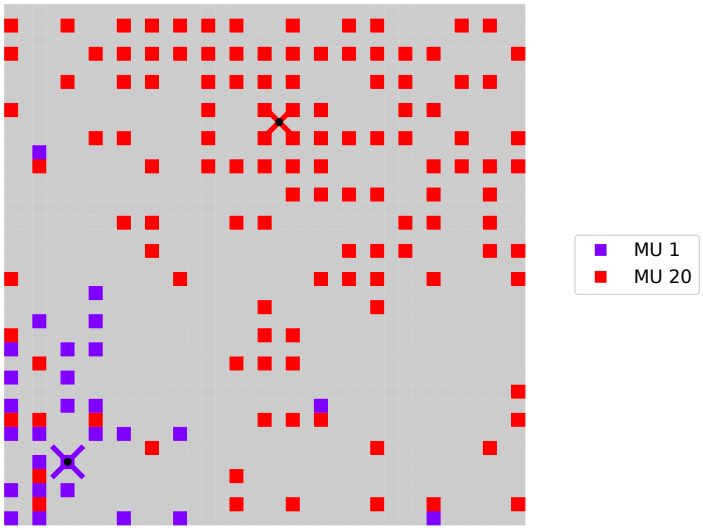
Next, we investigate how the surface EMG signals are influenced by several parameters of the simulation. We begin by studying EMG of only a single activated MU in the muscle.

The first scenario contains 20 MUs that have an exponentially progressing number of fibers as shown in Fig. 1.4a. The progression follows the function $y = c\,1.1^x$ for an appropriate constant $c > 0$. The MU assignment is created using method 1a of the algorithm described in **??**, where the MU territories are centered around given points and neighboring fibers are never part of the same MU.

Figure 1.4b shows the fibers that are assigned to the smallest and largest MUs, MUs 1 and 20. For this visualization, the muscle cross-section is mapped to the large gray square and every colored small square corresponds to one fiber. The purple and red crosses designate the center of MU territories for MU 1 and 20, respectively. In consequence, the fibers of MU 1 are mostly located at the bottom left of the cross-section and the fibers of MU 20 are mostly located in the upper right regions of the muscle cross-section. The visualization shows that the fibers of the same MU always have some spacing between them, which is due to the construction of the MU assignment algorithm.



(a) Exponential distribution of motor unit sizes. The diagram shows the motor unit numbers with the corresponding sizes or fibers counts of the MUs.



(b) Fibers that belong to motor units 1 and 20.

Figure 1.4: Assignment of the 1369 fibres to 20 motor units used in the simulation scenario for fiber based electrophysiology.

We begin with a simulation scenario where only a single MU is stimulated and study the effect on the surface EMG. The fibers of the respective MU are stimulated with a frequency $f = 24\,\text{Hz}$ starting at time $t = 0\,\text{ms}$. Each of the $13 \times 13 = 1369$ fibers consists of a mesh with 1481 nodes, the 3D mesh of the muscle contains $19 \times 19 \times 38 = 13\,718$ nodes and the 3D mesh of the fat layer contains $37 \times 5 \times 38 = 7030$ nodes. The domains are partitioned to 27 processes. The subcellular model of Hodgkin and Huxley is used, yielding a total number of more than $8.1 \cdot 10^6$ degrees of freedom. The timestep widths are $dt_{0D} = dt_{\text{splitting}} = 2.5 \cdot 10^{-3}\,\text{ms}$, $dt_{1D} = 6.25 \cdot 10^{-4}\,\text{ms}$ and $dt_{3D} = 5 \cdot 10^{-1}\,\text{ms}$, leading to 4 subcycles for the 1D model in each splitting step and 200 splitting steps per solution of the bidomain equation.

We compute the linear systems for the initial potential flow problem to estimate fiber directions in the 3D domain and for the bidomain equation, which is solved in every

timestep using a conjugate gradient solver. The program uses the `FastMonodomainSolver` class for the electrophysiology model. The Thomas algorithm solves the linear system of the diffusion problem. We use the `"vc"` optimization type and employ the scheme to only compute active fibers and the subcellular problems that are not in equilibrium.

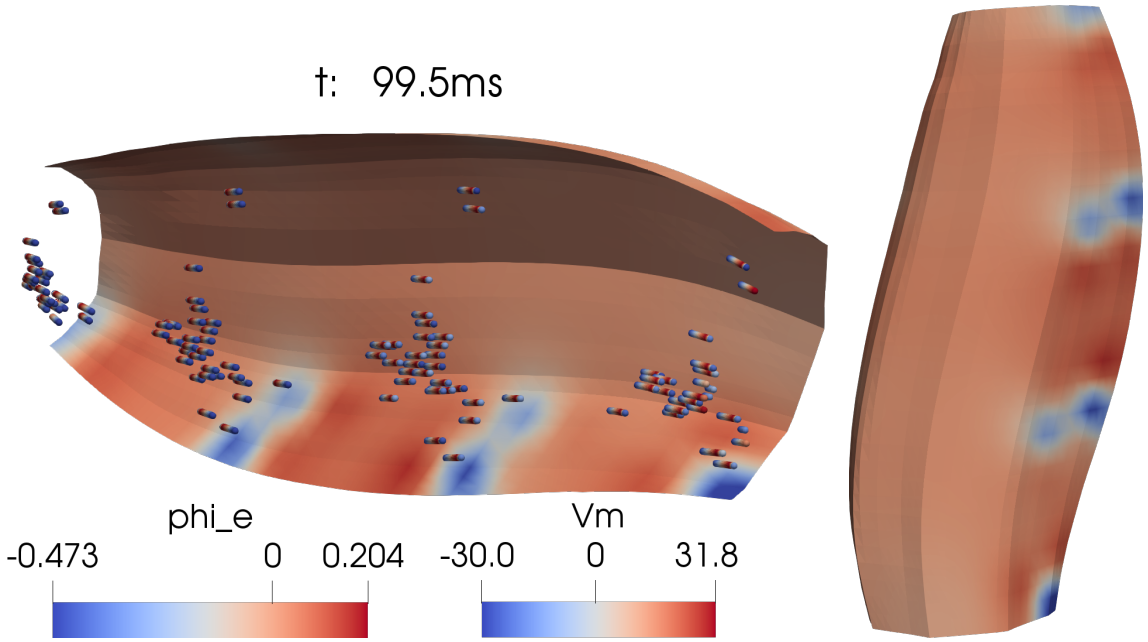
The computation of a simulated time span with $t_{\text{end}} = 100\,\text{ms}$ on a AMD EPYC 7742 64-core processor with 2492 MHz base frequency and 1.96 TB RAM takes approximately 100 s in the scenario that activates only the smallest MU and 126 s in the scenario that activates only the largest MU.

Figure 1.5 shows the result for the scenario of activating the smallest MU, MU 1. In Fig. 1.5a, the surface is shown in the background and colored according to the extracellular potential ϕ_e, which represents the EMG signal. The muscle volume is not shown, instead, the active parts of the respective fibers are displayed as tubes in the 3D domain. Their color visualizes the value of the transmembrane voltage V_m. In every of these small tube segments, the rising and declining shape of an action potential can be observed by the color progression from blue over orange to red for the rising part and back to blue for the declining part.

In this scenario, the fibers of MU 1 are stimulated three times within the first 100 ms at 0 ms, 41.6 ms and 83.3 ms. The innervation zone contains the starting points for the propagating stimulus on every fiber. The scenario positions the neuromuscular junctions randomly within the central 10 % of every muscle fiber. The activated parts of the fibers visible in Fig. 1.5a correspond to the propagated action potentials of the last two stimulations in this scenario.

By comparing the results in Fig. 1.5a with the fiber distribution in Fig. 1.4b, it can be seen that fibers of MU 1 are located opposite of the muscle surface, which is at the upper side of the cross-sectional square diagram in Fig. 1.4b. The left side of the diagram in Fig. 1.4b corresponds to the lower part of the skin in Fig. 1.5a. This part of the skin is closer to the activated fibers and, thus, the effect on the surface EMG is highest for this region.

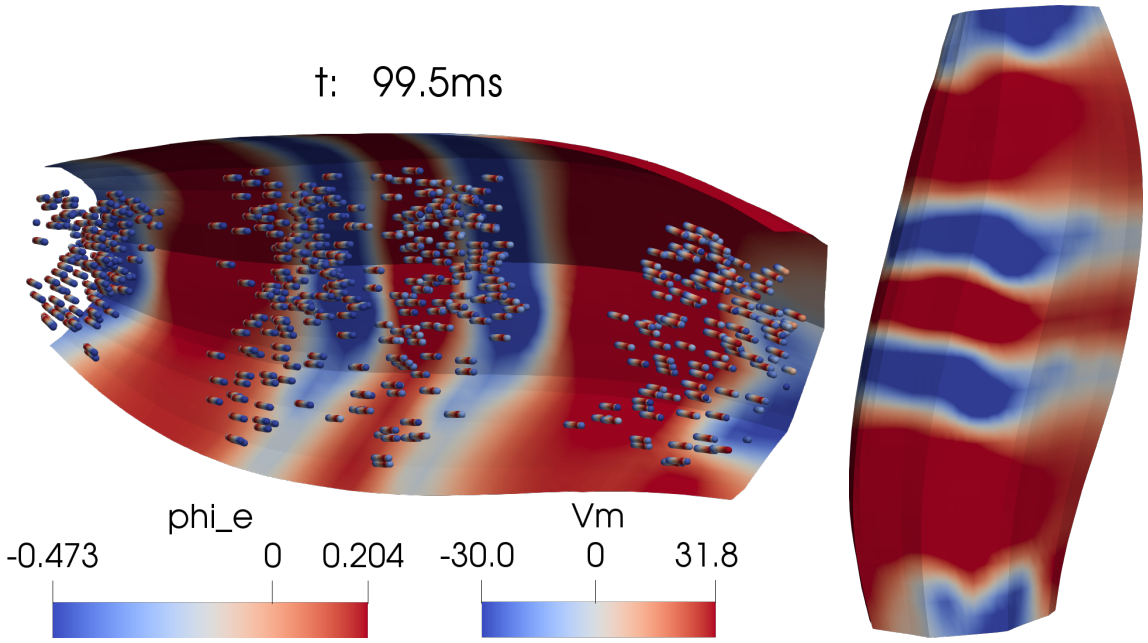
Figure 1.5b shows the skin surface as seen from the back in Fig. 1.5a. The active region is located at the right side in this image. It can be seen that the active region on the skin surface, which results from fibers of the activated MU 1, only spans a small portion of the surface.



(a) Membrane voltage V_m at active parts of the fibers (foreground) and EMG signals ϕ_e on the skin surface (background).

(b) Resulting surface EMG.

Figure 1.5: Simulation result at $t = 99.5$ ms where only motor unit 1 is activated.



(a) Membrane voltage V_m at active parts of the fibers (foreground) and EMG signals ϕ_e on the skin surface (background).

(b) Resulting surface EMG.

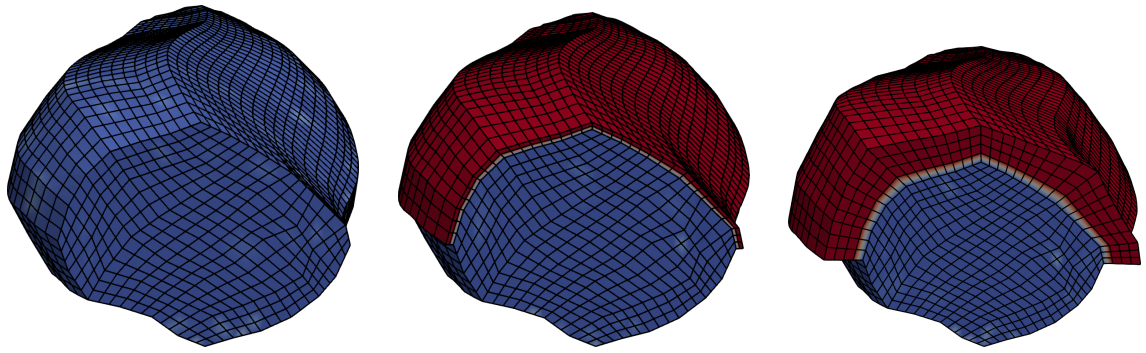
Figure 1.6: Simulation result at $t = 99.5$ ms where only motor unit 20 is activated, analogous to Fig. 1.5.

Figure 1.6 shows the analoguous scenario that activates MU 20 instead of MU 1 is shown. Figure 1.6a shows that, now, more fibers are activated as MU 20 is larger than MU 1. According to the MU layout in Fig. 1.4b, the active fibers are also located closer to the skin surface. This layout results in a stronger EMG signal compared to the previous scenario.

The color coding in the two scenarios in Figures 1.5 and 1.6 is identical and it can be seen that the absolute value of the extracellular potential ϕ_e is larger in the scenario of MU 20. For the scenario with MU 1 in Fig. 1.5, the value range of the extracellular potential ϕ_e is $[-0.473\,\text{mV}, 0.204\,\text{mV}]$. For the scenario with MU 20 in Fig. 1.6, it is $[-0.834\,\text{mV}, 0.579\,\text{mV}]$, which is more than double the range.

Figure 1.6b shows the overall EMG signal on the skin surface for MU 20. Compared to the result of MU 1 in Fig. 1.5b, nearly the inverse region is activated. It can, thus, be observed that the EMG signal is highly influenced by the location and size of the MUs. MUs with territories closer to the skin surface have a larger effect on the EMG signals than MUs that are located further away. As seen in Fig. 1.5b, the influence of fibers completely vanishes for more than a certain distance. On the contrary, the effects of several close fibers add up such that large MUs located near the surface have the most impact on the resulting EMG signal.

1.4.3 Effects of the Fat Layer on the Electromyography Signal



(a) Muscle mesh without fat layer. (b) Muscle with thin fat layer. (c) Muscle with thick fat layer.

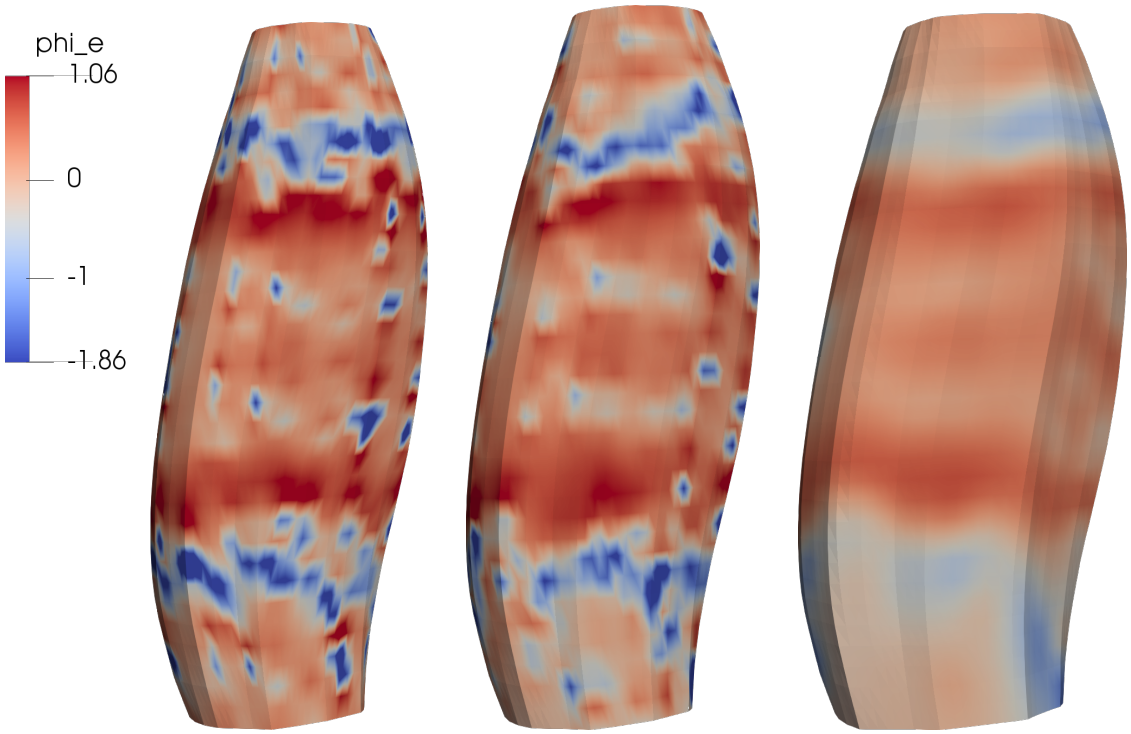
Figure 1.7: Meshes for the muscle domains (blue) and the layer of adipose tissue (red) used in the study to compare different fat layer widths.

In the next study, we investigate the effect of the fat layer on the resulting EMG signals. The same scenario as in the previous section is used, except that the size of the body fat

domain is varied and the activated MUs are chosen differently. We consider the domains and meshes shown in Fig. 1.7: Scenario (a) only considers the muscle domain without additional fat layer. Scenario (b) adds a thin fat layer with thickness of 2 mm, discretized by two layers of finite elements. Scenario (c) considers a fat layer with thickness of 1 cm and four layers of elements. The scenario in the previous section also used this thick fat layer.

In this series of experiments, the first 10 MUs are activated with different stimulation frequencies ranging from 7 Hz for the smallest MU to 15.15 Hz for MU 10. The runtime of the simulation for one scenario on the same hardware as in the previous section is approximately 9 min.

Figure 1.8 shows the simulation results at $t = 100$ ms for the three scenarios with different fat layers. The figure uses the same color coding for the extracellular potential ϕ_e in all three scenarios. It can be seen that the volume conduction in the fat layer significantly smooths the resulting EMG signal, especially for the thick fat layer. The scenarios with no fat layer and the thin fat layer also exhibit a small difference. This effect has implications for experimental studies, where the EMG recordings capture the less resolved spatial information, the more tissue is located between the muscle and the surface electrodes.



(a) Simulation without fat layer.

(b) Simulation with a thin fat layer.

(c) Simulation with a thick fat layer.

Figure 1.8: Simulated surface EMG signals for the different fat layers shown in Fig. 1.7.

How To Reproduce

The simulations in this section use the examples `examples/electrophysiology/fibers/fibers_emg` and `examples/electrophysiology/fibers/fibers_fat_emg` with the variables file `20mus_fat_comparison.py`.

The scenario data that is necessary to run the simulations are given in the repository at github.com/dihu-stuttgart/performance in the directory `opendihu/18_fibers_emg`. The main scripts that runs the simulations for the two sections are the following:

```
./run_single_MUs.sh
./run_compare_fat_layer.sh
```

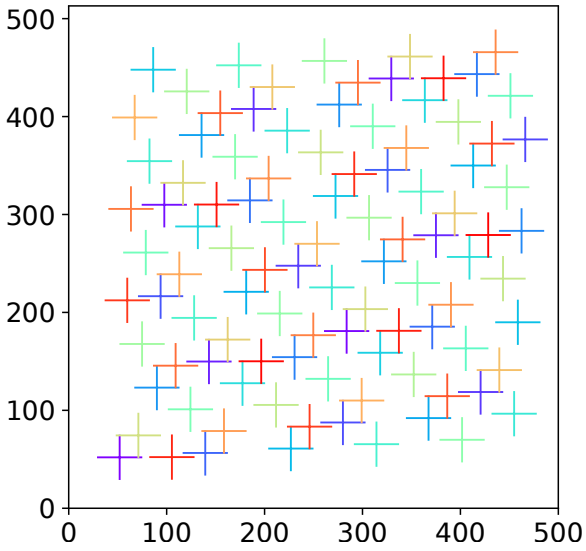


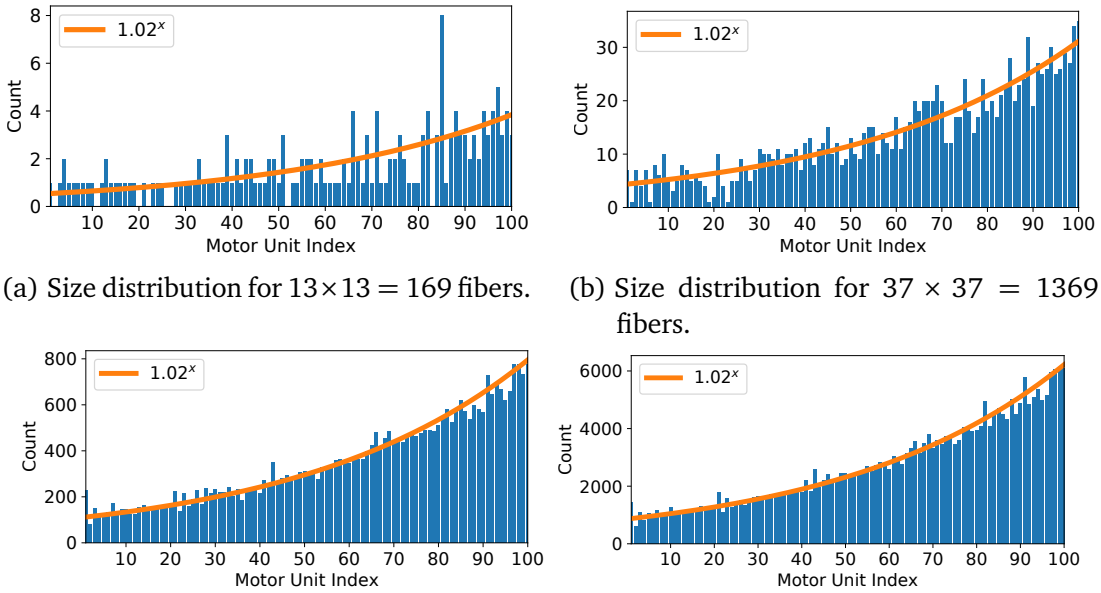
Figure 1.9: MU territory center points. The shown center points of the 100 motor units are used in all different scenarios within the study of different mesh widths.

1.4.4 Effects of the Mesh Width on the Electromyography Signal

Next, we compare the simulated EMG signal for different numbers of fibers and resolutions of the 3D mesh. We consider a scenario with 100 MUs and increase the spatial resolution and the number of processes that execute the computation on the supercomputer Hawk at the High Performance Computing Center Stuttgart.

We compute scenarios with between 1369 and 273 529 fibers. The specified number of 100 MUs have to be assigned to these numbers of fibers for each scenario. We use the method 1a of the algorithm described in **??**. The MU territories are centered around quasi-randomly generated center points, as shown in Fig. 1.9. It can be seen that the MU territory center points are homogeneously distributed in space.

For every fiber, the algorithm assigns a MU with a close center point with higher probability than a MU whose center is located further away. The total number of fibers per MU is progressing exponentially for the MUs from 1 to 100. The progression is described by an exponential function with basis 1.02. Figure 1.10 shows the MU size distributions for four scenarios with increasing numbers of fibers from 169 to 273 529. For 169 fibers in Fig. 1.10a, not all 100 MUs get associated with a fiber. Further, it can be seen that the error of the actual size distribution to the exponential function decreases with increasing number of fibers. For the largest scenario in Fig. 1.10d, the MU sizes range from 602 to 6097 fibers.



(a) Size distribution for $13 \times 13 = 169$ fibers.

(b) Size distribution for $37 \times 37 = 1369$ fibers.

(c) Size distribution for $187 \times 187 = 34\,969$ fibers.

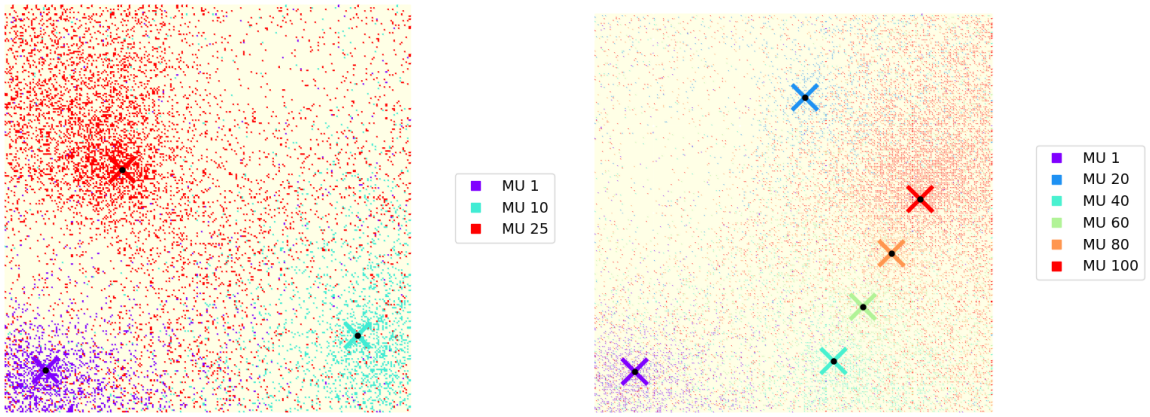
(d) Size distribution for $523 \times 523 = 273\,529$ fibers.

Figure 1.10: Distribution of the sizes of the 100 MUs in the scenarios with different number of fibers.

The number of approximately $3 \cdot 10^6$ fibers in the largest scenario matches the realistic number in a biceps muscle [Mac84]. The number of MUs can be higher in reality, e.g., by a factor of 5 [Fei55]; [Mac06]. Thus, the modeled MUs in this scenario can be seen as a combination of multiple real MUs. Especially the smallest MUs, which in reality can consists of only some dozens of fibers, are lumped by the first few MUs in our scenario. We restrict the number of MUs to 100 to be able to simulate the same problem also with smaller resolutions, e.g., with only 169 fibers.

As described in **??**, the MU assignment algorithm asserts that neighboring fibers are part of different MUs by splitting the assignment problem for the given set of fibers into four smaller problems and then interleaving the results of the four parts. Figure 1.11a shows the first such part, where 25 MUs are associated to a subset of the fibers for the largest scenario with 273 529 fibers. It can be seen that the three visualized MUs are largely clustered around their MU territory centers.

The final association of fibers and MUs is given in Fig. 1.11b. Six selected MUs are shown of which the first, MU 1, corresponds to the first MU in Fig. 1.11a. The figure shows that the fibers, especially the ones of the larger MUs, are distributed far across the muscle. Comparing the smallest MU, MU 1, with the largest MU, MU 100, gives an impression of the MU size differences in this scenario.



(a) Partial problem with a quarter of the whole set of fibers and 25 MUs, which occurs in the algorithm 1a described in **??**. Only three MUs are shown. The MU territory centers are indicated by crosses.

(b) The resulting assignments of MUs to fibers after four parts similar to (a) have been combined, here only shown for six MUs.

Figure 1.11: Association of MUs to the fibers. The square domain corresponds to a cross-section in the muscle, every colored point is one fiber and the color corresponds to the MU.

#fibers	3D stride		2D surface mesh	3D dofs (k=1000)	0D dofs	#proc.	#comp. nodes
	x, y	z					
$37^2 = 1369$	2	8	19×186	67 k	8109 k	144	3
$67^2 = 4489$	2	4	34×371	428 k	26 592 k	448	7
$109^2 = 11\,881$	2	3	55×495	1497 k	70 383 k	1152	18
$187^2 = 34\,969$	2	2	94×741	6547 k	207 156 k	3600	57
$277^2 = 76\,729$	2	1	139×1481	28 614 k	454 542 k	7744	121
$523^2 = 273\,529$	2	1	262×1481	101 661 k	1620 M	26 912	421

Table 1.1: Parameters of spatial discretization and parallel partitioning for the EMG study. The 3D stride refers to the stride with which the 3D mesh is generated from the 0D points. The 2D surface is the output of the EMG and corresponds to one face of the 3D mesh.

The numeric parameters of the simulations are the same as in the last section. The scenario is computed for a simulation time span of 1 s. The MUs are activated in a ramp every 2 ms such that all MUs are active after 200 ms. The fiber radius and the stimulation frequency for the MUs are exponentially distributed with basis 1.02, similar to the MU size. The fiber radius increases from 40 µm to 55 µm and the stimulation frequency decreases from 24 Hz to 7 Hz for MUs 1 to 100. A random frequency jitter of 10 % is assumed.

The surface to volume ratio A_m of the membrane is determined by assuming a cylindrical shape and can be computed from the fiber radius r by $A_m = 2/r$ [Klo20]. We model 70 % slow twitch and 30 % fast twitch fibers. Accordingly, the membrane capacitance C_m is set to $C_m = 0.58 \frac{\mu F}{cm^2}$ for the 70 smallest MUs and to $C_m = 1 \frac{\mu F}{cm^2}$ for the 30 largest MUs.

Table 1.1 lists the spatial discretization and parallel partitioning parameters. The first column shows the number of fibers. Their number increases, however, the mesh resolution of every 1D fiber mesh stays constant at 1480 elements per fiber. The stride that defines the 3D mesh is given in the second and third columns. The stride in radial direction of the muscle, i.e., in the x and y coordinate directions, stays constant. Because the fiber density increases, the 3D mesh is refined accordingly. The stride along the fibers, i.e., in z direction is reduced such that the mesh widths of the 3D mesh in all three coordinate directions remain balanced.

The resulting EMG recordings of each simulation are described by 2D meshes, which contain the values of the 3D muscle meshes without fat layer on the surface at one side of the muscle. The fourth column in Tab. 1.1 lists the dimensions of these surface meshes.

The next two columns list the number of dofs in the 3D mesh and the number of dofs in all fibers. For these scenarios, it is not practical to output the 3D mesh or the 1D fiber meshes in regular time intervals, because this would produce large amounts of data that could hardly be processed. Instead, we only output the 2D surface mesh in the ParaView format every 10 ms.

The last two columns in Tab. 1.1 show the numbers of processes and compute nodes that are used on Hawk. One compute nodes contains 128 physical cores and always four cores share a 16 MiB level three (L3) cache. However, we decide to use only 64 cores per compute node, i.e., two cores per L3 cache, because measurements showed that this reduces the overall computation times more than is lost by the decreased parallelism of not using all physical cores on a compute node. For example, the total computation time of this scenario with a timespan of 1 s is 2 h20 min for the scenario with 76 729 fibers and 7744 processes.

Figures 1.12 and 1.13 show the resulting surface EMG signals for different resolutions. The color visualizes the value of the extracellular potential ϕ_e according to the shown color bar. Because of sign conventions in the definitions of the electric potentials, the spikes in the EMG signals, which result from the action potentials, are negative.

The resulting electric potentials in Fig. 1.12 exhibit different regions of higher activation that move over time from the center of the muscle towards its ends. The size of these regions at time $t = 179.5\,\text{ms}$ decreases from Fig. 1.12a to Fig. 1.12d as the mesh width decreases. Dark colored strong signals can be seen, which mainly correspond to fibers that are located directly underneath the shown muscle surface. Apart from these strong signals, also weaker artifacts occur, which are shown in yellow and orange colors. They result from the superposition of several fibers of the same or different MUs. The number of recognizable weak signals is higher for the simulations with higher numbers of fibers and finer mesh resolution.

The four scenarios in Fig. 1.12 share the material parameters, territory centers and relative size distributions of the 100 MU and the activation scheme. However, the location of the neuromuscular junctions is determined randomly and varies between the scenarios. Therefore, the resulting EMG signals are not refined images of each other. However, a similarity of activated regions on a coarse scale can be observed in all scenarios.

Figure 1.13 shows two more scenarios with large numbers of fibers at times of $t \approx 1\,\text{s}$ and $t \approx 0.4\,\text{s}$. The scenario in Fig. 1.13a simulates approximately $76 \cdot 10^3$ fibers, which is in the order of a third of the realistic number of fibers in the biceps muscle. This scenario can also be interpreted as only activating a third of the available fibers in the muscle, resulting in the respective lower percentage of maximum voluntary contraction force.

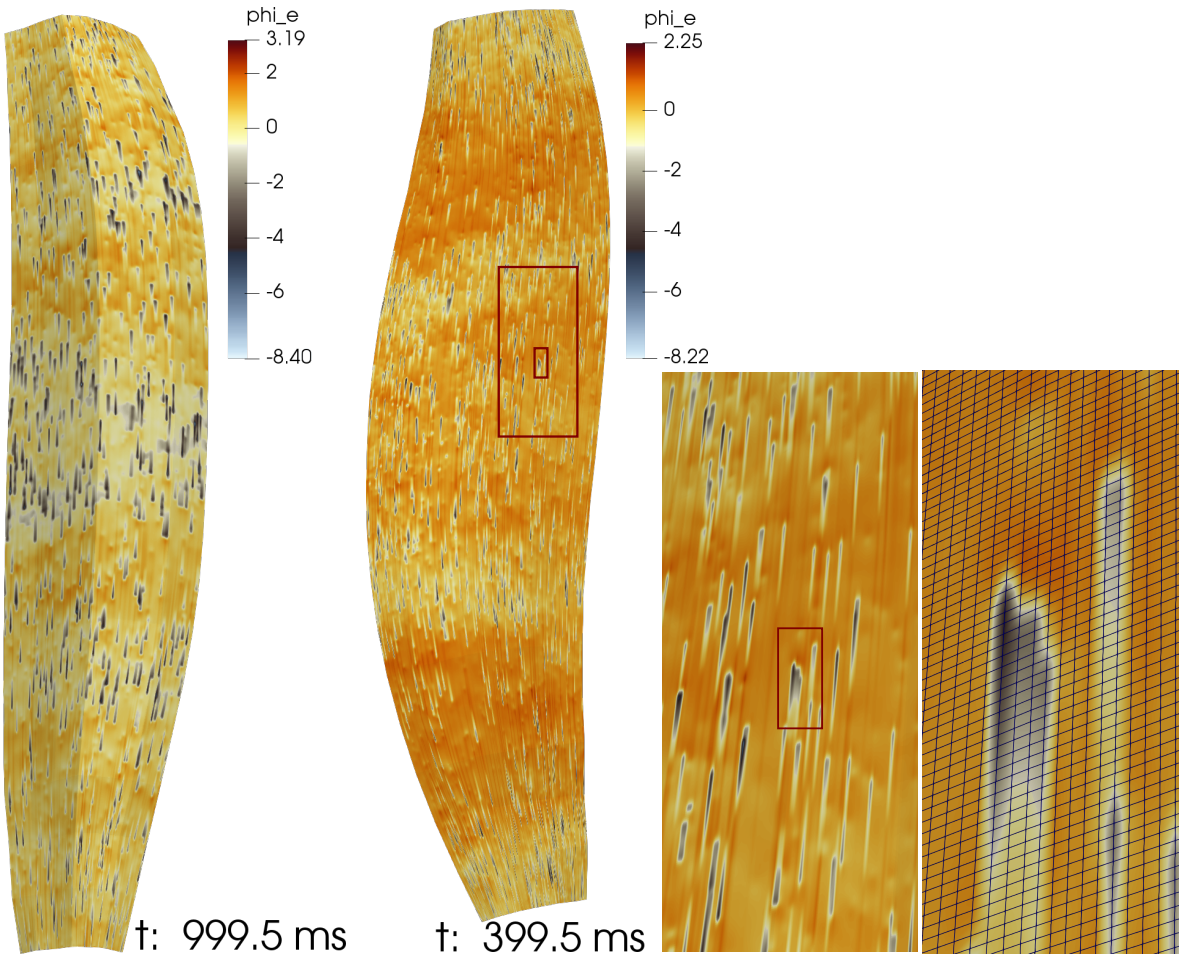
Figure 1.13b shows the scenario where a realistic number of $273 \cdot 10^3$ fibers was simulated. The computational effort for these two scenarios can only be tackled with High Performance Computing. The last two rows of Tab. 1.1 show that 121 respectively 421 compute nodes were used for the computations.

Figures 1.13c and 1.13d present details of the simulated surface EMG of the scenario in Fig. 1.13b. The cut outs are indicated by the red boxes in Figures 1.13b and 1.13c. Figure 1.13d also visualizes the elements of the 2D and 3D meshes. In both scenarios of Fig. 1.13, the 3D mesh is as finely resolved in z direction (vertical in Fig. 1.13d) as the muscle fibers. In transversal direction (horizontal in Fig. 1.13d), the element sizes are twice as large as the spacing between the fibers.

(a) 1369 fibers. (b) 4489 fibers. (c) 11 881 fibers. (d) 34 969 fibers.

Figure 1.12: Simulated surface EMG signals for different numbers of fibers and different mesh widths of the 3D mesh, see Tab. 1.1 for details. The same color coding of the EMG signal ϕ_e is used in the four scenarios.

The right side of Fig. 1.13d shows two almost aligned action potentials that propagate towards the top of the image. The upper action potential originates from a fiber that is located at the center between the element boundaries. Its electric potential is distributed to two adjacent nodes on the surface mesh, having the same activated values. In contrast, the lower action potential results from a fiber that is directly located on the element boundaries. It can be seen that the activation on the surface decays rapidly in transverse (horizontal) direction, as the neighbor elements already almost exhibit the same electric potential as the background level. This underlines the importance to use finely resolved meshes to accurately represent surface EMG signals.



(a) 76 729 fibers. (b) 273 529 fibers. (c) Detail view. (d) Detail view.

Figure 1.13: Simulated surface EMG signals with realistic fiber counts, continued from Fig. 1.12. Figure (a) shows a scenario with 76 729 fibers, which is approximately a third of the realistic number for the biceps muscle. Figures (b)-(d) show the result with a realistic number of 273 529 fibers. (c) shows a detail view of (b) indicated by the outer red box. (d) shows another zoomed in view of (b) and (c), also indicated by the red boxes.

How To Reproduce

Use the following commands to run the EMG simulation of the biceps muscle with fat layer and electrodes:

```
cd $OPENDIHU_HOME/examples/electrophysiology/fibers/fibers_fat_emg/
  ↪ build_release
mpirun -n 16 fibers_fat_emg ../settings_fibers_fat_emg.py 50mus.py
cd out/50mus
plot_emg.py ./electrodes.csv ./stimulation.log 25900 26000    # plot
  ↪  the result, here for time span 25.9s - 26s
```

1.4.5 Simulation of EMG electrodes

While the surface EMG simulation results as presented in the last section in Fig. 1.12 are suited for insights into the temporal and spatial variation of the electric potential, real experiments are constraint to capture values at the discrete locations of electrodes. For some applications such as the evaluation of EMG decomposition algorithms, it is beneficial to obtain simulated values at electrode locations.

One possiblity would be to extract nodal values from the simulated surface meshes to simulate electrodes. However, the distance between the nodes in the mesh is not constant in the whole mesh, whereas EMG electrode arrays have fixed inter electrode spacings. We, therefore, follow a different approach and allow to directly specify a grid of electrodes close to the muscle surface. These points are then mapped onto the surface of the muscle and the respective values are calculated by evaluating the finite element interpolant at the respective locations.

In OpenDiHu, a 2D grid of surface electrodes can be defined in the Python settings file by specifying the grid parameters and inter electrode distances. In result, the simulation distributes the electrodes to the processes according to the parallel partitioning of the 3D mesh, evaluates the computed EMG values at the respective locations and outputs them in a single text file of comma separated values.

Figure 1.14 shows simulation results of the fiber based electrophysiology model with 49 fibers, fat layer and an array of 12×32 electrodes. The electrodes are visualized as spheres. The muscle fibers below the fat layer are colored according to the transmembrane voltage V_m. Of the fat layer only the upper surface is shown and colored according to the

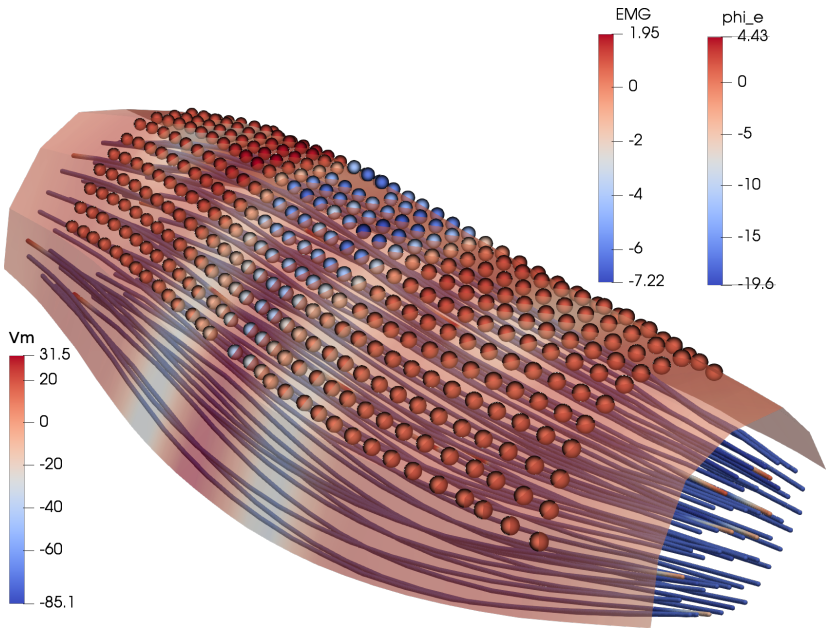
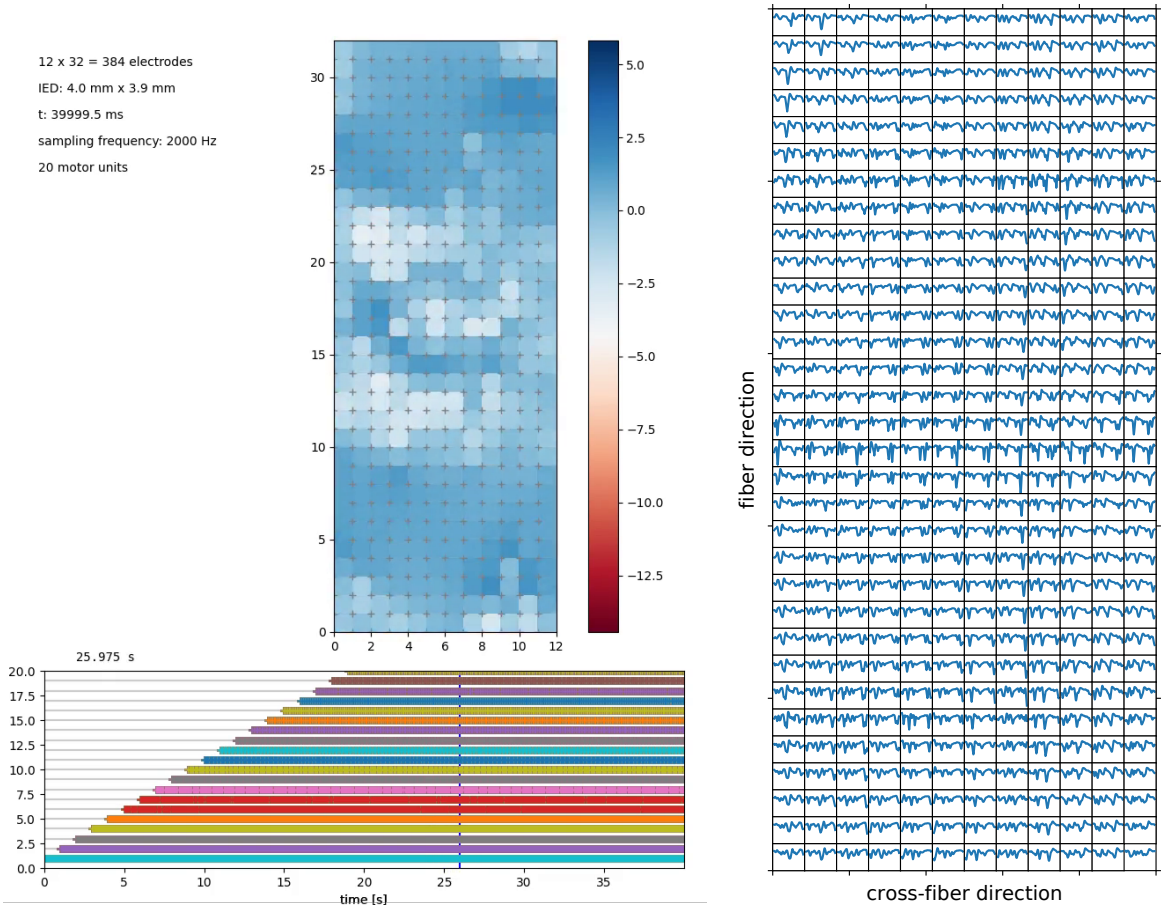


Figure 1.14: Simulation of surface EMG and capturing electrodes. The scenario contains 49 muscle fibers, a fat layer of which only the surface is shown and a grid of 12×32 equidistant electrodes.

extracellular potential ϕ_e. The EMG electrodes capture the values of the scalar field ϕ_e at their locations. The color coding for the electrodes has a different *EMG* color scale to make the resulting signals more distinguishable. Two activated bands across the muscle surface can be seen, which are also present in the electrode values.

To visually evaluate the simulated EMG signals at the electrodes, OpenDiHu provides utilities to create the visualizations shown in Fig. 1.15. Figure 1.15a shows a still image from an animation. On the upper right, the grid of electrodes is displayed. The EMG signal at the electrodes is given by the colored tiles and changes over time. At the bottom of the image, the activation times of the MUs are visualized. Every horizontal line corresponds to one MU. The colored markers indicate when the respective MU fires. As the shown example visualizes data for 40 s, the individual firing times are not distinguishable. In the animation, a vertical bar moves over the time axis and indicates the current simulation time. The picture displays the EMG values at time $t = 25.975$ s. The upper left of the image shows a text with static information about the dataset, containing the electrode grid size, the inter electrode distance (IED), the end time, the sampling frequency of the electrodes, i.e., the frequency with which the computed EMG signals values are stored to the output file, and the number of MUs.

Figure 1.15b shows another, static visualization of simulated EMG data. The diagram contains boxes for all electrodes in the 12×32 grid. The value of the EMG signal is plotted



(a) fibers mesh

(b) Surface EMG at 12×32 electrodes in the time range [25.9 s, 26 s].

Figure 1.15: Simulation of surface EMG using electrodes.

over time in every box for the respective electrode. Figure 1.15b visualizes the data of Fig. 1.15a for the time interval [25.9 s, 26 s]. The diagram enables experts to visually identify propagating action potentials from the tile columns. The propagation velocity of the action potentials can be estimated from the time shift of matching spikes in vertically adjacent boxes.

1.4.6 Decomposition of Surface EMG Signals

Surface EMG recordings are a valuable tool to gain insights into the neuromuscular system. They are used, e.g., for the diagnosis of muscular disorders and in clinical studies that aim to advance biomedical understanding.

As described earlier, the EMG signals on the skin surface originate from the activated

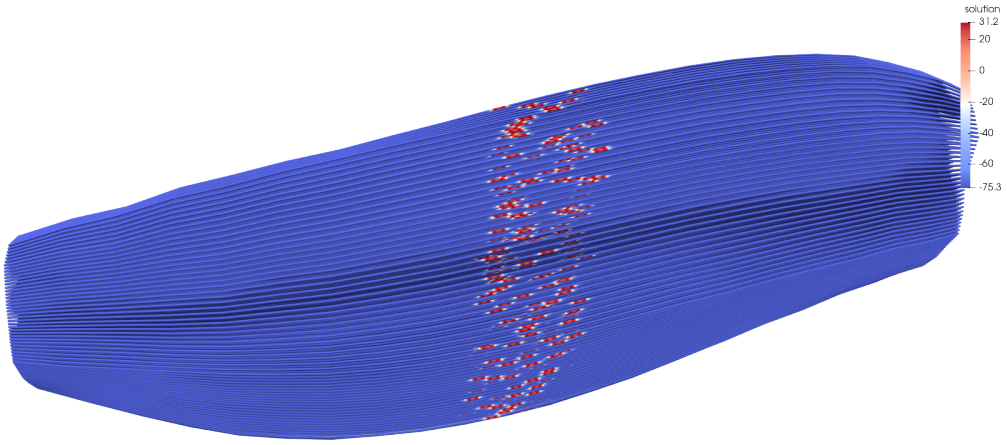


Figure 1.16: A simulation result that reveals the locations of the neuromuscular junctions. The figure depicts 1369 fibers after 1 ms that have initially been stimulated at the neuromuscular junction. The color coding corresponds to the membrane potential V_m, which has a positive value near the points of stimulation.

muscle fibers. Effects from volume conduction of action potentials on all muscle fibers are superpositioned and contribute to the EMG signal. The scaling of the contributions to the overall signal depends on several factors, such as the distance of the fibers to the skin surface. As all fibers in the same MU get activated simultaneously, each MU's contribution shows a characteristic "shape" in the resulting surface EMG signal. This shape is influenced by the number and location of the muscle fibers relative to the electrodes and the location of the neuromuscular junctions.

In our simulation, the location of the neuromuscular junctions is chosen pseudo-randomly (but deterministic) during initialization in the central 10 % of every muscle fiber. Figure 1.16 shows the state of a simulation with 1369 fibers at $t = 1\,\text{ms}$, where all fibers have been activated at $t = 0\,\text{ms}$. The color coding indicates the potential V_m of the membrane, which at the shown time has only depolarized near the locations of the neuromuscular junctions.

Methods exist that seek to decompose the surface EMG signal into the contributions of the individual MUs. One popular method is *Gradient Convolution Kernel Compensation* (gCKC) [Hol07a]; [Hol07b], which, in the following, will be outlined and then applied on simulated data.

Most decomposition methods, including the gCKC algorithm, assume that the EMG signal at an electrode is composed of the convolutional mixture of N filters of the MU activity. The activity of each MU $k \in \{1, \ldots, N\}$ is described by the innervation pulse trains that activate the fibers of MU k, given as a point process of neural inputs at stimulation

times φ_r. The source signal s_k in the muscle that represents the effect of MU k is described as a filter over these neural drives, $s_k(t) = \sum_r \delta(t - \varphi_r)$, where δ is the dirac delta function.

The vector of observed EMG values $\mathbf{x} \in \mathbb{R}^m$ at a time t is composed of the temporal convolution over L time-shifted sources $\mathbf{s}$ and a term $\boldsymbol{\omega}$ of additive Gaussion noise:

$$\mathbf{x}(t) = \sum_{\ell=0}^{L-1} \mathbf{H}(\ell)\,\mathbf{s}(t-\ell) + \boldsymbol{\omega}(t).$$

Here, $\mathbf{H}$ is the $m \times n$ mixing matrix for m observations and n MU sources and $\mathbf{s} = (s_k)_{1,\dots,n}$ is the vector of source signals. The sum over L previous values in this convolutive mixture can be reformulated by moving the summation into the matrix-vector product. The dimensions of the matrix $\mathbf{H}$ and the vector $\mathbf{s}$ are extended accordingly. Inversion of the extended mixture matrix yields the separation vectors, with which the innervation pulse trains φ_r of the MUs can be recovered from the recorded EMG signals $\mathbf{x}$. The gCKC algorithm performs the inversion indirectly by solving a derived optimization problem using a gradient descent scheme.

The gCKC decomposition algorithm is implemented in the DEMUSE software, a commercial, MATLAB based tool that allows automatic and semi-automatic EMG decomposition [Hol08]. In collaboration with Lena Lehmann from the *Institute of Signal Processing and System Theory* and the *Institute for Modelling and Simulation of Biomechanical Systems*, we evaluated the performance of gCKC decomposition on simulated surface EMG signals.

We simulate fiber-based electrophysiology scenarios with fat layer and 1369 fibers using the same model parameter as in Sec. 1.4.4. In the first scenario, a fat layer with thickness of 1 cm is modelled. The simulated EMG signal is sampled in an electrode array with a frequency of 2 kHz and a grid size of 12×32 fibers, as shown in Fig. 1.15.

Figure 1.17 shows the firing times of the 20 MUs in the first 10 s. The different MUs are initially activated every 100 ms to generate the shown "ramp" activation pattern, which later helps to identify the recovered MUs from the decomposition. From $t = 1.8$ s on, all MUs fire with their respective constant frequency, subject to jitter values of 10 %.

In this first scenario, the gCKC decomposition algorithm was applied on the first $t = 40$ s of simulated EMG data. The preconfigured algorithm in DEMUSE was used without manual intervention. While the simulated EMG recording consisted of an electrode grid of 12×32 fibers, only a rectangular subset of 5×13 channels at the lower center of

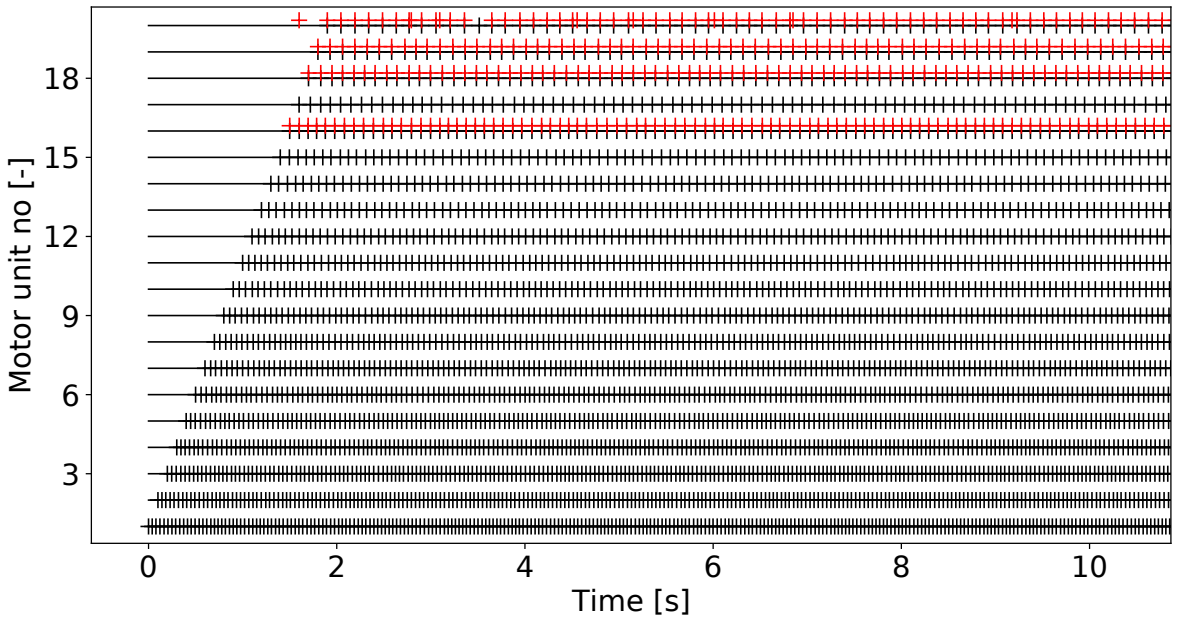


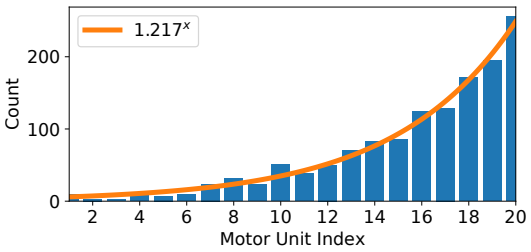
Figure 1.17: Match of EMG decomposition results with simulated data. The firing pattern over time for the 20 MUs in the simulation is shown by black markers. The recovered firing times of the gradient convolution kernel compensation algorithm are given by the red markers. The algorithm detected the four MUs 16, 18, 19 and 20.

the grid was used for the decomposition to mimic a realistic electrode array size. The DEMUSE software discarded four of these 65 channels as being invalid.

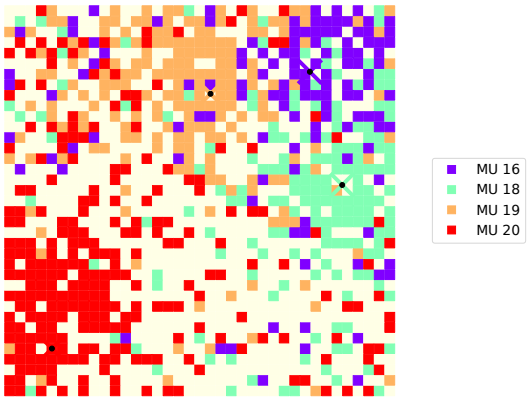
Figure 1.17 shows the innervation pulses that were detected by DEMUSE as red vertical markers. A time span of 50 s was simulated of which only the first 11 s are visualized in Fig. 1.17. DEMUSE found four MUs in this scenario, i.e., 20 % of the 20 simulated MUs. The recovered MUs were identified in the set of simulated MUs by matching the average firing frequency and the activation onset time in the ramp scheme. A first visual comparison with the original stimulation times given by the black markers shows a good agreement.

In this scenario, the association of fibers with MUs followed an exponential MU size progression with a basis of approximately 1.2, as shown in Fig. 1.18a. The smallest MU contained two fibers and the largest MU had 256 fibers. The method 1 described in **??** was used to generate the association between fibers and MUs.

Figure 1.18b depicts the location of the four MUs that were detected by DEMUSE. The detected MUs have the indices 16, 18, 19 and 20 and correspond to four of the five largest MUs. It can be seen that MUs 18 to 20 are located mainly in the upper half of the muscle

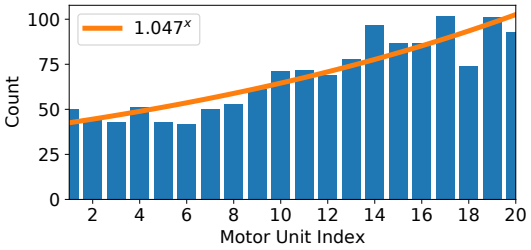


(a) Number of fibers of the 20 MUs in this scenario, following an exponential progression with basis 1.217.

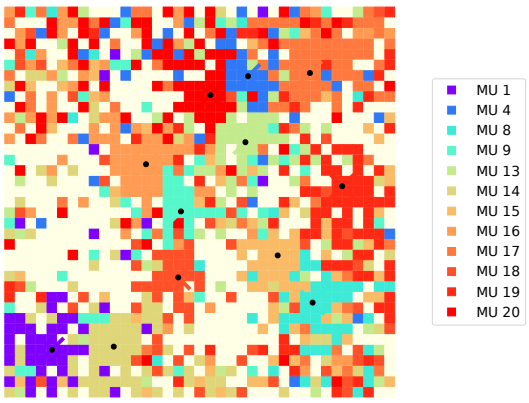


(b) Spatial location of the fibers of the four MUs that were recovered by the EMG decomposition algorithm, indicated by different colors. The territory center points of the MUs used in the generation algorithm are indicated by black dots. The yellow background area corresponds to other MUs, which were not detected.

Figure 1.18: Association of the fibers with motor units for the first scenario with 20 MUs, given in Fig. 1.17.



(a) Progression of MU sizes. The number of fibers per MU is more balanced in this scenario than in Fig. 1.18a.



(b) Spatial fiber arrangement of the MUs that were detected by the EMG decomposition algorithm, with their center points given by black dots.

Figure 1.19: Association of the fibers with motor units for the second scenario with 20 MUs, given in Fig. 1.20.

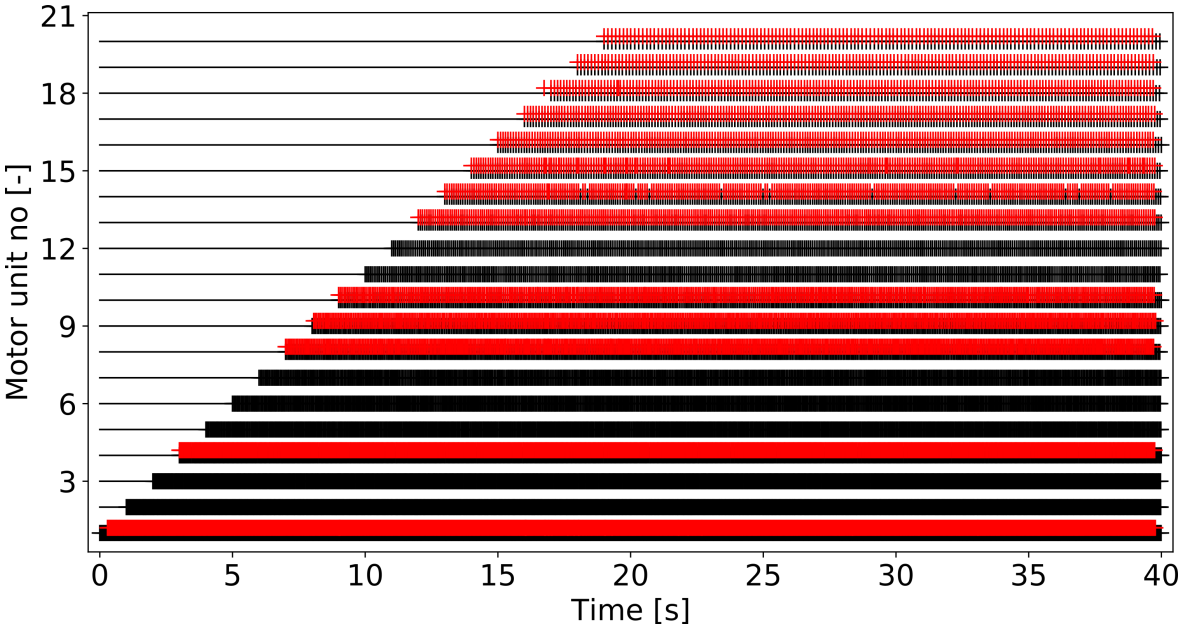


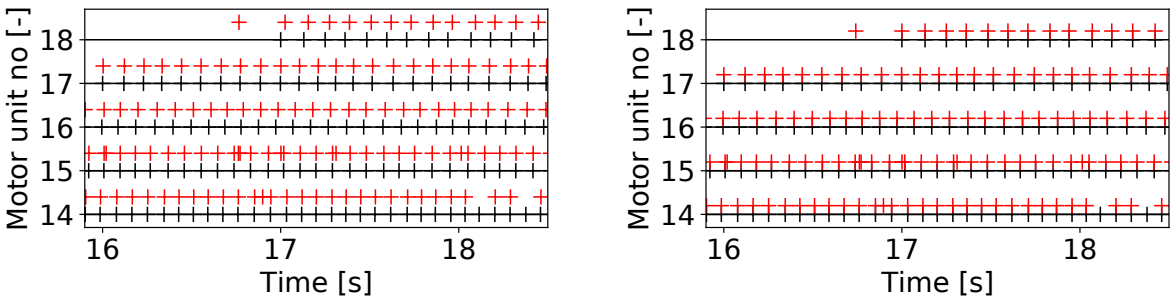
Figure 1.20: Activation pattern for the second scenario with 20 MUs. The activation times used in the simulation are shown as black markers, the recovered activation pulses of the EMG decomposition algorithm are shown as red markers.

cross-section, in proximity to the electrode array at the top of the diagram. The MU with the most fibers, MU 20, was detected by the decomposition algorithm even though it is located at the lower left of diagram at a large distance to the skin surface.

Two further scenario were simulated with the same parameters as the first scenario in Fig. 1.17, but instead with 50 and 100 MUs. In these datasets, DEMUSE was able to detect 8 and 12 MUs, which corresponds to 16 % and 12 %.

Moreover, another scenario with 20 MUs was computed, but the fat layer was varied to have a thickness of only 2 mm instead of 1 cm. In addition, the association scheme between MUs and fibers was changed to the one shown in Fig. 1.19. The exponential distribution of MU sizes only varied between 42 and 102 fibers per MU, corresponding to a basis in the exponential function of approximately 1.05 instead of 1.2.

Figure 1.20 shows the results of the EMG decomposition with the gCKC algorithm for this second scenario with 20 MUs. DEMUSE successfully decomposed the signal into 13 MUs, corresponding to 65 % of the 20 simulated MUs. DEMUSE also determined two additional MUs, which we do not consider part of the set of successfully recovered MUs. The first dataset only consists of ten innervation pulses, and the second pulse train contains high frequency oscillations. In this scenario, the software marked only one



(a) Original data where a constant time shift between the simulated (black) and recovered (red) markes can be seen, especially for MUs 16 and 18.

(b) Same data as in (a) with applied time offset correction. The applied time shifts for MUs 16 and 18 are −12.9 ms and −24.9 ms, respectively.

Figure 1.21: Excerpts of the detected firing times of MUs 14 to 18 in the second scenario with 20MUs. The stimulation times of the simulation are given by black markers, the recovered times are visualized by red crosses.

EMG recording channel as invalid, which means that more data was considered by the decomposition algorithm than in the first scenario with 20 MUs.

Similar to the previously presented scenario, the larger MUs were detected with a higher probability than the smaller MUs. In this scenario, the eight largest MUs were successfully found. Figure 1.19b shows the spatial arrangement of the detected MUs. The area of the muscle cross section that is occupied by undetected MUs is again distributed more distantly to the skin surface at the upper boundary. However, the recovered MUs 1 and 14 are nevertheless located at the lower boundary, i.e., in the most distant area from the EMG electrodes.

Next, we evaluate the quality of the innervation pulse trains that were recovered by the gCKC algorithm in our scenarios. We compare the stimulation times calculated by DEMUSE with the stimulation times of the simulation. Figure 1.21a shows an excerpt of the detected pulse trains of the second scenario with 20 MUs in Fig. 1.20, where the gCKC algorithm recovered 13 MUs. We observe for some MUs that the recovered stimulation times are consistently shifted in time. This effect is especially visible for MUs 16 and 18.

The reference times given by the black markers in Fig. 1.21a correspond to the times when the fibers were stimulated in the simulation in OpenDiHu. The detected MU activations in DEMUSE, however, correspond to the times when the MU action potential shapes in the EMG recording reached their maximum. Moreover, the exact times when particular MUs reach particular EMG electrodes depend on the distance of the electrodes

to the innervation points of the MUs. The further the electrodes are away from the neuromuscular junctions along the muscle, the higher is the delay of the recorded spikes to the corresponding innervation pulses. Thus, the constant time shifts in the pulses detected by the gCKC algorithm are valid and have to be accounted for in the evaluation of the decomposition performance.

We correct for these time shifts by adding constant time offsets Δt_k to the recovered innervation pulse trains. For every MU k, the algorithm finds the matching pairs of simulated and recovered pulses and optimizes the value of Δt_k such that the time differences in these pairs after shift correction get minimal.

Figure 1.21b shows the same extract of MU activity as in Fig. 1.21a with applied time offsets. The time offsets for MUs 14 to 18 in this example are given as

$$\Delta t_{14} = -2.4\,\text{ms}, \qquad \Delta t_{15} = -1.1\,\text{ms}, \qquad \Delta t_{16} = -12.9\,\text{ms},$$
$$\Delta t_{17} = -2.9\,\text{ms}, \quad \text{and} \qquad \Delta t_{18} = -24.9\,\text{ms}.$$

Figure 1.21b shows that the recovered pulses now match the simulated data very well. The non-matching pulses are clearly false positive detections.

To compare the recovered MU times between the scenarios, we evaluate metrics such as the rate of agreement. The MU firing times in the simulation serve as the ground truth to which we compare the recovered MU times. We identify true positive (TP), false positive (FP) and false negative (FN) recovered pulses, depending on whether a matching time to a recovered pulse can or cannot be found in the simulation data within a tolerance of $\varepsilon = 5\,\text{ms}$. The rate of agreement (RoA) between the gCKC algorithm output and the ground truth data is then computed by

$$\text{RoA} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}\,.$$

In the first scenario with 20 MUs in Fig. 1.17, the RoA for MUs 16,18 and 19 is above 99.7 % and slightly lower at 82.2 % for MU 20. Here, only 296 of the 334 detected pulses were true positives, corresponding to a precision of 88.6 %.

In the second scenario with 20 MUS presented in Fig. 1.20, all valid MUs except one have RoA values of above 94.5 %. Five MUs are even detected perfectly with 100 % rate of agreement. MU 9 is the only detected MU with a degraded RoA of approximately 57.9 %. However, the RoA improves to 98.1 %, if the tolerance ε for matching pulses is relaxed to 10 ms. This shows that the RoA metric also depends on a proper value for the tolerance

ε, and that some of the innervation pulse trains detected by DEMUSE can have varying accuracy in a range of less than 10 ms.

While the gCKC algorithm can be used for EMG decomposition of previously recorded signals in a controlled environment, it is less suited for real-time applications such as human-machine interfaces (HMI). The separation vectors that decompose the electrode signals and infer the MU innervation pulse trains can be computed in a training phase. However, their application on new data requires a certain history of previously captured signals to calculate the decomposed MU pulses. In consequence, the predictions are delayed, which is usually undesirable in HMI applications. Furthermore, the system is sensitive to noisy data.

A fundamentally different approach to EMG decomposition is the use of sequence-to-sequence learning methods provided by recurrent neural networks. The authors of [Cla21] used a gated recurrent unit (GRU) network for this task. The network was trained using the output of the gCKC algorithm and was subsequently able to decompose surface EMG signals into innervation pulse trains. The approach was shown to be robust and to outperform gCKC for low signal-to-noise ratios.

To assess, whether our simulations of surface EMG can be used for the supervised learning of GRU networks for EMG decomposition, we tried in a first step to reproduce the studies of [Cla21], where the GRU is trained with the output of the gCKC algorithm. Additionally, we trained a GRU network directly on the simulated EMG data. These tasks were carried out in the masters project of Srijay Kolvekar and were supervised by Lena Lehmann and me. For details on the methods and results, we refer to the literature [Cla21] and the project report [Kol21].

In this project, the EMG decomposition of a GRU network trained with raw innervation pulse trains obtained from the gCKC algorithm, similar to the literature, showed a large number of false positive and false negative predictions. However, a different setup using MU labels instead of raw pulse trains showed promising results. Every discrete point in time (according to the EMG sampling frequency) was either associated with the class of the currently active MU or with the background class, when no MU was activated at the time. This classification problem had a large class imbalance, as the background class was active for 86 % of the timesteps. The issue was mitigated by using class weights. The GRU network was trained with simulation data and yielded per-class rates of agreement of up to 72 % for the two scenarios with 20 MUs shown in Figures 1.17 and 1.20.

Figure 1.22 presents an excerpt of the resulting predictions of a GRU network that was trained with simulation results. We used the simulation of the second scenario with 20

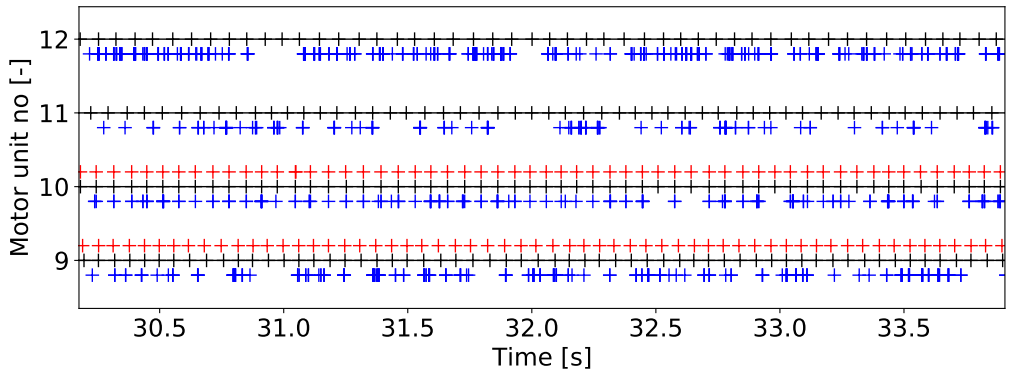


Figure 1.22: Comparison of innervation pulse train predictions of the gCKC algorithm (red), a GRU network (blue) and the ground truth data (black).

MUs, which is shown in Fig. 1.20. The black markers in Fig. 1.22 indicate the stimulation times used in the simulation. The red markers correspond to the recovered times by the gCKC algorithm. Out of the shown MUs, only MUS 9 and 10 were recovered by the gCKC algorithm. The blue markers denote the GRU predictions. Correction of time offsets was performed for both the gCKC and GRU outputs.

Figure 1.22 shows the best agreement between the two prediction methods for MU 10 with a RoA of 99.6 % for the gCKC algorithm and 72.2 % for the GRU network. In contrast to the gCKC algorithm, the GRU network predicts firings for all MUs. However, the quality is only acceptable for MUs that could also be detected by the gCKC algorithm. For MUs 11 and 12, the RoA for the GRU network is around 30 %.

In future work, the decomposition performance of the GRU networks could be improved by using different training data. For example, the ramp activation in the training data could be replaced by constant tetanic stimulations or training data where different single MUs are activated at a time could be investigated.

In conclusion, the gCKC algorithm is able to decompose artifically generated surface EMG signals. This means that our simulation can be used to evaluate the performance of EMG decomposition algorithms.

The number of detected MUs depends on the relation between MU sizes and on the distance of the MU territories to the electrodes. If the variance of the sizes of the activate MUs is small, such as in Fig. 1.19a, also MU are detected that are far away from the electrodes. If, in the opposite case, the sizes of active MUs are distributed over a large range such as in Fig. 1.18a, only the largest MUs are detectable.

In addition, the amount of adipose tissue between the electrodes and the muscle influences the number of MUs that can be recovered. In our studies, the performance of EMG decomposition was lower for all scenarios with thicker fat layer than for the scenario with a thin fat layer.

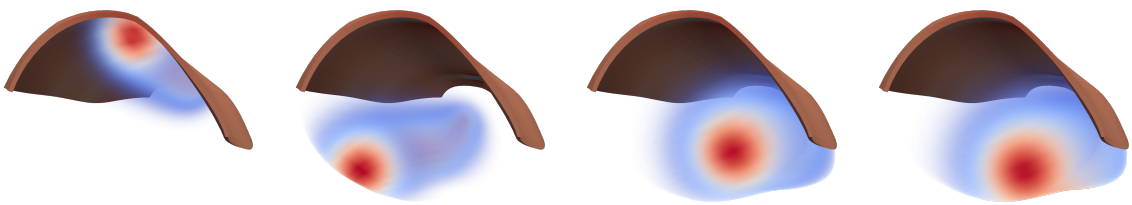
The rate of agreement of the determined pulse trains of the DEMUSE software was above 95 % in most of the cases. Correspondingly, the rate of false positives was low. A time shift between the recovered times and the ground truth data was observed for some pulse trains, however, it can be explained with the delay from first activation to the onset of the EMG signal. As a result, the time shift was corrected for the rate of agreement measurement.

A proof-of-concept implementation of GRU networks showed promising performance for predicting MU firing times from artifical EMG recordings. In future work, the algorithm has to be refined to be comparative to the gCKC algorithm.

begin unfinished text

begin unfinished text

1.5 Solver for the Multidomain Model



(a) f_r^0 (b) f_r^1 (c) f_r^2 (d) f_r^3

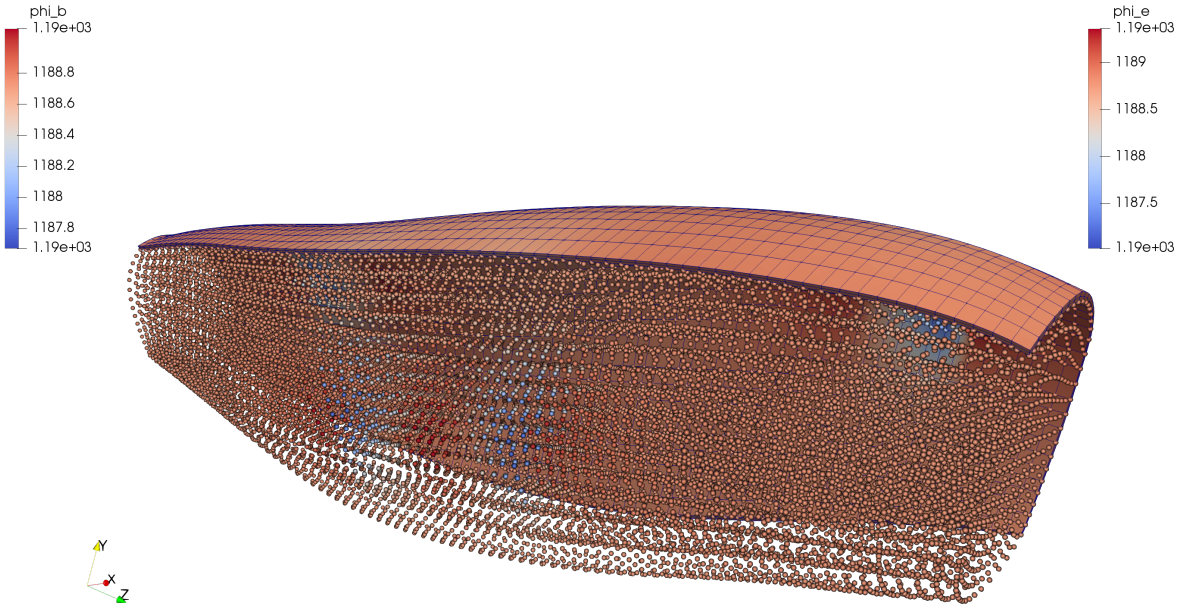
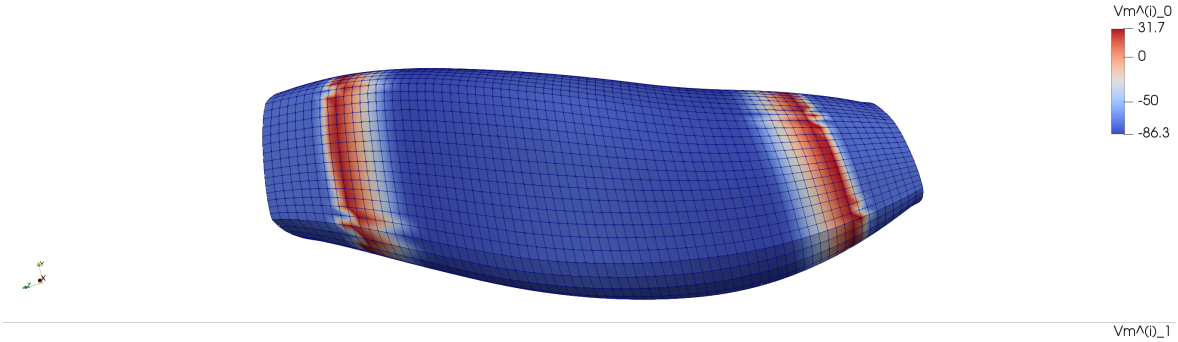
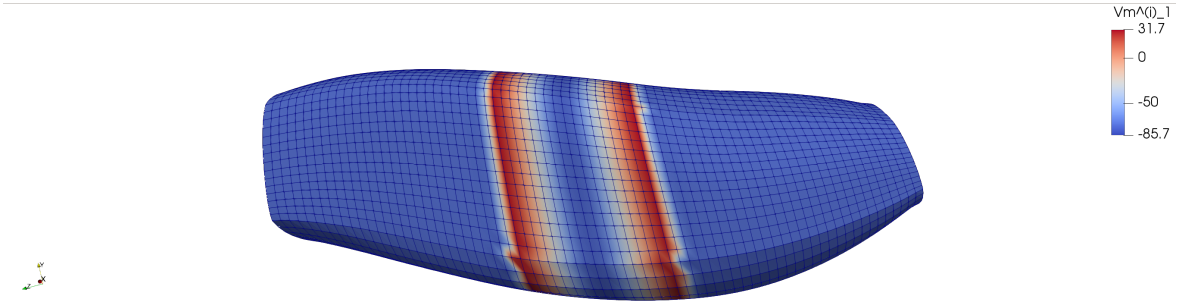
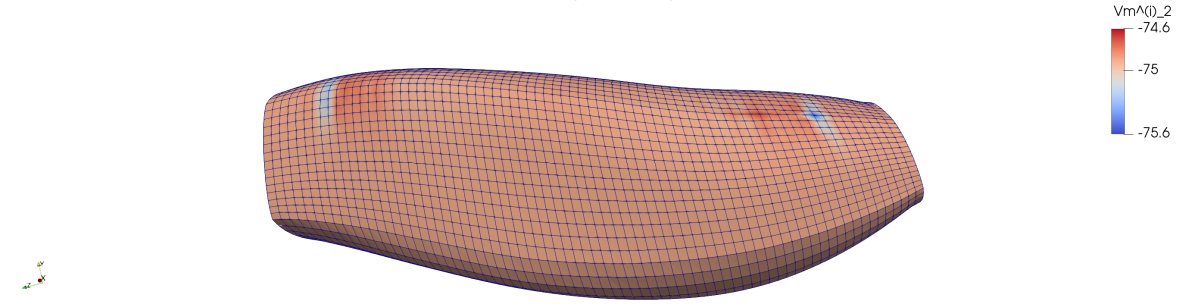


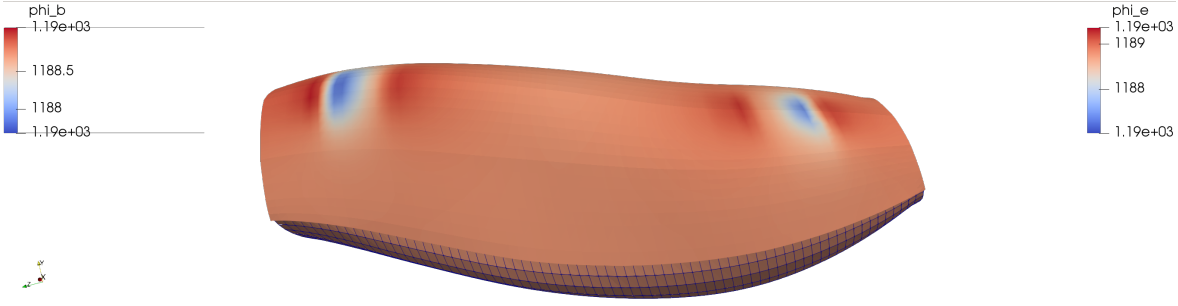
Figure 1.24: multidomain mesh





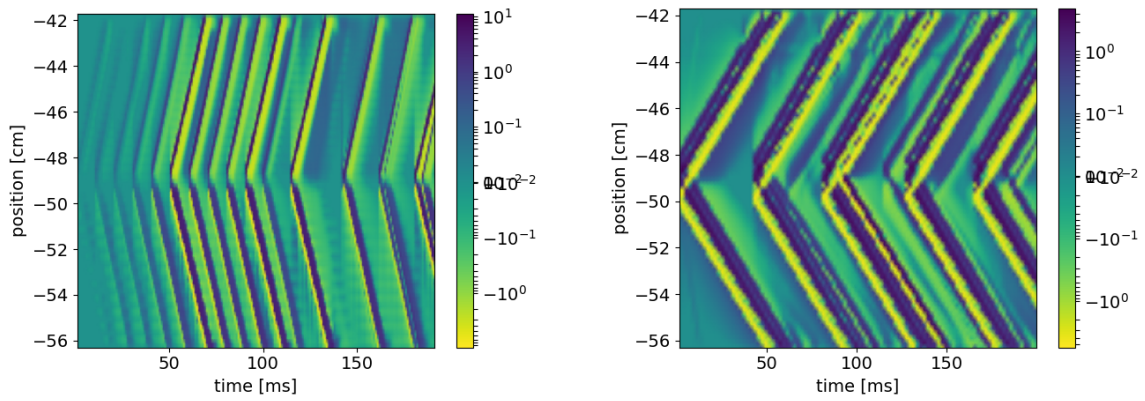
(a) V_m^0 and V_m^1





(b) V_m^2 and ϕ_b

Figure 1.25: gpu.



(a) Multidomain (b) Fibers

Figure 1.26: Fibers and Multidomain

1.6 Electrophysiology and Coupled Solid Mechanics

1.6.1 Linear Mechanics with artifical electrophysiology

1.6.2 Fiber Based Electrophysiology and Muscle Contraction

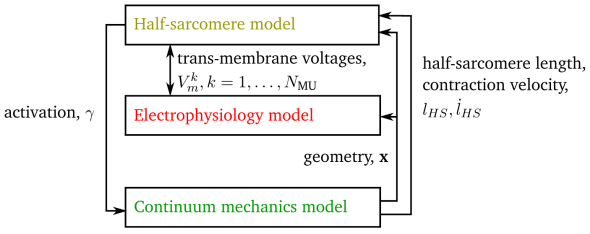


Figure 1.27: prestretch

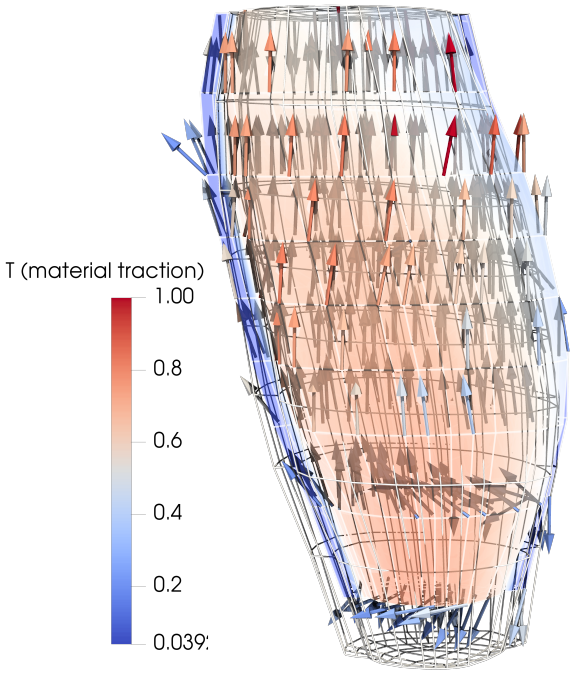


Figure 1.28: solver structure

1.6.3 Coupling with preCICE

1.6.4 With tendons preCICE

1.6.5 Multidomain and Muscle Contraction

1.7 Simulations of the Neuromuscular System

1.7.1 Simulation of Motoneurons With Fiber Based Electrophysiology

1.7.2 Neuromuscular system with Spindles and Prestretch

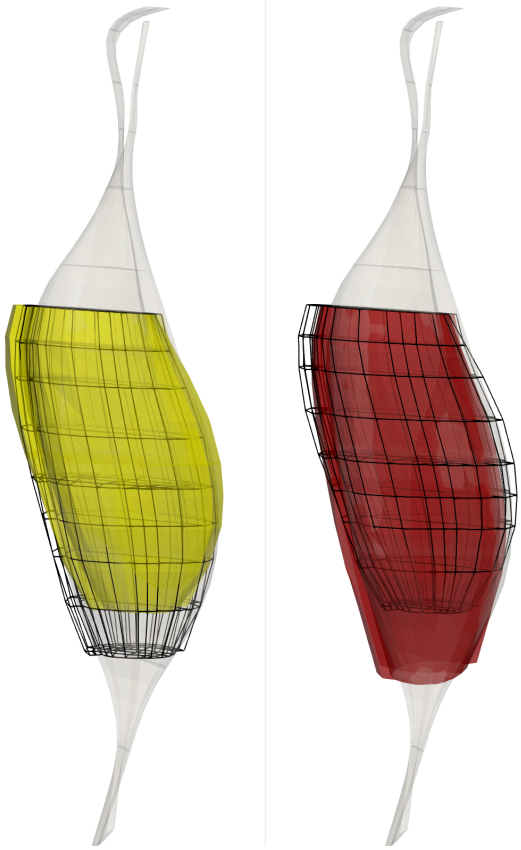


Figure 1.29: prestretch

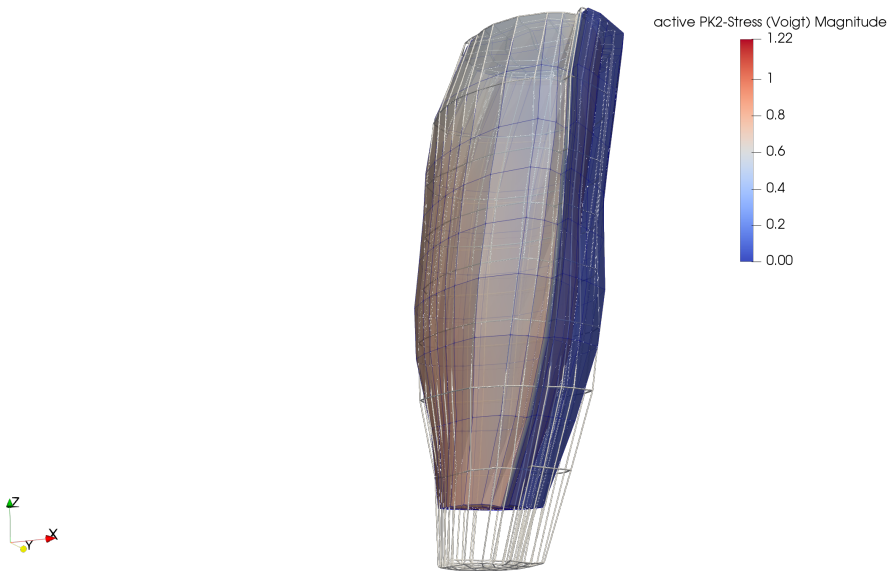


Figure 1.30: Reference and current configuration.

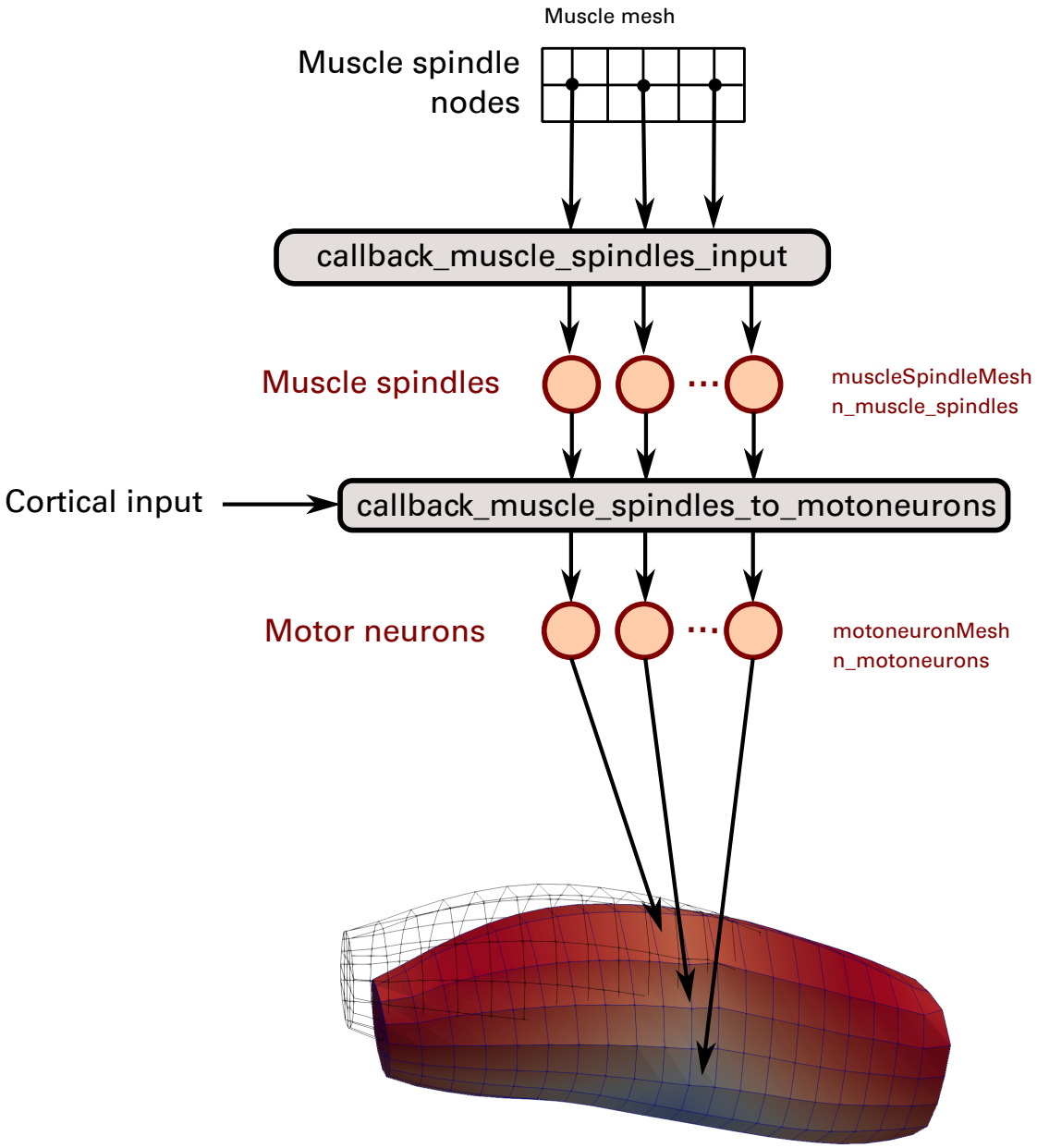


Figure 1.31: Data flow of simulation with spindles and motor neurons

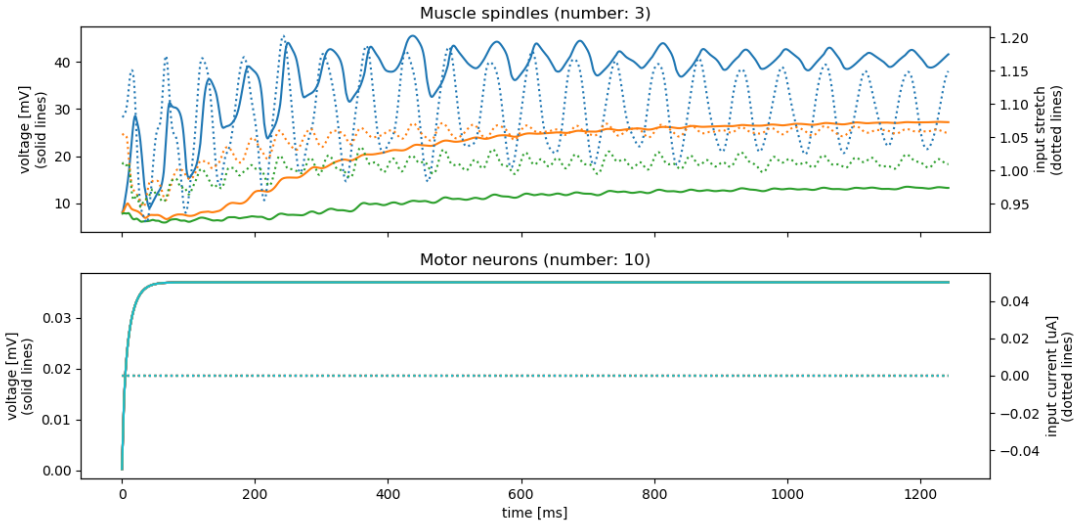


Figure 1.32: Activation data of sensors and neurons

1.7.3 Neuromuscular system with More Sensor Organs

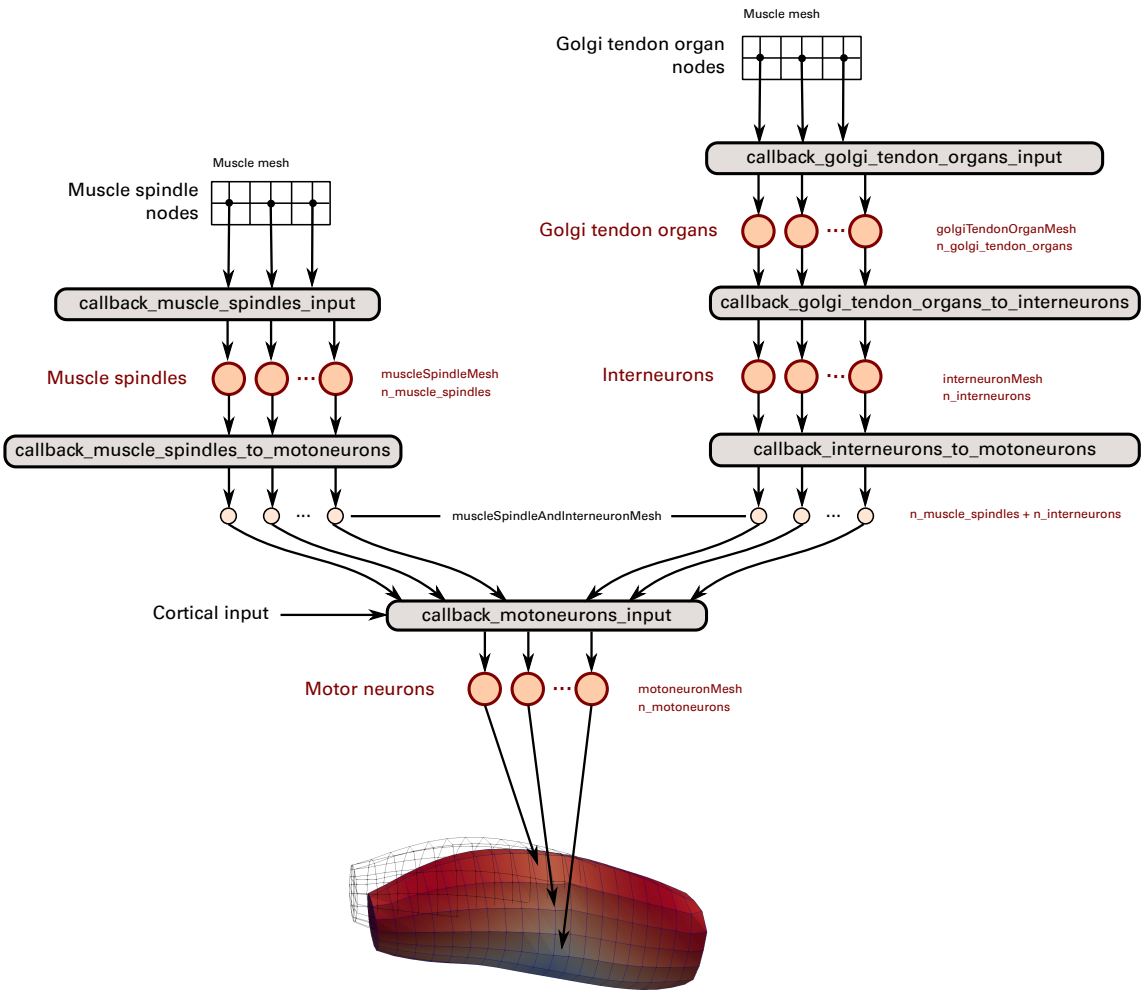


Figure 1.33: Data flow of simulation with spindles, Golgi tendon organs and motor neurons

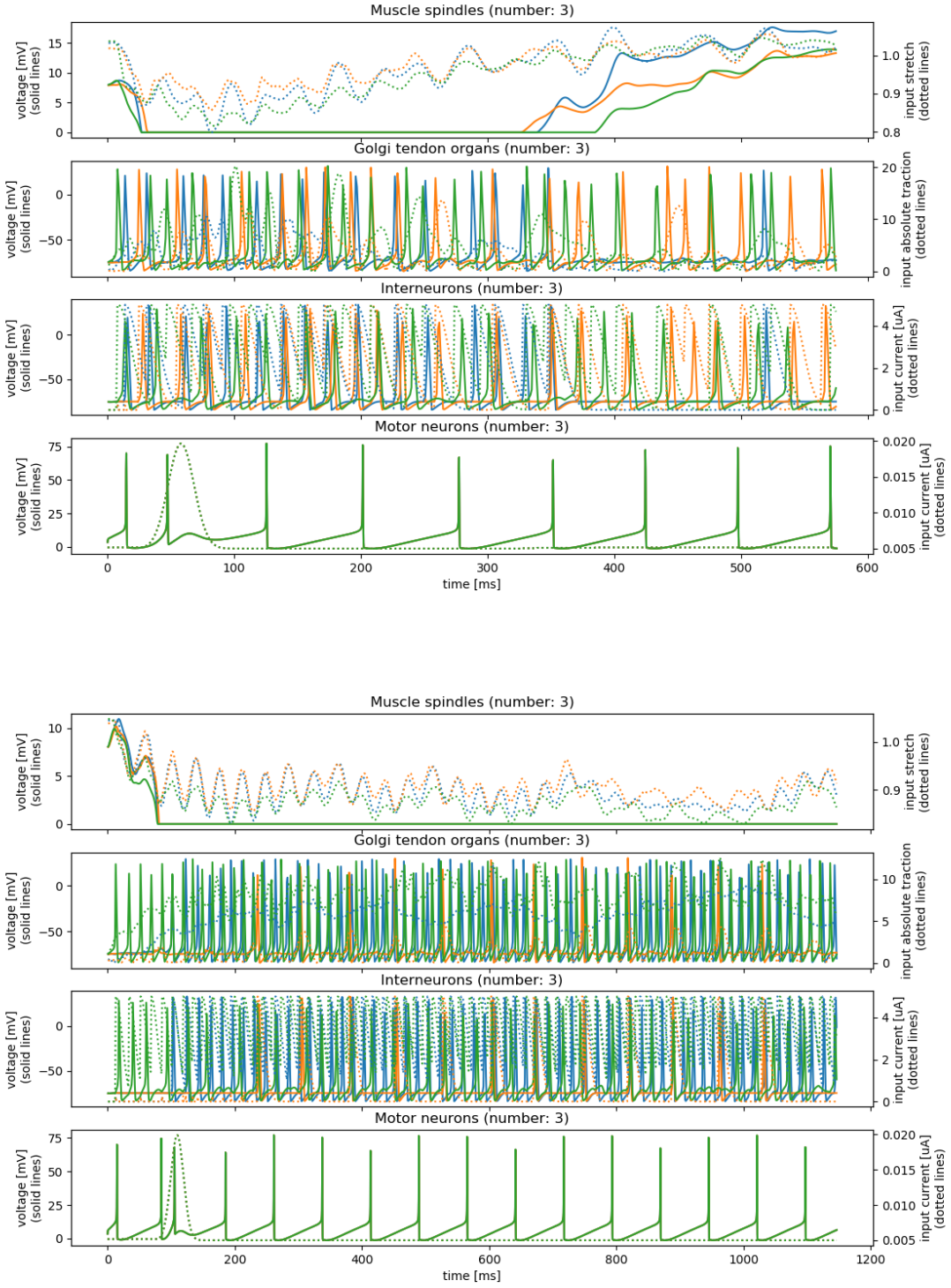


Figure 1.34: Activation data of sensors and neurons

end unfinished text

1.8 Performance Studies with OpenCMISS Iron

Next, we study the performance of the software by evaluating runtimes and parallel scalability for different solvers. We begin with OpenCMISS Iron as the baseline solver, which also implements parts of the multi-scale model considered in this work. The work of [Hei13] describes the implementation of the fiber based electrophysiology model coupled to a quasi-static hyperelastic material model with OpenCMISS. The implementation is parallelized for a hardcoded number of four processes and serves as the baseline code for the following studies.

We improved the performance of this solver for the multi-scale model by two actions: First, we evaluated and optimized the employed numeric schemes. Second, we implemented parallel partitioning for an arbitrary number of processes and evaluated different parallelization strategies. These changes were directly implemented in the OpenCMISS code. The improvements were also presented in a publication [Bra18]. In the following sections Sections 1.8.1 and 1.8.2, we describe the numeric improvements and the parallel partitioning strategies. In Sec. 1.8.3, we discuss the parallel weak scaling and memory consumption properties.

1.8.1 Numeric Improvements

The first numeric improvement is to replace the GMRES solver that is used to solve the 1D electric conduction problem on the muscle fibers by a faster direct solver.

As noted in **??**, the 1D electric conduction problem of the monodomain equation yields a tridiagonal system that can be solved with linear time complexity. The baseline solver code employs the restarted GMRES solver of PETSc, which is the default linear system solver in OpenCMISS Iron, as it is a robust choice for abitrary system matrices. More efficient solvers exist for symmetric positive definite systems such as the conjugate gradient scheme. Furthermore, the MUMPS package [Ame01] that can be interfaced in PETSc provides a parallel implementation of a direct, multi-frontal linear solver, which is able to exploit banded structures of the system matrix.

We study the runtime of these three solvers for different problem sizes of the 1D problem. The monodomain equation is solved on a single muscle fiber and the number of 1D elements is varied from 15 to 2807. The used timestep widths are $dt_{0D} = 10^{-4}$ ms and $dt_{1D} = 5 \cdot 10^{-3}$ ms. The end time of the simulation is 3 ms, yielding a total number of 600

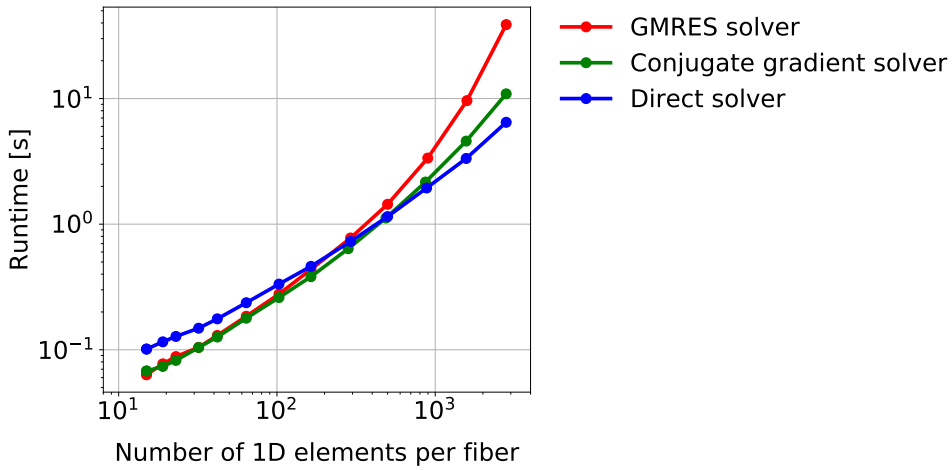


Figure 1.35: Numeric improvements in OpenCMISS: Runtime evaluation of different linear system solvers for a single muscle fiber with varying spatial resolution.

solutions of the linear system. The study is executed on an Intel Xeon E7540 processor with 24 cores, clock frequency of 1064 MHz and 506 GiB RAM.

Figure 1.35 shows the runtime of the GMRES, conjugate gradient and direct solvers for this problem in a double-logarithmic plot. It can be seen, that, for coarse discretizations with a low number of 1D elements per fiber, the GMRES and conjugate gradient solvers are faster than the direct solver. For finer discretizations, the conjugate gradient solver and the direct solver outperform the GMRES solver. For fibers with more than approximately 500 elements, the direct solver has the lowest runtime. Moreover, the direct solver exhibits an almost linear runtime complexity in terms of the problem size. This indicates that the solver is able to exploit the tridiagonal structure of the system matrix.

The second numeric improvement is the exchange of first-order accurate timestepping schemes by second-order schemes. For this exchange, we implemented the Strang operator splitting scheme and use it with the existing Crank-Nicolson implementation in OpenCMISS Iron and the Heun method, which was implemented by Aaron Krämer.

Numerical studies by Aaron Krämer presented in [Bra18] show that the relation $K = dt_{1D}/dt_{0D}$ between the timestep width dt_{1D} of the 1D electric conduction problem and the timestep width dt_{0D} of the 0D subcellular model has to be set to $K = 2$ and $K = 5$ for the Godunov and Strang splitting schemes, respectively, such that the errors of the 0D and 1D subproblems are balanced. To achieve a total error for the membrane potential V_m of approximately $8 \cdot 10^{-2}$, we can increase the required splitting timestep width $dt_{\text{splitting}}$

from $5 \cdot 10^{-4}$ ms for the Godunov splitting to $4 \cdot 10^{-3}$ ms for the Strang splitting scheme. This results in a runtime speedup of approximately 7.5.

To evaluate the total speedup of the described numeric improvements, we compare the runtimes without and with the improvements for a complete simulation of the fiber based electrophysiology model coupled with the elasticity model. A cuboid 3D domain is discretized by $2 \times 2 \times 2 = 8$ finite elements for the elasticity model and embeds $6 \times 6 = 36$ 1D fiber meshes. The number of 1D elements per fiber is varied between 576 and 239 400 to study the scaling behavior of the solvers depending on the problem size. The problem is solved in serial to avoid effects introduced by the parallelization.

The baseline implementation uses the Godunov splitting with forward and implicit Euler schemes for the 0D subcellular model and the electric conduction model, respectively. The linear system in the 1D problem is solved by a GMRES solver with relative residuum tolerance of 10^{-5} and restart after 30 iterations. Timestep widths of $dt_{0D} = 10^{-4}$ ms and $dt_{\text{splitting}} = dt_{1D} = 5 \cdot 10^{-4}$ ms are used. The improved scheme uses the Strang operator splitting with Heun and Crank-Nicolson schemes and timestep widths of $dt_{0D} = 2 \cdot 10^{-3}$ ms and $dt_{\text{splitting}} = dt_{1D} = 4 \cdot 10^{-3}$ ms. The direct solver is used for the linear system in the 1D problem. The solver for the 3D elasticity problem is the same for both implementations. A Newton scheme with residual tolerance of 10^{-8} is used and coupled to the 0D and 1D solvers with a coupling timestep width of $dt_{3D} = 1$ ms.

The present study and the studies in the next section are executed on the supercomputer *Hazel Hen* at the High Performance Computing Center Stuttgart. This Cray XC40 system contains compute nodes with two Intel Haswell E5-2680v3 processors with a base frequency of 2.5 GHz, 12 cores per CPU, 24 cores per compute node and 128 GB RAM per node.

Figure 1.36 shows the results of this study. In the upper part, the runtimes for different components of the simulation are indicated by different colors in a plot with double logarithmic scale. The runtimes for the baseline implementation are shown by solid lines and the runtimes including the improvements are shown by dashed lines. In the lower plot, the speedups from the baseline to the improved implementation are given.

The total runtime of the simulation is given by the black lines in the upper plot. It can be seen that the total runtime results almost completely from the 0D model solver, which is shown by the yellow lines. The 1D solver, given by the red lines, has the second most influence. The effects of the data mapping operations between the 3D mesh and the 1D fibers on the runtime are negligible. These data mapping operations consists of

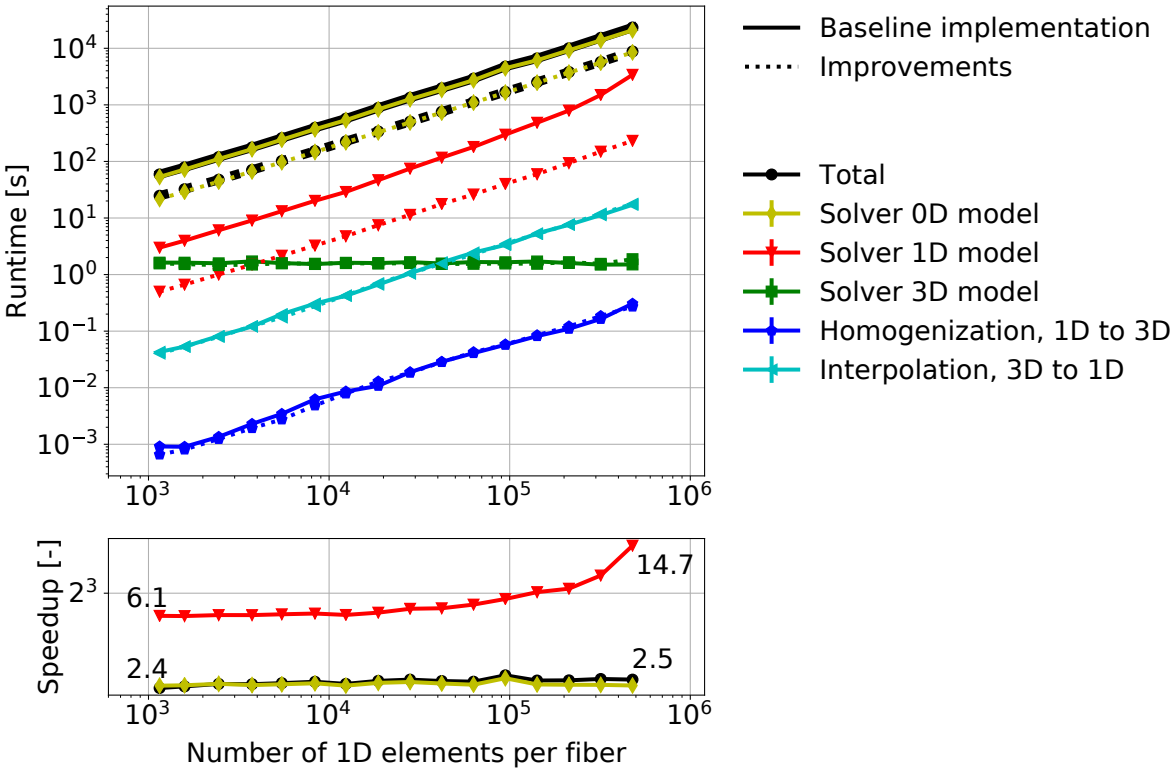


Figure 1.36: Numeric improvements in OpenCMISS: Study to evaluate the speedup of the improved implementation of the fiber-based electrophysiology and mechanics model in OpenCMISS.

the homogenization step from the 1D fibers to the 3D mesh and the interpolation step from the 3D mesh to the 1D fibers.

The runtimes for almost all problem parts increase linearly for increasing mesh resolution of the 1D fibers. Only the runtime of the 3D problem stays constant, as the 3D mesh is unchanged for the different runs.

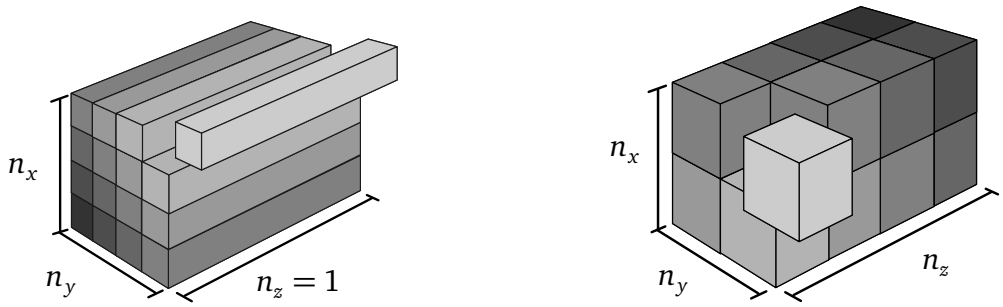
Significant runtime improvements of the new implementation compared to the baseline implementation can be seen in the lower plot of Fig. 1.36 for the 0D solver and the 1D solver. The speedup for the 0D solver is constant at approximately 2.5. The speedup resulting from the improved linear system solver in the 1D problem is approximately 6.1 for coarse meshes and increases to 14.7 for the finest mesh. This increase for high mesh resolutions results from the higher runtime of the GMRES solver for large problem sizes in the baseline implementation. The overall speedup is similar to the speedup of the 0D problem, as the 0D solver is responsible for the most runtime of the computation.

This study shows how numeric investigations can help to reduce the total runtime, in this case by a factor of 2.5. Moreover, the solver of the 0D model has the most potential

to further speed up computation times.

1.8.2 Parallel Partitioning Strategies

To exploit parallelism and, thus, further reduce the computation times, we implemented a generic domain decomposition for the studied problem in OpenCMISS Iron. Like in OpenDiHu, the 3D mesh can be partitioned to an arbitrary number of $n_x \times n_y \times n_z$ subdomains. The embedded 1D fibers are aligned with the z axis and are partitioned by the same cut planes as the 3D mesh.



(a) "Pillar-like" domain decomposition with $n_z = 1$.

(b) "Cube-like" domain decomposition

Figure 1.37: Different parallelization strategies that are implemented in the OpenCMISS model. This figure shows two approaches how the domain can be partitioned to 16 subdomains.

Figure 1.37 shows two exemplary partitionings. If the domain is only partitioned in x and y directions, the fibers are not split into multiple subdomains. As a result, we get "pillar" subdomains as shown in Fig. 1.37a. An alternative approach is to subdivide the domain in all three coordinate directions such that the subdomains are approximately cube shaped, as shown in Fig. 1.37b.

OpenCMISS Iron already provides the functionality to create partitioned unstructured meshes. However, every mesh has to be partitioned into non-empty subdomains for all processes. Thus, it is not possible to use individual meshes for the 1D fibers. In the baseline implementation of the model by [Hei13], all 1D fiber meshes are instead realized as a single mesh, whose node positions are set according to the positions of the individual fibers. This facilitates the implementation of the 0D subcellular model solvers and 1D model solvers, as the implementation has to deal with only a single mesh.

To allow for an arbitrary partitioning as in Fig. 1.37, we assigned the 1D elements of the single fiber mesh to the same processes as the subdomains of the 3D mesh. Furthermore, we reimplemented the data mapping between the 1D mesh and the 3D mesh, which was hardcoded for four processes.

In the following, we investigate the effect of different partitioning strategies on the overall runtime of the solver. The idea is that, for pillar-like partitionings as in Fig. 1.37a, the 1D problems could potentially be solved faster, as the fibers, which are aligned in z-direction, are not subdivided to multiple processes. On the other hand, the partitioning to cubes in Fig. 1.37b requires less communication in the solution of the 3D problem as the cubes minimize the surface of each subdomain and, in consequence, the amount of data to be exchanged. We evaluate how these effects influence the runtime for the pillar-like partitioning, the cube partitioning and all other possible partitionings specified by numbers of subdomains $n_x \times n_y \times n_z$.

Our test case uses a 3D mesh with $12 \times 12 \times 144$ elements. To reduce the runtime contribution of the 0D/1D electrophysiology problem and the memory consumption of the solver, only two 1D elements per 3D element are included. The numeric parameters are the same as for the improved scenario presented in Fig. 1.36. The simulations are executed on 12 compute nodes of the supercomputer Hazel Hen with 12 processes per node.

We partition the 3D domain to 144 processes using different combinations of n_x, n_y and n_z such that $n_x\, n_y\, n_z = 144$. For every partitioning, we compute the average surface area of the boundary of every subdomain. Figure 1.38 shows the resulting runtime in relation to this average boundary area. The pillar-like partitioning uses $12 \times 12 \times 1$ subdomains and exhibits the largest boundary surface area, corresponding to the last point in Fig. 1.38. The cube partitioning consists of $6 \times 6 \times 4$ subdomains and corresponds to the first data point with the smallest boundary area.

The plot shows that the runtime of the 3D solver increases approximately linearly with the amount of communication, which is expected. The partitioning with the largest average surface area has a runtime that is approximately four times as large as the runtime for the smallest surface area.

Moreover, the plot shows that the partitioning scheme has no significant influence on the runtime of the 1D solver. The reason is that the implementation does not fully reflect the decoupled nature of the individual problems of the fibers. As noted before, one big linear system has to be solved that contains the degrees of freedom of all fibers. The degrees of freedom are ordered by PETSc such that the nodes within every subdomain are

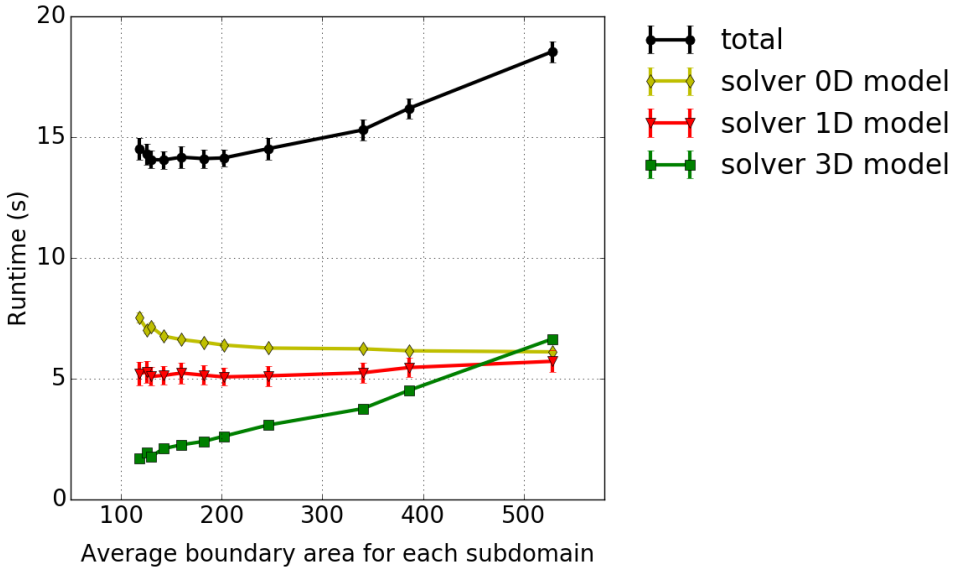


Figure 1.38: Runtime of the solvers for different partition shapes, from cube partitions on the left to pillar partitions on the right. (This figure has also been published in [Bra18] under a creative commons license.)

consecutive. If a subdomain contains (parts of) multiple fibers, their degrees of freedom are not necessarily consecutive in the solution vector and communication is required.

1.8.3 Weak Scaling Study and Memory Consumption

Next, we evaluate the parallel weak scaling properties of the overall solver. We increase the number of elements in the 3D mesh from 1232 to 8640 and the total number of 1D elements in all fibers from 14 784 to 103 680. Correspondingly, the number of processes increases from 24 to 192, such that the amount of work per process stays approximately constant. Each scenario is computed with two different partitioning schemes, once with a pillar-like partitioning and once with a cube partitioning. For the exact problem sizes, numbers of cores and numbers of elements in the partitions, we refer to the paper [Bra18].

Figure 1.39 shows the resulting runtimes of the different components of the simulation. It can be seen that the runtime stays approximately the same for all problem sizes. The observable differences in runtime within the same solver, especially for the last two data points, can be explained by a slighly different ratio of element count to process count,

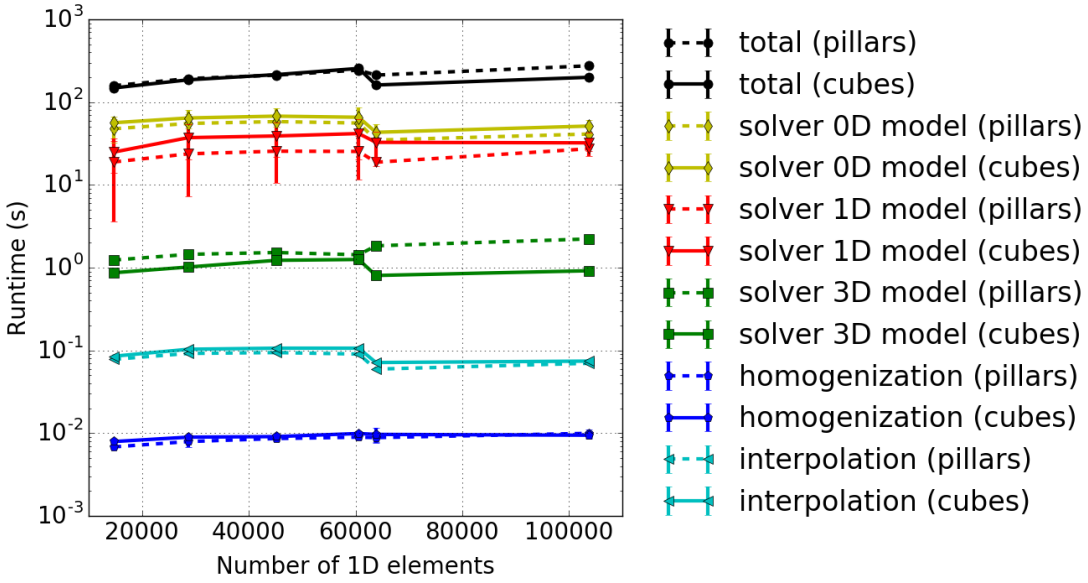


Figure 1.39: Parallel weak scaling study of a scenario with the pillars and cubes partitionings. (This figure has also been published in [Bra18] under a creative commons license.)

which results from the goal to use the pillar and cubes partitioning schemes while not exceeding the available main memory.

The runtimes of the pillars and cubes partitioning schemes are depicted by dashed and solid lines, respectively. The pillars partitioning exhibits shorter runtimes for the 1D solver and longer runtimes for the 3D solver compared to the cubes partitioning. In total, the runtime is not significantly different for the different partitioning strategies.

A limiting factor for the construction of weak scaling studies with this implementation is the high memory consumption. Figure 1.40 shows the total memory consumption per process at the end of the runtime of the simulations in Fig. 1.39. The used memory is visualized by purple lines. The dashed line again corresponds to the pillars partitioning and the solid line corresponds to the cubes partitioning.

As can be seen, the memory consumption per process monotonically increases with the total number of 1D elements. At the same time, however, the number of elements per process stays approximately constant in this weak scaling setting. The last data point is close to the memory limit of 128 GB/24 $\approx$ 4.967 GiB, which is reached when 24 processes are executed on a compute node of the supercomputer Hazel Hen.

A difference between the pillar partitions and the cube partitions is the size of the

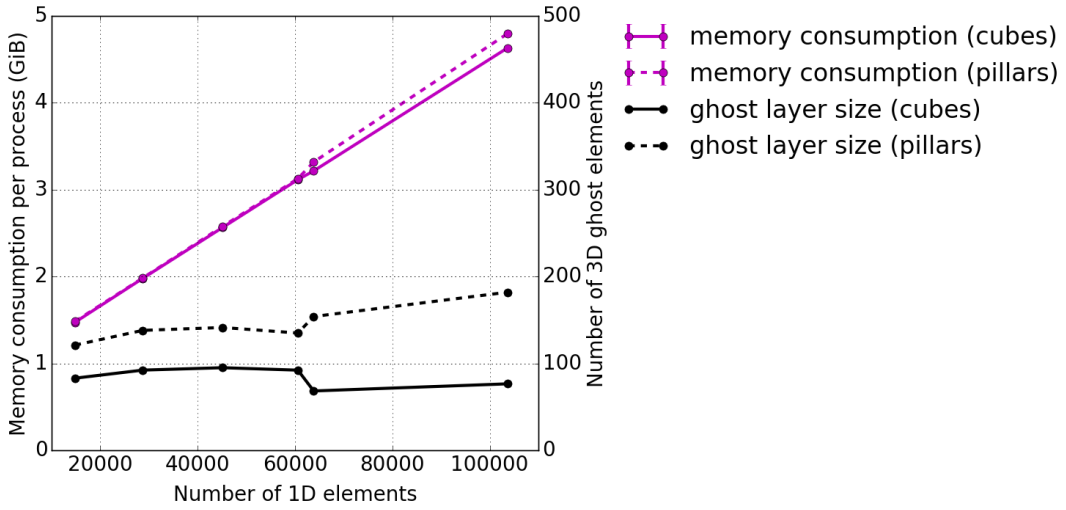


Figure 1.40: Memory consumption per process at the end of the simulation corresponding to the weak scaling study of Fig. 1.39. (This figure has also been published in [Bra18] under a creative commons license.)

subdomain surfaces and the corresponding size of the ghost layer. Fig. 1.40 shows the number of 3D ghost elements for the scenarios with cubes and pillars by the black lines. In OpenCMISS, a ghost element on a process is an element that contains ghost nodes, which are owned by a different process. The ghost elements serve as data buffers for communication during the assembly of the finite element matrices, similar to OpenDiHu.

The plot in Fig. 1.40 shows that the number of ghost elements is higher for the pillar partitioning scheme than for the cubes scheme, as expected. In consequence, the memory consumption per process is also slightly higher for the pillar partitioning. However, this effect is negligible compared to the high absolute value of the required memory and does not explain this effect.

The increase in memory results from the organization of parallel partitioned data in OpenCMISS Iron. On every process, global mesh topology information such as mappings between global indexing and local indexing is stored for the element numbers, node numbers and degree of freedom numbers. While this overhead in storage is negligible for moderately parallel scenarios, it counteracts the domain decomposition approach for higher degrees of parallelism.

Numerous functions and algorithms in the OpenCMISS Iron code rely on this type of global information. Thus, eliminating the parallelism constraint by reorganizing the data structures is a highly involved task. Especially the initialization of the parallel partitioning

heavily uses this global information. This initialization includes, e.g., the distribution of elements and nodes to the subdomains on the processes, the determination of the ghost layers and dofs to send to and receive from neighbor processes, and the setup of local numbers for elements, nodes and degrees of freedom.

We addressed the elimination of this use of global topology information in the initialization steps and developed and implemented appropriate local algorithms in OpenCMISS Iron. This resulted in major code changes that are difficult to oversee, also because of the lacking object orientation in the code base and the difficulty to test the functionality. Creating the required comprehensive set of unit tests for nearly all functionality of OpenCMISS would be a large task that remains to be done. Thus, these code changes could not be merged into the main trunk of OpenCMISS.

Even with these code changes, the memory problem is not yet solved. Another problem prior to the initialization step is that the mesh has to be specified from the user code in a global data structure. It is currently not possible to specify a mesh in a distributed way. Thus, OpenCMISS Iron can only use meshes that initially fit into the main memory on a single core.

Moreover, another issue is concerned with the data structures for matrices. Each process stores its local row indices and additionally a map from global to local row indices for all dofs of the global problem. This global-to-local map also contributes to the bad weak memory scaling and has to be eliminated as well. One possible approach is to use hash maps and only store the relevant portion of the mapping on every process. Work towards resolving this issue has been started by Lorenzo Zanon at the former SimTech Research Group on Continuum Biomechanics and Mechanobiology at the University of Stuttgart.

One reason for the generic mapping of matrix rows, which uses global information, is that OpenCMISS Iron does not restrict discretization schemes to the finite element method, where the system matrix can be assembled from local element matrices within the subdomains. An example for a different scheme is the boundary element method.

In addition, there exist more parts in the code that use a similar global-to-local mapping and would also have to be changed to allow for a constant memory consumption per process, e.g, the boundary condition handling and the data mapping between the 3D mesh and the fibers.

In summary, fixing the issue of non-scaling memory consumption in OpenCMISS Iron corresponds to redeveloping a significant portion of the code. To preserve the generic functionality of OpenCMISS, some changes would require new algorithmic considerations

and complex workarounds. This development effort would have to be quick enough to keep up with the independent development of the normal OpenCMISS branch. After completion, the merge back into the main software trunk would only be possible if the branches had not diverged too far and after significant efforts have been put into testing and preserving the feature set of OpenCMISS.

On the other hand, developing the missing functionality from scratch and making sensible restrictions on the generality of the solved problems and used methods requires possibly less effort and allows to consider design goals such as performance, usability and extensibility from the beginning. In this sense, the OpenDiHu software project can be seen as a complement to OpenCMISS Iron. The mentioned restrictions are, e.g., the exclusive use of the finite element method and cartesian coordinates and the use of parallel partitioned structured meshes instead of the more complex parallelization of unstructured meshes.

1.9 Performance Studies of the Electrophysiology Solver in OpenDiHu

Next, we investigate the runtime performance of the solvers for the electrophysiology part of the multi-scale model in OpenDiHu.

1.9.1 Evaluation of Compiler Optimizations

One difference of the data organization in OpenDiHu compared to OpenCMISS Iron is the transposed memory layout for the storage of multiple instances of the 0D subcellular model. If the `simd` optimization type in the `CellmlAdapter` class is used, the components of the state vector $\mathbf{y}$ of all 0D model instances are consecutively stored. This storage order is the SoA memory layout that was described in **??**. It allows the compiler to automatically employ SIMD instructions and, thus, exploiting instruction-level parallelism.

We study the auto-vectorization performance of the GNU, Intel and Cray compilers to determine the effect of these SIMD instructions on the total runtime of the solver. The simulated scenario consists of one muscle fiber mesh with 2400 nodes, where the monodomain equation **??** gets solved. The subcellular model of Shorten is used. The time

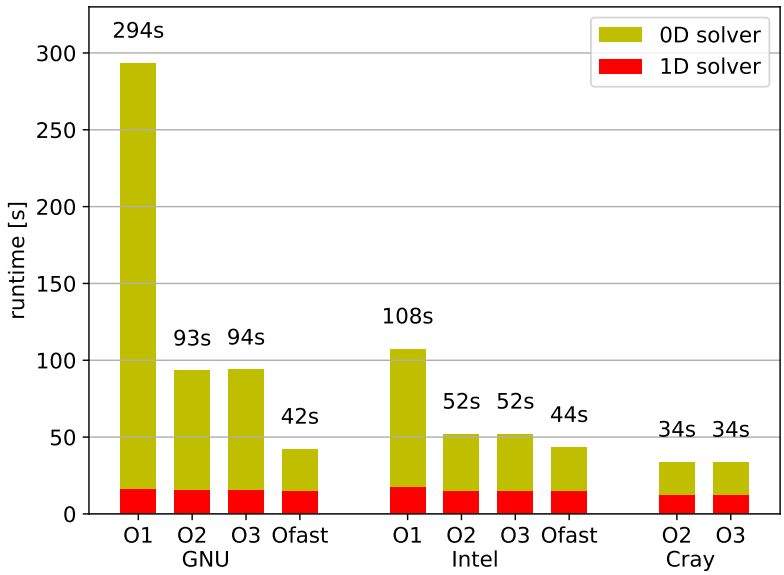


Figure 1.41: Comparison of auto-vectorization in different compilers. Runtime of the 0D and 1D solvers in the fiber based electrophysiology model with `simd` optimization type for different compilers and optimization flags.

steps widths are $dt_{0D} = 10^{-3}$ ms, $dt_{1D} = dt_{\text{splitting}} = 3 \cdot 10^{-3}$ ms and the model is computed up to a simulation end time of $t_{\text{end}} = 20$ ms.

We run the study on one compute node of the supercomputer Hazel Hen at the High Performance Computing Center in Stuttgart. This Cray XC40 system contains two 12-core Intel Haswell E-2680v3 CPUs with clock frequency of 2.5 GHz per dual-socket node, yielding 24 processors per compute node and contains 128 GB memory per compute node.

Figure 1.41 shows the runtime of the 0D and 1D part solvers for the three different compilers with varying optimization flags. As expected, the runtime of the 1D solver is not affected by the choice of compiler. The runtime of the 0D solver, however, varies greatly, as the compilers with different optimization flags are able to vectorize the code to a different extent.

For all compilers, the runtime stays approximately constant or decreases when a higher optimization level is chosen. A significant drop to less than half of the runtime is observed when switching from the O1 to the O2 optimization levels in the GNU and Intel compilers. This is mainly the result of the SIMD instructions, which are enabled starting from the O2 levels. The change to the aggressive optimization levels O3, which enables all available optimizations such as inlining and code transformations does not improve the runtime

further for all three evaluated compilers. Thus, vectorization is the main driver for good subcellular solver performance.

Another significant decrease in runtime can be observed for the `Ofast` optimization flags. For the GNU compiler, the runtime decreases again to less than half of the previous value, for the Intel compiler, the decrease is less prominent with approximately 15 %. The `Ofast` level performs optimizations that potentially change the behavior of the code. Especially floating-point arithmetic does no longer comply to the standardization rules of IEEE and ISO. Only finite numbers can be represented and the compiler is allowed to perform transformations in formulas that are mathematically correct, but not in terms of propagating rounding errors. The calculated values are correct as long as no invalid operations such as divisions by zero occur. The precision may decrease or even increase compared to `O3`. This is usually not an issue for the given simulations, however, divergence of the numeric solvers is not automatically detected in our code as no infinity values are available.

The comparison between the compilers shows that the Intel compiler create faster code than the GNU compiler and the Cray compiler creates faster code than the Intel compiler for the same optimzation levels. The performance of the `Ofast` flag is comparable between the GNU and Intel compilers. In total, the Cray compiler yields the best performance on this Cray hardware. The Cray compiler has a “whole-program mode”, which collects static information about all compilation units and allows, e.g., application-wide inlining during the linking step. The faster runtime is traded for longer compilation times. In our example, the compilation duration increases from approximately 10 min for the GNU and Intel compilers to over 2 h for the Cray compiler.

For all further simulations, we use the GNU compiler with the `Ofast` optimization flag, as it is freely available on all systems, has fast compilation times and shows good performance.

1.9.2 Evaluation of Code Generator Optimizations

Apart from the automatic compiler optimizations, also the code itself can be optimized by using efficient data structures and algorithms. **??** presented various options of our code generator, which potentially have an influence on the runtime of the subcellular solver. We compare all optimization options for a comprehensive scenario of a surface EMG simulation.

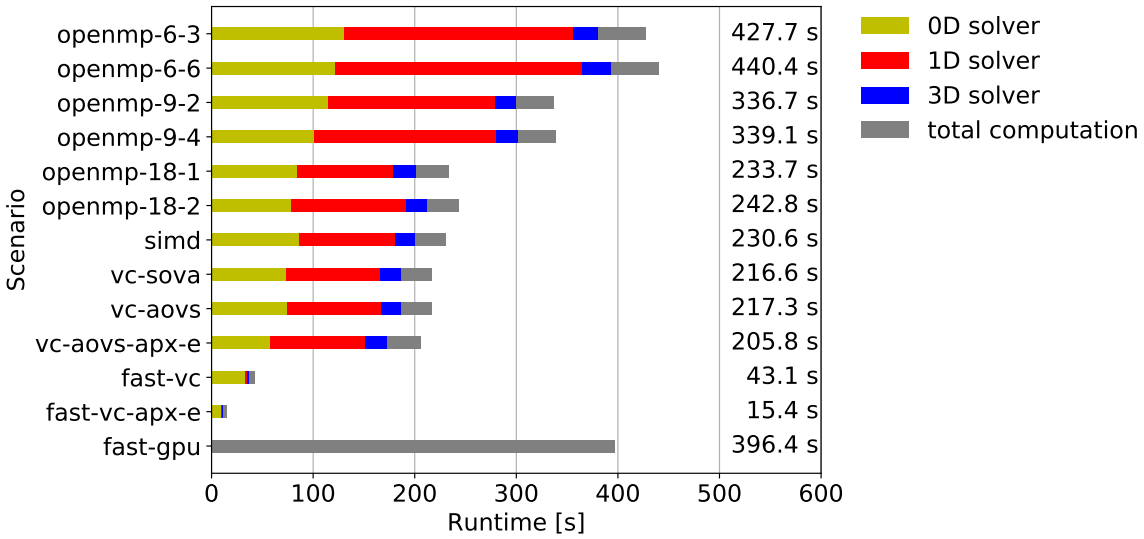


Figure 1.42: Efficiency model

The scenario solves the monodomain equation **??** on every 1D muscle fiber domain and is coupled to a 3D mesh where the bidomain equation **??** is solved. No body fat domain is considered in this scenario. We simulate 625 muscle fibers with 1481 nodes per fiber mesh and the subcellular model of Hodgkin and Huxley [Hod52]. This leads to a total number of 3 702 500 degrees of freedom to be solved for the 0D and 1D models. We run the code in parallel with 18 processes and a parallel partitioning of the 3D domain of $3 \times 2 \times 3$. Thus, every muscle fiber domain is distributed to three different processes. The 3D mesh contains 5239 nodes. Timestep widths of $dt_{1D} = 10^{-3}$ ms, $dt_{3D} = dt_{\text{splitting}} = 3 \cdot 10^{-3}$ ms and an end time of $t_{\text{end}} = 10$ ms are used and file output is disabled for this study.

We use a Intel Core i9-10980XE processor with 18 cores, base frequency of 3 GHz, maximum boost frequency of 4.8 GHz, cache sizes of 24.8 MiB, 18 MiB and 576 KiB and 31 GiB main memory. This processor is listed in the upper price segment of consumer hardware and can be considered a typical hardware for individual workstations in scientific research.

Figure 1.42 presents the results of the study for all available optimization types in our code generator. For every scenario, the bar chart shows the runtimes of the 0D subcellular solver in yellow color, the runtime of the 1D electric conduction solver in red color, the runtime for the 3D bidomain solver in blue color and the remaining runtime of the coupled solver scheme, which involves, e.g., data transfer between data structures and inter-process communication, in gray color. The presented runtimes are averaged

over several runs and over all processes per run.

The first six bars correspond to the `openmp` optimization type, which places OpenMP pragmas in the code and employs thread-based, shared memory parallelism. The scenario `openmp-`i`-`j refers to i MPI processes in total with j threads on every process. The problem is partitioned to i subdomains and the j OpenMP threads per subdomain simultaneously operate on the shared data structures of the subdomain. As a result, in the scenarios `openmp-6-3`, `openmp-9-2` and `openmp-18-0`, 18 threads are executed in total on the processor with 18 physical cores. The other scenarios, `openmp-6-6`, `openmp-9-4` and `openmp-18-1`, employ 36 threads.

It can be seen that the scenarios with the same number i of processes and varying number j of threads have similar runtimes. This shows that runtime is reduced mainly as a result of MPI parallelisation. The distribution of the runtime to the solvers allows further insights. Between the scenarios with the same number of processes, `openmp-6-3` and `openmp-6-6`, as well as between `openmp-9-2` and `openmp-9-4`, the runtime of the 0D solver decreases. This is a result of the higher number of OpenMP threads that is used to perform the same amount of work. For the last two scenarios, `openmp-18-0` and `openmp-18-1`, the runtime of the 0D solver shows no further decrease, as the 18-core processor is fully occupied as soon as 18 threads are used.

The effect of OpenMP parallelism on the 1D solver is even higher than on the 0D solver in this example. As the code generator using OpenMP parallelism is only responsible for the 0D problem, the performance of the 1D problem depends only on the partition size and workload defined by the parallel partitioning with i MPI processes. A reduction of the MPI parallelism has more impact than the resulting increased parallelism of the 0D solver. Thus, the approach with a high degree of OpenMP parallelism, such as in scenario `openmp-6-6` shows a worse performance than the approach with a higher MPI parallelism as in scenario `openmp-18-0`.

The next bar presents the runtime of the `simd` optimization type. As in all scenarios of this study, the GNU compiler with the `Ofast` flag is used and automatically vectorizes the subcellular model equations. This scenario is very similar to the `openmp-18-1` scenario, except that the OpenMP pragmas are omitted in the generated code. As a result, the runtimes are also similar to this scenario. A slight reduction in runtime is seen that results from the missing OpenMP initialization at every loop.

While the `simd` scenario relies on the auto-vectorization capabilities of the compiler, the `vc` scenarios, which are considered next, explicitely employ vector instruction, abstracted by the *Vc* and *std-simd* libraries.

The vc-sova scenario uses the Struct-of-Vectorized-Array (SoVA) memory layout and the barchart shows a slightly lower runtime of the 0D solver compared to the Array-of-Vectorized-Struct (AoVS) memory layout in the vc-aovs scenario.

The next considered scenario is vc-aovs-apx-e. It is the same as vc-aovs except that the exponential function is approximated by $\exp^*(x) = (1 + x/n)^n$ for $n = 1024$. The results show that this reduced the runtime of the 0D solver from 74.24 s to 58.02 s, which is a reduction of approximately 22 %.

Instead of generating code only for the 0D subcellular model and solving the 1D subcellular model using a direct solver of PETSc, we can also directly generate combined solver code for the 0D and 1D models and use the Thomas algorithm for the computation of the 1D model. This is done in the fast-vc scenario and reduces the runtime by a factor of nearly 5. In this approach, the exponential function can also be exchanged by its mentioned approximation. This is tested in the fast-vc-apx-e scenario and further decreases the total runtime of now only 15.4 s.

The two fast-vc scenarios demonstrate the performance of the AVX-512 vector instruction set of the used Intel processor. Its potential is only fully exploited, if the explicit vector instructions are generated in the code, as done in the vc scenarios.

The solution times for the last two mentioned scenarios can be further reduced if only the subcellular model instances are computed that are not in equilibrium. If enabled, this reduction depends on the activation pattern of the fibers. For the sake of the present study, which aims to compare runtimes of the code generator, this option is not enabled.

The last considered optimization type in the code generator is presented in the scenario fast-gpu. In this scenario, the program is only run with one MPI process. The total computation of the 0D and 1D models is offloaded to a GPU using OpenMP 4.5 pragmas in the generated code. We use the same simulation scenario and CPU hardware for this run. The computer is equipped with a NVIDIA GeForce RTX 3080 GPU with 8704 CUDA-cores, 10 GB of memory and a Thermal Design Power (TDP) of 320 W. The processing power is 29.77 TFLOPS for single precision and 465.1 GFLOPS for double precision operations. We use only double precision operations for the computation of the models. In this scenario, only the total runtime is measured. The bar chart shows a total solver runtime of 396 s, which far slower than the CPU computations. Possible reasons are that the GPU is targeted at single precision performance and that the employed GPU code by the OpenMP functionality of the GNU compiler is not optimal.

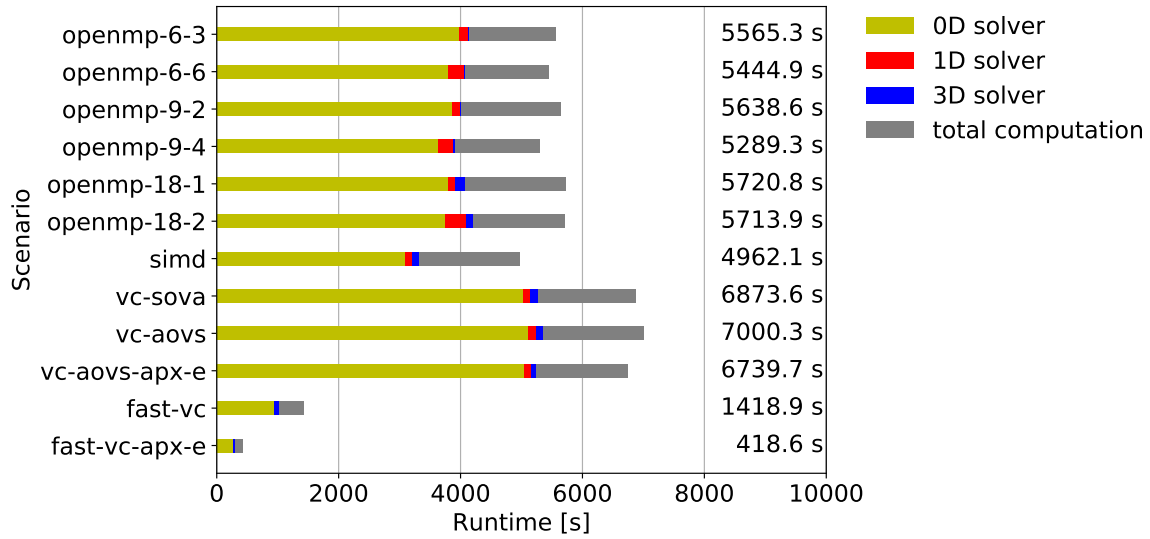


Figure 1.43: Efficiency model shorten

In the considered example, which uses the Hodgkin and Huxley subcellular model with a state vector $\mathbf{y} \in \mathbb{R}^4$, the workload of the 0D and 1D solvers was in the same range. Other subcellular models with higher workloads exist. In the next study, we repeat the same measurements with the subcellular model of Shorten et al. [Sho07b], which has a state vector $\mathbf{y} \in \mathbb{R}^5 7$. Whereas the solver for the model of Hodgkin and Huxley needs to compute 4 ODEs and 9 algebraic equations in every timestep, the solver for the Shorten model computes 57 ODEs and 71 algebraic equations in every timestep.

As the computational effort to solve one instance of the subcellular model increases, we compute a different simulation scenario for the next study. We use 49 fibers with 1481 nodes each and a 3D mesh with 23 696 degrees of freedom. The total number of degrees of freedom is 4 087 560, which is similar to the number 3 707 739 in the previous study. The simulation end times is 3 ms. For this subcellular model, smaller timestep widths of $dt_{\text{splitting}} = dt_{\text{1D}} = dt_{\text{0D}} = 2.5 \cdot 10^{-5}$ ms and $dt_{\text{3D}} = 1.0 \cdot 10^{-1}$ ms are used to ensure convergence of the solver.

Figure 1.43 shows the resulting runtimes for different scenarios in a bar chart analogous to Fig. 1.42. It can be seen that the solver time for the 0D model now dominates the total runtime. In the `openmp` scenarios, the runtime for the 0D solver decreases again, if more threads are used in total. Contrary to the previous study, the total runtime profits from this runtime reduction as the 0D part is significant enough for the total runtime. Another difference to the results in the previous study is that the durations for the 0D

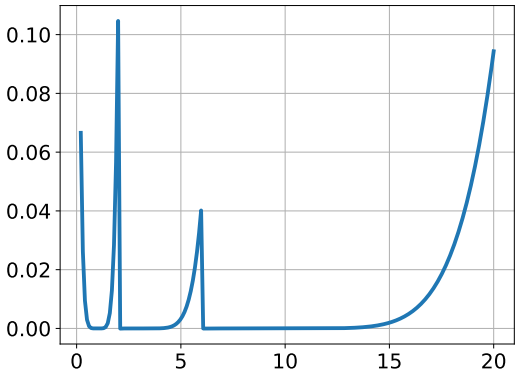


Figure 1.44: Relative error of the piecewise Taylor approximation of the log function as used in the vectorized simulation code.

model is nearly the same for every combination of number of MPI processes i and number of OpenMP threads j. This shows that the overhead of starting the OpenMP threads, which in the previous study was responsible for larger compute times of the models, is now amortized by the larger overall workload.

The performance in the `simd` scenario is, again, comparable with the `openmp-18-1` scenario and shows a slightly smaller runtime due to the missing OpenMP thread initializations.

A difference to the previous study can be seen for the `vc` scenarios. In the present study with the subcellular model of Shorten et al., the runtimes for the `vc-sova`, `vc-aovs`, and `vc-aovs-apx-e` are all higher than for the previous auto-vectorized scenarios. In the study with the Hodgkin and Huxley subcelluar model, the `vc` scenarios reduces the runtime.

This effect originates from the operations in the subcelullar equations. The Shorten model contains a large number of $\log(x)$ functions. These are especially compute intense and not supported in the abstraction layer of the AVX-512 instructions provided by the *std-simd* library. Instead, their serial counterparts are called. The auto-vectorization of the compilers, however, is able to employ the respective vectorized functions, which explains the better performance in the `openmp` and `simd` scenarios. For the future, we expect that the respective functionality becomes available in the *std-simd* library, which would automatically increase the performance for these optimization types. For processors without AVX-512 support but with the AVX2 instruction set, the library *Vc* is used, which supports the respective functions and, thus, yields the expected performance in the `vc` scenarios. Whereas AVX-512 has a SIMD lane width of eight double values, AVX2 only supports SIMD lanes with 4 double values.

To mitigate this effect, we also approximate the log function by a numeric approximation. We define the approximated logarithm function $\log^*(x)$ by its piecewise Taylor polynomials of sixth order around the points $x = 1$, 3 and 9 with discontinuities at the points $x = 2$ and $x = 6$. Figure 1.44 shows the absolute relative error for the range between 0.2 and 20 which, in this range, is bounded by 0.105. However, better convergence of the 0D-1D problem is achieved, if the approximated log function $\log^*$ is the inverse of the approximated exponential function $\exp^*$. We therefore apply one Newton iteration of the problem

$$F(y) = \exp^*(y) - x \stackrel{!}{=} 0$$

to the Taylor approximation value y, which is identical to substracting $(1 - x/\exp^*(x))$ from the computed result y. It only involves one evaluation of the approximated exponential function.

The scenario `fast-vc` in Fig. 1.43 generates unified solver code for both 0D and 1D models, but does not include this approximation. The approximated exponential and logarithm functions are included in the scenario `fast-vc-apx-e`. It can be seen that the total runtime is highly reduced compared to the auto-vectorized scenarios.

How To Reproduce

The simulations in this section use the example `examples/electrophysiology/fibers/fibers_emg` with the variables files `optimization_type_study.py` and `shorten.py`. The commands for the individual runs are executed by the following scripts:

```
cd $OPENDIHU_HOME/examples/electrophysiology/fibers/fibers_emg/
  ↪ build_release
../old_scripts/run_optimization_type_study.sh
../old_scripts/run_optimization_type_study_shorten.sh
```

The utility to create the plots from the generated `logs/log.csv` files can be found in the repository at github.com/dihu-stuttgart/performance in the directory `opendihu/18_fibers_emg`:

```
./plot_optimization_type_study_shorten.py
./plot_optimization_type_study.py
```

1.10 Parallel Strong Scaling and Comparison with OpenCMISS Iron

After the performance of different optimization types has been evaluated for a single scenario in the last section, we now conduct a strong scaling study with the found most performant optimization type and compare the runtimes to the reference software OpenCMISS Iron.

1.10.1 Evaluation of Runtimes

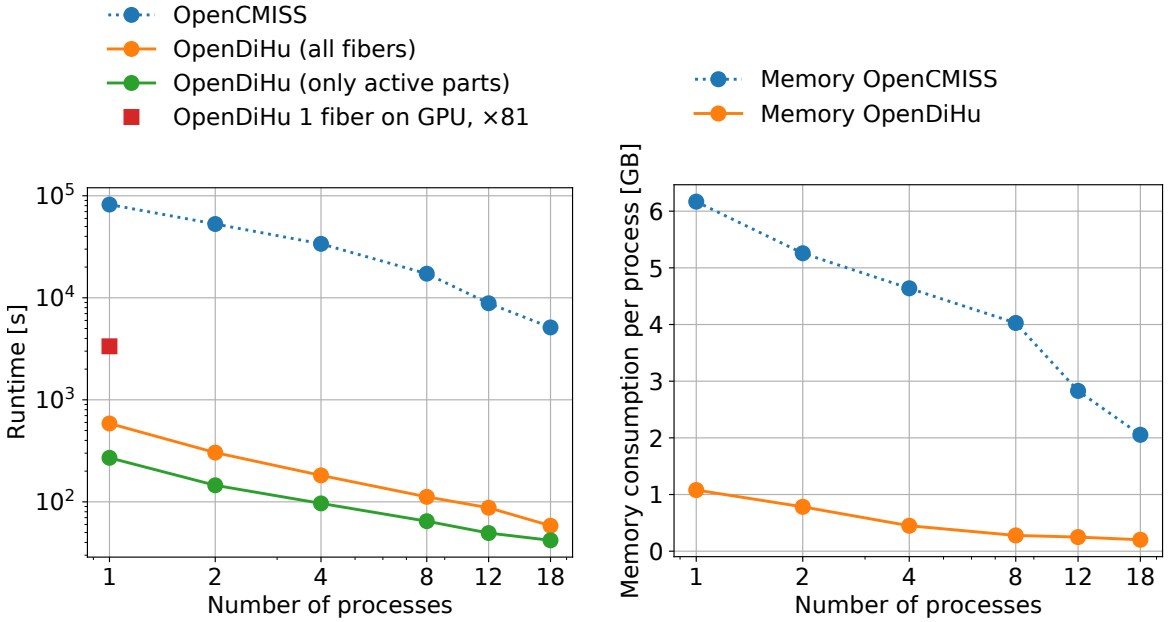
For a fair comparison, we take care to exactly compute the same scenario with both software packages. The simulated scenario uses the same model as in the previous section: fiber based electrophysiology with the monodomain model given by **??** on every muscle fiber, including the 1D electric conduction along the fibers and the 0D subcellular model of Shorten et al. [Sho07a]. The fibers are coupled with the bidomain equation in **??**, which is solved on the 3D domain to yield the EMG signals. No fat layer is considered in this study as this is not available in the OpenCMISS implementation.

The simulated scenario contains 81 fibers with 1480 elements each, a coarse 3D mesh with 775 nodes and 6 718 591 degrees of freedom in total. We use our improved OpenCMISS setup, which is discussed in Sec. 1.8.1 and employs the second order numerical time stepping schemes and the improved linear solvers: The monodomain model is solved using a Strang operator splitting with Crank-Nicolson and Heun's methods. A conjugate-gradient solver is used for the linear system of the bidomain equation.

We use timestep widths of $dt_{0D} = 10^{-4}$ ms, $dt_{1D} = dt_{\text{splitting}} = 5 \cdot 10^{-4}$ ms, $dt_{3D} = 10^{-1}$ ms and a simulation end time of $t_{\text{end}} = 2$ ms. During this time, the resulting values are written to output files 20 times after every 0.1 ms. The fibers are assigned to 10 MUs that are activated in a ramp every 0.2 ms from $t = 0$ ms to $t = 1.8$ ms.

Within the strong scaling study, the same scenario is computed with different numbers of processes, ranging from one to 18 in this case. We use the cubes partitioning strategies presented in Sec. 1.8.2 in both OpenCMISS and OpenDiHu. The study is executed on the same Intel Core i9-10980XE processor as the studies in the previous section. We measure the total user time of the simulation program, which includes the time for initialization, computation of system matrices and the duration of file output. However, the majority of the runtime is spent in the numerical solvers.

In the OpenDiHu program, we use the setup corresponding to the `fast-vc` scenario in the last section with enabled approximation of log and exp functions. We measure the runtime of two variants. The first variant computes all subcellular models and performs the same work as the OpenCMISS Iron implementation. In the second variant, the adaptive computation described in **??** is enabled, which only computes fibers that have been activated and the subcellular models that are not in equilibrium. For the chosen ramp activation pattern of the MUs, only approximately half of the subcellular model instances is computed in the second variant.



(a) Runtime of the simulation programs for OpenCMISS (blue) and two variants of OpenDiHu (orange and green), see the description in the text for details.

(b) Memory consumption per process at the end of the program.

Figure 1.45: Strong scaling study of fiber based electrophysiology and comparison between the implementations of OpenDiHu and OpenCMISS Iron. The same scenario is computed for both software packages and for increasing numbers of processes from one to 18.

Figure 1.45a shows the resulting runtimes in this study. It can be seen that the runtime decreases monotonically for higher numbers of processes for all three tested simulations. The OpenDiHu implementation exhibits lower runtimes for all numbers of processes. The reduction in runtime between OpenCMISS Iron and the first OpenDiHu variant is given by a factor of circa 100 with a maximum factor of 186 for 4 processes. In addition, the second OpenDiHu variant approximately halves the runtimes as expected, because only half of the subcellular models are computed.

In summary, the improvements to the EMG simulation software that are described in this work include the numeric improvements in Sec. 1.8.1 with a speedup of 2.5, the software improvements with a speedup of over 100 and a measured maximum of 186, and the algorithmic improvement of adaptive 0D model computations, whose speedup factor is scenario dependent. In the present study, the two latter factors, i.e., the speedup between the improved OpenCMISS Iron software and the OpenDiHu scenario with adaptive computation, give a combined maximum speedup of 363 for the measurement with two processes.

Moreover, the computation of this study was also carried out with the gpu optimization type in OpenDiHu, using the same GPU as in the last section. One process was started on the CPU, which offloaded the computational work of the 0D and 1D problems to the GPU. However, the GPU memory was not sufficient for the computation of all 81 fibers. Therefore, we only compute one fiber, but keep the rest of the simulation scenario equal to the other measurements. The req square in Fig. 1.45a shows the measured runtime multiplied by the factor 81 for compensation. As in the studies of the previous section, the computation on the GPU has higher runtimes than the computation on the CPU.

As the memory consumption was a limiting factor for parallelism in OpenCMISS Iron as shown in Sec. 1.8.3, we also measure the memory consumption per process at the end of the simulation in both software frameworks. Figure 1.45b shows the result for OpenCMISS and OpenDiHu. The two variants of OpenDiHu have the same memory consumption characteristics, as the only difference between the variant is that the computation of certain subcellular models is switched on or off.

It can be seen that the increased parallelism leads to a reduction of the used memory per process in both softwares. OpenDiHu approaches a saturation value of 200 MiB for eight and more processes. For OpenCMISS, the memory consumption is higher, but reduces more quickly also for higher numbers of processes. However, the relation between the two curves increases from a value of 6.168 GiB : 1.078 GiB$GB = 5.7$ for one process to 2.054 GiB : 0.202 GiB = 10.2 for 18 processes.

As a result, this study shows a large memory efficiency improvement in OpenDiHu compared with the OpenCMISS Iron software. For OpenCMISS, the memory scaling in this parallel strong scaling scenario is not as bad as in the parallel weak scaling considered in Fig. 1.40 in Sec. 1.8.3. However, the total memory for all processes still increases to approximately $18 \cdot 2.054\,\text{GiB} \approx 37\,\text{GiB}$, which is higher than the main memory capacity of the used processor.

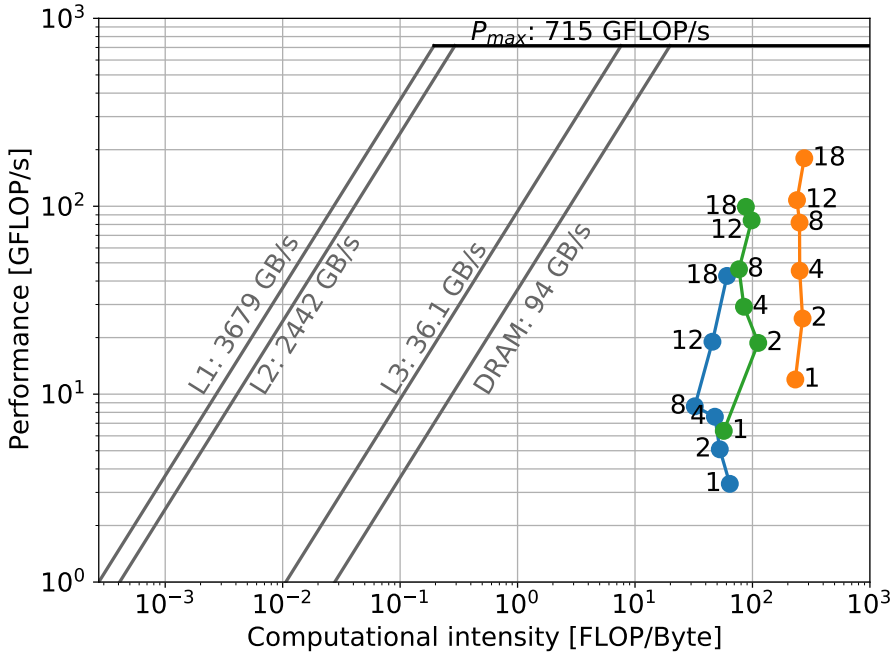


Figure 1.46: Roofline model of the strong scaling study. The blue, green and orange data points correspond to the runs of the OpenCMISS and OpenDiHu variants, as given in Fig. 1.45a.

1.10.2 Roofline Model

To further investigate the computational behavior, we also present the performance measurements of the solvers in a roofline model. Figure 1.46 shows the resulting diagram with the data points of all CPU runs in the study. The x axis shows the computational intensity of the simulation which is measured in double-precision floating-point operations (FLOPs) per byte of data that is transferred between the CPU and the main memory and caches. The y axis measures the performance in GFLOPs per second. The highest possible performance is given by the peak performance of the processor, which is $P_{\text{max}} = 715\ \frac{\text{GFLOP}}{\text{s}}$ in this case. Furthermore, the performance is limited by the amount of payload data that can be transferred to the CPU over the memory bus. The memory bandwidths of the L1, L2 and L3 caches and the main memory (DRAM) correspond to the shown diagonals in Fig. 1.46 and form the "roofline" of the model.

We measured the memory bandwidths of the Caches and the peak performance using the Empirical Roofline Tool [Yan20]. The main memory bandwidth was retrieved from the processors' documentation. To position the simulation runs of the study in the roofline

model, we used hardware counters to count floating point instructions and memory access operations. For the runs with OpenCMISS Iron and OpenDiHu, we started the hardware counters 90 s, respectively 15 s after the begin of the simulation such that the initialization phase was not included in the measured data. The counters were kept active for 15, 30 and 60 seconds, depending on the expected runtimes of the different runs. The counted numbers of events were then divided by the acquisition time to yield the required rates of memory bandwith and floating-point performance.

Figure 1.46 shows the measured points in the roofline model corresponding to the curves in Fig. 1.45a visualized by the same colors as in Fig. 1.45a. All data points are right of the memory bandwidth limits, which indicates that the simulation is compute bound. The highest computational intensity and performance are both achieved by the OpenDiHu variant given in orange color, which computes all fibers. The values for the adaptive variant given in green color are lower, as less computations are performed and a higher portion of the runtime and compute power is spent on determining which subcellular model has to be computed. The two metrics are lowest for the OpenCMISS runs given in blue color.

The roofline diagram shows the data points for all parallel runs and the number of processes is noted in the plot. The run with 18 processes is the most meaningful, as this means that the whole processor is employed. The largest performance for the OpenDiHu run in orange color has a value of 180.157 $\frac{\text{GFLOP}}{\text{s}}$ which corresponds to 25.2 % of the peak performance. Qualitatively speaking, this is a very good value. The rated 100 % of peak performance for the processor are practically unreachable. For example, the peak performance assumes only fused multiply add operations and requires a power management that maximized the employment of the boost clock frequency in the processor. These conditions are not fulfilled in our realistic computations. The performance values of the runs with 18 processes for the green and blue data points are 13.9 % and 6.0 %, respectively.

Furthermore, the measurements with lower process counts can also be assessed with a scaled down peak performance according to the fraction of used cores. However, this assessment is slightly off, as, e.g., the frequency boost can constantly activate for a processor that only does computations on a single core. The performance for the OpenDiHu run with one process given by the orange point is 11.966 $\frac{\text{GFLOP}}{\text{s}}$, which corresponds to 30.1 % of the fractional peak performance of 715 $\frac{\text{GFLOP}}{\text{s}}/18 = 39.7\ \frac{\text{GFLOP}}{\text{s}}$.

How To Reproduce

The scripts to run the studies in this scenario and to create the plots are available in the repository at github.com/dihu-stuttgart/performance in the directory `opendihu/20_fibers_emg_avx_opencmiss`:

```
./0_run.sh
```

The directory also contains a script that performs all steps to install OpenCMISS, if needed.

Note, the studies in the previous and the current sections were carried out on the computer with hostname `pcsgs05` in the institute network at the time of writing.

1.11 Performance Measurements on the GPU

In the previous two sections, measurements were made on a GPU, which produced worse results than the CPU code. The used GPU was a NVIDIA GeForce RTX 3080, a high-end consumer graphics card, which mainly targeted graphics rendering perfomance using single-precision operations. The ratio between double-precision and single-precision performance was 1:64. However, single-precision calculations were found to not yield a stable subcellular model solver, as the precision is too low.

In this section, we conduct two studies, where the first study uses the same GPU hardware as before. The second study is executed on a NVIDIA Quadro GP100 GPU, which has a double-precision to single-precision performance ratio of 1:2. The rated double-precision performance of the Quadro card is 5.168 TFLOPS, which is ten times higher than the value of 465.1 GFLOPS for the GeForce card.

The used CPU hardware contains a dual-socket CPU with two 12-core Intel Xeon Silver 4116 processors with 2.1 GHz base frequency and 3 GHz maximum turbo frequency, yielding a total core count of 24, and being equipped with main memory of 188 GiB.

Computational hardware can be compared by its average thermal design power dissipation (TDP). For the studies in the previous two sections, the TDP values for the used CPU and GPU were 165 W and 320 W. For the studies in the current section, the values for CPU and GPU are $2 \cdot 85\,\mathrm{W} = 170\,\mathrm{W}$ and 235 W. This indicates that the used hardware

is in a comparable electrical power range, however the GPU is more specialized for our double-precision needs.

1.11.1 Strong Scaling with the GPU for the Hodgkin-Huxley Model

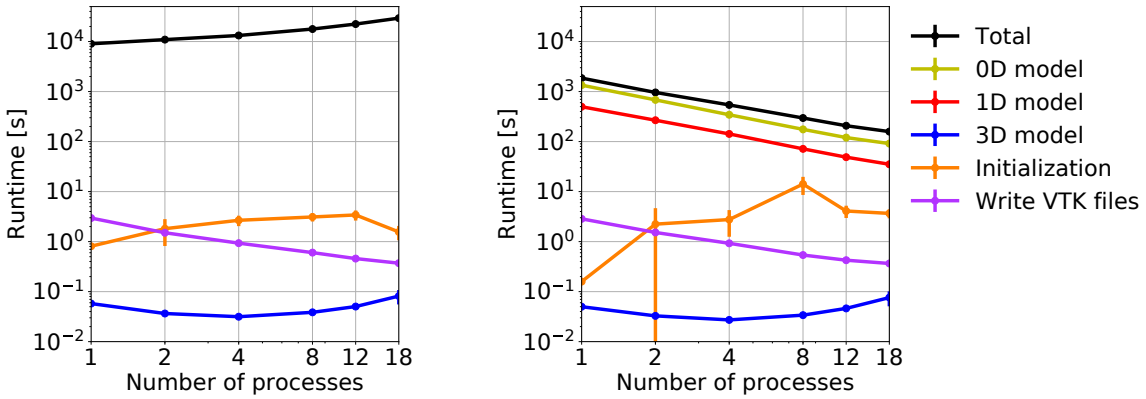
While the studies in the last two sections only used one process on the CPU, which prepared and interfaced the offloaded computations on the GPU, we now additionally consider parallelism on the CPU.

The first study compares the strong scaling with and without GPU acceleration. The runs with GPU acceleration partition the computational domain as usual to multiple subdomains, which are handled by dedicated processes on the CPU. The solution of the 0D and 1D models is performed on the GPU, and every process independently transfers its part of the computational work to the same GPU. The 3D model is solved using MPI parallelism on the CPU. We compare this setup with a pure CPU based strong scaling study.

The scenario solves the fiber-based electrophysiology model without fat layer with 169 fiber meshes of 1480 elements each and a 3D mesh with 1480 elements. The subcellular model of Hodgkin and Huxley [Hod52] is used. The computation uses the same numeric parameters as in the first study in Sec. 1.9.2, a 3D solver timestep of $dt_{3D} = 4 \cdot 10^{-1}$ ms and a simulation end time of 10 ms. Moreover, the setup equals the settings of the `fast-vc` and `fast-gpu` scenarios in Sec. 1.9.2.

Figure 1.47 presents the results for the two studies with and without GPU usage. Figure 1.47b shows the runtimes of different parts in the simulation of the CPU-only strong scaling study. It can be seen that the 0D computations account for most of the runtime, followed by the 1D computations. The 0D computations involve the solution of the Hodgkin-Huxley subcellular model. The 1D computations consists of solving 1D problems in serial using the Thomas algorithm. The 3D solver time is negligible as the 3D problem is only solved every 13 333 timesteps for the VTK file output of EMG values every 0.4 ms, which corresponds to an EMG sampling frequency of 2.5 kHz. Also the runtimes for initialization and the file output are very low compared with the runtimes of the computations.

It can be seen in Fig. 1.47b that the total runtime decreases with higher process counts in this strong scaling study. The parallel efficiency $E_p = T_1/(T_p\, p)$ for p processes reaches



(a) Study where every process on the CPU offloads the 0D and 1D model computations to the GPU.

(b) CPU-only study.

Figure 1.47: Strong scaling study with and without GPU usage. A scenario with 169 fibers and the subcellular model of Hodgkin and Huxley is simulated.

$E_p = 65.2\,\%$ for $p = 18$ processes. The plot shows that the 0D and 1D solvers and the VTK file output have good strong scaling properties, whereas the initialization and the solution of the 3D model contain serial code portions that prohibit optimal scaling.

Figure 1.47a shows the analogue study, where the 0D and 1D computations are offloaded to the GPU. The runtimes of these individual model parts are not explicitely measured, only the total runtime is known.

The plot shows in increasing total runtime for higher CPU parallelism. The runtimes of initialization, file output and the 3D solver are as in the CPU-only study. The increasing total runtime shows that the GPU is better at solving the complete 0D and 1D problems given by one CPU process than the case where the same computation is split to several parts and provided by different MPI processes. The benefit of using multiple CPU processes to interface the GPU in this study is, thus, only that the VTK output functionality gets parallelized. However, this effect is negligible.

An absolute comparison between the runtimes in Fig. 1.47a and Fig. 1.47b also reveals that the scenarios for one to 18 processes using the GPU have 4.8 to 181 times longer total runtimes. In this study, the memory transfer between the CPU and the GPU has a low influence on the total runtime, as this transfer only happens before and after the 3D model is solved. The measured runtimes therefore correspond to the computation on the GPU.

1.11.2 Evaluation of Hybrid CPU and GPU Computation for the Shorten Model

Whereas previously the subcellular model of Hodgkin and Huxley was solved on the GPU, we now switch to the more compute intense model of Shorten et al. [Sho07a]. This model has higher memory demands, such that it is not possible to solve it with OpenMP 4.5 for a muscle fiber mesh with 1481 nodes on the GeForce RTX 3080 GPU. As noted before, we use the NVIDIA Quadro GP100 GPU for the next study. This GPU is also not capable of solving the whole set of 1481 models instances per fiber times 169 fibers at the same time. For one instance of the subcellular model, 57 state and rate variables each, and 71 intermediate variables have to be stored, along with other data, such as element lengths for every element.

Thus, we follow a hybrid approach. We parallelize the scenario to 27 processes on the CPU. Only one process offloads its subdomain to the GPU. In this way, both the CPU and the GPU take part in the computation and the available hardware capabilities are fully exploited.

The scenario and the numeric parameters are the same as described for the study with the Shorten model in Sec. 1.9.2. 49 muscle fibers are used and parallelized to $3\times3\times3 = 27$ subdomains. Figure 1.48 visualizes the partitioning of the fibers by different colors and the EMG values ϕ_e on the muscle surface at the simulation end time of $t_{\text{end}} = 10\,\text{ms}$.

Figure 1.49 visualizes the runtimes of two runs. The first bar only employs the CPU and provides the reference for the measured runtimes. The second bar corresponds to the hybrid run, where one process employs the GPU. The 0D and 1D solver runtimes in the second bar are averaged over the CPU computations. The total runtime involves both the CPU and the GPU computations. It can be seen that the total runtime given by the total bar heights is higher for the hybrid run. Here, the CPU processes have to wait until the GPU process completes.

1.11.3 Conclusion

Several scenarios with computations of the 0D subcellular and 1D electric conduction models on the GPU have been evaluated. Section 1.9.2 compared the runtime for the Hodgkin-Huxley subcellular model on 625 fibers with implementations on the CPU. In Sec. 1.10.1, the computation with the Shorten subcellular model was measured for one fiber. Section 1.11.1 conducted a strong scaling study with the Hodgkin-Huxley model

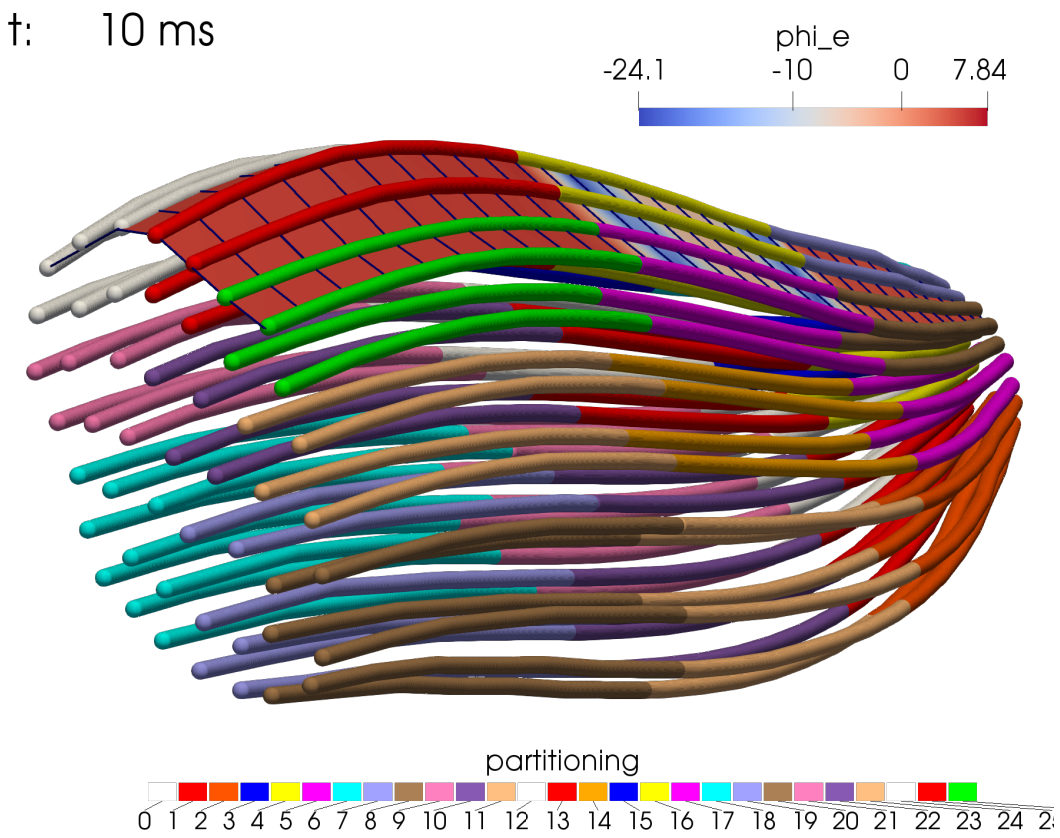


Figure 1.48: Partitioning of the 169 fibers to 27 processes used in the runtime study with hybrid CPU-GPU usage. The image also shows the resulting EMG signals ϕ_e on the muscle surface.

on 169 fibers and Sec. 1.11.2 evaluated a hyprid approach, where the Shorten model on different fibers is computed on the GPU and the CPU at the same time. This last study used a GPU with higher double-precision performance than the previous studies.

In all of these studies, the GPU computations could not compete with their CPU counterparts. The GPU computations relied on target-specific CUDA code, which was automatically generated by the OpenMP 4.5 pragmas produced by the code generator in OpenDiHu. The CPU computations used highly optimized CPU code with explicit vector instructions, which is close to optimal as shown by the roofline model in Sec. 1.10.2. Thus, the comparison considers different levels of optimization. We cannot conclude in general that the GPU is less suited to solve the the fiber-based electrophysiology models than the CPU.

The required GPU support of OpenMP in the GNU compiler is functional to the extend

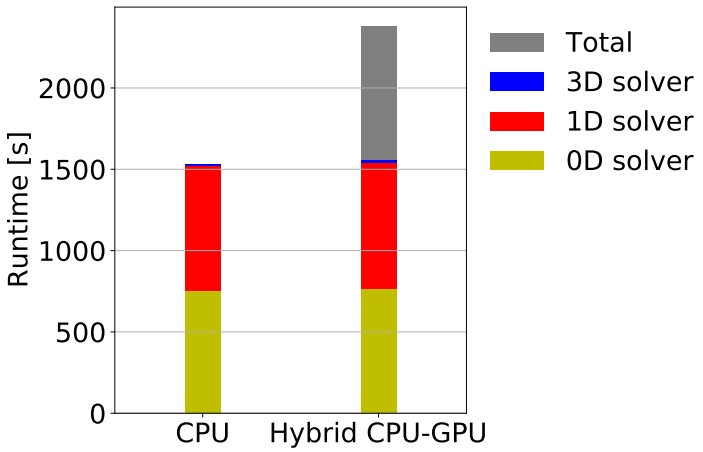


Figure 1.49: Runtime study for a CPU-only computation and a hybrid parallelization that employs both the CPU and the GPU. The compute-intense subcellular model of Shorten et al. is simulated.

needed in our studies only since GCC version 11, which, at the time of writing, is still experimental and not yet released. Further performance gains can be expected in the future as compiler development advances. One problem is also the high memory requirements for the subcellular models, which only allow a certain number of subcellular model instances to be computed on given hardwar. It is also not clear, if the memory consumptions will improve with later compiler versions.

Despite the lower performance, it was shown that OpenDiHu can be used to solve the monodomain equation **??** with different subcellular models on the GPU. Switching between the CPU and GPU versions can be accomplished by only changing the `optimizationType` parameter between `"vc"` and `"gpu"` and hybrid strategies, where some processes use `"vc"` and others `"gpu"`, have been demonstated.

In future work, the performance issue of the GPU computations can be addressed by using different technologies to access the compute power of GPUs. Examples are to use the CUDA programming language or the C++ based abstraction layer for acceleration hardware SYCL [19]. OpenDiHu already provides a reference implementation for such improvements by the code generator, which outputs model specific code with OpenMP pragmas for GPU offloading. This code implements proper economical data transfer between the devices and contains hints how to distribute the workload on the GPU by the respective pragma placements. It could be used as a starting point to integrate further "optimization types" in the code generator.

How To Reproduce

The scripts to run the studies and to create the plots for Figures 1.47 and 1.49 are available in the repository at github.com/dihu-stuttgart/performance in the directories `opendihu/16_hodgkin_huxley_gpu` and `opendihu/17_shorten_gpu`.

1.12 Parallel Scaling of the EMG model in High Performance Computing

After the parallel scaling of the multi-scale model in moderately parallel scenarios up to 27 processes has been investigated in the previous sections, we now study the parallel scalability in High Performance Computing scenarios with larger degrees of parallelism. We conduct these studies on the supercomputer Hawk at the High Performance Computing Center Stuttgart (HLRS).

In the following, Sec. 1.12.1 presents a weak scaling study, which scales the problem size up to a realistic number of 270 000 muscle fibers. Then, Sec. 1.12.2 shows measurements of the scaling behavior for the 1D model solver and gives details on MPI rank placement policies.

1.12.1 Weak Scaling of the Fiber Based Electrophysiology Model

We simulate the fiber based electrophysiology model with EMG values on the muscle surface and the subcellular model of Hodgkin and Huxley [Hod52]. Corresponding simulation results of this scenario, also for the highly parallel runs, are presented in Sec. 1.4.4.

In this weak scaling study, the number of fibers and number of processes is varied while their relation is kept approximately constant. The scenarios are chosen such that there are approximately 10 fibers per process, while maintaining a cube-liked partitioning scheme in the 3D domain. The 0D subcellular and the 1D electric conduction problems on the fibers are solved with the `FastMonodomainSolver` class and the `"vc"` optimization type, which is the fastest available option in OpenDiHu. The Strang operator splitting scheme with Heun's method and the implicit Euler scheme are used. The 3D problem is solved

using the conjugate gradient solver of PETSc and a relative tolerance on the residual norm of 10^{-5}.

The used timestep widths are $dt_{0D} = 10^{-3}$ ms, $dt_{1D} = dt_{\text{splitting}} = 2 \cdot 10^{-3}$ ms and $dt_{3D} = 1$ ms. Because only the scaling behaviour is of interest in this study, the simulation end time is set to $t_{\text{end}} = 2$ ms.

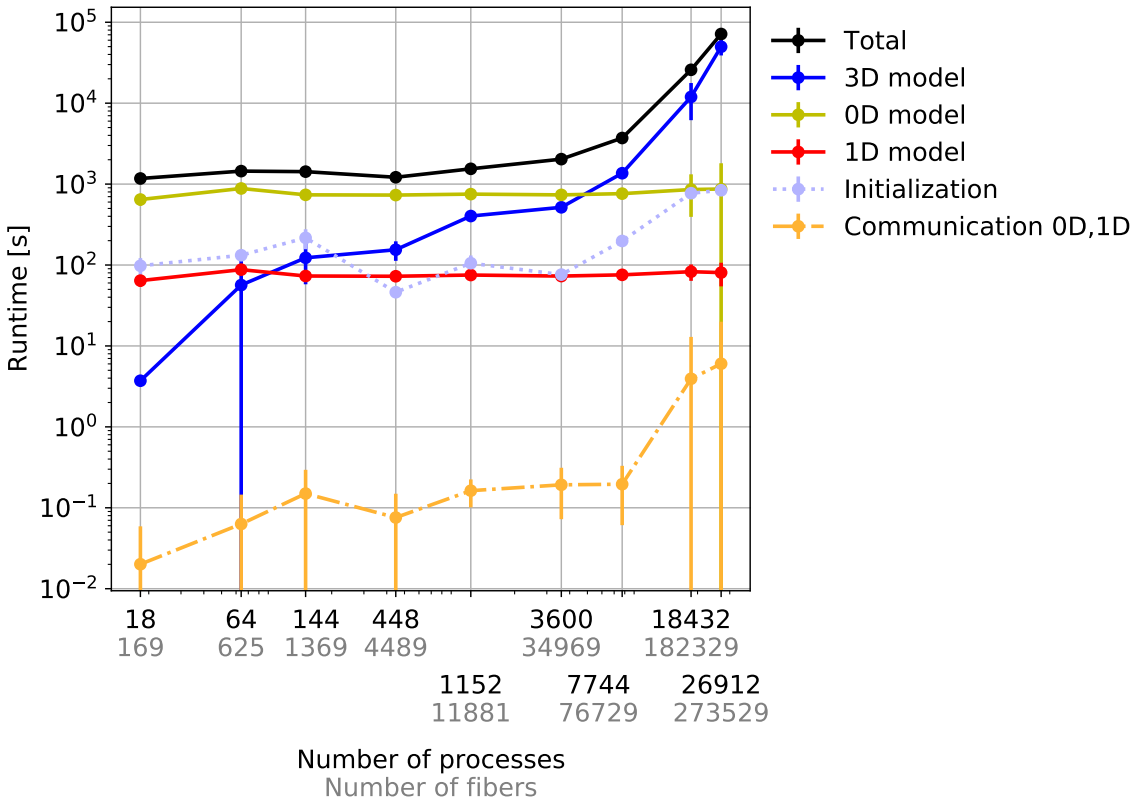


Figure 1.50: Weak scaling of the fiber based electrophysiology model on the supercomputer Hawk simulating up to more than 270 000 muscle fibers.

Figure 1.50 presents the resulting runtimes for the different parts of the simulation program: the solvers of the 0D, 1D and 3D models, the runtime for initialization and the runtime for the communication in the `FastMonodomainSolver`, as explained in **??**. To relate the initialization runtime to the runtime of the solvers in a realistic scenario with longer simulation times, all runtimes except for the initialization are scaled to a simulation end time of 1 s.

The results show perfect weak scaling properties of the 0D and 1D solvers, given by

# processes	18	64	144	448	1152	3600	7744	26 912
# iterations	72	115	176	339	561	1056	1636	2807

Table 1.2: Number of iterations of the conjugate gradient solver for the weak scaling study presented in Fig. 1.50.

the yellow and red lines. Both is expected due to the construction and the parallel partitioning. The 0D problems are “embarassingly parallel” and are solved independent of each other. In the 1D problem solver, the values are transferred to a dedicated process, where the serial Thomas algorithm is employed. Thus, the solution of all 1D problems is also performed independent of each other, except for the communication cost. The plot shows a very small runtime for this communication, which is given by the orange dashed line, even for higher parallelism.

The time for initialization involves parsing the Python script, which for the last data point requires 35.1 s, parallel file access and read operations of the mesh file, assembly of the 3D stiffness matrix and solution of the potential flow problem to obtain the fiber directions in the 3D mesh, which contains approximately 10^8 dofs for the last data point, code generation, compilation, linking and loading of the shared library for the subcellular problem, and initialization of all internal data structures.

Loading the mesh input file from the file system is the part of the initialization that requires the most runtime. The dotted light blue line in Fig. 1.50 shows that the initialization time increases to a maximum value of 839 s for the largest problem size.

The runtime of the 3D model is shown by the blue line in Fig. 1.50. This part of the model is responsible for the highest portion of the total runtime starting from the scenario with 7744 fibers and 76 729 processes. This increase is two-fold: first, the communication cost increases for a larger number of processes. Second, the number of iterations in the conjugate gradient solver increases for a larger number of unknowns.

In this weak scaling study, the number of conjugate gradient solver iterations increases from 72 for the first data point to 2807 for the last data point, as listed in Tab. 1.2. The 3D problem of the last data point has 10^9 dofs. The exact numbers of dofs are also listed in Tab. 1.1 in Sec. 1.4.4. Currently, the solution of the 3D problem uses no preconditioner. In future work, a multigrid solver could be employed for preconditioning, which could improve the weak scaling for large problem sizes.

In summary, the solution of the multi-scale model for fiber based electrophysiolgy without fat layer exhibits a very good weak scalability for up to $35 \cdot 10^3$ fibers. For larger

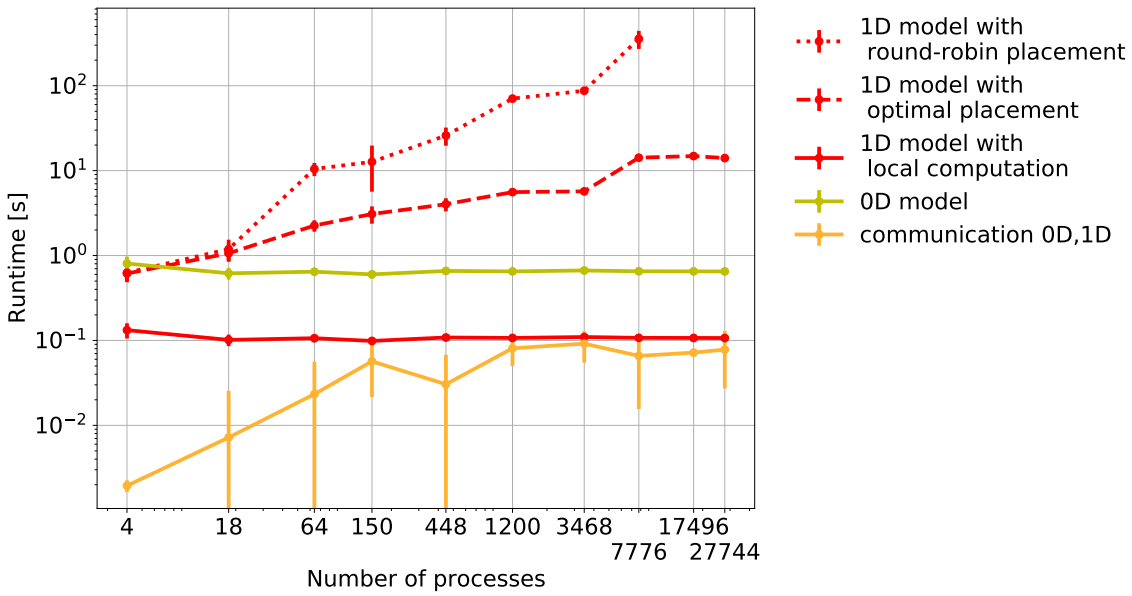


Figure 1.51: Rank placement strategy

problem sizes, the solution of the 3D problem dominates and the weak scaling behavior deteriorates. However, the solution times are still feasible, as such large problems have been successfully solved in Sec. 1.4.4 of this work.

1.12.2 Weak Scaling of the 1D Solver and MPI Rank Placement

Next, we compare the different approaches to solve the 1D electric conduction part of the monodomain equation on the fibers. Figure 1.51 shows a similar weak scaling study to the one in the last section with slightly different process counts. The study was carried out on the supercomputer Hazel Hen, Cray XC30 which was installed until 2020 at the High Performance Computing Center Stuttgart. The same problem as in the last section is solved, and, again, the relation between fibers and processes is 10:1. The partitionings were chosen in accordance with the compute nodes of Hazel Hen, which had 24 cores. The detailed problem sizes and partitionings can be found in [Mai19].

Figure 1.51 shows the runtimes of the same 0D and 1D solvers as in the last section by the solid yellow line for the 0D problem, the solid red line below for the 1D problem and the solid orange line for the communication. The perfect weak scaling for these parts is in line with the previous observations.

In addition, we compare the weak scaling of the 1D solution if a parallel conjugate

gradient solver of PETSc is used for the problem on every fiber, instead of the serial Thomas algorithm. This setup corresponds to the vc–aovs scenario presented in Sec. 1.9.2. As already noted earlier, the performance of this approach is worse. The dashed and dotted lines in Fig. 1.51 present the runtime of this approach for two different MPI rank placement strategies, but for the identical program. It can be seen that the runtime increases with higher numbers of processes in both curves. This effect is the result of the 1D fiber problems being distributed to more processes, as the total number of processes increases.

For example, in the scenarios with 1200 and 3468 processes, all fibers are distributed to 12 different processes. For the last three data points of the dashed curve, all fibers are distributed to 24 processes. As a result, the runtime to solve the 1D problems in the measurements with 1200 and 3468 processes is approximately equal, but lower than the runtime for the last three data points, where twice the amount of processors take part in the solution of a single 1D problem.

The difference between the dotted and dashed red curves is a different strategy to place the processes on the compute nodes. The dashed curve with the lower runtime corresponds to a placement of all the processes that share fibers on the same compute node. As the subdomain indices in the $n_x \times n_y \times n_z$ partitioning increase fastest in x-direction, then in y-direction and then in z-direction and the fibers are aligned with the z direction, the set of processes on a compute node contains MPI ranks that are offset with a constant stride of $n_x n_y$. This has to be ensured by explicit rank pinning to the compute nodes.

If no such measures are taken, the default placement of MPI ranks to compute nodes occurs consecutively by filling the compute nodes in order. This corresponds to a round-robin placement of the MPI ranks that compute the same fiber, which is the worst possible way of distributing MPI ranks to compute nodes. All processes that compute a fiber are all located on different nodes and have to communicate over compute node boundaries. Figure 1.51 shows the resulting runtimes by the upper, dotted red curve. The difference in runtime increases to one order of magnitude with higher number of cores.

Thus, it is important to properly handle MPI rank placement on compute clusters with multiple compute nodes. In consequence, we also configured rank placement on the supercomputer Hawk accordingly in all our studies on this system.

1.13 Performance Studies of the Solid Mechanics Solver

TODO

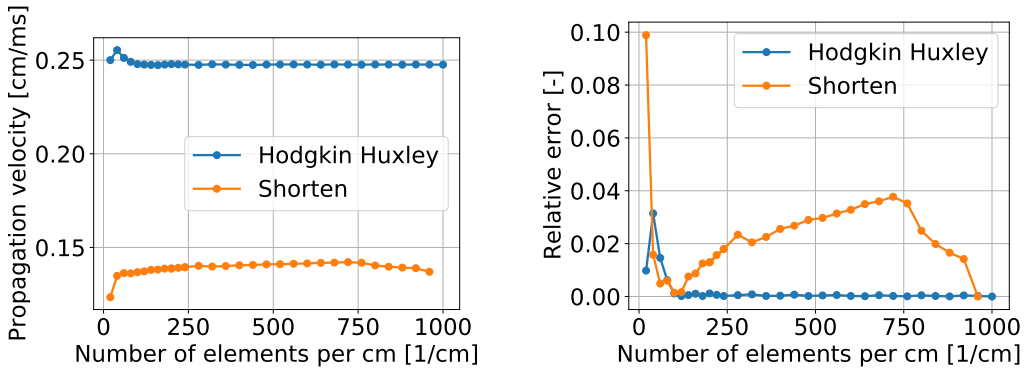
1.14 Studies of Numeric Properties

Next, we perform numeric studies to evaluate used mesh resolutions and solvers. Section 1.15 addresses the mesh width of the 1D problem. Then, Sec. 1.15.1 evaluates different numerical solver choices for the multidomain model.

1.15 Effect of the Mesh Width on the Action Potential Propagation Velocity

The numeric parameters of a simulation scenario, such as mesh widths and timestep widths should be chosen such that the resulting numerical errors are balanced between all model components. In the simulation of activated muscle fibers, the propagation velocity of the action potentials is an important quantity, which also influences the macroscopic outcome of EMG recordings. Thus, we investigate the effect of numerical parameters on the propagation velocity.

Figure 1.52



(a) Propagation velocities over spatial resolution of the 1D mesh.

(b) Relative error of the propagation velocities over spatial resolution of the 1D mesh.

Figure 1.52: Propagation velocities of action potentials for the subcellular models of Shorten and Hodgkin-Huxley. This study is used to determine the 1D mesh width.

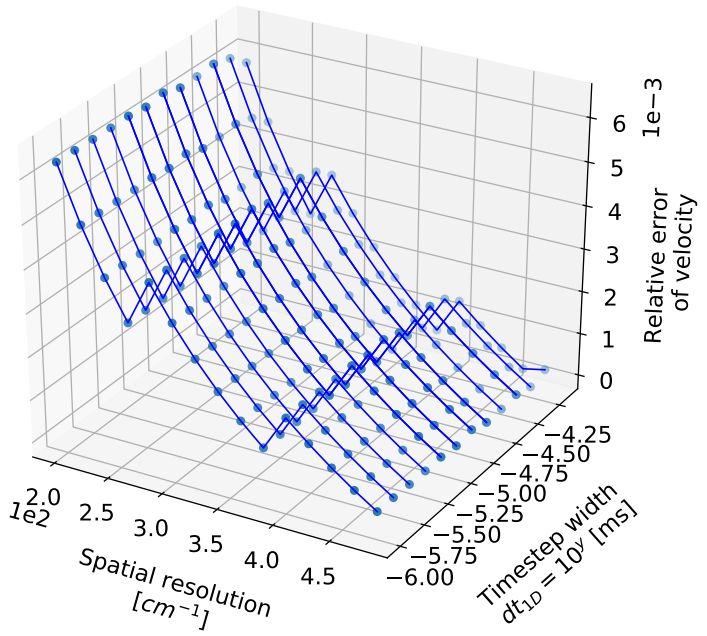


Figure 1.53: Error of the action potential propagation velocity for varying mesh width and timestep width.

1.15.1 Linear Solvers for the Multidomain Problem

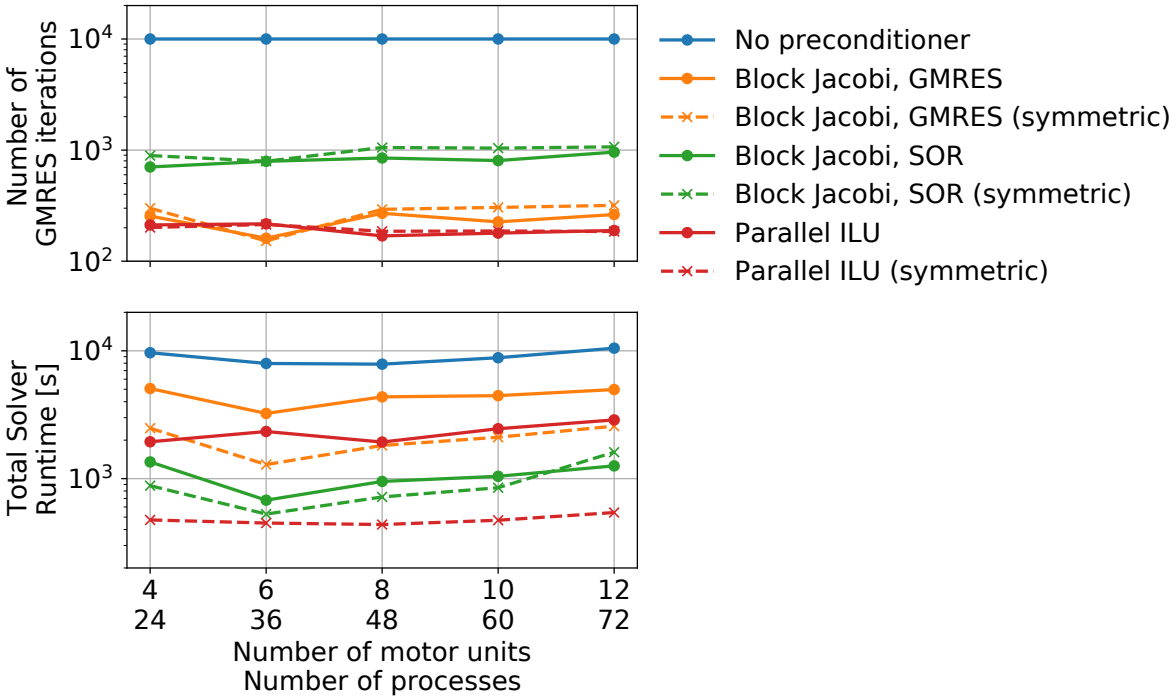


Figure 1.54: Multidomain solvers

In the next study, we investigate the parallel weak scaling behavior for the multidomain model with fat domain, i.e. **????????** or models (b1),(c) and (d) in **??**. The subcellular model of Shorten et al. [Sho07a] is used. The goal is to evaluate different preconditioners for the linear solution of the system of equations. The model is discretized by a Strang operator splitting with Heun's method for the subcellular model and an implicit Euler scheme for the multidomain equations. The timestep widths of all schemes are $dt_{0D} = dt_{\text{multidomain}} = dt_{\text{splitting}} = 5 \cdot 10^{-4}$ ms, the end time is 10^{-1} ms.

We partition the 3D domain of a 50 024-node muscle mesh and a 37 000-node fat mesh with 24, 36, 48, 60 and 72 processes and simulate 4,6,8,10 and 12 MUs. As the square-shaped system matrix contains one row of blocks for every MU plus one row of blocks for the fat mesh, the total number of rows scales nonlinearly with the number of MUs. In consequence, the system matrices in the scenarios contain 279 720, 379 768, 479 816, 579 864 and 679 912 rows and columns.

We solve the linear system of the multidomain equations by a GMRES solver, because the system matrix is non-symmetric. The stopping threshold on the residual norm is set to 10^{-15} and the specified maximum number of iterations is 10^4. Figure 1.54 shows the

number of GMRES iterations in the upper plot and the total runtimes of the preconditioner and solver in the lower plot.

For every preconditioner, the preconditioning is either performed using the non-symmetric system matrix (solid lines) or using a symmetric matrix that is obtained by taking all diagonal blocks of the system matrix (dashed lines).

The reference measurement is given by using no preconditioner. It can be seen that the specified tolerance is not achieved and the maximum number of 10^4 iterations is reached.

Two versions of the block Jacobi preconditioner of PETSc are evaluated. The block Jacobi divides the system matrix into blocks of smaller problems that are each solved individually. The first variant uses a GMRES solver, the second variant uses a SOR (successive over relaxation) solver, i.e., a Gauss-Seidel scheme. Another solver is ”Euclid“ [Hys01] from the HYPRE package, a parallel implementation of incomplete LU factorization using graph partitioning and a two-level ordering strategy.

1.16 Parallel in Time master thesis

maybe

1.17 Output file sizes

no

1.18 dx-dt dependencies

no

1.19 Load balancing bachelor thesis

no

1.20 Application of opendihu within the field of robotics

no

2 Conclusion and Future Work

2.1 Future Work

Bibliography

[19] *SYCL Specification, SYCL integrates OpenCL devices with modern C++*, Version 1.2.1, Khronos Group Inc., 2019

[Ame01] **Amestoy**, P. R. et al.: *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications 23.1, 2001, pp. 15–41, doi:10.1137/S0895479899358194, eprint: https://doi.org/10.1137/S0895479899358194, https://doi.org/10.1137/S0895479899358194

[Bra18] **Bradley**, C. P. et al.: *Enabling detailed, biophysics-based skeletal muscle models on HPC systems*, Frontiers in Physiology 9.816, 2018, doi:10.3389/fphys.2018.00816

[Cla21] **Clarke**, A. K. et al.: *Deep learning for robust decomposition of high-density surface emg signals*, IEEE Transactions on Biomedical Engineering 68.2, 2021, pp. 526–534, doi:10.1109/TBME.2020.3006508

[Fei55] **Feinstein**, B. et al.: *Morphologic studies of motor units in normal human muscles*, Cells Tissues Organs 23.2, 1955, pp. 127–142

[Hei13] **Heidlauf**, T.; **Röhrle**, O.: *Modeling the chemoelectromechanical behavior of skeletal muscle using the parallel open-source software library opencmiss*, Computational and Mathematical Methods in Medicine 2013, 2013, pp. 1–14, doi:10.1155/2013/517287, http://dx.doi.org/10.1155/2013/517287

[Hod52] **Hodgkin**, A. L.; **Huxley**, A. F.: *A quantitative description of membrane current and its application to conduction and excitation in nerve.* The Journal of Physiology 117.4, 1952, pp. 500–544

[Hol07a] **Holobar**, A.; **Zazula**, D.: *Multichannel blind source separation using convolution kernel compensation*, IEEE Transactions on Signal Processing 55.9, 2007, pp. 4487–4496, doi:10.1109/TSP.2007.896108

[Hol07b] **Holobar**, A.; **Zazula**, D.: *Gradient convolution kernel compensation applied to surface electromyograms*, Independent Component Analysis and Signal Separation, ed. by **Davies**, M. E. et al., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 617–624, isbn:978-3-540-74494-8

[Hol08] **Holobar**, A.; **Zazula**, D.; **Merletti**, R.: *Demusetool-a tool for decomposition of multichannel surface electromyograms*, 2008

[Hys01] **Hysom**, D.; **Pothen**, A.: *A scalable parallel algorithm for incomplete factor preconditioning*, SIAM Journal on Scientific Computing 22.6, 2001, pp. 2194–2215, doi:10.1137/S1064827500376193, eprint: https://doi.org/10.1137/S1064827500376193, https://doi.org/10.1137/S1064827500376193

[Klo20] **Klotz**, T. et al.: *Modelling the electrical activity of skeletal muscle tissue using a multi-domain approach*, Biomechanics and Modeling in Mechanobiology 19.1, 2020, pp. 335–349, ISSN: 1617-7940, doi:10.1007/s10237-019-01214-5, https://doi.org/10.1007/s10237-019-01214-5

[Kol21] **Kolvekar**, S.: *Gated Recurrent Unit Network for Decomposition of Synthetic High-Density Surface Electromyography Signals*, MA thesis, Pfaffenwaldring 47, 70569 Stuttgart, Germany: Institute for Signal Processing and System Theory, University of Stuttgart, 2021

[Mac06] **MacIntosh** B., R.; **Gardiner** P., F.; **McComas** A., J.: *Skeletal Muscle: Form and Function*, Second, Human Kinetics, 2006

[Mac84] **MacDougall**, J. D. et al.: *Muscle fiber number in biceps brachii in bodybuilders and control subjects*, Journal of Applied Physiology 57.5, 1984, PMID: 6520032, pp. 1399–1403, doi:10.1152/jappl.1984.57.5.1399, eprint: https://doi.org/10.1152/jappl.1984.57.5.1399, https://doi.org/10.1152/jappl.1984.57.5.1399

[Mai19] **Maier**, B. et al.: *Highly parallel multi-physics simulation of muscular activation and EMG*, COUPLED PROBLEMS 2019, 2019, pp. 610–621

[Sho07a] **Shorten**, P. R. et al.: *A mathematical model of fatigue in skeletal muscle force contraction*, Journal of Muscle Research and Cell Motility 28.6, 2007, pp. 293–313, doi:10.1007\%2Fs10974-007-9125-6, http://dx.doi.org/10.1007%5C%2Fs10974-007-9125-6

[Sho07b] **Shorten**, P. R. et al.: *A mathematical model of fatigue in skeletal muscle force contraction*, Journal of muscle research and cell motility 28.6, 2007, pp. 293–313

[Yan20] **Yang**, C.: *Empirical Roofline Tool (ERT)*, https://crd.lbl.gov/departments/computer-science/par/research/roofline/software/ert/, 2020, https://crd.lbl.gov/departments/computer-science/par/research/roofline/software/ert/