

Supplementary Information for 'Evaluating Computational Efforts and Physiological Resolution of Mathematical Models of Cardiac Tissue'

Karoline Horgmo Jæger¹, James D. Trotter¹, Xing Cai¹,
Hermenegild Arevalo¹, and Aslak Tveito¹

¹Simula Research Laboratory, Norway

Contents

S1 Membrane model formulation	2
S1.1 The membrane potential	3
S1.2 Stimulation current	3
S1.3 Membrane currents	3
S1.4 Ca^{2+} dynamics	7
S1.5 Ca^{2+} fluxes	8
S1.6 Nernst equilibrium potentials	10
S2 Definition of biomarkers	15
S2.1 Conduction velocity (CV)	15
S2.2 Action potential duration (APD)	15
S2.3 Time of arrival (ToA)	15
S3 Numerical implementation of the models	16
S3.1 Implementation of KNM and SKNM	16
S3.2 Implementation of BD and MD	16
S3.3 Implementation of EMI	16
S4 Parallel solvers for EMI	18
S4.1 MFEM configuration	24

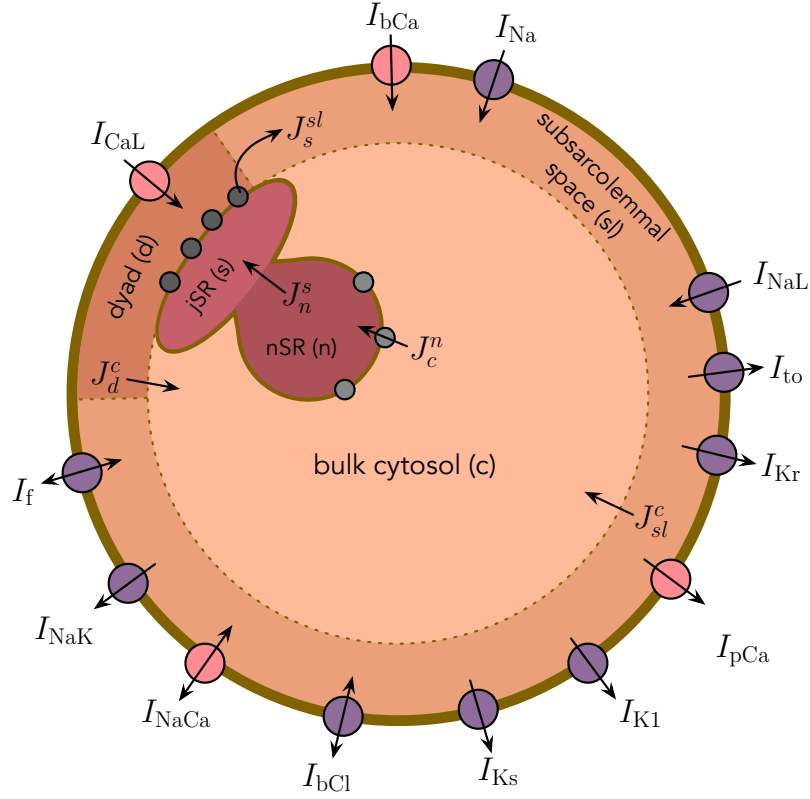


Figure S1: Schematic illustration of the components of the membrane model. The illustration is taken from [1].

S1 Membrane model formulation

Here we describe the full membrane model formulation, taken directly from [1]. In this formulation, the membrane potential (v) is given in units of mV, and the Ca^{2+} concentrations are given in units of mM. All currents are expressed in units of A/F, and the Ca^{2+} fluxes are expressed as mmol/ms per total cell volume (i.e., in units of mM/ms). Time is given in ms. The parameters of the model are given in Tables S1–S6.

S1.1 The membrane potential

The membrane potential is governed by the equation

$$\begin{aligned} \frac{dv}{dt} = & -(I_{\text{Na}} + I_{\text{NaL}} + I_{\text{CaL}} + I_{\text{to}} + I_{\text{Kr}} + I_{\text{Ks}} + I_{\text{K1}} \\ & + I_{\text{NaCa}} + I_{\text{NaK}} + I_{\text{pCa}} + I_{\text{bCl}} + I_{\text{bCa}} + I_{\text{f}} + I_{\text{stim}}), \end{aligned} \quad (\text{S1})$$

where I_{stim} is an applied stimulus current, and I_{Na} , I_{NaL} , I_{CaL} , I_{to} , I_{Kr} , I_{Ks} , I_{K1} , I_{NaCa} , I_{NaK} , I_{pCa} , I_{bCl} , I_{bCa} , and I_{f} are membrane currents specified below.

S1.2 Stimulation current

In our simulations, I_{stim} is given as a constant current of size -40 A/F for the 1D simulations and -200 A/F for the 2D simulations. The I_{stim} current is applied until the membrane potential reaches a value of -40 mV. We let the simulation run for 20 ms before applying the stimulation current to allow the membrane potential to reach a new steady state when parameter changes are applied (i.e., when we include a border zone with an increased extracellular potassium concentration, see Figure S2).

S1.3 Membrane currents

The currents through the voltage-gated ion channels on the cell membrane are in general given on the form

$$I = go(v - E),$$

where g is the channel conductance, v is the membrane potential and E is the equilibrium potential of the channel. Furthermore, $o = \prod_i z_i$ is the open probability of the channels, where z_i are gating variables, either given as a function of the membrane potential or governed by equations of the form

$$z'_i = \frac{1}{\tau_{z_i}}(z_{i,\infty} - z_i). \quad (\text{S2})$$

The parameters τ_{z_i} and $z_{i,\infty}$ are specified for each of the gating variables of the model in Table S7.

Fast sodium current The fast sodium current, I_{Na} , is given by

$$I_{\text{Na}} = g_{\text{Na}} o_{\text{Na}}(v - E_{\text{Na}}), \quad (\text{S3})$$

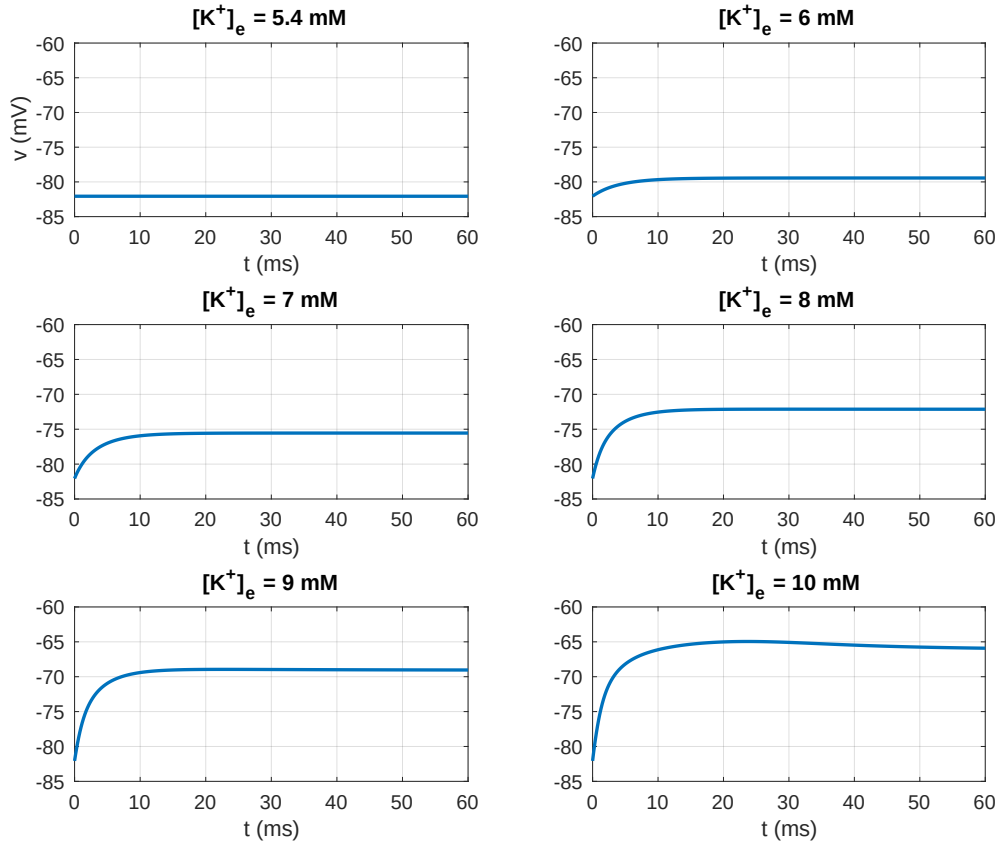


Figure S2: Change in membrane potential resulting from the increase in extracellular potassium concentration used to represent the infarction border zone. The plots show the results of membrane model simulations with initial conditions corresponding to the default extracellular potassium concentration of 5.4 mM and 1 Hz pacing.

where the open probability is given by

$$o_{\text{Na}} = m^3 j, \quad (\text{S4})$$

and m and j are gating variables governed by equations of the form (S2).

Late sodium current The late sodium current, I_{NaL} , is given by

$$I_{\text{NaL}} = g_{\text{NaL}} o_{\text{NaL}} (v - E_{\text{Na}}), \quad (\text{S5})$$

where the open probability is given by

$$o_{\text{NaL}} = m_L h_L, \quad (\text{S6})$$

and m_L and h_L are gating variables governed by equations of the form (S2).

Transient outward potassium current The transient outward potassium current, I_{to} , is given by

$$I_{\text{to}} = g_{\text{to}} o_{\text{to}} (v - E_{\text{to}}), \quad (\text{S7})$$

where the open probability is given by

$$o_{\text{to}} = q_{\text{to}} r_{\text{to}}, \quad (\text{S8})$$

and q_{to} and r_{to} are gating variables governed by equations of the form (S2).

Rapidly activating potassium current The rapidly activating potassium current, I_{Kr} , is given by

$$I_{\text{Kr}} = g_{\text{Kr}} \sqrt{\frac{[K^+]_e}{5.4 \text{ mM}}} o_{\text{Kr}} (v - E_{\text{K}}), \quad (\text{S9})$$

where

$$o_{\text{Kr}} = x_{\text{Kr1}} x_{\text{Kr2}}, \quad (\text{S10})$$

and the dynamics of x_{Kr1} and x_{Kr2} are governed by equations of the form (S2).

Slowly activating potassium current The slowly activating potassium current, I_{Ks} , is given by

$$I_{\text{Ks}} = g_{\text{Ks}} o_{\text{Ks}} (v - E_{\text{Ks}}), \quad (\text{S11})$$

where

$$o_{\text{Ks}} = x_{\text{Ks}}^2, \quad (\text{S12})$$

and the dynamics of x_{Ks} is governed by an equation of the form (S2).

Inward rectifier potassium current The inward rectifier potassium current, I_{K1} , is given by

$$I_{K1} = g_{K1} \sqrt{\frac{[K^+]_e}{5.4 \text{ mM}}} o_{K1} (v - E_K), \quad (\text{S13})$$

where

$$o_{K1} = \frac{a_{K1}}{a_{K1} + b_{K1}}, \quad (\text{S14})$$

$$a_{K1} = \frac{1}{1 + e^{0.2(v - E_K - 59)}}, \quad (\text{S15})$$

$$b_{K1} = \frac{0.5e^{0.08(v - E_K + 5)} + e^{0.06(v - E_K - 594)}}{1 + e^{-0.5(v - E_K + 5)}}. \quad (\text{S16})$$

Hyperpolarization activated funny current The hyperpolarization activated funny current, I_f , is given by

$$I_f = g_f o_f (v - E_f), \quad (\text{S17})$$

where

$$o_f = x_f, \quad (\text{S18})$$

and the dynamics of x_f is governed by an equation of the form (S2).

L-type Ca^{2+} current The L-type Ca^{2+} current, I_{CaL} , is given by

$$I_{\text{CaL}} = g_{\text{CaL}} o_{\text{CaL}} \frac{(2F)^2 v}{RT} \frac{0.341 c_d e^{\frac{2Fv}{RT}} - 0.341 c_e}{e^{\frac{2Fv}{RT}} - 1}, \quad (\text{S19})$$

where

$$o_{\text{CaL}} = df(1 - f_{\text{Ca}}), \quad (\text{S20})$$

and the dynamics of d , f and f_{Ca} are governed by equations of the form (S2).

Background currents The background currents, I_{bCa} and I_{bCl} are given by

$$I_{\text{bCa}} = g_{\text{bCa}} (v - E_{\text{Ca}}), \quad (\text{S21})$$

$$I_{\text{bCl}} = g_{\text{bCl}} (v - E_{\text{Cl}}). \quad (\text{S22})$$

Sodium-calcium exchanger The Na^+ - Ca^{2+} exchanger current, I_{NaCa} , is given by

$$I_{\text{NaCa}} = \bar{I}_{\text{NaCa}} \frac{e^{\frac{\nu F v}{RT}} [\text{Na}^+]_i^3 c_e - e^{\frac{(\nu-1)F v}{RT}} [\text{Na}^+]_e^3 c_{sl}}{s_{\text{NaCa}} \left(1 + \left(\frac{K_{\text{act}}}{c_{sl}} \right)^2 \right) \left(1 + k_{\text{sat}} e^{\frac{(\nu-1)F v}{RT}} \right)}, \quad (\text{S23})$$

where

$$s_{\text{NaCa}} = K_{\text{Ca},i} [\text{Na}^+]_e^3 \left(1 + \left(\frac{[\text{Na}^+]_i}{K_{\text{Na},i}} \right)^3 \right) + K_{\text{Na},e}^3 c_{sl} \left(1 + \frac{c_{sl}}{K_{\text{Ca},i}} \right) + K_{\text{Ca},e} [\text{Na}^+]_i^3 + [\text{Na}^+]_i^3 c_e + [\text{Na}^+]_e^3 c_{sl}.$$

Sarcolemmal Ca^{2+} pump The current through the sarcolemmal Ca^{2+} pump, I_{pCa} , is given by

$$I_{\text{pCa}} = \bar{I}_{\text{pCa}} \frac{c_{sl}^2}{K_{\text{pCa}}^2 + c_{sl}^2}. \quad (\text{S24})$$

Sodium-potassium pump The current through the Na^+ - K^+ pump, I_{NaK} , is given by

$$I_{\text{NaK}} = \bar{I}_{\text{NaK}} \frac{f_{\text{NaK}}}{1 + \left(\frac{K_{\text{NaK}}^{\text{NaK}}}{[\text{Na}^+]_i} \right)^4} \frac{[\text{K}^+]_e}{[\text{K}^+]_e + K_{\text{K},e}}, \quad (\text{S25})$$

where

$$f_{\text{NaK}} = \frac{1}{1 + 0.12 e^{-0.1 \frac{F v}{RT}}} + \frac{0.037}{7} \left(e^{\frac{[\text{Na}^+]_e}{67}} - 1 \right) e^{-\frac{F v}{RT}}. \quad (\text{S26})$$

S1.4 Ca^{2+} dynamics

The Ca^{2+} dynamics are governed by

$$\frac{dc_d}{dt} = \frac{1}{V_d} (J_{\text{CaL}} - J_d^b - J_d^c), \quad \frac{db_d}{dt} = \frac{1}{V_d} J_d^b, \quad (\text{S27})$$

$$\frac{dc_{sl}}{dt} = \frac{1}{V_{sl}} (J_e^{sl} - J_{sl}^c - J_{sl}^b + J_s^{sl}), \quad \frac{db_{sl}}{dt} = \frac{1}{V_{sl}} J_{sl}^b, \quad (\text{S28})$$

$$\frac{dc_c}{dt} = \frac{1}{V_c} (J_{sl}^c + J_d^c - J_c^n - J_c^b), \quad \frac{db_c}{dt} = \frac{1}{V_c} J_c^b, \quad (\text{S29})$$

$$\frac{dc_s}{dt} = \frac{1}{V_s} (J_n^s - J_s^{sl} - J_s^b), \quad \frac{db_s}{dt} = \frac{1}{V_s} J_s^b, \quad (\text{S30})$$

$$\frac{dc_n}{dt} = \frac{1}{V_n} (J_c^n - J_n^s), \quad (\text{S31})$$

where c_d is the concentration of free Ca^{2+} in the dyad, b_d is the concentration of Ca^{2+} bound to a buffer in the dyad, c_{sl} is the concentration of free Ca^{2+} in the SL compartment, b_{sl} is the concentration of Ca^{2+} bound to a buffer in the SL compartment, c_c is the concentration of free Ca^{2+} in the bulk cytosol, b_c is the concentration of Ca^{2+} bound to a buffer in the bulk cytosol, c_s is the concentration of free Ca^{2+} in the jSR, b_s is the concentration of Ca^{2+} bound to a buffer in the jSR, and c_n is the concentration of free Ca^{2+} in the nSR. The expressions for the fluxes are specified below.

S1.5 Ca^{2+} fluxes

Flux through the SERCA pumps The flux from the bulk cytosol to the nSR through the SERCA pumps is given by

$$J_c^n = \bar{J}_{\text{SERCA}} \frac{\left(\frac{c_c}{K_c}\right)^2 - \left(\frac{c_n}{K_n}\right)^2}{1 + \left(\frac{c_c}{K_c}\right)^2 + \left(\frac{c_n}{K_n}\right)^2}. \quad (\text{S32})$$

Flux through the RyRs The flux from the jSR to the SL compartment is given by

$$J_s^{sl} = J_{\text{RyR}} + J_{\text{leak}}, \quad (\text{S33})$$

where J_{RyR} represents the flux through the active RyR channels and J_{leak} represents the flux through the RyR channels that are always open, given by

$$J_{\text{RyR}} = p \cdot r \cdot \alpha_{\text{RyR}}(c_s - c_{sl}), \quad (\text{S34})$$

$$J_{\text{leak}} = \gamma_{\text{RyR}} \cdot \alpha_{\text{RyR}}(c_s - c_{sl}), \quad (\text{S35})$$

respectively. Here, p is the open probability of the active RyR channels given by

$$p = \frac{c_d^3}{c_d^3 + \kappa_{\text{RyR}}^3}, \quad (\text{S36})$$

and r represents the fraction of RyR channels that are not inactivated and is governed by the equation

$$\frac{dr}{dt} = -\frac{J_{\text{RyR}}}{\beta_{\text{RyR}}} + \frac{\eta_{\text{RyR}}}{p}(1 - r). \quad (\text{S37})$$

Passive diffusion fluxes between compartments The passive diffusion fluxes between compartments are given by

$$J_d^c = \alpha_d^c(c_d - c_c), \quad (\text{S38})$$

$$J_{sl}^c = \alpha_{sl}^c(c_{sl} - c_c), \quad (\text{S39})$$

$$J_n^s = \alpha_n^s(c_n - c_s). \quad (\text{S40})$$

Buffer fluxes The fluxes of free Ca^{2+} binding to a Ca^{2+} buffer are given by

$$J_d^b = V_d(k_{\text{on}}^d c_d(B_{\text{tot}}^d - b_d) - k_{\text{off}}^d b_d), \quad (\text{S41})$$

$$J_{sl}^b = V_{sl}(k_{\text{on}}^{sl} c_{sl}(B_{\text{tot}}^{sl} - b_{sl}) - k_{\text{off}}^{sl} b_{sl}), \quad (\text{S42})$$

$$J_c^b = V_c(k_{\text{on}}^c c_c(B_{\text{tot}}^c - b_c) - k_{\text{off}}^c b_c), \quad (\text{S43})$$

$$J_s^b = V_s(k_{\text{on}}^s c_s(B_{\text{tot}}^s - b_s) - k_{\text{off}}^s b_s). \quad (\text{S44})$$

Membrane fluxes The membrane fluxes, J_{CaL} , J_{bCa} , J_{pCa} , and J_{NaCa} , are given by

$$J_{\text{CaL}} = -\frac{\chi C_m}{2F} I_{\text{CaL}}, \quad J_{\text{pCa}} = -\frac{\chi C_m}{2F} I_{\text{pCa}}, \quad (\text{S45})$$

$$J_{\text{bCa}} = -\frac{\chi C_m}{2F} I_{\text{bCa}}, \quad J_{\text{NaCa}} = \frac{\chi C_m}{F} I_{\text{NaCa}}, \quad (\text{S46})$$

where I_{CaL} , I_{bCa} , I_{pCa} , and I_{NaCa} are defined by the expressions given above. Furthermore,

$$J_e^{sl} = J_{\text{NaCa}} + J_{\text{pCa}} + J_{\text{bCa}}. \quad (\text{S47})$$

Parameter	Description	Value
V_d	Volume fraction of the dyadic subspace	0.001
V_{sl}	Volume fraction of the SL compartment	0.028
V_c	Volume fraction of the bulk cytosol	0.917
V_s	Volume fraction of the jSR	0.004
V_n	Volume fraction of the nSR	0.05
χ	Cell surface to volume ratio	$0.6 \mu\text{m}^{-1}$

Table S1: Default geometry parameters of the membrane model.

S1.6 Nernst equilibrium potentials

The Nernst equilibrium potentials for the ion channels are defined as

$$E_{\text{Na}} = \frac{RT}{F} \log \left(\frac{[\text{Na}^+]_e}{[\text{Na}^+]_i} \right), \quad (\text{S48})$$

$$E_{\text{Ca}} = \frac{RT}{2F} \log \left(\frac{[\text{Ca}^{2+}]_e}{c_{sl}} \right), \quad (\text{S49})$$

$$E_{\text{K}} = \frac{RT}{F} \log \left(\frac{[\text{K}^+]_e}{[\text{K}^+]_i} \right), \quad (\text{S50})$$

$$E_{\text{Ks}} = \frac{RT}{F} \log \left(\frac{[\text{K}^+]_e + 0.018[\text{Na}^+]_e}{[\text{K}^+]_i + 0.018[\text{Na}^+]_i} \right), \quad (\text{S51})$$

$$E_{\text{Cl}} = \frac{RT}{F} \log \left(\frac{[\text{Cl}^+]_e}{[\text{Cl}^+]_i} \right), \quad (\text{S52})$$

$$E_{\text{f}} = -17 \text{ mV}, \quad (\text{S53})$$

for the parameter values given in Table S2.

Parameter	Description	Value
C_m	Specific membrane capacitance	0.01 pF/ μm^2
F	Faraday's constant	96.485 C/mmol
R	Universal gas constant	8.314 J/(mol·K)
T	Temperature	310 K
$[\text{Ca}^{2+}]_e$	Extracellular Ca^{2+} concentration	1.8 mM
$[\text{Na}^+]_e$	Extracellular sodium concentration	140 mM
$[\text{Na}^+]_i$	Intracellular sodium concentration	8 mM
$[\text{K}^+]_e$	Extracellular potassium concentration	5.4 mM
$[\text{K}^+]_i$	Intracellular potassium concentration	120 mM
$[\text{Cl}^-]_e$	Extracellular chloride concentration	150 mM
$[\text{Cl}^-]_i$	Intracellular chloride concentration	15 mM

Table S2: Physical constants and ionic concentrations of the membrane model.

Parameter	Value	Parameter	Value
g_{Na}	12.6 mS/ μF	g_{CaL}	0.12 nL/($\mu\text{F ms}$)
g_{NaL}	0.025 mS/ μF	g_{bCa}	0.00055 mS/ μF
g_{to}	0.27 mS/ μF	$\bar{I}_{\text{NaCa}}$	4.9 $\mu\text{A}/\mu\text{F}$
g_{Kr}	0.025 mS/ μF	$\bar{I}_{\text{pCa}}$	0.068 $\mu\text{A}/\mu\text{F}$
g_{Ks}	0.003 mS/ μF	$\bar{J}_{\text{SERCA}}$	0.00024 mM/ms
g_{K1}	0.37 mS/ μF	α_{RyR}	0.0075 ms^{-1}
g_{f}	0.0001 mS/ μF	α_d^c	0.0017 ms^{-1}
g_{bCl}	0.007 mS/ μF	α_{sl}^c	0.15 ms^{-1}
$\bar{I}_{\text{NaK}}$	1.8 $\mu\text{A}/\mu\text{F}$	α_n^s	0.012 ms^{-1}

Table S3: Conductance values and similar parameters for each of the membrane currents and intracellular Ca^{2+} fluxes of the membrane model.

Parameter	Flux	Value
K_c	J_c^n	0.00025 mM
K_n	J_c^n	1.7 mM
β_{RyR}	J_s^{sl}	0.038 mM
γ_{RyR}	J_s^{sl}	0.001
κ_{RyR}	J_{RyR}	0.015 mM
η_{RyR}	J_s^{sl}	0.00001 ms ⁻¹

Table S4: Parameters for the intracellular Ca²⁺ fluxes of the membrane model.

Parameter	Current	Value
k_{sat}	I_{NaCa}	0.3
ν	I_{NaCa}	0.3
K_{act}	I_{NaCa}	0.00015 mM
$K_{\text{Ca},i}$	I_{NaCa}	0.0036 mM
$K_{\text{Ca},e}$	I_{NaCa}	1.3 mM
$K_{\text{Na},i}$	I_{NaCa}	12.3 mM
$K_{\text{Na},e}$	I_{NaCa}	87.5 mM
$K_{\text{Na},i}^{\text{NaK}}$	I_{NaK}	11 mM
$K_{\text{K},e}$	I_{NaK}	1.5 mM
K_{pCa}	I_{pCa}	0.0005 mM

Table S5: Parameters for the membrane currents of the membrane model.

Parameter	Compartment	Value
B_{tot}^c	Bulk cytosol	0.07 mM
k_{on}^c	Bulk cytosol	$40 \text{ ms}^{-1}\text{mM}^{-1}$
k_{off}^c	Bulk cytosol	0.03 ms^{-1}
B_{tot}^d	Dyad	1.2 mM
k_{on}^d	Dyad	$100 \text{ ms}^{-1}\text{mM}^{-1}$
k_{off}^d	Dyad	1 ms^{-1}
B_{tot}^{sl}	Subsarcolemmal space	0.9 mM
k_{on}^{sl}	Subsarcolemmal space	$100 \text{ ms}^{-1}\text{mM}^{-1}$
k_{off}^{sl}	Subsarcolemmal space	0.15 ms^{-1}
B_{tot}^s	Junctional SR	27 mM
k_{on}^s	Junctional SR	$100 \text{ ms}^{-1}\text{mM}^{-1}$
k_{off}^s	Junctional SR	65 ms^{-1}

Table S6: Parameters for the Ca^{2+} buffers of the membrane model.

Current	Gate	z_∞	α_z	β_z	τ_z
I_{Na}	m	$\frac{1}{(1 + e^{(-57-v)/9})^2}$	$0.13e^{-((v+46)/16)^2}$	$0.06e^{-((v-5)/51)^2}$	$\alpha_m + \beta_m$
	j	$\frac{1}{(1 + e^{(v+72)/7})^2}$	$\begin{cases} 0, & \text{if } v \geq -40 \\ \frac{\begin{pmatrix} -2.5 \cdot 10^4 e^{0.2v} \\ -7 \cdot 10^{-6} e^{-0.04v} \end{pmatrix} (v+38)}{1 + e^{0.3(v+79)}}, & \text{otherwise} \end{cases}$	$\begin{cases} \frac{0.6e^{0.06v}}{1 + e^{-0.1(v+32)}}, & \text{if } v \geq -40 \\ \frac{0.02e^{-0.01v}}{1 + e^{-0.14(v+40)}}, & \text{otherwise} \end{cases}$	$\frac{1}{\alpha_j + \beta_j}$
I_{NaL}	m_L	$\frac{1}{1 + e^{(-43-v)/5}}$	$\frac{1}{6.8e^{(v+12)/35}}$	$8.6e^{-(v+77)/6}$	$\alpha_m + \beta_m$
	h_L	$\frac{1}{1 + e^{(v+88)/7.5}}$			200 ms
I_{CaL}	d	$\frac{1}{1 + e^{-(v+5)/6}}$	$\frac{1 - e^{-\frac{v+5}{6}}}{0.035(v+5)}$		$\alpha_d d_\infty$
	f	$\frac{1}{1 + e^{(v+35)/9}} + \frac{0.6}{1 + e^{(50-v)/20}}$	$\frac{1}{0.02e^{-(0.034(v+14.5)^2)} + 0.02}$		α_f
	f_{Ca}	$\frac{1.7c_d}{1.7c_d + 0.012}$	$\frac{1}{1.7c_d + 0.012}$		α_{Ca}
I_{to}	q_{to}	$\frac{1}{1 + e^{(v+53)/13}}$	$\frac{39}{0.57e^{-0.08(v+44)} + 0.065e^{0.1(v+46)}}$	6	$\alpha_{q_{\text{to}}} + \beta_{q_{\text{to}}}$
	r_{to}	$\frac{1}{1 + e^{-(v-22.3)/18.75}}$	$\frac{14.4}{e^{0.09(v+30.61)} + 0.37e^{-0.12(v+24)}}$	2.75	$\alpha_{r_{\text{to}}} + \beta_{r_{\text{to}}}$
I_{Kr}	x_{Kr1}	$\frac{1}{1 + e^{-(v+20.7)/4.9}}$	$\frac{450}{1 + e^{-(v+45)/10}}$	$\frac{6}{1 + e^{(v+30)/11.5}}$	$\alpha_{x_{\text{Kr1}}} \cdot \beta_{x_{\text{Kr1}}}$
	x_{Kr2}	$\frac{1}{1 + e^{(v+88)/50}}$	$\frac{3}{1 + e^{-(v+60)/20}}$	$\frac{1.12}{1 + e^{(v-60)/20}}$	$\alpha_{x_{\text{Kr2}}} \cdot \beta_{x_{\text{Kr2}}}$
I_{Ks}	x_{Ks}	$\frac{1}{1 + e^{-(v+3.8)/14}}$	$\frac{990}{1 + e^{-(v+2.4)/14}}$		$\alpha_{x_{\text{Ks}}}$
I_{f}	x_{f}	$\frac{1}{1 + e^{(v+78)/5}}$	$\frac{1900}{1 + e^{(v+15)/10}}$		$\alpha_{x_{\text{Ks}}}$

Table S7: Specification of the parameters z_∞ and τ_z , for $z = m, j, m_L, h_L, d, f, f_{\text{Ca}}, q_{\text{to}}, r_{\text{to}}, x_{\text{Kr1}}, x_{\text{Kr2}}, x_{\text{Ks}}$ and x_{f} in the equations for the gating variables (S2).

S2 Definition of biomarkers

S2.1 Conduction velocity (CV)

In the 1D strand simulations in Figures 2–3, the first cell of the cell strand (or a corresponding length) is stimulated. The conduction velocity is computed as the distance between the centers of cells 10 and 20 divided by the duration of time between when the membrane potential in these two points reaches a value above -20 mV.

S2.2 Action potential duration (APD)

The action potential duration at 90% repolarization is computed from the membrane potential in the center of cell number 10. It is defined by a threshold membrane potential value,

$$v_{\text{th}} = v_{\text{max}} - 0.9(v_{\text{max}} - v_{\text{min}}), \quad (\text{S54})$$

where v_{max} and v_{min} are the maximum and minimum values, respectively, of the membrane potential obtain over time. The APD90 value is defined as the difference in time between when the membrane potential crosses this threshold value during the upstroke/depolarization phase of the action potential and during the repolarization phase (see Figure S3).

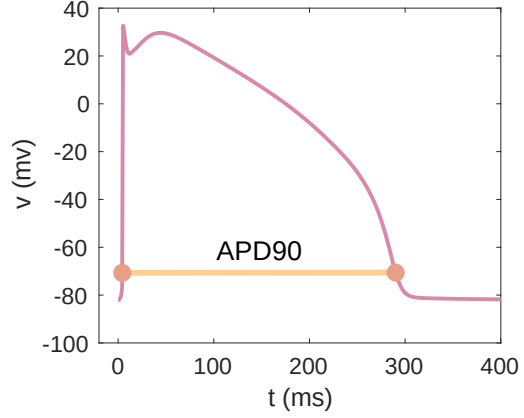


Figure S3: Definition of APD90.

S2.3 Time of arrival (ToA)

The time of arrival in a spatial point is defined as the point in time when the membrane potential in that point reaches a value ≥ 0 mV.

S3 Numerical implementation of the models

In this subsection we describe the implementation of the numerical solution algorithms for the five models EMI, KNM, SKNM, BD, and MD.

S3.1 Implementation of KNM and SKNM

The numerical solution of KNM and SKNM are implemented in C++. A classical first-order operator splitting scheme is used to split the linear and the non-linear parts of the systems (see, e.g., [2]). Here the non-linear part of the systems is the part concerning the membrane dynamics, I_{ion}^k and F_k . These non-linear ordinary differential equations are solved using a first-order Rush-Larsen method [3] with code generated using the Gotran code generator [4] and OpenMP parallelization [5]. The remaining linear equations are solved using the backward Euler method. The resulting linear systems are solved using the biconjugate gradient stabilized method (BiCGSTAB) from the MFEM finite element library [6, 7]. For KNM, we use a block Jacobi preconditioner [8]. Unless otherwise stated, we use the time step $\Delta t = 10 \mu\text{s}$ in the solution of KNM and SKNM.

S3.2 Implementation of BD and MD

The numerical solution of BD and MD are also implemented in C++ using the MFEM finite element library [6, 7]. The meshes are generated using gmsh [9], and we use first-order finite elements. Like for KNM and SKNM, we split the system of equations into linear and non-linear parts using a first-order operator splitting scheme [2]. Again, the non-linear part of the systems is the part concerning the membrane dynamics, I_{ion} and F , and these are solved using a first-order Rush-Larsen method [3] with code generated using the Gotran code generator [4] and OpenMP parallelization [5]. The remaining linear equations are solved using the backward Euler method and the linear systems are solved using the generalized minimal residual method (GMRES). For BD, we use a block Jacobi preconditioner [8]. Unless otherwise stated, we use the time step $\Delta t = 10 \mu\text{s}$ and the spatial resolution $\Delta x = 10 \mu\text{m}$ in the solution of BD and MD.

S3.3 Implementation of EMI

The EMI meshes are generated using gmsh [9]. To solve the EMI equations, we apply the temporal and spatial splitting algorithm introduced in [10] (see Algorithm 1). In this splitting algorithm, the non-linear membrane dynamics

Initial conditions: $v^{k,0}, s^{k,0}, w^{k,0}, u_e^0$, for all k .

for $n = 1, \dots, N_t$:

Step 1: For all k , find $s^{k,n}$ and $\bar{v}^k$ at the nodes of the membrane Γ_k by solving a time step Δt from $(s^{k,n-1}, v^{k,n-1})$ of

$$v_t^k = -\frac{1}{C_m} I_{\text{ion}}(v^k, s^k), \quad (\text{S55})$$

$$s_t^k = F(v^k, s^k). \quad (\text{S56})$$

Define $\bar{u}_e = u_e^{n-1}$, $\bar{w}^k = w^{k,n-1}$ for all k .

for $j = 1, \dots, N_{\text{it}}$:

Step 2:

for $m = 1, \dots, M_{\text{it}}$:

For every k , find $\bar{u}_i^k$ by solving

$$\nabla \cdot \sigma_i \nabla \bar{u}_i^k = 0 \quad \text{in } \Omega_i^k, \quad (\text{S57})$$

$$\bar{u}_i^k + \frac{\Delta t}{C_m} n_i^k \cdot \sigma_i \nabla \bar{u}_i^k = \bar{v}^k + \bar{u}_e \quad \text{at } \Gamma_k, \quad (\text{S58})$$

$$-n_i^k \cdot \sigma_i \nabla \bar{u}_i^k = \frac{1}{R_g} \bar{w}^k + C_g \frac{\bar{w}^k - w^{k,n-1}}{\Delta t} \quad \text{at } \Gamma_{k,\tilde{k}}, \quad (\text{S59})$$

where $\tilde{k}$ denotes each of the neighboring cells of cell k .

Update $\bar{w}^k = \bar{u}_i^k - \bar{u}_i^k$ at $\Gamma_{k,\tilde{k}}$ for all k and $\tilde{k}$.

end

Step 3: Find $\bar{u}_e$ by solving

$$\nabla \cdot \sigma_e \nabla \bar{u}_e = 0 \quad \text{in } \Omega_e, \quad (\text{S60})$$

$$\bar{u}_e = 0 \quad \text{at } \partial\Omega_e^D, \quad (\text{S61})$$

$$n_e \cdot \sigma_e \nabla \bar{u}_e = 0 \quad \text{at } \partial\Omega_e^N, \quad (\text{S62})$$

$$n_e \cdot \sigma_e \nabla \bar{u}_e = -n_i^k \cdot \sigma_i \nabla \bar{u}_i^k \quad \text{at } \Gamma_k \text{ for all } k. \quad (\text{S63})$$

end

Define $u_e^n = \bar{u}_e$, $u_i^{k,n} = \bar{u}_i^k$, $w^{k,n} = \bar{w}^k$ for all k .

Step 4: Define $v^{k,n} = u_i^{k,n} - u_e^n$ at Γ_k for all k .

end

Algorithm 1: Spatial and temporal splitting algorithm applied to solve the EMI model equations, taken from [10, 11]. Here, $\bar{v}^k$, $\bar{u}_e$, $\bar{w}^k$, and $\bar{u}_i^k$ are temporary solutions of v^k , u_e , w^k , and u_i^k , respectively. Moreover, N_{it} and M_{it} denote the number of outer and inner iterations of the algorithm. In our computations, we use $N_{\text{it}} = M_{\text{it}} = 1$.

(I_{ion}^k and F_k) are treated in Step 1 like in the operator splitting for KNM, SKNM, BD and MD. We use a first-order Rush-Larsen method [3] with code generated using the Gotran code generator [4] and OpenMP parallelization [5] to solve these ordinary differential equations.

In addition, operator splitting techniques are applied so that in each time step, the linear system of model can be split spatially into distinct parts that can be handled separately [10]. More specifically, the linear system is split into one equation for each cell (Step 2) that may be treated independently of each other (i.e., in parallel). In addition, we get one system for the extracellular part of the domain that can be handled separately (Step 3). Consequently, the sub-problems needed to be solved for each time step are standard Laplace problems with Dirichlet, Neumann or Robin boundary conditions. We have used one iteration of this algorithm for each time step, but the accuracy of the splitting algorithm could potentially be improved by increasing the number of iterations (see [10]).

In our implementation, we use first-order finite elements from the MFEM finite element library [6, 7]. The resulting linear systems for the intracellular and extracellular systems are solved using the conjugate gradient method (CG) with a Gauss-Seidel preconditioner [8]. We also consider the use of CG with an algebraic multigrid (AMG) preconditioner, namely the BoomerAMG implementation from *hypre* [12], as well as two different sparse direct solvers, UMFPACK [13] and SuperLU [14]. BoomerAMG and SuperLU are parallelised using OpenMP, whereas the other solvers do not benefit from multi-threading. The intracellular systems are solved in parallel using OpenMP [5]. Unless otherwise stated, we use the time step $\Delta t = 10 \mu\text{s}$ and the spatial resolution $\Delta x = 5 \mu\text{m}$ in the solution of EMI.

S4 Parallel solvers for EMI

As noticed in Figure 4 in the paper, the computation time for the EMI solver is very significantly longer than for the other models. This is reasonable since EMI offers sub-cellular resolution whereas the other models average many cells (MD, BD) or represent every cell (KNM, SKNM) without sub-cellular resolutions. Since EMI clearly is the most accurate representation of the biophysics involved, it is of great importance to speed up the simulations. In Algorithm 1, the EMI model has been broken down to components that invite parallel solution in the sense that all computations can be performed independently for every cell except for the extracellular potential that binds everything together. In this section, we consider steps to speed up this part of the computations. In particular, we describe numerical experiments

Threads	CG-Gauss-Seidel	CG-AMG (hypre)	UMFPACK	SuperLU
1	127.90	21.26	15.69	25.10
2	–	12.34	–	18.77
4	–	7.60	–	15.71
8	–	6.39	–	15.82
16	–	5.04	–	16.43
24	–	5.23	–	17.37

Table S8: Execution time (in seconds) for solving the extracellular equations. We perform 10 time steps of the EMI model using the same setup as in Figure 4, where the extracellular system consists of 453 572 unknowns. Four different solvers are used, and we report the total execution time for all time steps. The measurements are obtained on a 24-core AMD Epyc 7413 CPU, where up to 24 threads are used if the solver is parallel (i.e., CG-AMG and SuperLU). Otherwise, only the serial execution time is reported.

concerned with the shared-memory parallel performance of the EMI solver implementation. The experiments are carried out on a 24-core AMD Epyc 7413 CPU.

Recall that the problem described in Figure 4 requires about 16 hours to simulate a configuration of 13×65 cells for 4 000 time steps, using a sequential CG-Gauss-Seidel solver for the linear system arising from the extracellular equation each time. A profiling shows that more than 80 % of the execution time is spent on solving the extracellular equations (S60)–(S63), about 10 % on solving the intracellular equations (S57)–(S59), a mere 1 % on solving the membrane model (S55)–(S56), and a small, remaining fraction is mostly spent on moving data and updating the solution between time steps and solver iterations.

Based on this observation, we first consider alternative solvers for the extracellular equations. In addition to CG with Gauss-Seidel preconditioning, we also consider hypre (i.e., CG with AMG preconditioning) as well as the sparse direct solvers UMFPACK and SuperLU. Table S8 shows execution times for solving the extracellular equations for 10 time steps using the different solvers. Only sequential execution times are reported for CG-Gauss-Seidel and UMFPACK, whereas parallel execution times using up to 24 threads are reported for hypre and SuperLU.

Speed-up using multiple threads. In the sequential case, hypre, UMFPACK and SuperLU yield speedup factors of 6.0, 8.2 and 5.1, respectively,

over the baseline solver with CG and Gauss-Seidel preconditioning. SuperLU benefits from using up to 4 or 8 threads, which results in about 60 % improvement over its sequential runtime. At this point, SuperLU reaches the same performance as UMFPACK, about 15.7 seconds. Additional threads do not provide any further benefit and only result in poor parallel efficiency. By comparison, hypre benefits from using up to 16 threads and attains a speedup of $4.2\times$ over its sequential runtime. It is thus the fastest among the candidates considered for solving the extracellular equations in parallel, and it results in a speedup of $25\times$ compared to the sequential baseline solver.

Direct vs. iterative solvers. In the case of UMFPACK and SuperLU, the reported times include both the initial factorization as well as solving triangular linear systems during each time step. Although factorization is performed only once, it accounts for nearly half of the runtime for UMFPACK in the current example. On the other hand, if the number of time steps increases, more time will be spent on performing the triangular solves, and the factorization time eventually becomes negligible. By this reasoning, direct solvers can be regarded as more competitive than they would seem at first. Supposing that we ignore factorization, then the execution time for UMFPACK is 8.5 seconds. So, even in this case, hypre remains the fastest alternative.

Parallel performance of EMI model. Besides the extracellular equations (S60)–(S63), parallel processing is also beneficial when solving the intracellular equations (S57)–(S59) and the membrane model (S55)–(S56). Table S9 shows the execution time when using up to 24 threads to solve the EMI model for 10 time steps. The top half of the table shows the initial parallel performance of the code, whereas the bottom half shows the parallel performance after applying certain code optimizations that are discussed in detail below. The execution times are also broken down into five main parts, including solving the membrane model, as well as assembling and solving linear systems for the intracellular and extracellular equations. Based on our earlier findings, hypre is used to solve the linear systems arising from the extracellular equations.

Good speed-up for membrane and intracellular models. Both membrane model and intracellular solvers show nearly perfect scalability with speedups of $21.2\times$ and $20.8\times$, respectively, when using 24 threads. As we saw earlier, the extracellular solve exhibits a more modest speedup of $4.6\times$. This is expected for iterative solvers of large, sparse linear systems [15], whose

Threads	Total	Membrane	Intracellular		Extracellular	
			Assembly	Solve	Assembly	Solve
1	72.704	2.669	13.947	5.048	20.994	22.238
2	58.738	1.346	13.966	2.551	21.012	12.114
4	52.247	0.663	13.949	1.276	21.016	7.556
8	50.047	0.337	13.948	0.658	20.959	6.425
16	47.888	0.172	13.975	0.337	20.990	4.765
24	47.797	0.126	13.947	0.243	20.943	4.808
1	61.125	2.603	9.371	4.972	15.969	21.303
2	33.505	1.323	4.747	2.504	8.455	11.861
4	19.582	0.622	2.388	1.259	4.744	7.187
8	16.915	0.337	1.212	0.644	5.184	6.406
16	14.569	0.176	1.050	0.336	5.251	4.917
24	16.896	0.120	0.998	0.254	7.485	5.170

Table S9: Execution time (in seconds) for solving the EMI model.

We perform 10 time steps of the EMI model with the same setup as in Figure 4 and using hypre to solve the extracellular equations. Measurements are obtained on a 24-core AMD Epyc 7413 CPU, where up to 24 threads are used. The top half of the table shows the initial parallel performance before optimization and parallelization of the right-hand side assembly, whereas the bottom half of the table shows the parallel performance after optimizing and parallelizing the assembly of right-hand sides.

Cells	Total	Membrane	Intracellular		Extracellular	
			Assembly	Solve	Assembly	Solve
13×65	16.90	0.12	1.00	0.25	7.49	5.17
26×65	34.65	0.25	2.20	0.49	13.48	12.16

Table S10: Execution time (in seconds) for solving the EMI model for different meshes. We perform 10 time steps of the EMI model, using hypre to solve the extracellular equations. Measurements are obtained using 24 OpenMP threads on a 24-core AMD Epyc 7413 CPU.

performance is limited by memory bandwidth, a resource that is shared by all cores. Solving the intracellular equations does not suffer from this issue due to solving many, small linear systems that are likely to fit within on-chip caches on each CPU core.

Inefficient assembly of right-hand sides. In any case, a much bigger concern is posed by the *assembly* of right-hand side vectors for the extracellular and intracellular equations, which takes place during every time step. (Note that left-hand side matrices are assembled only once and their contribution to the overall simulation time is negligible.) In the sequential case, assembly of right-hand sides accounts for half of the overall execution time. Moreover, this part of the computation does not currently benefit from parallel execution, and therefore constitutes a larger share of the overall execution time as more threads are used. Next, we therefore describe code optimizations that aim to parallelize the assembly of right-hand side vectors.

Parallel assembly for the intracellular equations. The intracellular linear systems (S57)–(S59) are all independent, and multiple threads can therefore be used to assemble each of the right-hand sides independently and in parallel. In principle, this is achieved by adding an OpenMP parallel region and worksharing-loop construct (i.e., `#pragma omp parallel for`) to the loop that assembles the intracellular right-hand sides. Unfortunately, this alone is not enough, as we observe severe slowdowns when increasing the number of threads. We traced the cause to the OpenBLAS library, which uses locking to ensure thread-safety in a multithreaded setting. By default, MFEM relies on an external BLAS library to perform factorization and inversion of small, dense matrices during assembly.

To work around the issue, we disable the use of BLAS and fall back to

MFEM’s native routines for dense matrix inversion. The result is, first, a performance improvement of 30–50 % from avoiding the overhead of invoking BLAS to invert small matrices. Second, and more importantly, assembly of right-hand sides can now be performed in parallel for the intracellular equations, and it scales almost perfectly up to 8 threads, see the lower part of Table S9. It is likely that 8 threads are sufficient to saturate the memory bandwidth, and additional threads therefore have little impact on the performance.

Parallel assembly for the extracellular equations. Right-hand side assembly for the extracellular equations (S60)–(S63) is sequential, and limitations in MFEM prevent parallelization for the particular case of using tetrahedral mesh elements. We work around this limitation by adding our own OpenMP-parallel version of the right-hand side assembly, where atomic operations are used to avoid race conditions from multiple threads updating the same element in the right-hand side vector. Ultimately, parallelization speeds up the right-hand side assembly for the extracellular equations by 3.37 times when using 4 threads. Additional threads degrade the performance because of contention and overhead associated with the use of atomic operations.

Due to MFEM being a general finite element framework that supports different kinds of meshes and elements, we observe a non-trivial overhead in its implementation of assembly of right-hand side vectors. In the future, we plan to investigate if a simpler, bespoke code can provide the features that we need while offering further opportunities for optimizing the code.

Feynmann optimality. Finally, Table S10 compares the execution time of the parallel EMI solver for tissues of two different sizes. More specifically, we double the size of the tissue to $2.6\text{ mm} \times 1.3\text{ mm}$ and the number of cells to 26×65 . Roughly speaking, the total execution time also doubles, which indicates that the amount of work performed by the solver is proportional to the tissue area and number of cardiac cells. In [16], Feynman formulated a rule for the complexity of a simulation stating that the computing efforts should only increase linearly with the space-time volume under consideration. For the EMI model, this means that the CPU efforts should only increase linearly with the number of cardiomyocytes and the time interval under consideration. Table S10 indicates that this is the case in these simulations.

S4.1 MFEM configuration

The numerical experiments presented in Section S4 use MFEM 4.5.2, which is configured with the following options:

- `CMAKE_BUILD_TYPE=Release`
- `MFEM_USE_MPI=YES`
- `MFEM_USE_OPENMP=YES`
- `MFEM_THREAD_SAFE=YES`
- `MFEM_USE_LAPACK=NO`
- `MFEM_USE_METIS=YES` and `MFEM_USE_METIS_5=YES`
- `MFEM_USE_HYPRE=YES`
- `MFEM_USE_SUITESPARSE=YES`
- `MFEM_USE_SUPERLU=YES`

We note especially that the option `MFEM_THREAD_SAFE=YES` is required to ensure thread-safe versions of certain MFEM functions are used. This is necessary due to assembling and solving multiple linear systems in parallel. In addition, the use of BLAS to invert small, dense matrices during assembly of right-hand side vectors is disabled with the option `MFEM_USE_LAPACK=NO`.

Besides MFEM, several additional software packages are needed, including METIS 5.1.0, hypre 2.26.0, SuiteSparse 5.12.0, SuperLU_dist 8.1.0 and OpenMPI 4.1.4. We further note that OpenMP support is enabled in hypre (i.e., `--with-openmp`) and SuperLU_dist (i.e., `enable_openmp=ON`). The code is compiled using GCC 11.4.0.

Finally, the listing below shows the parallel code that was used for assembling right-hand side vector of the extracellular equations. The code is based on the function `LinearForm::Assemble()` from MFEM. It is somewhat simplified because only boundary integrals are required for our particular case. A parallel region and worksharing-loop construct were added, as well as atomic operations to avoid race conditions when updating the entries of the right-hand side vector. Furthermore, we note that calls to the function `Mesh::GetBdrElementTransformation()` were substituted with a thread-safe version to avoid race conditions and thus ensure correct results.

```

void linearform_assemble_omp(LinearForm & b)
{
    b = 0.0;

    /* Only boundary integrals are needed in our case. */
    Array<LinearFormIntegrator*> *boundary_integs = b.GetBLFI();
    if (boundary_integs->Size())
    {
        FiniteElementSpace *fes = b.FESpace();
        Mesh *mesh = fes->GetMesh();

        #pragma omp parallel for
        for (int i = 0; i < fes -> GetNBE(); i++)
        {
            Array<int> vdofs;
            IsoparametricTransformation eltrans;
            DofTransformation *doftrans;
            Vector elemvect;

            mesh -> GetBdrElementTransformation (i, &eltrans);
            for (int k=0; k < boundary_integs->Size(); k++)
            {
                (*boundary_integs)[k]->AssembleRHSElementVect(
                    *fes->GetBE(i), eltrans, elemvect);
                double * bx = b.HostReadWrite();
                double * elemvectx = elemvect.HostReadWrite();
                for (int l = 0; l < vdofs.Size(); l++) {
                    int j = vdofs[l];
                    if (j >= 0) {
                        #pragma omp atomic
                        bx[j] += elemvectx[l];
                    } else {
                        #pragma omp atomic
                        bx[-1-j] -= elemvectx[l];
                    }
                }
            }
        }
    }
}

```

References

- [1] Karoline H Jæger, Verena Charwat, Bérénice Charrez, Henrik Finsberg, Mary M Maleckar, Sam Wall, Kevin Healy, and Aslak Tveito. Improved computational identification of drug response using optical measurements of human stem cell derived cardiomyocytes in microphysiological systems. *Frontiers in Pharmacology*, 10:1648, 2020.
- [2] Joakim Sundnes, Glenn Terje Lines, and Aslak Tveito. An operator splitting method for solving the bidomain equations coupled to a volume conductor model for the torso. *Mathematical Biosciences*, 194(2):233–248, 2005.
- [3] Stanley Rush and Hugh Larsen. A practical algorithm for solving dynamic membrane equations. *IEEE Transactions on Biomedical Engineering*, 4:389–392, 1978.
- [4] Johan Hake, Henrik Finsberg, Kristian Gregorius Hustad, and George Bahij. Gotran – General ODE TRANslator, 2020. <https://github.com/ComputationalPhysiology/gotran>.
- [5] Leonardo Dagum and Ramesh Menon. OpenMP: An industry-standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998.
- [6] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cervený, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. MFEM: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021.
- [7] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cervený, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, et al. MFEM: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021.
- [8] Joachim Berdal Haga, Harald Osnes, and Hans Petter Langtangen. Efficient block preconditioners for the coupled equations of pressure and deformation in highly discontinuous media. *International Journal for Numerical and Analytical Methods in Geomechanics*, 35(13):1466–1482, 2011.

- [9] Christoph Geuzaine and Jean-François Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79:1309–1331, 2009.
- [10] Karoline H Jæger, Kristian Gregorius Hustad, Xing Cai, and Aslak Tveito. Efficient numerical solution of the EMI model representing the extracellular space (E), cell membrane (M) and intracellular space (I) of a collection of cardiac cells. *Frontiers in Physics*, 8:539, 2021.
- [11] Karoline H Jæger, Kristian Gregorius Hustad, Xing Cai, and Aslak Tveito. Operator splitting and finite difference schemes for solving the EMI model. In *Modeling Excitable Tissue*, pages 44–55. Springer, Cham, 2020.
- [12] hypre: High performance preconditioners. <https://llnl.gov/casc/hypre>, <https://github.com/hypre-space/hypre>.
- [13] Timothy A. Davis. Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, 2004.
- [14] Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2):110–140, 2003.
- [15] Hartwig Anzt, Erik Boman, Rob Falgout, Pieter Ghysels, Michael Heroux, Xiaoye Li, Lois Curfman McInnes, Richard Tran Mills, Sivasankaran Rajamanickam, Karl Rupp, Barry Smith, Ichitaro Yamazaki, and Ulrike Meier Yang. Preparing sparse solvers for exascale computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190053, 2020.
- [16] Richard P Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982.