VIII International Conference on Computational Methods for Coupled Problems in Science and Engineering
COUPLED PROBLEMS 2019
E. Oñate, M. Papadrakakis and B. Schrefler (Eds)

HIGHLY PARALLEL MULTI-PHYSICS SIMULATION OF MUSCULAR ACTIVATION AND EMG

B. MAIER*, N. EMAMY*, A. KRÄMER‡ AND M. MEHL*

*Institute for Parallel and Distributed Systems, University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
e-mail: {benjamin.maier, nehzat.emamy, miriam.mehl}@ipvs.uni-stuttgart.de
web page: https://www.ipvs.uni-stuttgart.de/abteilungen/sgs

‡Institute of Applied Analysis and Numerical Simulation, University of Stuttgart
Allmandring 5b, 70569 Stuttgart, Germany
e-mail: aaron.kraemer@ians.uni-stuttgart.de
web page: https://www.ians.uni-stuttgart.de/cmcs

Key words: Musculo-skeletal, High Performance Computing, Multi-scale

Abstract. Simulation of skeletal muscle activation can help to interpret electromyographic measurements and infer the behavior of the muscle fibers. Existing models consider simplified geometries or a low number of muscle fibers to reduce the computation time. We demonstrate how to simulate a finely-resolved model of biceps brachii with a typical number of 270.000 fibers. We have used domain decomposition to run simulations on 27.000 cores of the supercomputer *HazelHen* at HLRS in Stuttgart, Germany. We present details on *opendihu*, our software framework. Its configurability, efficient data structures and modular software architecture target usability, performance and extensibility for future models. We present good parallel weak scaling of the simulations.

1 INTRODUCTION

The neuro-muscular system allows humans to perform a wide range of tasks. A skeletal muscle, with its adaptable neural recruitment process and numerous muscle fibers can perform fine, coordinated movements such as gripping a small needle, powerful actions such as lifting weights, and enduring, repetitive actions such as running a marathon.

Due to the difficulty of observing internal activities in in-vivo experiments, simulation plays an important role in understanding the neuro-muscular system. What can indeed be measured on a live subject is the electric potential over time. This process, known as electromyography (EMG), can be performed either on the surface or intramuscularly via needle electrodes. Although signal-processing techniques can help us understand the functioning of the neuromuscular system [1], weak signals, shifting of electrodes during

movement, and muscle fiber cross-talk [2] make it difficult to identify precisely which muscle fibers have been activated. Simulating the muscle activation process can thus help improve post-processing techniques for interpreting high-density EMG measurements and provide insights into those processes not detectable via measurements.

Both analytical and numerical models exist to perform in-silico experiments regarding the electromyography of a skeletal muscle. Analytical models provide closed formulas for the resulting signal [3] and are typically limited to simplified geometries (e.g. cylinders) and homogeneous material properties. For example, the authors in [4] derive a formulation for a single fiber and a plate electrode. A finite-element-based numerical model that also includes bone, fat and skin tissue is presented by [5]. Here, however, the domain is cylindrical and only a single action potential is simulated. In [6] the authors consider an axis-symmetric muscle geometry and simulate prescribed fiber shortening by adjusting the diffusion tensor. The electric conduction through the tissue volume is discretized with 100,000 finite elements using a commercial solver. However, this work does not account for the actual muscle geometry and for the activation of fibers organized into motor units.

The authors of [7] describe a framework to model biophysically motivated muscle activation and contraction and the resulting EMG. They study 390 fibers in a cuboid geometry. They also illustrate a realistic geometry by simulating the tibialis anterior muscle with 2700 fibers. However, the latter simulation is presented only for a short time span with little contraction. They report long computation and file I/O times using the open-source software OpenCMISS. In our previous work [8] we parallelized the OpenCMISS code to run on up to 768 cores. We detected issues with the memory consumption, which prevented further parallel scaling.

In this work, we present the computational framework *opendihu* to perform parallel simulations of the neuro-muscular system. We implement a highly-resolved, physically based, multi-scale model of muscular activation and intramuscular EMG. Our approach uses the same models for the action potential and predicting the intramuscular EMG as in [7]. However, we do not include the model for tissue contraction. We use the real geometry of the musculus biceps brachii and simulate 273,529 muscle fibers, which is a typical number for the biceps brachii [9]. The overall problem contains 1.6 billion degrees of freedom for the activation model and 37 million for the computation of the EMG. Our code executes on 27,744 cores of the supercomputer *HazelHen* in Stuttgart and allows simulations with runtimes in the range of several hours.

In section 2 we describe the model used in our simulation. We motivate the design goals of our software framework *opendihu* and describe the software components in section 3. In section 4 we perform weak scaling studies that are evaluated in section 5.

2 MODEL

A skeletal muscle comprises numerous muscle fibers. These fibers are innervated by axons of the nervous system at the innervation zone, which is located at the center of the fibers. Motor neurons in the spinal cord generate stimuli that lead to electric action

potentials on the muscle fibers. Upon activation of a fiber, the action potential travels from the innervation zone to both ends of the fiber.

All muscle fibers that are connected to the same motor neuron are excited simultaneously and form a motor unit (MU). In our simulations, we assumed 100 MUs. The number of fibers per MU was distributed exponentially. We employed a recruitment model that activated smaller MUs more frequently than larger MUs, as in [7].

In the following, we describe the model of action potential propagation and contribution to intramuscular EMG signals.

2.1 The model of activation and electromyography

Our simulation is based on the framework described in [7], which is briefly summarized here. The multi-scale model is comprised of the following three ordinary and partial differential equations, solved on 0D, 1D and 3D domains, respectively:

$$\frac{\partial \mathbf{y}}{\partial t} = f(\mathbf{y}, V_m), \tag{1}$$

$$\frac{\partial V_m}{\partial t} = \frac{1}{A_m\, C_m} \left(\sigma_{\text{eff}} \frac{\partial^2 V_m}{\partial x^2} - A_m\, I_{\text{ion}}(\mathbf{y}, V_m) \right) \quad \text{in } \Gamma_t, \tag{2}$$

$$\operatorname{div}\left((\boldsymbol{\sigma}_e + \boldsymbol{\sigma}_i) \operatorname{grad}(\phi_e) \right) + \operatorname{div}\left(\boldsymbol{\sigma}_i \operatorname{grad}(V_m) \right) = 0 \quad \text{in } \Omega_t, \tag{3}$$

where $\mathbf{y}$ and V_m are the vector of subcellular states and transmembrane potential, respectively. A_m is the fibers' surface to volume ratio. C_m is the capacitance of the membrane and σ_{eff} is the effective conductivity of the muscle fibers. I_{ion} describes the ionic current passing across the membrane. The conductivity tensors in the extra and intracellular domains of the bidomain model are represented as $\boldsymbol{\sigma}_e$ and $\boldsymbol{\sigma}_i$ and the extracellular potential is ϕ_e.

The first equation, eq. (1), describes subcellular processes and is a system of ordinary differential equations (ODEs) with state vector $\mathbf{y}$. We use the Hodgkin-Huxley model [10], which contains 4 state variables. In the following, we refer to it as the 0D model.

The second equation, eq. (2), is the 1D monodomain equation. We model each muscle fiber as a 1D domain Γ_t embedded in the 3D muscle domain Ω_t. We solve the monodomain equation, eq. (2), for each of these 1D domains. The connection to the 0D model is made through the states $\mathbf{y}$, which are computed at equidistant points on the 1D domains by solving the system of ODEs, eq. (1).

The third equation of the model is the bidomain equation, eq. (3), which provides the extracellular potential, ϕ_e, in the 3D muscle domain, Ω_t. This is in fact the electric signal that could be measured by the intra-muscular electromyography. The required transmembrane potential, V_m, in the 3D domain is projected from the values of V_m in the embedded 1D domains using a trilinear interpolation.

2.2 Discretization and solution

We solve the coupled 0D and 1D equations of the model, eqs. (1) and (2), with the Strang operator splitting approach [11]. The time derivatives are discretized using finite differences. We integrate the 1D reaction-diffusion equation by the Crank-Nicolson scheme and the 0D equations with Heun's method. Thus, the splitting scheme would be second order consistent.

For the spatial discretization of eq. (2), we use the Finite Element Method with linear 1D elements and solve eq. (1) at each node. We solve the 3D part, eq. (3), with the Finite Element Method using 3D linear hexahedron elements. The 3D and 1D discretizations share some nodes, such that the 1D fiber meshes are aligned with the edges of the nearby 3D elements, which they pass through. The 3D elements have a coarser discretization than the 1D elements. A visualization of the meshes is given in fig. 1a.

For the 3D model, eq. (3), the intra and extra-cellular conductivity tensors, $\boldsymbol{\sigma}_i$ and $\boldsymbol{\sigma}_e$, are chosen such that the conductivity is 10 times higher in the direction of fibers than in the transverse direction. We estimate the fibers' directions by considering streamlines of a potential flow, which we solve inside the muscle geometry, a valid approach approved by the literature [12, 13]. The potential flow problem is described by a Laplace equation for the pressure with Dirichlet boundary conditions of 0 and respectively 1 at the longitudinal ends of the muscle.

This auxiliary potential flow problem above, the 1D monodomain and 3D bidomain equations involve solution of linear systems of equations. Only the Crank-Nicolson scheme of the 1D equation yields a symmetric positive definite system matrix, for which we use a conjugate-gradient solver. For the solution of the other two system of equations, we employ a GMRES solver.

3 SOFTWARE

We implement a solver for the described model within our novel software framework, named *opendihu*. This framework provides functionality to solve static and dynamic multi-physics problems, spatially discretized in 1D, 2D and 3D by the finite element method. Our core design goals of the framework are usability, performance and extensibility. In the following section, we describe how these goals are realized in our framework.

3.1 Design goals

The first design goal is usability. From the user's perspective, running and examining a simulated scenario is possible without altering the technical C++ implementation. Changing the geometry, tuning discretization parameters or evaluating different model parameters can be done in a python script, which will be parsed by the simulation program at runtime.

The second design goal is performance. Parallel execution reduces runtime by distributing the computational workload to multiple CPUs. Vectorization allows to calculate

multiple values at once with single instructions. The internal data representations enable these paradigms and avoid time-consuming data copy. The framework runs efficiently on supercomputers as well as normal workstations with a smaller numbers of CPUs.

The third design goal is extensibility, which allows to create solvers for new models. The object-oriented architecture allows to combine existing components to new solvers and to implement new ones. A testing infrastructure ensures that existing functionality is preserved.

3.2 Software components

Opendihu is an open-source software framework[1]. To remain lightweight, it is closely built upon established open-source projects. The parallel message passing interface (MPI) is used to enable distributed memory parallelism. Linear algebra and solvers are provided by OpenBLAS[2] and PETSc[3]. A Python interpreter with Numpy and Scipy is included for the configuration interface.

Installation on different linux systems, supercomputers as well as workstations, is managed by a SCons[4] based build system. Supported compilers are GNU, Intel and Cray. Compliance to the C++14 standard is enforced by compiler flags that disallow warnings and add pedantic errors, which ensure high code quality.

The code comprises four parts: data representation, finite element method related functionality, solvers, and I/O functions.

Data representation. Native parallel PETSc data structures of vectors and sparse matrices are used. Vector fields have the "Struct of Array" layout, where each component is stored consecutively in the memory. This allows cache-efficient data accesses and enables instruction-level parallelism.

Finite Element discretization. As the base for the Finite Element Method, *opendihu* implements assembly of the stiffness and mass matrices for the generalized Laplace operator. Quadratures for 1D, 2D and 3D domains are performed using Gauss, Newton-Cotes and Clenshaw-Curtis schemes. Ansatz functions can be chosen as linear and quadratic Lagrange and cubic Hermite functions.

We have implemented three mesh types: (i) the general, unstructured mesh with arbitrary topology, (ii) the structured mesh, whose elements can be efficiently iterated over by nested for loops and (iii) the regular mesh with a fixed mesh width in all dimensions, for which the system matrix assembly uses time-efficient, pre-computed stencils.

[1] https://github.com/maierbn/opendihu, release 1.0 is used for the studies in this paper
[2] https://www.openblas.net
[3] https://www.mcs.anl.gov/petsc/
[4] https://scons.org/

Data can be mapped linearly between arbitrary meshes that occupy the same physical space, e.g. between one-dimensional muscle fibers and a 3D continuum representation of the tissue.

Solvers. Solvers and time stepping schemes are implemented as C++ class templates. Multiple schemes can be nested at compile-time to create multi-physics solution schemes. Examples of implemented schemes are the forward and backward Euler integration, Heun's method, Crank-Nicolson scheme, and Godunov and Strang operator splittings.

The main C++ program of every simulation consists of the definition of a compound class of such nested solvers and a "run" command. This allows users to get a quick overview over how the model is solved, by inspecting the minimalist main source file.

Input and output. Inputs to the simulation program include the configuration, geometry data and subcellular model. The configuration is done in a python script, where all parameters can be computed and specified. The script is initially parsed by the simulation program. During the time stepping, callback functions allow to alter parameters and boundary conditions, which we use for stimulating the MUs at prescribed physical times. The geometry data can be either specified in this script or provided as a binary file, which is then loaded in parallel using the distributed MPI read functions. This is needed to input large amounts of data in highly parallel runs.

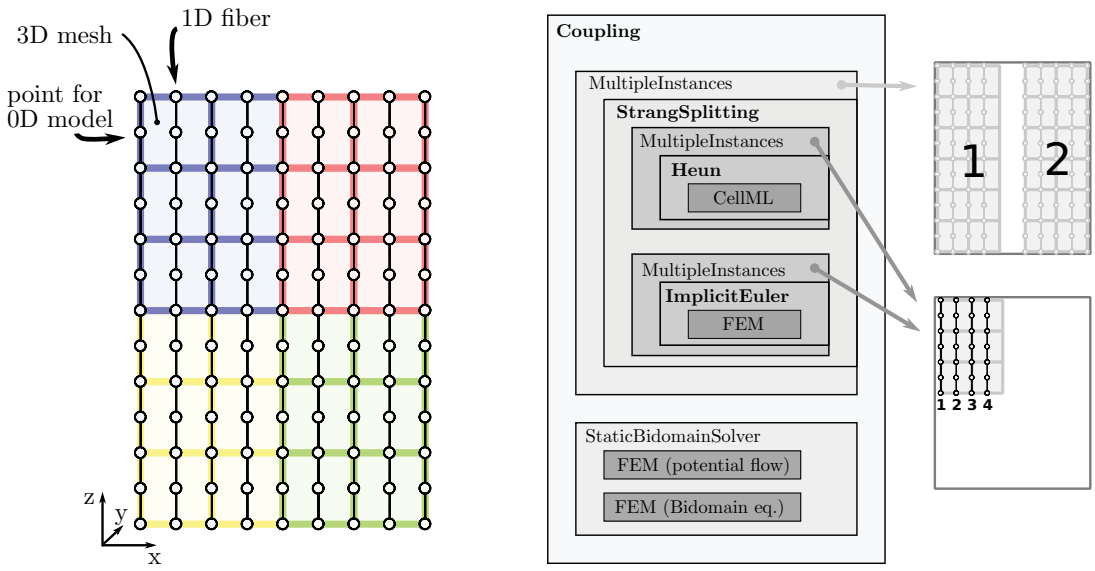
The subcellular model, eq. (1), is loaded from a CellML file, a standard description for biomechanical ODE models. *Opendihu* implements a novel source-to-source compiler, which combines the code for multiple instances of such models and, by enabling vectorization, reduces the solution time.

Output of simulation results can be written in various formats. A binary, python-pickle-based format and the ASCII-based Exfile format are used for small amounts of data, to be postprocessed with the *opendihu* matplotlib scripts and the OpenCMISS-Zinc based utility, respectively. For large amounts of data, *opendihu* implements MPI-parallel output to binary VTK files to be viewed with ParaView or can use the adaptable IO System 2 (ADIOS2) for parallel output in a binary-packed format.

3.3 Parallelization and solver structure

In our muscle simulations, we use structured grids for the spatial discretization. A structured grid is divided by planar cuts into subdomains for multiple processes, as depicted in fig. 1a. Thus, each subdomain contains parts of multiple fibers, where each fiber is handled by multiple processes.

The parallel solver structure is depicted in fig. 1b. The innermost solvers are the Heun scheme operating on the 0D model, eq. (1) using the CellML description, and the Crank-Nicolson scheme for solving the diffusion equation, eq. (2), for which the spatial operator is discretized by the finite element method. These schemes are wrapped in



(a) Visualization of meshes **(b)** Program structure

Figure 1: (a) 0D, 1D and 3D meshes and subdomains for 4 processes (different colors). (b) Composite solver structure. Each box corresponds to a class template in the program, time stepping schemes in bold font. For the *MultipleInstances* classes, the corresponding domains are depicted on the right.

MultipleInstances classes, which subsequently call them for all fibers in the subdomain. Thus, all processes that participate in the computation of fibers can solve the diffusion equation in parallel. Figure 1b at the bottom right visualizes the used data in this step. The Strang splitting scheme uses these two solution schemes to solve the propagation of the action potential in a set of fibers that share processes. The surrounding *MultipleInstances* class calls the splitting scheme subsequently for all of these sets of fibers, as visualized at the top right of fig. 1b. Finally, the solver for the 3D bidomain equation, eq. (3), is called after a specified time span by the top-level *Coupling* class.

4 RESULTS

To examine the efficiency of our software framework, we compare runtimes with the simulation of the tibialis anterior muscle in [7]. It consists of 2700 fibers, each comprising 50 instances of the Shorten subcellular model [14]. We replicate the discretization but only consider a cuboid geometry, since this does not influence the runtime. Moreover, we only solve the electrophysiology without the model for contraction, which in [7] was discretized using only 12 3D elements. We run our simulation using a single processor on a computer with an Intel(R) Xeon(R) CPU E7-8880 v3, which features the same microarchitecture (Ivy Bridge) but a lower frequency (2.3 GHz instead of 3.2 GHz) compared to the hardware used in [7]. Whereas the authors in [7] reported 3 h runtime to solve for 0.2 ms, *opendihu* computes the same time span in 4 min 40 s overall runtime, including file output, which is

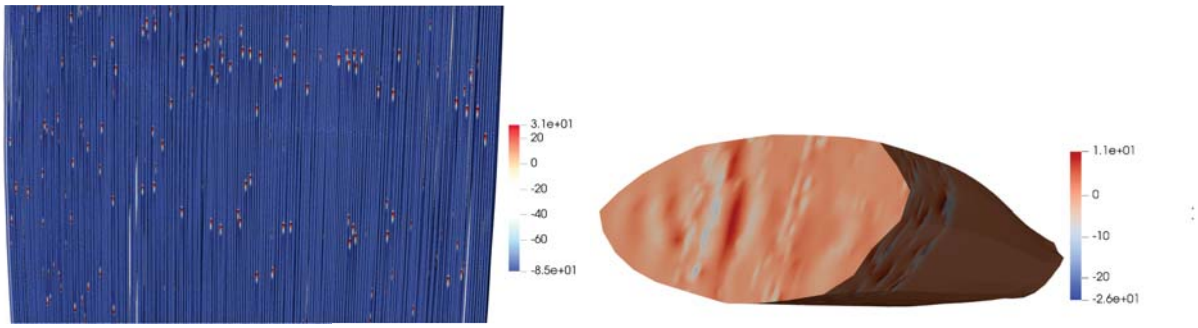


Figure 2: Activation of the biceps brachii muscle. Left: propagation of the action potentials, V_m, along the fibers (zoomed in). Right: contours of the extracellular potential, ϕ_e, at a cross section of the muscle.

a reduction by a factor of 38. On a modern laptop with an Intel Core i5-6300U CPU and frequency of 2.4 GHz the runtime using a single processor reduces to 80 s.

4.1 Weak scaling

In the following, we present studies of parallel scalability. Our scenario is the action-potential propagation in the biceps brachii muscle. The geometry is extracted from the Visible Human Male of the Visible Human Project [15]. The model is described in section 2. The stimulation times and material parameters are the same as in [7].

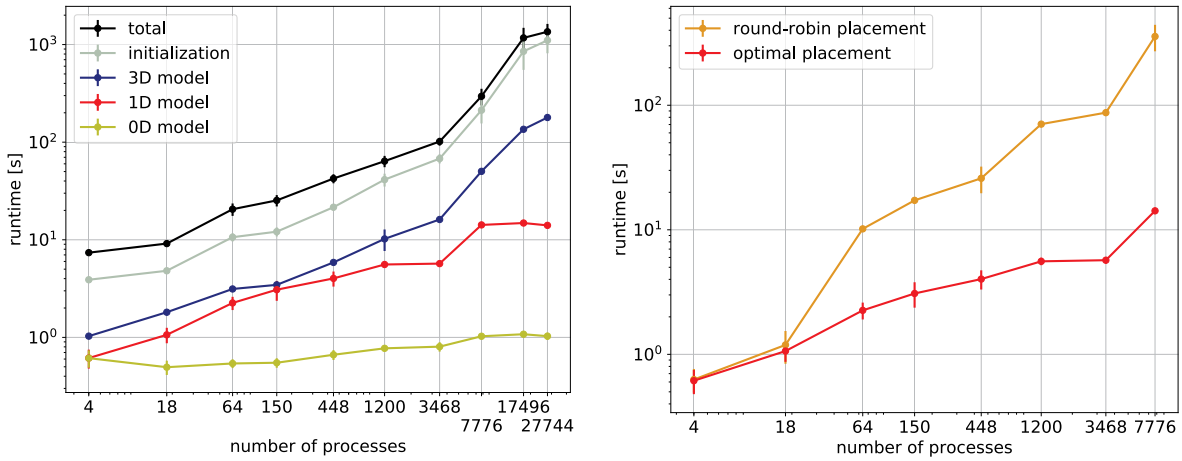
We compile our program with the Cray compiler and execute it on the supercomputer *HazelHen* in Stuttgart, Germany. This is a Cray XC40 computer with a total of 7712 compute nodes, each with 2 Intel Haswell E5-2680v3 CPUs at 2.5 GHz and 24 cores per node. Figure 2 shows the simulation results using 4489 fibers at time t =100 ms.

The weak scaling studies are depicted in fig. 3a. The runtimes are measured while increasing the size of problem and number of processes at the same time as listed in table 1. The size of problem is increased by increasing the number of muscle fibers. Each fiber is discretized by 1479 1D elements, leading to an element length of approximately 100 µm. The number of 3D elements is coupled to the number of fibers. In the x and y spatial directions, there are 3×3 fibers going through each of the 3D elements and in the z direction (direction of the fibers) there are 3 1D elements per each 3D element.

The time steps are 1.5 µs for Heun's method, 1 µs for the Crank-Nicolson scheme and 3 µs for the Strang splitting. The 3D bidomain equation is solved with 10.000 iterations of the GMRES solver in the interval of 1 ms. The simulation time for the weak scaling study is also 1 ms, which corresponds to one time step of the outermost loop.

We choose the number of fibers such that there are approximately 10 fibers per process for every measurement. We start from 4 processors and increase the number of processors up to the last data point of 27,744 processes for 273,529 fibers.

The parallel partitioning is performed based on the following two conditions. First, the number of subdomains in the fiber direction has to be a factor of 24, which is the number of cores per each compute node. Second, the number of subdivisions in the three



(a) Weak scaling

(b) Rank placement strategy

Figure 3: Parallel scaling experiments. (a) Weak scaling by increasing the problem size proportional to the number of processes. (b) Effect of rank placement strategy on the runtime of the 1D problem. The Round-robin places the processes corresponding to one fiber on different compute nodes. The optimal placement places the corresponding processes for each fiber on the same node.

coordinate directions should be similar. The first condition ensures that each fiber is computed on only one computational node, the second condition reduces the surface between neighboring subdomains and thus minimizes the communication in the solution of the 3D model.

The weak scaling shows a perfect behavior for the 0D model, as can be seen from the yellow curve in fig. 3a. This is expected, since all instances of the 0D model are independent of each other. The runtimes for solution of the 1D model, shown by the red curve in fig. 3a, is only dependent on the numbers of processes per fiber, which are given in the first column of table 1. The runtime increases slightly for higher process counts. This is also expected, as the solution of the 1D model requires communication of all processes of the respective fiber.

The duration of the 3D problem increases, because all processes are involved in the solution using the GMRES solver, although the processes' subdomains are constructed as cuboids in order to make the communication surfaces as small as possible. This supports a fast solution of the 3D solver, because of the small amount of communication required between the processes. On the other hand, this partitioning increases the runtime for the 1D solver, as the elements of the 1D domains are distributed to multiple processes. However, this is acceptable, because the 3D solver takes a longer runtime.

The runtime for initialization, which includes e.g. parallel reading of the geometry files, assembly of the stiffness matrices and compilation of the CellML code, is the largest portion of the overall runtime. This is because the study only considers the runtimes for one (the first) timestep. For an enough long simulation time, the portion of initialization

Table 1: Parameters of the weak scaling study

partitioning $x \times y \times z$	fibers	fibers per process	total degrees of freedom
$2 \times 2 \times 1 = 4$	49	12.3	297,984
$3 \times 3 \times 2 = 18$	169	9.4	1,032,160
$4 \times 4 \times 4 = 64$	625	9.8	3,797,216
$5 \times 5 \times 6 = 150$	1369	9.1	8,303,280
$7 \times 8 \times 8 = 448$	4489	10.0	27,201,100
$10 \times 10 \times 12 = 1200$	11,881	9.9	72,124,720
$17 \times 17 \times 12 = 3468$	34,969	10.1	212,187,268
$18 \times 18 \times 24 = 7776$	76,729	9.9	464,686,624
$27 \times 27 \times 24 =$17,496	182,329	10.4	1,102,902,304
$34 \times 34 \times 24 =$27,744	273,529	9.9	1,656,579,616

would become negligible.

4.2 Process placement

In the next study, we investigate the effect of placing MPI processes on hardware cores in different ways. We use the same parameters as in the weak scaling study and consider the duration of solving the 1D model for two different rank placement strategies. The red curve in the results in fig. 3b shows a scenario, where for each 1D domain all processes are located on the same compute node. This strategy is also used in the weak scaling study. For the orange curve all processes are placed in a round-robin fashion. Thus, the processes that compute a fiber are all located on different nodes. Although the two strategies are almost the same for small number of cores, as the number of cores increases, the difference becomes prominent in the order of 25 times higher runtimes, if the processes for an instance of the 1D problem are no longer on the same computational node.

5 DISCUSSION

The modular structure of *opendihu* and the use of C++ class templates allows to create composite solver combinations for multi-scale problems, as demonstrated with the simulations in this paper. As the compiler knows the final structure of solvers and meshes, it can optimize for performance. This is a difference to OpenCMISS Iron, where procedural programming is used and composite solvers are only selected at runtime.

We have demonstrated that by targeting efficiency, runtimes can be reduced by a significant factor in comparison to the OpenCMISS Iron used in [7]. This enables us to solve problems of large sizes and allows for studying some effects that are not visible in small-sized scenarios.

The Finite Element meshes used for High Performance Computing have a regular structure. Therefore, subdividing the cuboids by planar cuts allows us to generate a

parallel domain decomposition with equally sized subdomains. Unlike for tetrahedral or unstructured quadrilateral grids, the neighbor information for each subdomain and element is given implicitly, which avoids time-intense lookup operations and improves memory consumption. Considering a regular grid is sufficient when simulating the belly of a fusiform muscle, like the biceps brachii. For more irregular geometries, unstructured grids have to be used, which *opendihu* supports for serial execution so far.

Our weak scaling study demonstrates that the High Performance Computing is applicable to the type of multi-physics problems considered. The considered multi-physics formulation contains numerous instances of the 0D and 1D models, which can be solved independently of each other. This allows to simulate the realistic number of fibers for biceps brachii in about 4 min runtime per millisecond simulation time, excluding initialization and file output. The study of process placement emphasizes the importance of a proper placement on the supercomputer.

6 CONCLUSION

We have presented the novel open-source software framework *opendihu* for multi-physics simulation of skeletal muscle. By employing C++ and Python codes, it combines usability and performance. For serial execution, we achieved a speedup of 38 for a particular simulation, compared to simulation software OpenCMISS.

On the supercomputer *HazelHen*, we simulated the propagation of the action potentials and resulting intramuscular EMG signals using a highly-resolved, multi-physics activation model of a realistic biceps brachii with a typical number of 273,529 muscle fibers. We have run simulations on 27,744 cores and examined the weak scaling behavior providing reasonable runtimes.

REFERENCES

[1] R. Merletti and P. Parker. *Electromyography - Physiology, Engineering, and Noninvasive Applications*. John Wiley & Sons, 2004.

[2] D. Farina, R. Merletti, and D. F. Stegeman. *Biophysics of the Generation of EMG Signals*, chapter 4, pages 81–105. Wiley-Blackwell, 2005.

[3] Luca Mesin. Volume conductor models in surface electromyography: Computational techniques. *Computers in Biology and Medicine*, 43(7):942 – 952, 2013.

[4] Nonna A. Dimitrova, Alexander G. Dimitrov, and George V. Dimitrov. Calculation of extracellular potentials produced by an inclined muscle fibre at a rectangular plate electrode. *Medical Engineering & Physics*, 21(8):583 – 588, 1999.

[5] M. M. Lowery, N. S. Stoykov, A. Taflove, and T. A. Kuiken. A multiple-layer finite-element model of the surface emg signal. *IEEE Transactions on Biomedical Engineering*, 49(5):446–454, May 2002.

[6] L. Mesin, M. Joubert, T. Hanekom, R. Merletti, and D. Farina. A finite element model for describing the effect of muscle shortening on surface emg. *IEEE Transactions on Biomedical Engineering*, 53(4):593–600, April 2006.

[7] M. Mordhorst, T. Heidlauf, and O. Röhrle. Predicting electromyographic signals under realistic conditions using a multiscale chemo-electro-mechanical finite element model. *Interface Focus*, 5(2):1–11, February 2015.

[8] Chris P. Bradley, Nehzat Emamy, Thomas Ertl, Dominik Göddeke, Andreas Hessenthaler, Thomas Klotz, Aaron Krämer, Michael Krone, Benjamin Maier, Miriam Mehl, Tobias Rau, and Oliver Röhrle. Enabling detailed, biophysics-based skeletal muscle models on hpc systems. In *Front. Physiol.*, 2018.

[9] J D MacDougall, D G Sale, Stephen Alway, and J R Sutton. Muscle fiber number in biceps brachii in bodybuilders and control subjects. *Journal of applied physiology: respiratory, environmental and exercise physiology*, 57:1399–403, 12 1984.

[10] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[11] G. Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 5(3):506–517, 1968.

[12] Hon Fai Choi and Silvia S. Blemker. Skeletal muscle fascicle arrangements can be reconstructed using a laplacian vector field simulation. *PLOS ONE*, 8(10):1–7, 10 2013.

[13] Geoffrey G. Handsfield, Bart Bolsterlee, Joshua M. Inouye, Robert D. Herbert, Thor F. Besier, and Justin W. Fernandez. Determining skeletal muscle architecture with laplacian simulations: a comparison with diffusion tensor imaging. *Biomechanics and Modeling in Mechanobiology*, 16(6):1845–1855, Dec 2017.

[14] P. R. Shorten, P. O'Callaghan, J. B. Davidson, and T. K. Soboleva. A mathematical model of fatigue in skeletal muscle force contraction. *Journal of Muscle Research and Cell Motility*, 28(6):293–313, 2007.

[15] Victor Spitzer, Michael J. Ackerman, Ann L. Scherzinger, and David Whitlock. The Visible Human Male: A Technical Report. *Journal of the American Medical Informatics Association*, 3(2):118–130, 03 1996.