Noname manuscript No.
(will be inserted by the editor)

Multi-physics multi-scale HPC simulations of skeletal muscles

Aaron Krämer · Benjamin Maier · Tobias Rau · Felix Huber · Thomas Klotz · Thomas Ertl · Dominik Göddeke · Miriam Mehl · Guido Reina · Oliver Röhrle

Received: date / Accepted: date

Abstract We present a highly scalable framework for the simulation of skeletal muscles as a neuromuscular system. Our work is based on previous implementations of a complex model coupling different physical phenomena on different temporal and spatial scales, in particular bio-chemical processes on the cellular level (as ordinary differential equations), electrical signal propagation along muscle fibers (as many one-dimensional diffusion equations), and the EMG signal in the three-dimensional muscle at organ scale. Our contribution is a new software toolbox that allows to generate simulation code suited for the execution on a supercomputer from the commonly used XML-based model-description used in the respective community. We present different variants of this code generation for CPUs, specific optimizations for our muscle model, and our in situ visualization approach that allows us to minimize data transfer between simulation, and the visualization front-end (such as the Powerwall at VISUS[1]) along with results for numerical experiments on up to approximately seven thousand cores of the CRAY XC40 system Hazel Hen[2] for more than 180,000 muscle fibers. The original implementation was limited to 4 cores. Compared to our previous work in [4], we further enhanced scalability, but in particular also node-level performance.

Aaron Krämer
Institute of Applied Analysis and Numerical Simulation
University of Stuttgart
E-mail: aaron.kraemer@ians.uni-stuttgart.de

all other authors
Institute of Applied Analysis and Numerical Simulation
Institute for Modelling and Simulation of Biomechanical Systems
Institute of Parallel and Distributed Systems
Stuttgart Center for Simulation Science
Visualization Research Center
University of Stuttgart

[1] https://www.visus.uni-stuttgart.de/
[2] https://www.hlrs.de/systems/cray-xc40-hazel-hen/

Keywords Neuromuscular system · Code generation · Equation coupling · Performance optimization · In situ visualization

1 Introduction

The human musculoskeletal system can support many different actions, from applying strong forces to very tactile movements. Even ‘simple’ tasks like grabbing an object involve highly coordinated actions, i.e., voluntary contractions of skeletal muscles. Understanding the underlying mechanism of neuromuscular control is challenging and subject to active research. Besides obtaining new insights into the basic understanding of the neuromuscular system itself, a detailed understanding paves the way to many highly beneficial applications, including but not limited to personalized rehabilitation, patient-specific prosthesis design, passenger safety in crash scenarios, surgical planning tools, human-computer interfaces, ergonomic assessment of the workplace and the design of assistive tools. The desired degree of detail and complexity within models for (parts of) the neuro-musculoskeletal system requires the coupling of different physical phenomena on different temporal and spatial scales, e.g., models describing the mechanical or electrical state of the muscle tissue on the organ scale and the bio-chemical processes on the cellular scale (cf. Sect. 2.1). One of the few non-invasive and clinically available diagnostic tools to obtain insights into the functioning (or disfunctioning) of the neuromuscular system are electromyographic (EMG) recordings. They measure the activation-induced potentials on the skin surface, and rely on the direct connection between the discharge of spinal motorneurons and the resulting action potentials of the associated muscle fibers. Due to the high complexity of the neuromuscular system, conclusions from the EMG signals to the behavior of the motorneurons are notoriously difficult, in particular, as methods to decode the activation mechanisms cannot be fully validated based on experimental data.

Using synthetically generated simulation data can provide important contributions to the analysis of EMG signals. However, being able to take into account all these different processes on different scales requires a flexible multi-scale, multi-physics computational framework and significant compute power. Over the past years, we have worked towards this goal, see [4] for intermediate results that we update here. We emphasize that only the joint development of model improvements, changes to the mathematical algorithms, suitable HPC techniques and, often forgotten, appropriate visualization techniques to generate a feedback loop with the domain scientist can lead to true progress.

In this chapter, we describe and analyze the HPC-related improvements we achieved, both in terms of runtime and scalability, but also in terms of an in situ visualization on Hazel Hen, the Tier-0 system installed at the HLRS in Stuttgart, that peaked at position #8 in the TOP500 in November 2015. In particular, we demonstrate techniques that enable the efficient simulation of muscles comprising a realistic number of 100,000s of muscle fibers. The remainder of this chapter is organized as follows: In Sect. 2 we present the multi-scale model and the associated numerical schemes. Sect. 3 covers the implementation and the optimizations we framed. Results are discussed in Sect. 4, and we conclude with a summary of the most important findings and an outlook to future work in Sect. 5.

2 Model and Discretization

A skeletal muscle consists of millions of sarcomeres which are arranged in series as a myofibril. Many myofibrils arranged in parallel then form a muscle fiber. About 10.000 to 1.000.000 muscle fibers form the active tissue of a muscle, depending on the type of muscle. Muscle fibers are much longer than wide. Also, action potentials which initiate the force generation in selected fibers only spread along these fibers, not transversely. Thus, all sarcomeres across a muscle fiber behave similarly, and we can represent all sarcomeres of such a muscle fibers' cross section as a zero-dimensional *subcell*. This is a common procedure in skeletal muscle modelling. In addition to a diffusive spreading of the action potential, there is a reactive subcellular behavior which can have both a supporting and a suppressive effect on the action potential propagation.

2.1 Model Formulation

Our muscle model describes the evolution of intra-cellular bio-chemical compositions, the propagation of action potentials and EMG signals. Action potentials are treated by modelling the physical quantity of transmembrane potential. A detailed description of the underlying bio-physical view on this subject is given in [15]. The interplay of the three components — bio-chemical compositions $\boldsymbol{y}$, transmembrane potentials V_m, and extracellular potential ϕ_e — is governed by a multi-scale coupling of the following differential equations, each on different domains:

$$\frac{\partial \boldsymbol{y}}{\partial t} = G\left(\boldsymbol{y}, V_\mathrm{m}, I_\mathrm{stim}\right) \qquad \text{on } \Omega_s^i, \qquad (1)$$

$$\frac{\partial V_\mathrm{m}}{\partial t} = \frac{1}{A C_\mathrm{m}}\left(\frac{\partial}{\partial x}\left(\sigma_\mathrm{eff}\,\frac{\partial V_\mathrm{m}}{\partial x}\right) - A I_\mathrm{ion}\big(\boldsymbol{y}, V_\mathrm{m}, I_\mathrm{stim}\big)\right) \qquad \text{on } \Omega_f^j, \qquad (2)$$

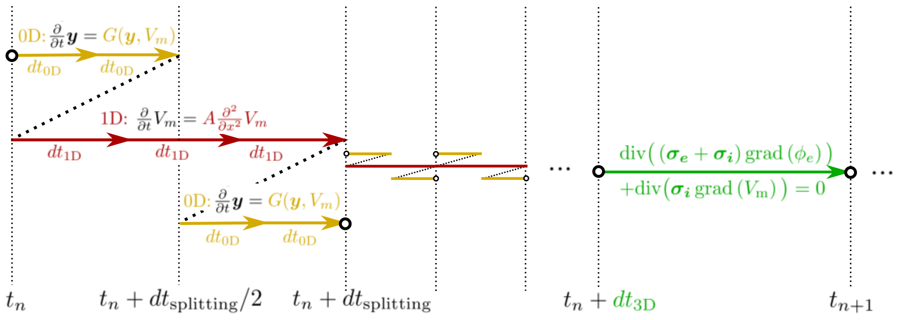
$$0 = \operatorname{div}\Big(\left(\boldsymbol{\sigma_e} + \boldsymbol{\sigma_i}\right)\operatorname{grad}\left(\phi_e\right)\Big) + \operatorname{div}\Big(\boldsymbol{\sigma_i}\operatorname{grad}\left(V_\mathrm{m}\right)\Big) \qquad \text{on } \Omega_m. \qquad (3)$$

Here, Ω_s^i, $i = 1, \dots, N_s$ are locations of N_s zero-dimensional subcells. Ω_f^j with $j = 1, \dots, N_f$ are the one-dimensional representations of muscle fibers, and Ω_m is the three-dimensional muscle domain. It holds $\Omega_s^i \subsetneq \Omega_f^j$ for any subcell belonging to muscle fiber j, and $\Omega_f^j \subsetneq \Omega_m$ for any muscle fiber.

Eqn. (1) was originally formulated by Hodgkin and Huxley [12]. We refer to their article for a more detailed presentation of the right-hand side G. G is a nonlinear function, depending on the subcellular state $\boldsymbol{y}$, on the muscle fibers' transmembrane potential V_m, and the stimulation current I_stim stemming from one of many motor units of the central nervous system. Eqn. (1) forms a closing condition for the *Monodomain Equation* (2), which describes the evolution of the transmembrane potential V_m along a muscle fiber. The muscle fibers' surface to volume ratio is called A, C_m is the capacitance of its membrane, σ_eff its effective conductivity. The ionic current passing across the membrane is described by I_ion. By Eqn. (3) a description of the extracellular potential ϕ_e is given throughout the three-dimensional tissue. We refer to it as *stationary Bidomain Equation*. The conductivity tensors in the extra- and intra-cellular domains are denoted as $\boldsymbol{\sigma_e}$ and $\boldsymbol{\sigma_i}$. The solution variable of Eqn. (3), ϕ_e, corresponds to EMG signals over time at any spatial point of interest.

2.2 Discretization and Solution

The discretization and solution of Eqns. (1)–(3) follows the method described in [4] and will be summarized briefly in the following.



0D: $\frac{\partial}{\partial t}\boldsymbol{y} = G(\boldsymbol{y}, V_m)$
dt_0D
dt_0D
1D: $\frac{\partial}{\partial t}V_m = A\frac{\partial^2}{\partial x^2}V_m$
dt_1D
dt_1D
dt_1D
0D: $\frac{\partial}{\partial t}\boldsymbol{y} = G(\boldsymbol{y}, V_m)$
dt_0D
dt_0D
$\mathrm{div}\big((\boldsymbol{\sigma_e} + \boldsymbol{\sigma_i})\,\mathrm{grad}\,(\phi_e)\big)$
$+\mathrm{div}\big(\boldsymbol{\sigma_i}\,\mathrm{grad}\,(V_\mathrm{m})\big) = 0$
$\cdots$
$\cdots$
t_n
$t_n + dt_\mathrm{splitting}/2$
$t_n + dt_\mathrm{splitting}$
$t_n + dt_\mathrm{3D}$
t_{n+1}

Fig. 1 Overall time stepping for the solution of the system of Eqns. (1)–(3). The diagram shows a sequence of Strang splitting steps for Eqns. (1) (yellow) and (2) (red) followed by a solution of Eqn. (3) (green). Since Eqn. (3) is stationary, it holds $t_n + dt_\mathrm{3D} = t_{n+1}$. The complete sequence is called a global time step.

We discretize the spatial operators of the Monodomain Equations (2), by one-dimensional (1D) Finite Elements with linear ansatz functions, yielding 1D muscle fiber meshes. We use three-dimensional Finite Elements with linear ansatz functions to discretize the stationary Bidomain Equation (3) in the muscle volume Ω_m. Multiple 1D muscle fiber meshes are embedded in the 3D domain of the muscle volume. The value of the transmembrane potential V_m, is linearly interpolated and mapped between the 1D and the 3D meshes discretizing Ω_f^j and Ω_m.

The time discretization is visualized in Fig. 1. We apply a Strang operator splitting with time step $dt_\mathrm{splitting}$ to the Monodomain system, Eqns. (1) and (2), and, thus, solve the diffusion (1D) and reaction (0D) terms alternatingly. For the reaction term, we use (multiple) time steps of Heun's method with time step dt_0D. For the diffusion equation, we use (multiple) time steps of a second order consistent Crank-Nicolson scheme with time step dt_{1D}. We couple the stationary Bidomain Equation (3) uni-directionally to the Monodomain system, Eqns. (1) and (2). After the sequence of Strang splitting steps the coupling variable V_{m} gets transferred from Ω_f^j to Ω_m. By solving the stationary Bidomain Equation (3), a new approximation for the extracellular potential ϕ_e is found, which represents the current EMG signal.

3 Implementation and Optimizations

We solve the model in our open-source software framework *opendihu*[3]. It allows to compose a nested structure of solvers and numerical schemes, and thus serves as a playground tool to identify optimal schemes for the problem at hand. It consists of a core C++ library that builds on various data management and solver packages, such as MPI, ADIOS2[4], PETSc [2], MUMPS [1] and Hypre [5].

Formulated models, e.g., for subcellular kinetics, can be integrated using the community standard CellML [8], an XML based description. A simulation program for a specific model comprises compile-time and runtime code. A short C++ main file defines the graph of solvers by means of C++ templates at compile time, whereas at runtime, a Python script gets interpreted. It defines parameters and callback functions to modify them during the simulation. More details on the software structure are available in a previous publication [14].

The solution of the governing equations is suited for parallel computation, as has been discussed more detailed in [14]. The Monodomain Equations on all the muscle fibers are independent of each other. Furthermore, all instances of the subcellular Eqn. (1) are independent of each other and can be computed concurrently. Only the computation on the 3D domain, Eqn. (3), involves information in the whole domain Ω_m. We partition the computational domain into disjoint subdomains for the available processes. As we use a structured 3D mesh, this can be done easily by defining appropriate index sets in x, y and z direction. The 1D muscle fiber meshes for Ω_f^j are partitioned in the same way as the 3D mesh for Ω_m, such that every process owns a distinct part of the overall spatial domain.

In this section, we focus on the compute-node level performance of the Monodomain system, Eqns. (1) and (2), as well as efficient in situ visualization.

3.1 CellML

CellML is an XML based description, established in the bioengineering community for subcellular models. Any CellML model abstracts the right hand

[3] https://github.com/maierbn/opendihu/
[4] https://github.com/ornladios/ADIOS2

side of a differential-algebraic equation by formulating the following function:

$$\begin{pmatrix} \texttt{rates} \\ \texttt{algebraics} \end{pmatrix} = \texttt{cellml}\,(\texttt{states}, \texttt{constants})\,. \tag{4}$$

The input consists of the vector of `constants` as well as the vector of `states` for which the corresponding vector of `rates` are computed. The vector of `algebraics` contains additional output values. Each scalar has a unique name in the CellML description, by which it is also identified by *opendihu*.

Various tools exist to manipulate and solve CellML models. A programming interface (API) as well as the CellML website provide means to use CellML models in various programming languages such as C, MATLAB and Python. The open-source tool OpenCOR [9] can be used to edit, simulate and analyze such models in a user-friendly graphical user interface. It focuses on single-cell models and, thus, is not able to simulate the Monodomain Equation. In the domain of cardiac electrophysiology simulations, Chaste [7] is a popular software framework. It provides a utility to convert CellML models for the use in the framework. A tool with similar purpose is Myokit [6]. Another software environment is OpenCMISS [3] which also can be used to simulate the Monodomain Equation.

Our solution differs from existing tools in that it is targeted at skeletal muscle simulations at high resolution on supercomputers. Through the tight integration in *opendihu*, specific optimizations are possible such as explicit vectorization. In our system comprising Eqns. (1) and (2), the `states` vector contains the transmembrane potential V_m as well as the subcellular state $\boldsymbol{y}$. All parameters such as the stimulation current I_stim are part of the `constants` vector. The `algebraics` part is empty for the present model of EMG simulation. It contains values in other scenarios, e.g., when muscle contraction is considered. If required, our tool is also capable of handling `algebraics`. Fig. 2 summarizes the steps starting from a given CellML file to solving Eqns. (1) and (2) carried out by our software *opendihu*.

3.2 General Optimizations of the Code Generator

As shown in Fig. 2, a code generator produces optimized C code from a CellML problem description. The naive way to solve all the instances of a CellML problem on a process required for Eqn. (2) leads to storing the state vectors in an Array-of-Struct (AoS) memory layout, where all state variables for a 0D unit or a 1D mesh point are close in memory. This approach is used, e.g., in OpenCMISS [3].

Using a Struct-of-Array (SoA) memory layout instead, where all instances of a state variable are contiguously stored, and additionally modifying the iteration loops such that the outer loop is over instances and the inner loop over the states allows cache-efficient memory access and to make use of the single-instruction multiple-data (SIMD) paradigm. We provide two specific types of code generation, `"comp"` and `"vc"`. Both make use of the SIMD paradigm. The

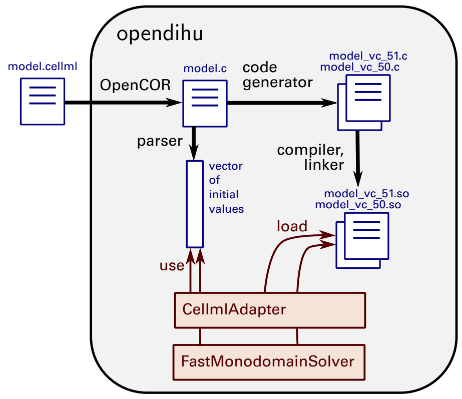


Fig. 2 Schematic representation of information flow for the CellML subsystem of *opendihu*. First, the command line interface of OpenCOR is used to convert the model to a C code file. The initial values for all states get parsed from this file and will be used later to initialize the state vector. Additionally, all compute instructions are parsed into an internal syntax tree. Then, the code generator produces optimized C code that can solve as many independent instances of the CellML model as are assigned to each MPI rank by the global domain decomposition.

`comp`-code generator relies on the compilers capability of auto-vectorization by employing the available instruction set, e.g., leading to simultaneous computations of 4 double values with AVX2 or 8 double values with AVX-512. The Hazel Hen system uses the AVX2 instruction set with 4 double values.

Experiments show that relying on the compilers auto-vectorization does not lead to optimal results. I.e., for a system with AVX2, our measurements show that the runtime decreases approximately by a factor of two. Therefore, we additionally implement explicit vectorization within the `vc`-code generator. We use the C++ library *Vc*[5] which abstracts SIMD instructions and compiles code that uses the respective instruction sets. The used memory layout for this implementation is Array-of-Vectorized-Struct (AoVS) and the outer loop iterates over the CellML instances in groups according to the vector register length. Where the computation in the CellML model branches, i.e., for piecewise-defined functions, all cases have to be computed and the inactive values have to be masked out.

Profiling the code for different CellML models shows that about half the time is spent in evaluating the exponential function. Therefore we approximate

$$\exp(x) \approx \exp^*(x) = \left(1 + \frac{x}{n}\right)^n, \qquad n = 1024 = 2^{10}.$$

This computation can be performed using only 1 addition and 11 multiplications. In the considered CellML models, the values for x are bounded by $|x| < x_{\max} = 12$, for which the relative error is $|(\exp^* - \exp)(x_{\max})/\exp(x_{\max})| < 0.07$. Furthermore, we found that potentiation only occurs with exponents

[5] https://vcdevel.github.io/Vc-1.4.1

that are whole numbers. We replace the generic `pow` function by a more efficient, recursive implementation.

3.3 Specific Optimizations for the Monodomain Equation

Further specific performance improvements in *opendihu* are employed within the `FastMonodomainSolver` class. As shown in Fig. 2, the class also uses the code generator and replaces the `CellmlAdapter`. The implementation exploits the particular structure of the Monodomain Equation.

When using the general `CellmlAdapter`, the solution of the 1D diffusion problem in Eqn. (2) is done using a parallel conjugate gradient (CG) solver of PETSc, which requires costly communication between processes. The `FastMonodomainSolver` exploits the fact that, for the whole muscle, we solve many 1D problems (one for each muscle fiber), each of them small enough to be solved sequentially with the simple linear complexity Thomas' algorithm. Since the 3D problem requires a three-dimensional domain decomposition as described in [14], all data required to solve the 0D-1D system for a given fiber have to be first communicated to one particular MPI rank (within a node). This rank then performs the 0D/1D computations serially for a time span of dt_{3D}. Afterwards, all data are communicated back to the original location, as needed for the subsequent computation of the 3D model. The selection of the process to do the respective computation for a given fiber follows a round-robin fashion such that all processes receive the same amount of workload.

The code for this computation is again generated by the code generator using the *Vc* library allowing arbitrary subcellular CellML models. Thus, the whole algorithm uses vector instructions and avoids unnecessary data copies. On the software side, this 'shortcut' algorithm partially specializes the same C++ class templates as the baseline version and therefore integrates well with the encompassing coupling scheme to the stationary Bidomain Equation. Also the same Python settings file can be used for both variants. This means that the change between the two schemes appears user-friendly in one line of code.

3.4 In situ Visualization

As the large volume of simulation output in our envisaged research questions poses a challenge for a posteriori analysis by domain experts (see Sect. 1), we implemented and tested an in situ visualization setup. Besides reducing the data output, our setup has the capability to produce the image data on- or off-site as well as connecting the visualization to the VISUS Powerwall for interactive exploration of the high resolution visualization while the simulation is still running. We focus on the on-site visualization.

This means that one visualization instance is spawned on the same nodes the simulation is running on. The visualization instance collects all data produced on this node and renders a part of the final image. All partial images are then depth-correctly composited using IceT [16]. The nonexistence of GPUs on Hazel Hen made it necessary to rely on the CPUs for rendering. Thus, we implemented a new pipeline into the visualization framework MegaMol [11] that uses the ray tracing engine OSPRay [20] for rendering [17]. The original OpenGL-based rendering in MegaMol is performed in the corresponding renderer modules and implicitly composited via the default framebuffer. The new CPU rendering path processes data in the geometry modules and gathers all data in a single rendering module that owns the complete scene graph and therefore is able trigger a single render, global pass. Using ray tracing or optionally path tracing, global lighting and shading are available. These novel geometry modules can also be daisy-chained to easily compose complex scenes using a normal MegaMol project graph. However, in a distributed rendering scenario each instance of the visualization still generates a partial image from local data, which makes a composition step necessary to obtain the final image.

In general, we prefer the loosely-coupled in situ scheme [13], because this scheme increases the robustness of the joint simulation and visualization. Therefore, they do not run in the same MPI world and just communicate via small notification messages. However, this kind of scheme could not be realized on Hazel Hen, because executing two different programs on the same nodes is deactivated. As a fallback solution we run the setup in a single MPI world, where the simulation starts a visualization instance on each node (see Fig. 3). During initialization *opendihu* requests a connection with the local MegaMol instance via ZeroMQ [6] and checks if MegaMol has already initialized the module graph and is therefore ready for rendering. Depending on the communication pattern, each instance of *opendihu* that runs on the same node has to register at the MegaMol instance.

The connection uses MegaMol's built-in LUA interface [10] for notification and acknowledgments. After that, *opendihu* performs several simulation steps and writes data to the shared memory of the node. Upon finishing this, MegaMol gets notified with the explicit virtual 'file name'. MegaMol then maps the shared memory and processes the data for rendering. Note that a time step is synchronized in MegaMol, which means that MegaMol uses the information how many *opendihu* instances run on a node to synchronize changing the displayed data to the latest complete simulation step.

The file format used for all transactions is based on bp-files as written by ADIOS2 [7]. This offers a lot of flexibility to the setup and also enables parallel I/O. For example, the simulation can either handle every *opendihu* instance on a node independently or write a single file and limit the communication with MegaMol to a single instance per node. This reduces communication overhead, avoids potential connection issues and reduces the module graph overhead of MegaMol.

Connecting from anywhere to the head node of the visualization with another MegaMol instance starts the image transfer to the remotely connected

[6] https://zeromq.org/

[7] https://github.com/ornladios/ADIOS2

instance. The on-site and off-site MegaMol instances also communicate via the LUA interface, enabling the user to interact with the visualization that is running on-site and transporting only images off-site.

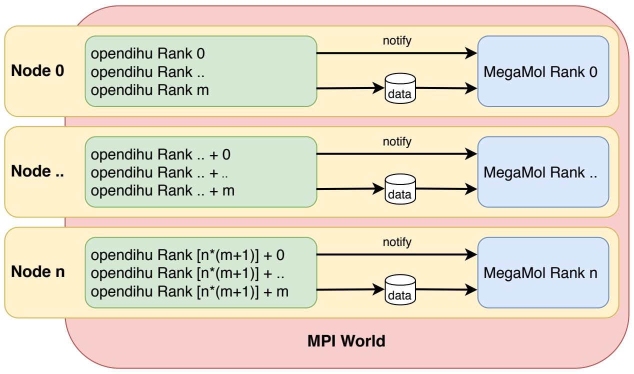


Fig. 3 Schematic of the in situ coupling. On each node a visualization instance is started and all *opendihu* processes communicate availability of their data once that is written into shared memory. The communication is a short message about new data from one or multiple simulation processes. To avoid rendering mixed images of old and new data, the visualization puts the new data on hold until all processes have signaled the availability of new data. Additionally, a connected client can send visualization parameters such as camera manipulation commands to the on-site visualization application, which is used for an interactive exploration of the data.

4 Results and Discussion

In our evaluation, we use a realistic geometry of a Biceps Brachii muscle, which was extracted from the Visible Man dataset [19]. Fig. 4 shows a simulation result with colored electric transmembrane potential V_m on multiple muscle fibers. Different improvements are analyzed in detail in the following.

The performance measurements are carried out on the supercomputer Hazel Hen at HLRS in Stuttgart. Per compute node, it contains two Intel Haswell E5-2680v3 CPUs at 2.5 GHz with 24 cores.

4.1 Choice of Compiler

At first, we investigate the auto-vectorization performance of the GNU, Intel and Cray compilers, and vary optimization levels both for the program and in the `CellmlAdapter`. We compute the system of Eqns. (1) and (2) with the subcellular model of Shorten et al. [18] for one muscle fiber with $n_p = 2400$ nodes, time step widths $dt_\mathrm{0D} = 10^{-3}ms, dt_\mathrm{1D} = dt_\mathrm{splitting} = 3 \cdot 10^{-3}ms$ and end time $t = 20ms$. 24 processes are used on one compute node of Hazel Hen. The total runtimes for the 0D and 1D parts are depicted in Fig. 5.

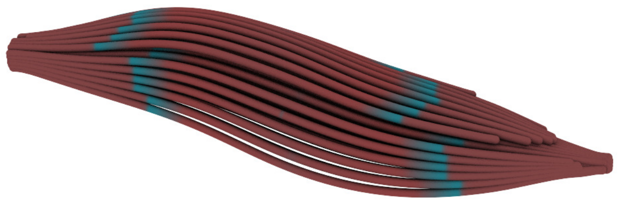


Fig. 4 Simulation result for multiple muscle fibers. The color represents the value of the electric transmembrane potential V_m.

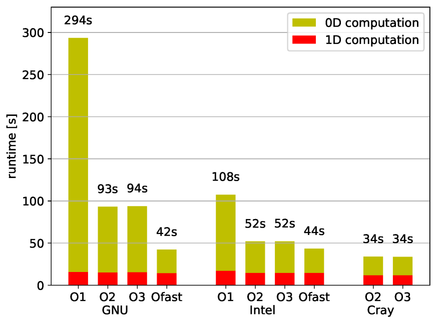


Fig. 5 Runtime of the 0D and 1D models with different compilers and optimization levels. The data is averaged over 120 runs.

It can be seen, that for all compilers computation times for the 0D model decrease if the optimization level is increased from `O1` to `O2` or `O3` and further to `Ofast`. The optimization levels `O2` and `O3` behave similarly for the given program. This shows that vectorization, which is already enabled by the `O2` flag, contributes most to the decrease in runtimes. For the same reason, the computation of the 1D model does not profit from a higher optimization level of the compiler. When using `Ofast`, compliance regarding the IEEE and ISO rules for floating-point arithmetic and math functions gets disregarded, which means for example that infinite math is not supported. This is not an issue for our simulation code. As can be seen in the measurement of the GNU and Intel compilers, this optimization level yields a further increase in performance with more than 2 times faster code for the GNU compiler. Comparing the compilers, it can be seen, that, for the same optimization level, the Cray compiler produces faster code than the Intel compiler, which in turn produces faster code than the GNU compiler. The best result is achieved with the Cray compiler and the `O3` flag. However, the compile time increases to over 2 hours compared to approximately 10 minutes with the other two compilers.

4.2 Monodomain Solver Improvements

In a small scale preliminary study, we measure the performance of the code generator and its explicit vectorization. We solve the Monodomain system, Eqns. (1) and (2), using the `FastMonodomainSolver` with the `vc`-code generator on the same scenario as in 4.1. We use two processes on an Intel Skylake I5-6300U dual-core processor with 2.4 GHz base frequency, which has hardware counters for scalar and vectorized floating point operations. During computation, we measure a performance of 19.6 GFlops which corresponds to 25.5% of the theoretical peak performance of 76.8 GFlops for this processor. This is an indication of good performance and results from the explicit SIMD instructions employed by the code generator.

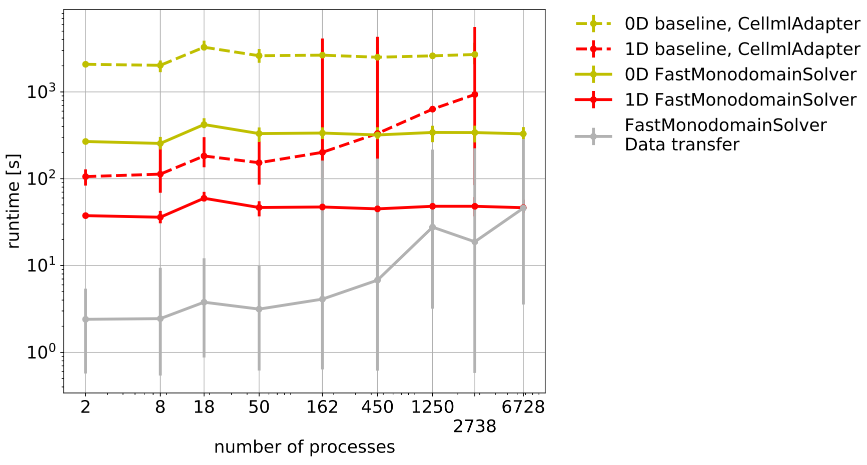


Fig. 6 Comparison of runtimes for solving the Monodomain system, Eqns. (1) and (2), using the `CellmlAdapter` (dashed lines) and the `FastMonodomainSolver` (solid lines). Depicted is the runtime in a weak scaling setup as well as indications for minimum and maximum values over all processes for each data point.

The next study compares weak scaling runtimes for the Monodomain system solved with the `CellmlAdapter` and the `FastMonodomainSolver` with the subcellular model of Hodgkin-Huxley [12]. The results are depicted in Fig. 6. The number of processes increases from 2 to 6728, and the number of muscle fibers from 49 to 182,329. The spatial resolution within the fibers stays constant, with a number of $n_{1\text{D}} = 1480$ elements per muscle fiber. We solve the complete model including the EMG, which requires communication of the respective quantities between the 3D and 0D/1D domains. For the sake of this study, only the runtimes of the 0D-1D solution as well as the additional time for communication in the `FastMonodomainSolver` are shown. The baseline solvers (dashed lines in Fig. 6) use the `CellmlAdapter` with the `vc`-code generator and a parallel CG solver of PETSc for the 1D diffusion along fibers. The improved method (solid lines) uses the `FastMonodomainSolver`.

We use time step widths of $dt_{0\mathrm{D}} = 10^{-3}ms$, $dt_{1\mathrm{D}} = 2 \cdot 10^{-3}ms$, $dt_{\mathrm{splitting}} = 2 \cdot 10^{-3}ms$ and $dt_{3\mathrm{D}} = 10^{-1}ms$ and simulate one and two time steps of the 3D and 0D-1D problems, respectively. To obtain the correct relation of these runtimes, we consider a simulation end time of $10^{-1}ms$ and scale the measured durations according to the number of time steps in this simulation time span.

The results show a sharp increase in runtime for the 1D baseline solver which is due to increased communication. The computation of the 0D subcellular model which involves no communication is constant for both schemes. Compared to the baseline, the improved algorithm shows a runtime speedup of approximately 8 for the 0D solver and between 2 and 19 for the 1D solver.

Additionally, the 1D baseline solver shows large variations between minimum and maximum solution times on different processors. The wall time of the simulation would correspond to the maximum solution times which contributes to an even larger speedup for our new solver. For the improved solver, these variations are effectively eliminated. The data transfer time is also part of the runtime for the new solver, but it is negligible as it occurs only twice in the large time interval of $dt_{3\mathrm{D}}$.

4.3 Bidomain Solver Improvements

The weak scaling of the CG solver for the 3D stationary Bidomain Equation (3) has also been improved by updating the PETSc version used by *opendihu*. We mainly profit from an improved communication implementation in the newer PETSc version which affects the CG solver performance. Fig. 7 shows the runtime of the CG solver on Hazel Hen for a fixed number of solver steps. The number of unknowns per process is relatively small, so that the runtime is dominated by communication time. With the new PETSc version, the increase in runtime with larger number of processes that initially followed approximately a square root function is replaced by the expected logarithmic growth.

4.4 Weak Scaling of the Overall Model

Taking into account all improvements, in Fig. 8 we present weak scaling results with the same setting as in the weak scaling study in Sect. 4.2. It shows the relation of runtimes for the 0D, 1D and 3D models as well as initialization time for a realistic simulation with end time $1s$. For the CG solver for the 3D problem, we prescribe a fixed number of 10^4 iterations. The 3D time step width of $dt_{3\mathrm{D}} = 10^{-1}ms$ corresponds to a sampling frequency of a simulated EMG measurement device of 10kHz.

It can be seen that the 0D solver consumes most of the runtime, despite extensive use of SIMD instructions, whereas the 3D solver has the lowest

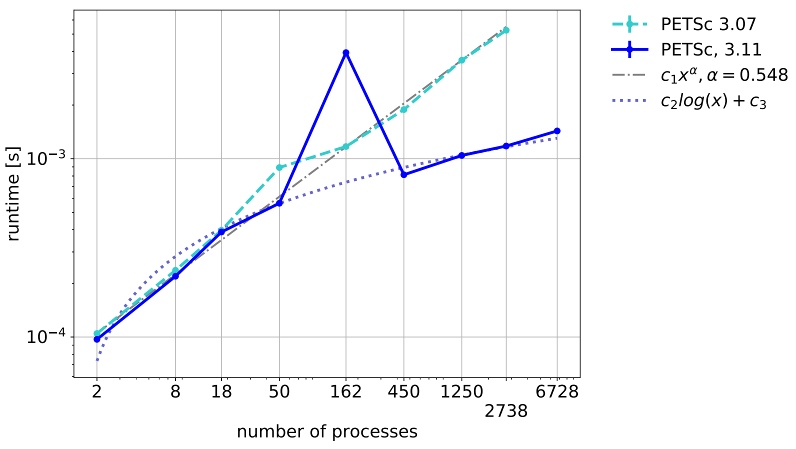


Fig. 7 Improvements in the CG solver due to newer PETSc version. Runtimes are shown for 3 CG iterations on Hazel Hen with approximately 248 unknowns per process. Each process is assigned to an unique core where all cores in a node are utilized. The peak at 162 processes is a reproducible outlier which we currently cannot attribute to a specific reason. To illustrate the different scaling behavior the dashed reference lines show fitted algebraic and logarithmic functions.

runtimes. The duration for initialization which mainly consists of reading the geometry data from input files increases with higher problem sizes, but remains significantly lower than the 0D and 1D solvers.

For the 3D solver, large spans between minimum and maximum runtimes can be seen, which are due to a very small number of processes that receive a remainder subdomain that is smaller than the average. In total, the 0D, 1D and 3D components of the simulation exhibit good weak scaling properties.

4.5 In situ Visualization

As a first step, we improved the I/O of the simulation to be compatible with the visualization framework and improve the parallel I/O performance. We simulate 1,369 fiber meshes and use 13,200 nodes in the 3D EMG mesh using 64 processes on an Intel Xeon E7-8880 v3 processor with 72 physical cores and SSD. The 3D mesh is written in parallel and at 40 time steps using the output file formats *Exfile*, *VTK* and *bp*. The *Exfile* format is a text-based format used by the OpenCMISS community. VTK is a base64-encoded, convenient format that can be visualized, e.g., by the program ParaView. The *bp* format of ADIOS2 is optimized for parallel output.

The total durations of writing the output and the resulting file sizes are summarized in Table 1. It can be seen that ADIOS2 clearly outperforms the other two formats both in file size and runtime. In the in situ scenario, where

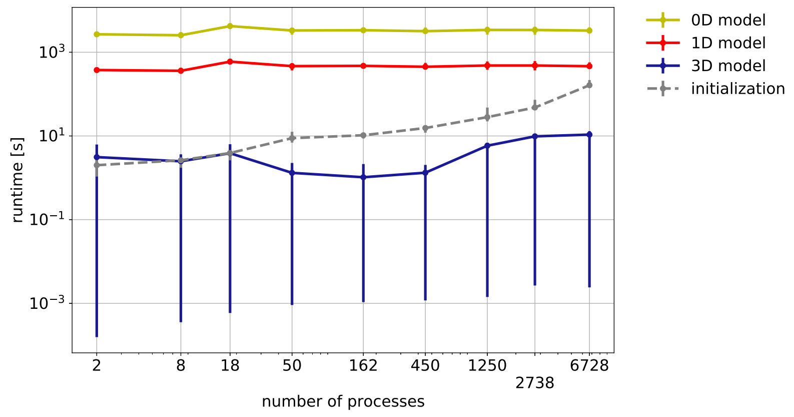


Fig. 8 Weak scaling results, runtimes of components of the simulation scaled to match a simulation time of 1*s*. The mean runtime of the 0D,1D and 3D solvers as well as the duration for initial loading of the fiber geometry is shown, with bars indicating minimum and maximum over all processes.

Table 1 Measurement of the I/O performance using three different file writers available in *opendihu*.

Format	File size [Mb]	output duration [s]
Exfile[8]	8800	311.06
VTK[9]	65	1.3
bp (ADIOS2)	33	0.68

data is only communicated via shared memory in our loosely-coupled setting (see Sect. 3.4), the output time is expected to be even shorter.

Previous setups show that MegaMol runs stable on 128 nodes, more nodes have not been tested yet. However, on Hazel Hen only single-node proof-of-concept runs have been conducted. To test the setup for the scale of a full-sized simulation, we have to prepare further tests. While waiting for availability of the new supercomputer Hawk[10], we recreated a small-scale setup with currently available hardware and adopted several new features from current MegaMol development, for example new interaction and abstraction patterns for camera control, a completely overhauled remote visualization component and a significantly extended ADIOS2 integration. Furthermore, the internal OSPRay data structures and internal communication patterns have been significantly improved. These features have been developed in parallel, but have not been tested at scale on an HPC system yet. In Fig. 9 a basic visualization is shown, while running in situ with the simulation. The small-scale setup consists of two

[8] http://opencmiss.org/
[9] https://vtk.org/
[10] https://www.hlrs.de/systems/hpe-apollo-hawk/

Intel Xeon Phi 7210 that run the simulation and the visualization on-site. The memory that the simulation maps for rendering depends completely on the system size, the parallelization of the simulation and the number of buffered time steps. The visualization framework's memory overhead varies slightly, depending on the framebuffer size, the number of ranks because of the image composition and the communication pattern. In this particular case, we measured a memory overhead of 660 Mb for the on-site running visualization using a full HD resolution and a single communication channel per node. We consider this to be reasonable to do an on-site visualization, since we also expect no significant deviations at a larger scale. However, for optimization we need to conduct further tests on the new system.

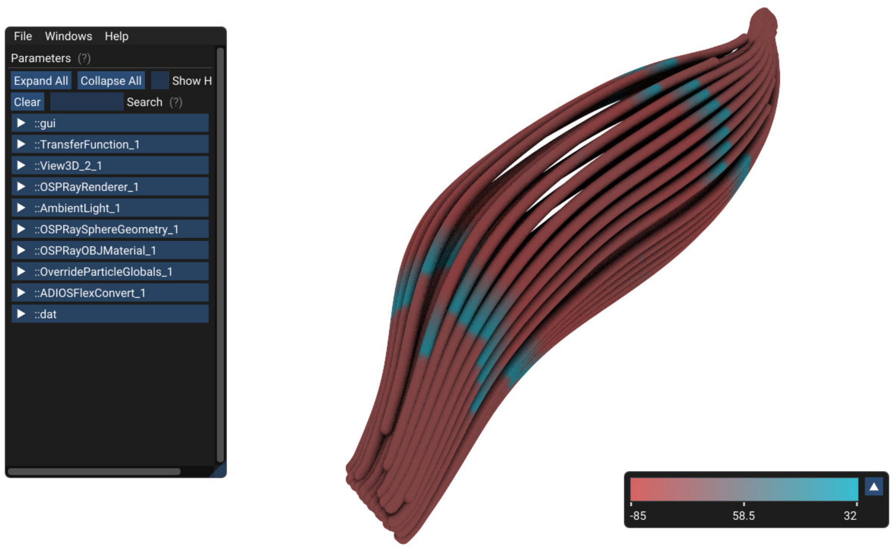


Fig. 9 In situ visualization of muscle fibers in the proof-of-concept in situ test case. In this case, we use a built-in *opendihu* simulation scenario with multiple muscle fibers and the mapped electric transmembrane potential, V_m. The simulation is connected to the visualization framework MegaMol.

5 Conclusion

We have presented scalability results for our new implementation of a neuromuscular system simulation. In particular, we could improve runtimes by up to a factor of 25 compared to previous results by optimal choices of compilers, optimized code generation exploiting AVX vectorization as well as the particular structure of the system of equations we are using. We enhanced the simulation by a communication-minimizing in situ visualization with MegaMol. Some results are preliminary due to the substantially reduced availability of Hazel

Hen and Hawk during the transition to the new system Hawk throughout the last six months. Since the HLRS systems have become available again shortly before the submission deadline, we will perform full runs and include performance results and discussion for the camera-ready deadline. In particular, we will verify stability and performance of different communication patterns in the in situ visualization approach at scale on Hawk and make measurements of the performance impact on the simulation and the additional memory consumption of the on-site visualization.

Conflict of interest

The authors declare that they have no conflict of interest.

Acknowledgements We thank the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) for supporting this work by funding - EXC2075 – 390740016 under Germany's Excellence Strategy. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech). Further funded by the International Research Training Group "Soft Tissue Robotics" (GRK 2198/1).

References

1. Amestoy, P., Buttari, A., L'Excellent, J.Y., Mary, T.: On the complexity of the block low-rank multifrontal factorization. SIAM Journal on Scientific Computing **39**(4), 1710–1740 (2017). DOI 16M1077192
2. Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F.: Efficient management of parallelism in object oriented numerical software libraries. In: E. Arge, A.M. Bruaset, H.P. Langtangen (eds.) Modern Software Tools in Scientific Computing, pp. 163–202. Birkhäuser Press (1997)
3. Bradley, C., Bowery, A., Britten, R., Budelmann, V., Camara, O., Christie, R., Cookson, A., Frangi, A.F., Gamage, T.B., Heidlauf, T., Krittian, S., Ladd, D., Little, C., Mithraratne, K., Nash, M., Nickerson, D., Nielsen, P., Nordbo, O., Omholt, S., Pashaei, A., Paterson, D., Rajagopal, V., Reeve, A., Röhrle, O., Safaei, S., Sebastián, R., Steghöfer, M., Wu, T., Yu, T., Zhang, H., Hunter, P.: OpenCMISS: A multi-physics & multi-scale computational infrastructure for the VPH/Physiome project. Progress in Biophysics and Molecular Biology **107**(1), 32–47 (2011). DOI 10.1016/j.pbiomolbio.2011.06.015
4. Bradley, C.P., Emamy, N., Ertl, T., Göddeke, D., Hessenthaler, A., Klotz, T., Krämer, A., Krone, M., Maier, B., Mehl, M., Tobias, R., Röhrle, O.: Enabling detailed, biophysics-based skeletal muscle models on HPC systems. Frontiers in Physiology **9**(816) (2018). DOI 10.3389/fphys.2018.00816
5. Brannick, J., Cao, F., Kahl, K., Falgout, R.D., Hu, X.: Optimal interpolation and compatible relaxation in classical algebraic multigrid. SIAM Journal on Scientific Computing **40**(3), A1473–A1493 (2018). DOI 10.1137/17M1123456
6. Clerx, M., Collins, P., [de Lange], E., Volders, P.G.: Myokit: A simple interface to cardiac cellular electrophysiology. Progress in Biophysics and Molecular Biology **120**(1), 100–114 (2016). DOI 10.1016/j.pbiomolbio.2015.12.008
7. Cooper, J., Spiteri, R.J., Mirams, G.R.: Cellular cardiac electrophysiology modeling with Chaste and CellML. Frontiers in Physiology **5**, 511 (2015). DOI 10.3389/fphys.2014.00511
8. Cuellar, A.A., Lloyd, C.M., Nielsen, P.F., Bullivant, D.P., Nickerson, D.P., Hunter, P.J.: An overview of CellML 1.1, a biological model description language. SIMULATION **79**(12), 740–747 (2003). DOI 10.1177/0037549703040939

9. Garny, A., Hunter, P.J.: OpenCOR: a modular and interoperable approach to computational biology. Frontiers in Physiology **6**, 26 (2015). DOI 10.3389/fphys.2015.00026
10. Gralka, P., Becher, M., Braun, M., Frieß, F., Müller, C., Rau, T., Schatz, K., Schulz, C., Krone, M., Reina, G., Ertl, T.: MegaMol – a comprehensive prototyping framework for visualizations. The European Physical Journal Special Topics **227**(14), 1817–1829 (2019). DOI 10.1140/epjst/e2019-800167-5
11. Grottel, S., Krone, M., Müller, C., Reina, G., Ertl, T.: MegaMol – Prototyping Framework for Particle-Based Visualization. IEEE Transactions on Visualization and Computer Graphics **21**(2), 201–214 (2015). DOI 10.1109/TVCG.2014.2350479
12. 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), 500–544 (1952)
13. Kress, J., Klasky, S., Podhorszki, N., Choi, J., Childs, H., Pugmire, D.: Loosely Coupled In Situ Visualization: A Perspective on Why It's Here to Stay. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV2015, pp. 1–6. Association for Computing Machinery (2015). DOI 10.1145/2828612.2828623
14. Maier, B., Emamy, N., Krämer, A.S., Mehl, M.: Highly parallel multi-physics simulation of muscular activation and EMG. In: COUPLED PROBLEMS 2019, pp. 610–621 (2019)
15. Mordhorst, M., Heidlauf, T., Röhrle, O.: Predicting electromyographic signals under realistic conditions using a multiscale chemo-electro-mechanical finite element model. Interface Focus **5**(2), 1–11 (2015). DOI 10.1098/rsfs.2014.0076
16. Moreland, K., Kendall, W., Peterka, T., Huang, J.: An Image Compositing Solution at Scale. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, pp. 25:1–25:10. ACM (2011). DOI 10.1145/2063384.2063417
17. Rau, T., Krone, M., Reina, G., Ertl, T.: Challenges and Opportunities using Software-Defined Visualization in MegaMol. In: N. Ferreira, L.G. Nonato, F. Sadlo (eds.) Workshop on Visual Analytics, Information Visualization and Scientific Visualization (WVIS) in the 30th Conference on Graphics, Patterns and Images (SIBGRAPI'17) (2017)
18. Shorten, P.R., O'Callaghan, P., Davidson, J.B., Soboleva, T.K.: A mathematical model of fatigue in skeletal muscle force contraction. Journal of Muscle Research and Cell Motility **28**(6), 293–313 (2007)
19. Spitzer, V., Ackerman, M.J., Scherzinger, A.L., Whitlock, D.: The Visible Human Male: A Technical Report. Journal of the American Medical Informatics Association **3**(2), 118–130 (1996). DOI 10.1136/jamia.1996.96236280
20. Wald, I., Johnson, G., Amstutz, J., Brownlee, C., Knoll, A., Jeffers, J., Günther, J., Navratil, P.: OSPRay - A CPU Ray Tracing Framework for Scientific Visualization. IEEE Transactions on Visualization and Computer Graphics **23**(1), 931–940 (2017). DOI 10.1109/TVCG.2016.2599041