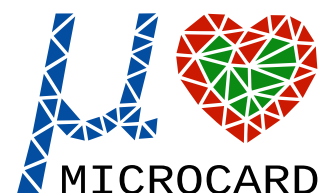


Centre of Excellence

MICROCARD-2



Deliverable 7.1 (public)

6-month benchmark report

PROJECT INFORMATION

Grant agreement number	101172576
Project acronym	MICROCARD-2
Project title	Numerical modeling of cardiac electrophysiology at the cellular scale
Start date of the project	1 November 2024
Duration of the project	30 months
Call	HORIZON-EUROHPC-JU-2023-COE-03 – Centres Of Excellence For Exascale HPC Applications
Topic	HORIZON-EUROHPC-JU-2023-COE-03-01
Scientific coordinator	Mark Potse (UBx)

DELIVERABLE INFORMATION

Deliverable number	7.1
Dissemination level	public
Work package	7
Tasks	7.1
WP leader	Simone Pezzuto
Lead beneficiary	UTrento
Authors	Fatemeh Chegini, Fritz Göbel, Julian Schramm, Joshua Steyer, James D Trotter
Reviewers	Robert Nürnberg, Xing Cai, Ana Alonso
Due date of Deliverable	30 April 2025 (M6)
Submission date	30 April 2025



EuroHPC
Joint Undertaking



**Co-funded by
the European Union**

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or EuroHPC. Neither the European Union nor the granting authority can be held responsible for them. The project is supported by the EuroHPC Joint Undertaking and its members (including top-up funding by ANR, BMBF, and Ministero dello sviluppo economico).

Contents

PROJECT SUMMARY	4
DELIVERABLE SUMMARY	4
1 Introduction	5
1.1 The EMI model	5
1.2 Weak formulation	6
1.3 Finite element discretization	7
1.4 Time integration method	8
1.5 Linear solver and preconditioner	8
1.6 Ionic models	8
1.7 Parallel computing in openCARP	9
2 Serial linear solver performance	9
2.1 Experimental setup	9
2.2 Linear solvers	10
2.3 Serial performance	11
3 Parallel performance and scalability	12
3.1 Single node performance with OpenMP	12
3.2 Single node performance with MPI	12
3.3 Weak scaling	14
4 Simulations with realistic cardiac tissues	15
4.1 Experimental setup	15
4.2 Results	17
References	18
5 Appendix: Ginkgo configuration	20

History of changes

version	date	change history	authors	partner
1	30/03/2025	Initial version	James D Trotter	Simula
2	02/04/2025	Weak scaling results	Fritz Göbel	TUM
3	07/04/2025	Section 1	Fatemeh Chegini	ZIB
4	16/04/2025	Section 4	Joshua Steyer	KIT
5	23/04/2025	Revisions	James D Trotter	Simula
6	24/04/2025	Section 3	Fritz Göbel	TUM
7	28/04/2025	Revisions	Simone Pezzuto	UTrento
8	29/04/2025	Revisions	Julian Schramm	ZIB

List of beneficiaries

nr	organisation name	short name	country	PI
1	Université de Bordeaux	UBx	France	Mark Potse
2	Università degli studi di Pavia	UPavia	Italy	Luca Pavarino
3	Zuse Institute Berlin	ZIB	Germany	Martin Weiser
4	Karlsruhe Institute of Technology	KIT	Germany	Axel Loewe
5	Technical University München	TUM	Germany	Hartwig Anzt
6	Simula Research Laboratory AS	Simula	Norway	Aslak Tveito
7	Università di Trento	UTrento	Italy	Simone Pezzuto
8	Inria	Inria	France	Olivier Aumage
9	Université de Strasbourg	Unistra	France	Vincent Loechner
10	MEGWARE Computer Vertrieb und Service GMBH	MEGWARE	Germany	Axel Auweter

PROJECT SUMMARY

Cardiac function is coordinated by an electric system whose disorders are among the most frequent causes of death and disease. Numerical models of this complex system are mature and widely used, but to match observations in aging and diseased hearts they need to move from a continuum approach to a representation of individual cells and their interconnections. This makes the problem more complex, harder to solve, and four orders of magnitude larger, necessitating exascale computers.

The EuroHPC-2019 MICROCARD project developed a simulation platform that can meet this challenge, by a joint effort of HPC experts, numerical scientists, biomedical engineers, and biomedical scientists, from academia and industry. The Centre of Excellence MICROCARD-2 will consolidate and scale up the MICROCARD results enabling digital twins of cardiac tissue.

We will further develop MICROCARD's numerical schemes, moving to second-order spatial discretization. We will develop mixed-precision preconditioners and data compression to reduce communication bandwidth. The highly successful efforts made in MICROCARD towards automated compilation of high-level model descriptions into optimized, energy-efficient system code for different CPUs and GPUs will be extended to upcoming architectures. We will continue efforts to robustify parallel remeshing software and add necessary functionality for parallel mesh partitioning and production of realistic synthetic tissue meshes needed for simulations.

The platform will be benchmarked with realistic test cases and be made accessible for a wide range of users with tailored workflows. It will be adaptable to similar biological systems such as nerves, and several of our products such as improved solvers, preconditioners, remeshers, and partitioners will be reusable in a wide range of applications.

DELIVERABLE SUMMARY

This report concerns the current capabilities and performance of the openCARP software as a tool for cardiac simulations at the cellular scale. Specifically, in MICROCARD-2, a solver for the Extracellular-Membrane-Intracellular (EMI) model [1, 2] has been developed for openCARP. This report presents benchmarks to evaluate the serial and parallel performance on a set of simple, idealised geometries. Additional experiments are presented to demonstrate the EMI solver's capabilities to also handle realistic cardiac tissues with hundreds of myocytes and millions of mesh elements.

1 Introduction

Numerical models and methods from MICROCARD are currently being integrated into openCARP [3, 4], the primary simulation framework adopted by MICROCARD-2 for modeling cardiac electrical activity. OpenCARP is an open-source software (the successor of the proprietary CARP software), designed for flexible and scalable simulations in cardiac electrophysiology. It aims to support a wide user base in research, education, and translational medicine by including well-established modeling workflows, a powerful scripting interface via Python (carputils), and support for detailed cell, tissue, and organ-scale simulations.

Moreover, openCARP is built on highly parallel and scalable components, such as PETSc [5] and Ginkgo [6] for solving linear systems, and it supports domain decomposition methods and MPI-based parallelism. These properties are essential for adapting to exascale architectures, enabling the simulation of highly detailed models, such as the Extracellular-Membrane-Intracellular (EMI) model [1, 2], at cellular resolution across large tissue volumes.

Ongoing work in MICROCARD-2 has led to a solver implementation for the EMI model, which is now available to users in a development version of openCARP. The solver is in the process of being integrated into a future openCARP release. In the meantime, the development version has been used to conduct the experiments and performance benchmarks described in this report.

Briefly outlined, the EMI solver in openCARP employs a spatial discretization based on the Finite Element Method (FEM) and a time discretization based on an implicit-explicit (IMEX) scheme. The vertex-based, first-order (P1) finite element formulation resolves the intracellular, membrane, and extracellular domains explicitly on a tetrahedral mesh. Each tetrahedron is associated with a specific subdomain (e.g., intra- or extracellular), and triangular elements are associated with interfaces between subdomains (e.g., gap junctions or membranes). The numerical method enforces flux continuity across the interfaces and captures the bidomain dynamics at the subcellular level. The discrete EMI system is constructed by assembling the stiffness and mass matrices corresponding to the intra- and extracellular spaces, as well as interface terms that couple these compartments through the transmembrane current. The result is a monolithic linear system with a block structure that couples the electric potentials in all three regions.

The rest of this section goes into further details about the openCARP-based solver for the EMI model.

1.1 The EMI model

The EMI (extracellular-membrane-intracellular) model of electrophysiology describes the myocardium as a collection of N pairwise disjoint myocytes, $(\Omega_i)_{i=1,\dots,N}$, which, together with the extracellular space Ω_0 , cover the whole domain, $\Omega \subset \mathbb{R}^d$, $d \in \{2, 3\}$, occupied by the myocardium, i.e., $\overline{\Omega} = \bigcup_{i=0}^N \overline{\Omega}_i$. Ions can diffuse within each myocyte and in the extracellular domain, subject to conductivities σ_i , which leads to intra- and extracellular electric potentials $u_i \in H^1(\Omega_i)$. Ion currents across the membranes are due to active or passive ion channels controlling the exchange of ions between myocytes and the extracellular space, whereas ion currents

across gap junctions between adjacent myocytes are due to passive ion channels. For simplicity, we refer to both cases as having a transmembrane current $n^T \sigma_i \nabla u_i$, which consists of the ion current I_{ij}^{ion} and the capacitive current $C_m \dot{v}_{ij}$. The ion current depends on the transmembrane voltage $v_{ij} = u_i - u_j$ defined on the membrane or gap junction $F_{ij} = \partial\Omega_i \cap \partial\Omega_j$, separating adjacent subdomains Ω_i and Ω_j , as well as the state w_{ij} of ion channels, which in turn follows a nonlinear dynamic given by a function R of the transmembrane voltage and ion channel states. The dynamics of gap junctions are assumed to be linear, meaning that the ion channel state w_{ij} is only effective between the intra- and extracellular space (i.e., $i = 0$ or $j = 0$).

This setting results in the partial differential algebraic system:

$$\begin{cases} -\nabla \cdot (\sigma_i \nabla u_i) = 0, & \text{in } \Omega_i, \\ -n_i^T \sigma_i \nabla u_i = C_m \dot{v}_{ij} + I_{ij}^{\text{ion}}(v_{ij}, w_{ij}), & \text{on } F_{ij}, i \neq j, \\ \dot{w}_{ij} = R(v_{ij}, w_{ij}), & \text{on } F_{ij}, i = 0 \text{ or } j = 0, \\ w_{ij} = 0, & \text{on } F_{ij}, i, j > 0. \end{cases} \quad (1)$$

Here, n_i denotes the unit outer normal of Ω_i , where $i = 0, \dots, N$. Note that $v_{ij} = -v_{ji}$, $I_{ij}^{\text{ion}}(v_{ij}, w_{ij}) = -I_{ji}^{\text{ion}}(v_{ji}, w_{ji})$, but $w_{ij} = w_{ji}$.

1.2 Weak formulation

To yield a weak formulation of the EMI model, we multiply the first equation in system (1) by a test function $\phi_i \in H^1(\Omega_i)$ and perform integration by parts:

$$\begin{aligned} 0 &= - \int_{\Omega_i} \nabla \cdot (\sigma_i \nabla u_i) \phi_i \, dx \\ &= \int_{\Omega_i} \nabla u_i^T \sigma_i \nabla \phi_i \, dx - \int_{\partial\Omega_i} n_i^T \sigma_i \nabla u_i \phi_i \, ds \\ &= \int_{\Omega_i} \nabla u_i^T \sigma_i \nabla \phi_i \, dx + \sum_{j \neq i} \int_{F_{ij}} \left(C_m \dot{v}_{ij} + I_{ij}^{\text{ion}}(v_{ij}, w_{ij}) \right) \phi_i \, ds. \end{aligned}$$

Summing over all subdomains yields the weak formulation for $u \in V = \prod_{i=0}^N H^1(\Omega_i)$,

$$0 = \sum_{i=0}^N \int_{\Omega_i} \nabla u_i^T \sigma_i \nabla \phi_i \, dx + \sum_{i=0}^N \sum_{j \neq i} \int_{F_{ij}} \left(C_m (\dot{u}_i - \dot{u}_j) + I_{ij}^{\text{ion}}(u_i - u_j, w_{ij}) \right) \phi_i \, ds \quad (2)$$

for all $\phi \in V$.

For brevity, we define the symmetric bilinear forms

$$\begin{aligned} a(u, \phi) &= \sum_{i=0}^N \int_{\Omega_i} \nabla u_i^T \sigma_i \nabla \phi_i \, dx, \\ p(\dot{u}, \phi) &= \sum_{i=0}^N \sum_{j \neq i} \int_{F_{ij}} C_m (\dot{u}_i - \dot{u}_j) \phi_i \, ds \\ &= \sum_{i=0}^N \sum_{i < j} \int_{F_{ij}} C_m (\dot{u}_i - \dot{u}_j) (\phi_i - \phi_j) \, ds, \end{aligned}$$

and the nonlinear form

$$\begin{aligned} f(\phi; u, w) &= \sum_{i=0}^N \sum_{j \neq i} \int_{F_{ij}} I_{ij}^{\text{ion}}(u_i - u_j, w_{ij}) \phi_i \, ds, \\ &= \sum_{i=0}^N \sum_{i < j} \int_{F_{ij}} I_{ij}^{\text{ion}}(u_i - u_j, w_{ij}) (\phi_i - \phi_j) \, ds, \end{aligned}$$

such that we can write (2) as

$$a(u, \phi) + p(\dot{u}, \phi) + f(\phi; u, w) = 0 \quad \forall \phi \in V. \quad (3)$$

1.3 Finite element discretization

Let \mathcal{T} be a conforming simplicial triangulation of Ω compatible with the subdomain structure, i.e., each $T \in \mathcal{T}$ is contained in the closure of exactly one subdomain Ω_i . (We assume that each Ω_i is polytopal.) We consider finite element spaces $V_{h,i} = \{u \in C(\overline{\Omega}_i) \mid \forall T \in \mathcal{T}, T \subset \overline{\Omega}_i : u|_T \in \mathbb{P}_p\}$ of piecewise polynomial and continuous functions (within each subdomain) and the product space $V_h = \prod_{i=0}^N V_{h,i}$ containing finite element functions which are discontinuous across the cell membranes. Using a Lagrange basis $(\varphi_k)_{k=1, \dots, N_h}$ of V_h , the standard Galerkin approach

$$a(u_h, \varphi_k) + p(\dot{u}_h, \varphi_k) + f(\varphi_k; u_h, w_h) = 0, \quad k = 1, \dots, N_h,$$

transforms the weak formulation in Eq. (3) into a differential-algebraic equation

$$M \dot{u}_h = -K u_h - b(u_h, w_h), \quad (4)$$

where $M_{kl} = p(\varphi_k, \varphi_l)$, $K_{kl} = a(\varphi_k, \varphi_l)$, $b_k(u_h, w_h) = f(\varphi_k; u_h, w_h)$, and the slight abuse of notation of identifying $u_h \in V_h$ with its coefficient vector $u_h \in \mathbb{R}^{N_h}$ with respect to the basis $(\varphi_k)_k$. K denotes the stiffness matrix assembled over all elements of the given mesh, and M denotes the mass matrix assembled over the faces on the interface mesh, such that the stiffness matrix is located on the block diagonal of the global matrix, while the mass matrix is located on the off-diagonal blocks. The density of the off-diagonal blocks reflects the connectivity between neighboring regions.

On the cell membranes between myocytes and the extracellular domain only, the pointwise ordinary differential equations (ODE) for w_{ij} yield the corresponding ODEs

$$\dot{w}_h = R_h(u_h, w_h) \quad (5)$$

for the finite element coefficients $w_h \in \mathbb{R}^{n_h}$. Note that the dimension N_h in Eq. (4) is usually much larger than the dimension n_h in Eq. (5), since the gating variables w_h are restricted to the nodes on the cell membranes.

1.4 Time integration method

Equations (4) and (5) form an index one differential-algebraic equation. We integrate it by a first order Implicit-Explicit (IMEX) operator splitting method. The scheme reads

$$(M + \tau K) \delta u^k = -\tau \left(K u_h^k + B_s I^{\text{ion}}(B u_h^k, w_h^k) \right) \quad (6)$$

$$u_h^{k+1} = u_h^k + \delta u^k \quad (7)$$

for the discretized solution u_h^k at time step t^k and $\tau = t^{k+1} - t^k$. Based on u_h^{k+1} the approximate solution to the gating variable w_h^{k+1} is computed by a compatible time integration scheme. This scheme takes account of the individual dynamics of w . The non-square matrices $B \in \mathbb{R}^{m \times n}$ and $B_s \in \mathbb{R}^{n \times m}$ represent the interpolation and scattering operators, respectively. Here, m is the number of faces on the surface mesh representing all membrane and gap junction interfaces, and n is the number of degrees of freedom (DoFs) in the volumetric mesh. The interpolation operator B maps the electrical potential from the nodal DoFs to the barycenter of each face on the interface mesh. Conversely, the scattering operator B_s distributes the computed transmembrane current—defined at the barycenters—back to the surrounding nodal values. To avoid ambiguity, ionic currents and gap junction fluxes are evaluated as piecewise constant at the barycenters of interface faces.

1.5 Linear solver and preconditioner

OpenCARP supports both PETSc and Ginkgo solver backends, allowing users to select their preferred solver and preconditioner through input parameters or command-line options. By default, PETSc's conjugate gradient (CG) solver with an incomplete Cholesky factorisation (ICC) preconditioner is configured. However, users can override this setup by providing a custom configuration file via the command line. An example is shown in [listing 1](#) on p. 20, which uses Ginkgo's CG solver together with a three-level algebraic multigrid (AMG) preconditioner.

1.6 Ionic models

The complexity and type of the chosen ionic models—especially those with a large number of state variables—can significantly affect the computational requirements of the solver. The EMI model in openCARP supports the use of multiple ionic models, which can be applied to the membranes or gap junctions of different myocytes. As a result, the functions R in equation (1) and R_h in equation (5), which govern the dynamics of the membranes and gap junctions, are considered to be spatially dependent. This enables users to implement physiological and pathological heterogeneity. To achieve this, *tag numbers* are assigned to specific mesh regions in the input files provided by the user to configure their simulations in openCARP.

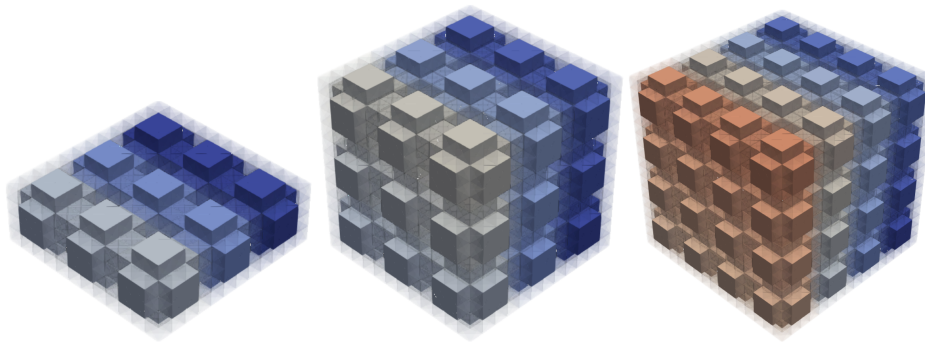


Figure 1: 3D grids of idealised myocytes.

1.7 Parallel computing in openCARP

To implement process-level parallelism with the requirement that myocytes not be split across partitions, the input mesh is pre-partitioned. Each mesh element is assigned a numerical *tag* corresponding to a subdomain or part of a subdomain. There is usually a tag for each myocyte and one or more tags for the immediately surrounding extracellular region. The partitioner then distributes the tag pairs corresponding to myocytes to processes, such as to balance workload and minimize partition surface.

A given process assembles a part of the global linear system, i.e., the matrix and right-hand side vectors, based on the mesh elements assigned to it. The MPI message passing model is used to communicate between processes when necessary, and parallel linear solvers from PETSc and Ginkgo do most of the work.

2 Serial linear solver performance

In this section, we evaluate the serial performance of what is typically the most time-consuming component of cell-by-cell electrophysiology models in openCARP, namely solving linear systems that arise from the finite element discretisation during every time step. The purpose is to establish single-core baselines to later compare against openCARP’s distributed-memory parallel solvers.

The problems considered here have little physiological relevance, but they are purposefully chosen because they shed light on the performance of the underlying numerical methods and software implementation. A more realistic representation of cardiac tissue is considered in [section 4](#), and future versions of these benchmark reports will incorporate even more physiologically relevant scenarios.

2.1 Experimental setup

To evaluate the linear solver, we use simple meshes consisting of regular 3D grids of idealised myocytes surrounded by a cuboid extracellular region (see [figure 1](#)). Each myocyte is shaped

like a $50 \times 50 \times 50 \mu\text{m}^3$ cube with cuboid extrusions with a dimension of $25 \times 50 \times 50 \mu\text{m}^3$ on each of its six faces. These extrusions form gap junctions, where they intersect with neighbouring myocytes. The volume of each myocyte is $500\,000 \mu\text{m}^3$, occupying 50 % of the mesh volume, while the extracellular region occupies the other half. Moreover, the mesh is generated with edge lengths of 8 to $12 \mu\text{m}$, resulting in 6 144 tetrahedral elements per myocyte. The extracellular region contains the same number of elements as all of the myocytes combined.

We note again that the geometry described above is not representative of real cardiac tissue, particularly because myocytes tend to be considerably smaller (see also [section 4](#)). Nevertheless, the geometry has the advantage of being simple to generate, and it provides a good starting point for evaluating linear solver efficiency.

The intra- and extracellular regions have isotropic conductivities of 2.0 S/m and 0.4 S/m, respectively. Cell membranes observe a default Aliev-Panfilov ionic cell model [7] with two state variables, and a forward Euler scheme is used to solve the ODEs. Gap junctions obey a simple, passive model [8] with a gap junction resistance of $0.0045 \text{ k}\Omega \text{ cm}^2$. These models are light in terms of computation and are negligible with respect to the overall simulation runtime for the benchmarks considered here. The time measurements reported in the following experiments therefore do not include the time for solving the ODEs.

A Dirichlet boundary condition is applied to at least one vertex in the extracellular domain, fixing its extracellular potential to zero, to avoid ambiguity and ensure a unique solution. Although it is sufficient to fix a single vertex, in practice, the Dirichlet condition is applied to all vertices on the boundary of the extracellular domain within a sphere of radius $10 \mu\text{m}$ centered at a position $12.5 \mu\text{m}$, $12.5 \mu\text{m}$ and $6.25 \mu\text{m}$ along the x-, y- and z-axis, respectively, from one corner of the grid.

The simulation is initiated with a 1 millisecond stimulus of $200 \mu\text{A}/\text{cm}^2$ applied to a single myocyte. The simulation is then carried out for 2 000 time steps with a step size of $10 \mu\text{s}$, i.e., a total simulated time of 20 milliseconds.

For the experimental setup described above, we observe that the electric potentials propagate in the expected manner, producing results that appear to be physiologically consistent.

2.2 Linear solvers

OpenCARP supports a variety of linear solvers accessed through the PETSc [9] or Ginkgo [10] libraries. In this evaluation, we include several iterative solvers based on the conjugate gradient (CG) method. The first, which is openCARP's default linear solver, uses PETSc's native CG together with a preconditioner based on incomplete Cholesky factorisation. We also include PETSc's and Ginkgo's unpreconditioned CG, PETSc's CG with algebraic multigrid (AMG) preconditioning provided by Hypre [11], and Ginkgo's native CG with AMG preconditioning. In all cases, we set the stopping criterion for the iterative methods to require a reduction of 10^{-8} in the relative difference of the unpreconditioned residual norm. Finally, we also include three factorisation-based direct solvers: PETSc's native Cholesky solver, SuiteSparse's CHOLMOD [12], and SuperLU_dist [13].

Solver	Preconditioner	Time [s]	Iterations	GFLOP/s
CG (PETSc)	ICC (PETSc)	3763.05	269.3	1.79
CG (PETSc)	None	6938.00	1387.8	2.92
CG (PETSc)	AMG (Hypre)	1598.73	10.0	Unavailable
CG (Ginkgo)	None	3154.32	Unavailable	Unavailable
CG (Ginkgo)	AMG (Ginkgo)	1750.84	Unavailable	Unavailable
Cholesky (PETSc)	Not applicable	1067.80	Not applicable	1.09
Cholesky (CHOLMOD)	Not applicable	163.38	Not applicable	3.12
LU (SuperLU_dist)	Not applicable	554.89	Not applicable	Unavailable

Table 1: Runtime (in seconds) for different solvers on a single core of an Intel Xeon Platinum 8360Y (Ice Lake) CPU. OpenCARP is used to carry out 2000 time steps of the EMI model solver. The mesh consists of a $4 \times 4 \times 4$ grid of idealised myocytes of 786 432 elements, resulting in a linear system with 176 497 unknowns. The mean iteration count per time step is also shown, where applicable.

While it does not feature in these experiments, we note that a new BDDC (Balancing Domain Decomposition by Constraints) preconditioner has been developed for Ginkgo and identified as a suitable candidate for EMI simulations [14]. It is currently in the process of being integrated into openCARP and will be investigated in the future.

2.3 Serial performance

Table 1 shows the serial runtime of different linear solvers for a mesh of $4 \times 4 \times 4$ myocytes on a single core of an Intel Xeon Platinum 8360Y (Ice Lake) CPU. The runtime reported by OpenCARP includes assembly of a right-hand side vector and subsequent solution of the linear system. The time for solving the ODEs is negligible (less than 2 seconds) and is not included in the reported time. For PETSc’s iterative solvers, we include the mean iteration count over all time steps. (This information is not currently available in openCARP for Ginkgo’s solvers.) Iteration counts are not applicable for the direct solvers. Furthermore, in the case of PETSc’s native solvers and CHOLMOD, we also show the rate of floating point operations (FLOPs) in GFLOP/s, as counted and provided by PETSc. This information is not directly available for other solvers at the moment.

As a baseline, we consider openCARP’s default linear solver, which is PETSc’s CG with an incomplete Cholesky factorisation preconditioner. The total runtime for this solver is 3763 seconds, or about 1 hour and 3 minutes. On average, the solver performs 269 iterations per time step and 1.79 GFLOP/s.

For this problem, we observe that the three direct solvers are the fastest. Compared to the default solver, the speedups are 3.5, 6.8, and 23.0 times for PETSc’s Cholesky solver, SuperLU_dist and CHOLMOD, respectively. The fastest serial solver for this problem is therefore the direct solver CHOLMOD from SuiteSparse. The direct solvers are followed by the multigrid-preconditioned CG solvers from Hypre and Ginkgo. Their performance is comparable, yielding speedups of $2.4\times$ and $2.1\times$, respectively, compared to the default solver.

In the serial case, the default linear solver performs considerably worse than the alternatives (except for PETSc’s unpreconditioned CG). That being said, it may still be a reasonable default choice because the solver supports distributed-memory parallelisation with MPI, unlike, e.g., CHOLMOD which only supports shared-memory parallelism through OpenMP and would therefore be limited to a single compute node. Furthermore, the default solver does not require additional third-party libraries beyond PETSc.

3 Parallel performance and scalability

This section describes three experiments to assess the parallel performance of the openCARP EMI solver. The first experiment concerns parallel performance on a single compute node using shared-memory parallelism with OpenMP. The second and third experiments use distributed-memory parallelism with MPI to investigate the strong scaling behaviour for a single node and the weak scaling behaviour in a setting with multiple nodes.

3.1 Single node performance with OpenMP

Several of the linear solvers available to openCARP support shared-memory parallelism using OpenMP. [Table 2](#) shows the runtime using up to 36 OpenMP threads on an Intel Xeon Platinum 8360Y CPU for a subset of the solvers that also support OpenMP. The experimental setup is the same as in [section 2.1](#).

Only Ginkgo’s unpreconditioned CG shows good scalability with a speedup of 16.6 times when using 36 threads. Ginkgo’s CG with AMG preconditioning and CHOLMOD show a small improvement with up to 4 or 8 threads, but thereafter appear to degrade in performance. The other solvers do not benefit from OpenMP parallelism at all. In the end, CHOLMOD remains the fastest solver for the single-node case when using 8 threads with a runtime of 110 seconds, corresponding to a speedup of 34 times compared to the serial baseline solver of PETSc’s CG with incomplete Cholesky factorisation preconditioning.

3.2 Single node performance with MPI

Instead of using OpenMP parallelism, the PETSc and Ginkgo backends also offer parallelism through MPI. We now study the performance of two different MPI-parallel, conjugate gradient solvers preconditioned with AMG: 1) PETSc’s CG with AMG from Hypre, and 2) Ginkgo’s CG with Ginkgo’s own AMG preconditioner. These experiments were run on the CPU queue on the EuroHPC machine Karolina, where each compute node has two AMD EPYC 7H12 CPUs with 64 cores each.

The experimental setup is the same as in [section 2.1](#), except that only 200 time steps are carried out. The PETSc backend was configured to use the default options for Hypre, whereas the configuration used for the Ginkgo backend is given in [listing 1](#) on [page 20](#).

Solver	Preconditioner	Time [s]				
		1 thread	2 threads	4 threads	8 threads	36 threads
CG (Ginkgo)	None	3154.32	1679.15	948.22	546.48	189.62
CG (PETSc)	AMG (Hypre)	1598.73	1619.16	1603.43	1622.89	1647.18
CG (Ginkgo)	AMG (Ginkgo)	1750.84	1518.87	1413.63	1505.27	2317.67
Cholesky (CHOLMOD)	Not applicable	163.38	130.74	114.16	110.11	133.72
LU (SuperLU_dist)	Not applicable	554.89	546.82	549.31	556.39	671.66

Table 2: Runtime (in seconds) for different solvers using up to 36 cores of an Intel Xeon Platinum 8360Y (Ice Lake) CPU. OpenCARP is used to carry out 2 000 time steps of the EMI model solver. The mesh consists of a $4 \times 4 \times 4$ grid of idealised myocytes of 786 432 elements, resulting in a linear system with 176 497 unknowns.

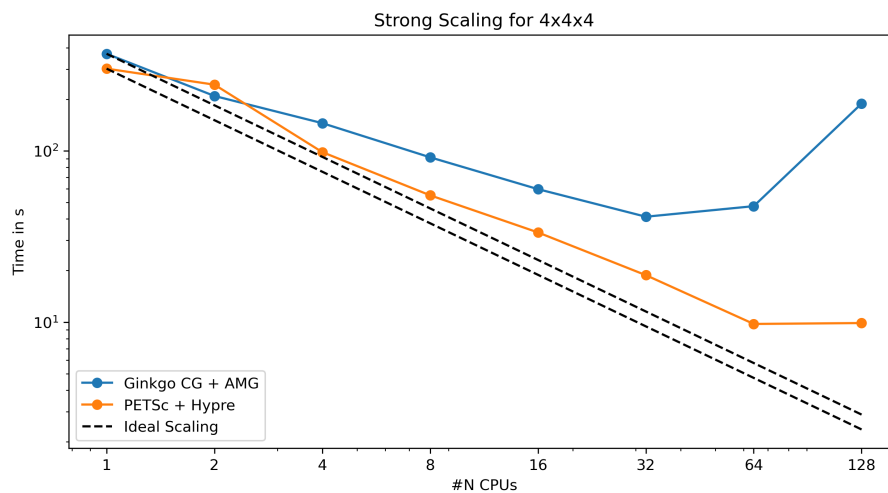


Figure 2: Runtime (in seconds) for the PETSc and Ginkgo linear solvers in a strong scaling experiment.

Table 3 and figure 2 consider strong scaling for the $4 \times 4 \times 4$ mesh on a single compute node, starting with one CPU core and doubling the number of CPU cores until a full node of 128 cores is utilized.

In the case of 1 CPU core, Hypre spends 31.8 seconds per iteration, whereas Ginkgo only takes 16.1 seconds per iteration. Ginkgo, however, performs about 2.4 times more iterations than Hypre, which may be expected due to the respective AMG preconditioners using different methods for coarsening and different smoothers.

Initially, both the PETSc and Ginkgo solvers scale close to optimal, until finally the local problems become too small and performance degrades, presumably due to increased communication overhead. Ginkgo, in particular, suffers from poor performance when 64 or 128 CPUs are used, and further performance analysis is needed to diagnose potential issues.

#CPUs	CG+Hypre (PETSc)		CG+AMG (Ginkgo)	
	Time [s]	Iterations	Time [s]	Iterations
1	301.89	9.50	368.72	22.96
2	243.40	9.43	209.05	23.15
4	98.25	9.41	145.05	22.84
8	54.95	9.45	91.60	22.92
16	33.36	9.41	59.69	21.93
32	18.77	9.50	41.20	21.85
64	9.74	9.47	47.51	22.10
128	9.87	9.47	188.72	22.33

Table 3: Runtime (in seconds) and mean iterations for 200 time steps on a mesh containing $4 \times 4 \times 4$ cells using PETSc and Ginkgo solvers on up to 128 cores of a dual-socket system with 2x AMD EPYC 7H12 CPUs.

#cells	CPU cores	CG+Hypre(PETSc)		CG+AMG(Ginkgo)	
		Time [s]	Iterations	Time [s]	Iterations
$3 \times 3 \times 1$	9	5.83	9.56	17.99	21.84
$3 \times 3 \times 2$	18	7.39	9.49	23.29	21.91
$3 \times 3 \times 3$	27	8.18	9.50	28.81	23.23
$4 \times 4 \times 4$	64	9.65	9.47	47.63	22.10
$5 \times 5 \times 5$	125	12.09	9.55	100.31	21.96
$6 \times 6 \times 6$	216	34.39	9.58	1201.55	23.54
$7 \times 7 \times 7$	343	66.95	9.55	2915.03	23.16

Table 4: Weak scalability: Runtime (in seconds) and mean iterations for 200 timesteps on meshes with an increasing number of cells. Each cell and the immediately surrounding extra-cellular subdomain are assigned to one CPU core, so the size of each local problem remains constant.

3.3 Weak scaling

We now consider a sequence of meshes of the type shown in [figure 1](#) with an increasing number of myocytes up to a $7 \times 7 \times 7$ grid with a total of 343 myocytes. These meshes are used to study the weak scaling behaviour of the solver by assigning to each MPI process one myocyte together with its immediately surrounding extracellular subdomain. In other words, the problem size per CPU core is held constant, as we increase the number of myocytes and the number of CPU cores used at the same rate.

[Table 4](#) and [figure 3](#) show the weak scaling behavior of the PETSc and Ginkgo solvers for simulations with 200 timesteps. While initially, the compute time grows only slowly and the increasing parallelism compensates for the larger problem size well, this is not the case once more than one compute node with 128 CPU cores is used and internode communication is necessary. This could potentially be relieved at least partly by larger local computations, which will be investigated in future reports.

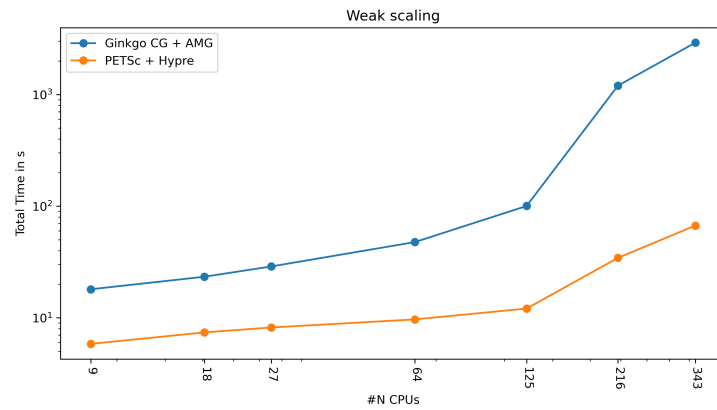


Figure 3: Runtime (in seconds) for the PETSc and Ginkgo linear solvers in a weak scaling experiment. One compute node has 128 CPU cores, and for problems where we assign more cores than that, inter-node communication over the network becomes a very time consuming factor.

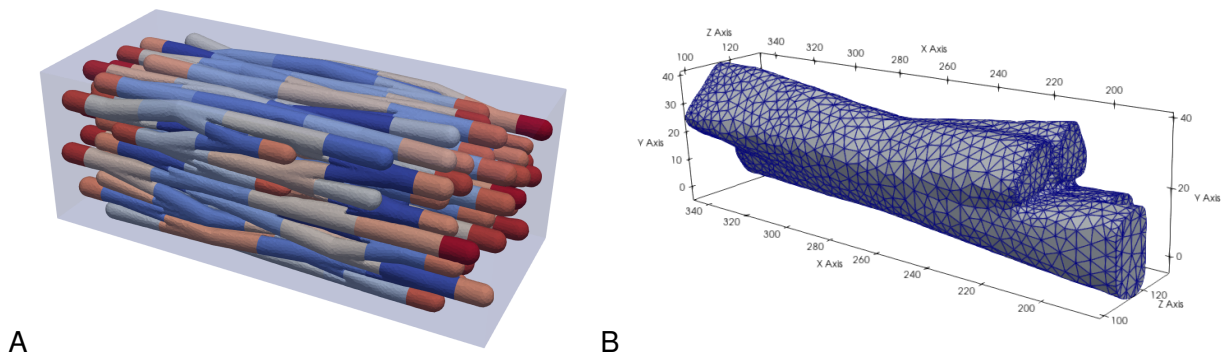


Figure 4: **A:** Synthetically generated mesh with realistic myocyte alignment. **B:** Shape and dimensions (in μm) of a typical myocyte in the mesh. Edges of mesh elements are shown by blue lines.

4 Simulations with realistic cardiac tissues

In this section, we evaluate the strong scaling performance of the EMI model implementation in openCARP using synthetic meshes that realistically represent myocyte shape, alignment and distribution [15]. The purpose is to demonstrate that openCARP's implementation of the EMI model has now matured to the point where it is feasible to conduct experiments with more realistic geometries.

4.1 Experimental setup

The mesh used for the simulations is shown in figure 4. Its computational domain extends over $665 \times 238 \times 287 \mu\text{m}^3$ and contains 331 individual myocytes of varying shapes and dimensions. An example of a typical myocyte is also shown in figure 4. Moreover, the volume of a typical myocyte is $115,961 \mu\text{m}^3$. The intracellular regions thus comprise 47.9 % of the overall volume, while the extracellular region comprises the other 52.1 %. The intracellular volume fraction is

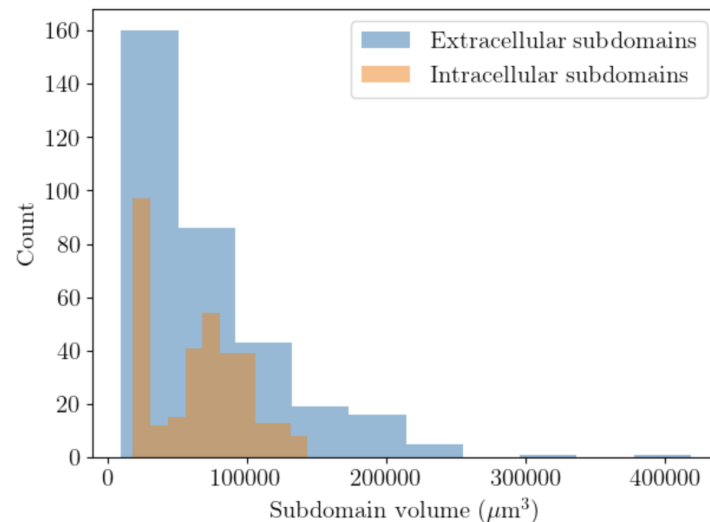


Figure 5: Histogram of the volumes of all subdomains in the mesh.

smaller than in real tissue (70 %) due to boundary effects. Note that the extracellular volume is also subdivided into 331 smaller parts of different sizes, thus one extracellular subdomain for each myocyte. The significance of these intra- and extracellular subdomains is that they are assigned to processes in a parallel simulation as described in [section 1.7](#).

Note that besides the “typical” myocytes, there are also small *caps* at the end of each strand of myocytes, near the edge of the domain. These are usually shown in red in [figure 4](#). Although they are smaller, these caps are also treated as individual myocytes and thus also form gap junctions with their neighbouring myocytes.

[Figure 5](#) shows the distribution of the volumes of myocytes and the subdivided extracellular subdomains. As opposed to the meshes evaluated in [section 2](#), the subdomains do not have the same size and there are subdomains whose volume differs significantly from the average volume, such as the smaller *cap* myocytes mentioned above. It is anticipated that uneven subdomain sizes may induce load imbalance for large-scale parallel simulations, unless the workload is distributed among processes in a careful way.

The edge length of the mesh is $4 \pm 1 \mu\text{m}$, resulting in a total of 6 665 245 mesh elements and 1 145 699 mesh nodes.

The electrophysiological parameters are otherwise chosen as in [section 2.1](#). An excitation propagation is triggered by activating a single myocyte. As before, a time step of $10 \mu\text{s}$ is used, but the total simulated time is 2 milliseconds, meaning that 200 time steps are carried out. The linear solver is configured to use the Conjugate Gradient (CG) solver from PETSc, combined with Algebraic Multigrid (AMG) preconditioning provided by Hypre.

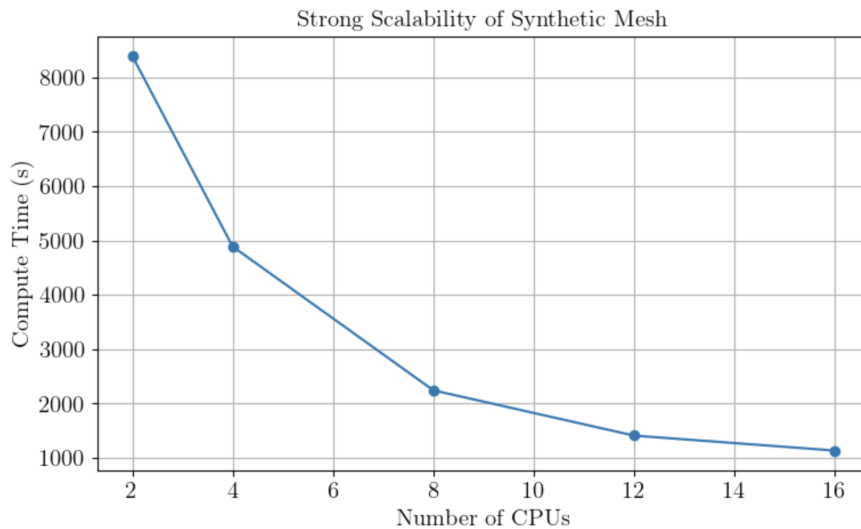


Figure 6: Strong scaling behaviour of simulations with the mesh given in figure 4.

# CPU cores	Time [s]	Speedup	Parallel efficiency	Iterations	Mean κ_2
2	8 396	1.00	100.0 %	10	1.9
4	4 889	1.72	85.8 %	10	1.7
8	2 242	3.74	93.5 %	9	1.8
16	1 133	7.41	92.6 %	10	1.9

Table 5: Runtime (in seconds) for the 331-myocyte mesh shown in figure 4 on up to 16 CPU cores of an Intel Core i9-10980XE CPU. Speedup and parallel efficiency compared to the baseline of 2 CPU cores is also shown. Finally, the mean number of iterations per time step is shown in addition to the condition number (κ_2).

4.2 Results

Figure 6 shows the runtime and strong scaling behaviour on an Intel Core i9-10980XE CPU, using 2 to 16 CPU cores. Further details are provided in table 5. A baseline of 2 CPUs is used because the serial simulations with 1 CPU core take a long time to complete.

The linear solver time for the case of 16 CPU cores amounts to nearly 20 minutes, or about 9 minutes 24 seconds to simulate 1 milliseconds of electrophysiological activity. However, table 5 also shows a high degree of parallel efficiency, above 90 %, when 16 CPU cores are used. These results indicate that load imbalance does not yet pose an issue at this scale, and that the runtime will be further reduced by using even more CPUs. Experiments with multiple nodes and larger number of CPUs will be investigated in future benchmark reports.

Finally, table 5 shows that the AMG preconditioner is effective, even for such a problem with a complicated and realistic geometry. This is corroborated by the low iteration counts and the low condition number κ_2 of the preconditioned problem. Furthermore, iteration counts and condition numbers are not significantly affected by parallel execution with increasing number of CPU cores.

References

- [1] Pierre-Elliott Bécue, Mark Potse, Yves Coudière. Microscopic Simulation of the Cardiac Electrophysiology: A Study of the Influence of Different Gap Junctions Models. In *Computing in Cardiology*, Maastricht, Netherlands, Sept. 2018. <https://hal.inria.fr/hal-01910679>.
- [2] K.H. Jøger, A.G. Edwards, W.R. Giles, A. Tveito. From millimeters to micrometers; re-introducing myocytes in models of cardiac electrophysiology. *Front. Physiol.*, 2021;12:763584.
- [3] Gernot Plank*, Axel Loewe*, Aurel Neic*, Christoph Augustin, Yung-Lin (Cary) Huang, Matthias Gsell, Elias Karabelas, Mark Nothstein, Jorge Sánchez, Anton Prassl, Gunnar Seemann*, Ed Vigmond*. The openCARP simulation environment for cardiac electrophysiology. *Computer Methods and Programs in Biomedicine*, 2021;208:106223. [doi:10.1016/j.cmpb.2021.106223](https://doi.org/10.1016/j.cmpb.2021.106223).
- [4] openCARP consortium, Christoph Augustin, Patrick M. Boyle, Fatemeh Chegini, Raphaël Colin, Matthias Gsell, Marie Houillon, Yung-Lin (Cary) Huang, Atoli Huppé, Kristian Gregorius Hustad, Elias Karabelas, Jonathan Krauß, Moritz Linder, Vincent Loechner, Axel Loewe, Lena Myklebust, Aurel Neic, Mark Nothstein, Gernot Plank, Anton Prassl, Jorge Sánchez, Gunnar Seemann, Tomas Stry, Arun Thangamani, Nico Tippmann, Tiago Trevisan Jost, Ed Vigmond, Eike Moritz Wülfers. openCARP, 2024. [doi:10.35097/70234yr69tr09krh](https://doi.org/10.35097/70234yr69tr09krh).
- [5] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, Hong Zhang. PETSc Web page. <http://www.mcs.anl.gov/petsc>, 2015. <http://www.mcs.anl.gov/petsc>.
- [6] Hartwig Anzt, Terry Cojean, Goran Flegar, Fritz Göbel, Thomas Grützmacher, Pratik Nayak, Tobias Ribizel, Yuhsiang Mike Tsai, Enrique S Quintana-Ortí. Ginkgo: A modern linear operator algebra framework for high performance computing. *ACM Transactions on Mathematical Software (TOMS)*, 2022;48(1):1–33.
- [7] Rubin R. Aliev, Alexander V. Panfilov. A simple two-variable model of cardiac excitation. *Chaos, Solitons & Fractals*, 1996;7(3):293–301. [doi:https://doi.org/10.1016/0960-0779\(95\)00089-5](https://doi.org/10.1016/0960-0779(95)00089-5).
- [8] Robert Plonsey. The formulation of bioelectric source-field relationships in terms of surface discontinuities. *Journal of the Franklin Institute*, 1974;297(5):317–324. [doi:https://doi.org/10.1016/0016-0032\(74\)90036-2](https://doi.org/10.1016/0016-0032(74)90036-2).
- [9] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, Jacob Faibussowitsch, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Hansol Suh,

- Stefano Zampini, Hong Zhang, Hong Zhang, Junchao Zhang. PETSc/TAO users manual. Technical Report ANL-21/39 - Revision 3.23, Argonne National Laboratory, 2025. doi:10.2172/2476320.
- [10] Hartwig Anzt, Terry Cojean, Goran Flegar, Fritz Göbel, Thomas Grützmacher, Pratik Nayak, Tobias Ribizel, Yuhsiang Mike Tsai, Enrique S. Quintana-Ortí. Ginkgo: A Modern Linear Operator Algebra Framework for High Performance Computing. ACM Transactions on Mathematical Software, Feb. 2022;48(1):2:1–2:33. doi:10.1145/3480935.
- [11] *hypr*: High performance preconditioners. <https://llnl.gov/casc/hypr>, <https://github.com/hypr-space/hypr>.
- [12] Yanqing Chen, Timothy A. Davis, William W. Hager, Sivasankaran Rajamanickam. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. ACM Trans. Math. Softw., Oct. 2008;35(3). doi:10.1145/1391989.1391995.
- [13] Xiaoye S. Li, James W. Demmel. Superlu_dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. ACM Trans. Math. Softw., June 2003;29(2):110–140. doi:10.1145/779359.779361.
- [14] Fritz Göbel, Terry Cojean, Hartwig Anzt. BDDC preconditioning on gpus for cardiac simulations. In Euro-Par 2023: Parallel Processing Workshops: Euro-Par 2023 International Workshops, Limassol, Cyprus, August 28 – September 1, 2023, Revised Selected Papers, Part II, page 265–268, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-48803-0_30.
- [15] Mark Potse, Luca Cirrottola, Algiane Froehly. A practical algorithm to build geometric models of cardiac muscle structure. In Proceedings of the 8th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS Congress 2022), Oslo, Norway, June 2022. <https://inria.hal.science/hal-03936963v1/file/potse-eccomas22.pdf>.

5 Appendix: Ginkgo configuration

The configuration file used for the Ginkgo linear solver in sections 3.2 and 3.3 is shown below.

```

1 {
2   "type": "solver::Cg",
3   "criteria": [
4     {
5       "type": "Iteration",
6       "max_iters": 100
7     },
8     {
9       "type": "ResidualNorm",
10      "reduction_factor": 1e-8
11    }
12  ],
13  "preconditioner": {
14    "type": "solver::Multigrid",
15    "mg_level": [
16      {
17        "type": "multigrid::Pgm",
18        "deterministic": true,
19        "max_unassigned_ratio": 0.25
20      }
21    ],
22    "criteria": [
23      {
24        "type": "Iteration",
25        "max_iters": 1
26      }
27    ],
28    "coarsest_solver": {
29      "type": "preconditioner::Schwarz",
30      "local_solver": {
31        "type": "solver::Cg",
32        "preconditioner": {
33          "type": "preconditioner::Isai",
34          "isai_type": "spd"
35        },
36        "criteria": [
37          {
38            "type": "Iteration",
39            "max_iters": 100
40          },
41          {
42            "type": "ResidualNorm",
43            "reduction_factor": 1e-3
44          }
45        ]
46      }
47    },
48    "max_levels": 3,
49    "min_coarse_rows": 1024,
50    "default_initial_guess": "zero"
51  }
52 }
```

Listing 1: Configuration used for the Ginkgo solver: A CG solver with a three-level AMG preconditioner with a local only coarse solver to reduce communication.