Computer Methods in Applied Mechanics and Engineering 423 (2024) 116875

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

A comparison of Algebraic Multigrid Bidomain solvers on hybrid CPU–GPU architectures

Edoardo Centofanti [a,*], Simone Scacchi [b]

[a] Dipartimento di Matematica, Università di Pavia, Via Adolfo Ferrata, 5, Pavia, 27100, Italy
[b] Dipartimento di Matematica, Università di Milano, Via Cesare Saldini, 50, Milano, 20133, Italy

ARTICLE INFO

Keywords:
Bidomain model
Finite Element Method
Parallel solvers
Algebraic multigrid preconditioners
Computational Electrocardiology
IMEX Methods

ABSTRACT

The numerical simulation of cardiac electrophysiology is a highly challenging problem in scientific computing. The Bidomain system is the most complete mathematical model of cardiac bioelectrical activity. It consists of an elliptic and a parabolic partial differential equation (PDE), of reaction–diffusion type, describing the spread of electrical excitation in the cardiac tissue. The two PDEs are coupled with a stiff system of ordinary differential equations (ODEs), representing ionic currents through the cardiac membrane. Developing efficient and scalable preconditioners for the linear systems arising from the discretization of such computationally challenging model is crucial in order to reduce the computational costs required by the numerical simulations of cardiac electrophysiology. In this work, focusing on the Bidomain system as a model problem, we have benchmarked two popular implementations of the Algebraic Multigrid (AMG) preconditioner embedded in the PETSc library and we have studied the performance on the calibration of specific parameters. We have conducted our analysis on modern HPC architectures, performing scalability tests on multi-core and multi-GPUs settings. The results have shown that, for our problem, although scalability is verified on CPUs, GPUs are the optimal choice, since they yield the best performance in terms of solution time.

1. Introduction

Developing physiologically and morphologically realistic and detailed computer models of integrated cardiac function is an important research area, which may lead to the development of personalized diagnostic techniques and targeted therapies, see e.g. [1–6]. Also multiscale data available to research can be included in these in-silico models in order to reach such an ambitious goal. The main drawback to this approach consists in the associated computational cost, which can rapidly become important in terms of solution time of the linear (or non linear) systems deriving from the discretization of the mathematical models employed to this purpose. High-performance computing (HPC) resources are crucial in facing this kind of problems, but great effort is still necessary to maximize their efficiency and ensure their sustainable utilization, minimizing waste and environmental impact. In particular, over the last decade, General-Purpose Graphic Processing Units (GPGPUs) have shown an extraordinary computational power with respect to using a large number of CPUs ($O(10^2)$-$O(10^4)$) to solve large scale (about $O(10^6) - O(10^7)$ DOFs) sparse linear systems. This has led to GPUs being a significant item in the budgets of companies and institutions interested in assembling an HPC cluster. Apart from computational resources, remarkable efforts have been made also in the development of efficient numerical schemes and techniques aimed at reducing the number of iterations of an iterative method employed to solve a large linear system. This is the case of the preconditioners, which can be tailored according to the problem to be addressed. The Algebraic Multigrid

* Corresponding author at: Dipartimento di Matematica, Università di Pavia, Via Adolfo Ferrata, 5, Pavia, 27100, Italy.
E-mail address: edoardo.centofanti01@universitadipavia.it (E. Centofanti).

https://doi.org/10.1016/j.cma.2024.116875
Received 17 November 2023; Received in revised form 23 January 2024; Accepted 21 February 2024
Available online 27 February 2024

(AMG) [7] is a versatile method that can be employed as an effective preconditioner for reducing Conjugate Gradient (CG) iterations when solving particularly expensive linear systems arising from the discretization of elliptic partial differential equations (PDEs), which are largely employed in the mathematical modeling of cardiac electrophysiology. In this work, we focused our efforts on solving efficiently the cardiac Bidomain model, a macroscopic representation of the cardiac tissue modeling the spatio-temporal evolution of the intra- and extracellular electric potentials. In the formulation considered throughout the work, it consists of two PDEs, an elliptic and a parabolic one, coupled with a system of ordinary differential equations (ODEs) representing the ionic currents through the cellular membrane. The model homogenizes both the intra- and the extracellular space, namely in each physical degree of freedom (DOF) both spaces coexist. Previous work on solvers involving AMG applied to the Bidomain system are [8–12], but other well known preconditioners for elliptic systems have been successfully tested and employed for this problem, such as geometric multigrid and domain decomposition preconditioners, as Multilevel Schwarz [13,14], Neumann Neumann [15,16] and BDDC [17,18]. The novelty of this work consists in having chosen and tested two different implementations of the AMG preconditioner and having benchmarked their performances on an HPC architecture involving both CPUs and GPUs, in order to verify if GPUs are effective for this particular problem and setup. A previous work on this topic [19] has shown a speedup of roughly a factor 10 when employing GPUs with respect to the best performance on CPU, but the benchmark is limited up to 20 GPUs and the Hypre implementation of AMG considered, BoomerAMG, was not yet implemented in order to run on GPUs. In this work we will also expand some of those results exploiting our *in-house* codebase, which considers a slightly different numerical scheme.

The rest of the paper is organized as follows: In Section 2 the Bidomain model is presented and the numerical schemes employed are explained and commented; in Section 3 we present the parameters related to our problem; in Section 4 we describe the AMG implementations and the importance of two different threshold parameters employed; in Section 5 we present the simulation setup and the HPC architectures considered; in Sections 6 and 7 we present the numerical tests performed and the corresponding results.

2. Model

The model studied in the following is the macroscopic Bidomain model of electrocardiology in the parabolic–elliptic formulation [20–23]. Denoting by $\Omega \subset \mathbb{R}^3$ the open, connected and bounded physical region occupied by the portion of myocardium of our interest, we represent the tissue as the superimposition of two anisotropic continuous media, called intra- and extracellular media. In this model, it is assumed that they coexist at every point and are separated by a distributed continuous cellular membrane. Given Ω and a time interval $(0,T) \in \mathbb{R}^+$, $T \in \mathbb{R}^+$, our problem consists of finding the extracellular potential $u_e : \Omega \times (0,T) \to \mathbb{R}$, the transmembrane potential $v : \Omega \times (0,T) \to \mathbb{R}$ and the gating and ionic concentration variables $\mathbf{w} : \Omega \times (0,T) \to \mathbb{R}^{N_w}$ and $\mathbf{c} : \Omega \times (0,T) \to \mathbb{R}^{N_c}$, respectively, such that:

$$\begin{cases} \chi C_m \dfrac{\partial v}{\partial v} - \operatorname{div}(D_i \nabla v) - \operatorname{div}(D_i \nabla u_e) + I_{ion}(v, \mathbf{w}, \mathbf{c}) = I^i_{app} & \text{in } \Omega \times (0,T), \\ -\operatorname{div}(D_i \nabla v) - \operatorname{div}((D_i + D_e)\nabla u_e) = I^i_{app} + I^e_{app} & \text{in } \Omega \times (0,T), \\ \dfrac{\partial \mathbf{w}}{\partial t} - \mathbf{R}(v, \mathbf{w}) = 0 & \text{in } \Omega \times (0,T), \\ \dfrac{\partial \mathbf{c}}{\partial t} - \mathbf{C}(v, \mathbf{w}, \mathbf{c}) = 0 & \text{in } \Omega \times (0,T), \\ \mathbf{n}^\top D_i \nabla (v + u_e) = 0 & \text{in } \partial\Omega \times (0,T), \\ \mathbf{n}^\top D_e \nabla u_e = 0 & \text{in } \partial\Omega \times (0,T), \\ v(\mathbf{x},0) = v_0(\mathbf{x}), \quad \mathbf{w}(\mathbf{x}),0 = \mathbf{w}_0(\mathbf{x}), \quad \mathbf{c}(\mathbf{x}),0 = \mathbf{c}_0(\mathbf{x}) & \text{in } \Omega, \end{cases} \tag{1}$$

where I_{ion} is the ionic current related to the ionic model. In this work we have employed the ten Tusscher–Panfilov [24] model (TP06). The functions $\mathbf{R}$ and $\mathbf{C}$ describe the dynamics of the gating and ionic concentration variables $\mathbf{w}$ and $\mathbf{c}$, respectively. I^i_{app} and I^e_{app} denote the applied currents in the intra- and extracellular region, respectively, that must satisfy the compatibility condition

$$\int_\Omega (I^i_{app} + I^e_{app})\, d\mathbf{x} = 0.$$

The anisotropy of the tissue, due to the fiber arrangement of cardiac myocytes (see e.g. [25]), is described by the conductivity tensors $D_{i,e}(\mathbf{x})$ at any point $\mathbf{x} \in \Omega$, which have the following definition:

$$\begin{aligned} D_{i,e}(\mathbf{x}) &= \sigma^{i,e}_l \mathbf{a}_l(\mathbf{x})\mathbf{a}_l^\top(\mathbf{x}) + \sigma^{i,e}_t \mathbf{a}_t(\mathbf{x})\mathbf{a}_t^\top(\mathbf{x}) + \sigma^{i,e}_n \mathbf{a}_n(\mathbf{x})\mathbf{a}_n^\top(\mathbf{x}) \\ &= \sigma^{i,e}_t I + (\sigma^{i,e}_l - \sigma^{i,e}_t)\mathbf{a}_l(\mathbf{x})\mathbf{a}_l^\top(\mathbf{x}) + (\sigma^{i,e}_n - \sigma^{i,e}_t)\mathbf{a}_n(\mathbf{x})\mathbf{a}_n^\top(\mathbf{x}), \end{aligned} \tag{2}$$

where $\mathbf{a}_l$, $\mathbf{a}_t$ and $\mathbf{a}_n \in [L^\infty(\Omega)]^3$ represent a triple of orthonormal principal axes, with $\mathbf{a}_l$ parallel to the local fiber direction, $\mathbf{a}_n$ and $\mathbf{a}_t$ orthogonal and tangent to the radial laminae, respectively, and both transversal to the fiber axis. Furthermore, we have denoted as $\sigma^{i,e}_l$, $\sigma^{i,e}_t$ and $\sigma^{i,e}_n$ the conductivity coefficients for the intra- and extracellular media along the corresponding directions $\mathbf{a}_l$, $\mathbf{a}_t$ and $\mathbf{a}_n$. Since the conductivity coefficients $\sigma^{i,e}_*$, $* = l,t,n$, are positive scalars, by definition of $D_{i,e}$ the operators $D_{i,e}(x)\nabla(\cdot)$ are uniformly elliptic. We recall that these tensors model the structure of the cardiac tissue, i.e. an ensemble of fibers rotating counterclockwise from epi- (the outermost protective layer of the heart) to endocardium (the innermost), organized as muscle foils running radially from epi- to endocardium.

2.1. Variational formulation

Let us define $V := H^1(\Omega)$ the usual Sobolev space and then define

$$\widetilde{V} = \{\psi \in V : \int_\Omega \psi = 0\},$$

$$U = V \times \widetilde{V} = \{u = (\phi, \psi) : \phi \in V, \psi \in \widetilde{V}\},$$

the L^2-inner product $(\phi, \psi) = \int_\Omega \phi\psi d\mathbf{x}\ \forall \phi, \psi \in L^2(\Omega)$, and the bilinear forms

$$a_{i,e}(\phi, \psi) = \int_\Omega (\nabla\phi)^\top D_{i,e}(\mathbf{x})\nabla\psi d\mathbf{x},$$

$$a(\phi, \psi) = \int_\Omega (\nabla\phi)^\top D(\mathbf{x})\nabla\psi d\mathbf{x}, \quad \forall \phi, \psi \in H^1(\Omega),$$

where $D = D_i + D_e$ is the bulk conductivity tensor.

The variational formulation of the Bidomain model reads as follows: given $v_0 \in L^2(\Omega)$, $\mathbf{w}_0 \in [L^2(\Omega)]^{N_w}$, $\mathbf{c}_0 \in [L^2(\Omega)]^{N_c}$, $I^{i,e}_{app} \in L^2(\Omega \times (0,T))$, find $v \in L^2(0,T;V)$, $u_e \in L^2(0,T;\widetilde{V})$, $\mathbf{w} \in L^2(0,T;[L^2(\Omega)]^{N_w})$ and $\mathbf{c} \in L^2(0,T;[L^2(\Omega)]^{N_c})$ such that for all $t \in (0,T)$

$$\begin{cases} c_m \dfrac{\partial}{\partial t}(v, \hat{v}) + a_i(v + u_e, \hat{v}) + (I_{ion}(v, \mathbf{w}, \mathbf{c}), \hat{v}) = (I^i_{app}, \hat{v}) & \forall \hat{v} \in V \\ a_i(v, \hat{u}_e) + a(u_e, \hat{u}_e) = 0 & \forall \hat{u}_e \in V \\ \dfrac{\partial}{\partial t}(w_j, \hat{w}) - (R_j(v, \mathbf{w}), \hat{w}) = 0, & \forall \hat{w} \in V,\ j = 1, \dots, N_w \\ \dfrac{\partial}{\partial t}(c_j, \hat{c}) - (C_j(v, \mathbf{w}, \mathbf{c}), \hat{c}) = 0, & \forall \hat{c} \in V,\ j = 1, \dots, N_c \end{cases}$$

with initial conditions $v = v_0$, $\mathbf{w} = \mathbf{w}_0$ and $\mathbf{c} = \mathbf{c}_0$ and where we have imposed $I^e_{app} = -I^i_{app}$.

2.2. Space and time discretization

Let $\mathcal{T}_h$ be a quasi-uniform tessellation of Ω having maximal diameter h and V_h be an associated conforming finite element space. Let us select a finite element basis $\{\phi_p\}_{p=1}^N$ of V_h, evaluate $M = \{m_{pj}\}$, the diagonal mass matrix, with the usual mass-lumping technique and $A_{i,e} = \{a^{i,e}(\phi_j, \phi_p)\}$ the symmetric intra- and extracellular stiffness matrices, having elements

$$a^{i,e}(\phi_j, \phi_p) = \int_\Omega D_{i,e}\nabla\phi_j \cdot \nabla\phi_p dx$$

By means of a standard Galerkin procedure, we can rewrite the semi-discrete Bidomain problem, discretized in space, in the following compact form:

$$\begin{cases} c_m \mathcal{M} \dfrac{d}{dt}\begin{bmatrix}\mathbf{v}\\ \mathbf{u}_e\end{bmatrix} + \mathcal{A}\begin{bmatrix}\mathbf{v}\\ \mathbf{u}_e\end{bmatrix} + \begin{bmatrix} M I_{ion}(\mathbf{v}, \mathbf{w}, \mathbf{c}) \\ \mathbf{0}\end{bmatrix} = \begin{bmatrix} M\mathbf{I}^i_{app} \\ \mathbf{0}\end{bmatrix} \\ \dfrac{d\mathbf{w}}{dt} = \mathbf{R}(\mathbf{v}, \mathbf{w}) \\ \dfrac{d\mathbf{c}}{dt} = \mathbf{C}(\mathbf{v}, \mathbf{w}, \mathbf{c}) \end{cases} \tag{3}$$

where we denote the block mass and stiffness matrices as

$$\mathcal{M} = \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \qquad \mathcal{A} = \begin{bmatrix} A_i & A_i \\ A_i & A_i + A_e \end{bmatrix}$$

and we define $\mathbf{v}$, $\mathbf{u}_e$, $\mathbf{w} = (\mathbf{w}_1, \dots, \mathbf{w}_{N_w})^\top$, $\mathbf{c} = (\mathbf{c}_1, \dots, \mathbf{c}_{N_c})^\top$, $R(\mathbf{v}, \mathbf{w}) = (R_1(\mathbf{v}, \mathbf{w}), \dots, R_{N_w}(\mathbf{v}, \mathbf{w}))^\top$, $C(\mathbf{v}, \mathbf{w}, \mathbf{c}) = (C_1(\mathbf{v}, \mathbf{w}, \mathbf{c}), \dots, C_{N_c}(\mathbf{v}, \mathbf{w}, \mathbf{c}))^\top$, $\mathbf{I}_{ion}(\mathbf{v}, \mathbf{w})$ and $\mathbf{I}^e_{app}$ as the coefficient vectors of finite element approximations of u_i, u_e, v, w_r, $R_r(v, w_1, \dots, w_{N_w})$, $C_r(v, w_1, \dots, w_{N_w}, c_1, \dots, c_{N_c})$, $I_{ion}(v, w_1, \dots, w_{N_w})$ and $I^{i,e}_{app}$ respectively. For this work in particular, we have discretized the equation through Q1 elements.

For the time discretization, we consider an implicit–explicit (IMEX) strategy, which consists of decoupling the ODEs from the PDEs and of treating the linear diffusion terms implicitly and the non-linear reaction terms explicitly. We then solve uncoupled the two Eqs. (3) in discretized form.

In particular, considering $\mathbf{v}^n$ and $\mathbf{u}^n_e$ at the timestep n, we solve the elliptic equation evaluating $\mathbf{u}^{n+1}_e$, and then we solve the parabolic equation in order to update the transmembrane potential $\mathbf{v}^{n+1}$. $\mathbf{w}^{n+1}$ and $\mathbf{c}^{n+1}$ are evaluated implicitly. We summarize the

Table 1
Geometric parameters of idealized LV domain.

Parameter	Value	Parameter	Value
a_1	2.2	$\theta_{\min}$	$-\frac{3\pi}{2}$
a_2	3.3	$\theta_{\max}$	$\frac{\pi}{2}$
b_1	2.2	$\phi_{\min}$	$-\frac{3\pi}{8}$
b_2	3.3	$\phi_{\max}$	$\frac{\pi}{8}$
c_1	5.9		
c_2	6.4		

Table 2
Number of points and elements for the "U-mesh" geometry.

Name	Physical DOFs	Elements
U-mesh 1	35,725	30,108
U-mesh 2	258,415	240,864
U-mesh 3	1,987,285	1,926,912

scheme, given $\mathbf{w}^n$, $\mathbf{v}^n$, $\mathbf{u}_e^n$, as

$$\begin{aligned}
&\mathbf{w}^{n+1} + \Delta t \mathbf{R}(\mathbf{v}^n, \mathbf{w}^{n+1}) && = \mathbf{w}^n \\
&\mathbf{c}^{n+1} + \Delta t \mathbf{C}(\mathbf{v}^n, \mathbf{w}^{n+1}, \mathbf{c}^{n+1}) && = \mathbf{c}^n \\
&(A_i + A_e)\mathbf{u}_e^n && = -A_i \mathbf{v}^n \\
&\left(\frac{c_m}{\Delta t} M + A_i\right) \mathbf{v}^{n+1} && = \frac{c_m}{\Delta t} M \mathbf{v}^n - A_i \mathbf{u}_e^n + \\
& && + M \mathbf{I}_{ion}(\mathbf{v}^n, \mathbf{w}^{n+1}, \mathbf{c}^{n+1}) + M \mathbf{I}_{app}^{i,n}.
\end{aligned}$$

Overall, at each timestep we solve once the linear system with matrix $A_i + A_e$, i.e. the discrete form of the elliptic equation, and once the linear system with matrix $\frac{c_m}{\Delta t} M + A_i$ deriving from the parabolic equation. Being the resulting systems very large, due to the number of DOFs considered, they must be solved with an iterative method. We have chosen in particular the Preconditioned Conjugate Gradient (PCG) method, since the matrix arising from the parabolic equation is symmetric positive definite whereas that arising from the elliptic equation is symmetric positive semi-definite. The preconditioners used for the parabolic and elliptic systems will be discussed in the next sections.

3. Parameters and setting

We have focused our study on two geometries: a truncated ellipsoid, modeling an idealized left ventricle (LV) and a realistic geometry, representing a patient specific LV at three different resolutions. The idealized LV is discretized by a structured hexahedral mesh, whereas the patient specific LV is discretized by an unstructured mesh consisting of irregular hexahedra.

The truncated ellipsoid is built following the parametric equations:

$$\begin{cases} x = a(r)\cos\theta\cos\phi, & \theta_{\min} \le \theta \le \theta_{\max} \\ y = b(r)\cos\theta\sin\phi, & \phi_{\min} \le \phi \le \phi_{\max} \\ z = c(r)\sin\phi, & 0 \le r \le 1, \end{cases} \tag{4}$$

where $a(r) = a_1 + r(a_2 - a_1)$, $b(r) = b_1 + r(b_2 - b_1)$ and $c(r) = c_1 + r(c_2 - c_1)$, with $a_{1,2}$, $b_{1,2}$, $c_{1,2}$ coefficients defining the main axes of the ellipsoid. The geometric parameters are reported in Table 1.

For what concerns the patient specific LV geometry, denoted in the following as "U-mesh" we have considered three different refinements, which are reported in Table 2. The mesh has been provided us by Marco Fedele at Mox Laboratory, Politecnico di Milano, and the fibers have been generated using the open-source code lifex-fiber [26], developed at Mox Laboratory, Politecnico di Milano; see also [4,27].

The ionic membrane model considered is the ten Tusscher-Panfilov model (TP06) [24,28]. The stimulus is applied for 1 ms with intensity of 350 mA/cm^3. Depending on the numerical tests performed, the simulation spans the first 5 ms of the excitation phase as well as 500 ms, corresponding to almost a full heartbeat, both with timestep of 0.05 ms.

4. Algebraic multigrid

We are interested in solving a problem of the form

$$A\mathbf{x} = \mathbf{f}, \tag{5}$$

with $A \in \mathbb{R}^{n\times n}$, $\mathbf{x}, \mathbf{f} \in \mathbb{R}^n$.

Being these kind of systems usually very large, using direct methods to solve them would mean employ an high complexity method (at most $O(n^3)$) which can take an unnecessary high amount of time.

As previously motivated, it is necessary to employ iterative methods and other strategies in order to get a faster, albeit approximate solution.

The main idea of the Algebraic Multigrid (AMG) algorithm is solving (5) by cycling through levels composed of coarse (i.e. smaller) linear systems and finding updates that, interpolated on the original space, improve the solution. In this way, the so called 'smooth error' e that is not eliminated through iterative relaxations, is removed by coarse-grid correction.

The idea is implemented by solving the residual equation $Ae = r$ on a coarser grid, then through interpolation the solution is brought back to the finer grid and finally the fine-grid approximation is updated $u \longleftarrow u + e$.

Given the matrix A with entries a_{ij}, for convenience sake, the matrix indices i, j are identified with grid points on a grid Γ, such that x_i denotes the value of $\mathbf{x}$ in (5) at the point i. The components playing a role in the AMG are the following:

1. M "grids", i.e. index sets $\Gamma = \Gamma^1 \supset \Gamma^2 \supset \ldots \supset \Gamma^M$.
2. M grid operators $A^1, A^2, \ldots, A^M$.
3. $M-1$ Interpolation operators $P^1, P^2, \ldots, P^{M-1}$.
4. $M-1$ Restriction operators $R^1, R^2, \ldots, R^{M-1}$.
5. $M-1$ Smoothers $S^1, S^2, \ldots, S^{M-1}$.

All these components are defined in the *setup phase* of the algorithm:

Algorithm 1 AMG setup phase

Set $\Gamma^1 = \Gamma$
for $k = 1;\ k < M;\ k++$ **do**
  Partition Γ^k into disjoint sets C^k and F^k.
  Set $\Gamma^{k+1} = C^k$.
  Define interpolation P^k.
  Define restriction R^k (often $R^k = (P^k)^\top$).
  $A^{k+1} \leftarrow R^k A^k P^k$
  Set up S^k.
end for

When the setup phase is completed, the algorithm proceeds with a recursively defined cycle.

Following the notation in [29], we call this phase 'MGV', since it is often addressed as 'Multigrid V-Cycle'. Other cycles (W and F) are possible, but in this work we will focus only on V. The steps are the following:

Algorithm 2 MGV($A^k, R^k, P^k, S^k, u^k, f^k$)

if $k == M$ **then**
  solve $A^M u^M = f^M$ with a direct solver.
else
  apply the smoother S^k μ_1 times to $A^k u^k = f^k$.
  $r^k \leftarrow f^k - A^k u^k$ ▷ Coarse grid correction step
  $r^{k+1} \leftarrow R^k r^k$
  apply MGV($A^{k+1}, R^{k+1}, P^{k+1}, S^{k+1}, e^{k+1}, r^{k+1}$) ▷ Recursive step
  $e^k \leftarrow P^k e^{k+1}$ ▷ Interpolation step
  $u^k \leftarrow u^k + e^k$ ▷ Correction
  apply the smoother S^k μ_2 times to $A^k u^k = f^k$.
end if

We note that the smoothing step is performed through a Richardson iteration of the form $u^k_{j+1} = u^k_j + \omega(S^k)^{-1}(f - Au^k_j)$, with S^k such that S can be $diag(A)$ (Jacobi), the lower part of A (Gauss–Seidel) or the ILU approx of A, and ω relaxation factor.

Throughout this work, we will employ two popular AMG implementations in the field of HPC and many-core systems: GAMG, which is the built-in AMG solver in the PETSc library [30] and BoomerAMG, provided within the Hypre library [31]. They are both contained in the PETSc library, the Portable, Extensible Toolkit for Scientific Computation, which includes a large suite of scalable parallel linear and nonlinear equation solvers, ODE integrators, and optimization algorithms [30].

A critical step consists of the construction of the restriction matrices R^k, which are involved in the coarsening phase.

Since algorithm 2 should not rely directly on the geometry of the domain, many algebraic techniques have been explored and developed over the years. Although many coarsening algorithms are available for the implementations employed [32–35], we have considered the ones suggested as default options which are, for GAMG, PETSc default AMG implementation, a modified Maximal Independent Set (MIS) algorithm [36,37], while for Hypre a Hybrid Maximal Independent Set (HMIS) algorithm [38].

Table 3
Technical data for Marconi100.

Theoretical peak performance	CPU (nominal/peak freq.)	691/791 GFlops
	GPU	31.2 TFlops
	Total	32 TFlops
Memory Bandwidth (nominal/peak freq.)	220/300 GB/s	

In the first phase of the modified MIS, A weighted graph is built from the nodes of the grid Γ, with weight on the edge between nodes i and j defined as $w_{ij} = \frac{a_{ij}}{\sqrt{a_{ii}a_{jj}}}$.

A threshold on the weight is thus set such that at each coarsening step all the edges with weight less that the threshold are cut. Then, a greedy MIS or an equivalent parallel implementation (for example Luby's algorithm [39]) is applied to the modified graph. We call S the resulting set. Clusters C_j are defined with the following procedure: for each $i \notin S$, $j \in S$, $i \in C_j$ if $w_{ij} = \max_{\tilde{j} \in S} w_{i\tilde{j}}$. Finally, prolongators for AMG are defined as

$$P_{ij} = \begin{cases} |C_j|^{-\frac{1}{2}} \text{ if } i \in C_j \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

In the first phase of HMIS instead, A is explored and for each row the coarsening nodes are chosen between the ones satisfying the condition

$$|a_{ij}| \geq \alpha \max_{k \neq i} |a_{ik}| \tag{7}$$

with α called *strong threshold* parameter. This brings to the definition of sets of *strongly connected nodes* which are used to define the sets C_j and prolongators (6).

In the following sections we will explore how the threshold and the strong threshold influence our code performances.

5. Architectures and technical setup

Except where otherwise stated, the numerical experiments have been performed on the Marconi100 cluster at CINECA laboratory. Marconi100 is a Linux Cluster with 980 nodes, each one including 2 × 16 cores IBM POWER9 AC922 @ 3.1 GHz and 4 NVIDIA Volta V100 GPUs with 16 GB memory per GPU and NVlink 2.0. Each node has a memory of 256 GB. In Table 3 are reported more technical details about its bandwidth. Computations have been performed using up to 16 nodes considering either CPU or GPU performance. The maximum number of cores was 512, while we used up to 64 GPUs. Our *in-house* code is written in C with CUDA kernels for solving the membrane model on the GPUs. The numerical aspects of the code are based on the PETSc library, while the communications between parallel processors is handled by MPI. Tests have been focused mainly on the performance and the behavior at different scales of the preconditioners used for the elliptic equation of the model. Therefore, we have tested two implementations of the AMG preconditioner, available in PETSc:

- The GAMG preconditioner, the default implementation in PETSc, on CPU (up to 512 cores, 16 nodes). We have also performed a few tests on GPU (up to 4 devices, 1 node), but our current setup did not allow to perform an exhaustive analysis on GPUs. In particular we have observed out of memory issues and a general suboptimal use of the device (GPU) resources with our implementation, likely related to high matrix–matrix product memory consumption on GPU with GAMG.
- The Hypre BoomerAMG preconditioner, on CPU (up to 512 cores, 16 nodes) and on GPU (up to 64 devices, 16 nodes).

Regarding the parabolic equation in the formulation considered, we have employed Block Jacobi preconditioner on CPU, while we have left the system unpreconditioned on GPU, since the Block Jacobi implementation on the device did not show a significant reduction in CG iterations and solution time compared to the unpreconditioned system. For the PETSc options employed in this work, refer to Appendix A. We have divided the results into subsections: For the structured mesh, in *Test 1* we tune the AMG threshold hyperparameter both for Hypre BoomerAMG and GAMG, in *Test 2* we have performed a strong scaling test, fixing a global dimension for the problem while changing the number of the CPUs and GPUs employed, while in *Test 3* we have performed a weak scaling test, keeping the local size (i.e. the size per processor or per GPU) of the problem unchanged, while varying the number of processors or GPUs.

For the unstructured mesh, in *Test 1* we have tuned the threshold like in the structured case, while in *Test 2* we have performed a strong scaling test, fixing the global size of the problem while varying the number of processors or GPUs for the three refinements of the mesh.

6. Numerical results on structured meshes

We have first considered the structured mesh, discretizing the truncated ellipsoid described in Section 2. Snapshots of the numerical solutions for the potentials on the epicardial surface as a structured mesh are shown in Fig. 5.

Table 4
AMG threshold calibration, structured mesh. Results for Hypre GPU solver. $\text{It}_{\text{ellip, mean}}$: average CG iterations per timestep for the elliptic solver. $T_{\text{ellip, mean}}$: average CG solution time (in s) per timestep for the elliptic solver.

Threshold	0.25	0.3	0.4	0.5	0.6	0.7
$\text{It}_{\text{ellip, mean}}$	23.30	19.29	13.92	22.81	24.57	32.11
$T_{\text{ellip, mean}}$ (s)	9.6E−02	7.4E−02	5.6E−02	1.0E−01	1.1E−01	1.5E−01

Table 5
AMG threshold calibration, structured mesh. Results for Hypre CPU solver. Same format as in Table 4.

Threshold	0.25	0.3	0.4	0.5	0.6	0.7
$\text{It}_{\text{ellip, mean}}$	6.84	7.76	11.90	6.27	8.84	11.30
$T_{\text{ellip, mean}}$ (s)	2.9E−02	3.1E−02	4.5E−02	2.4E−02	3.1E−02	3.6E−02

Table 6
AMG threshold calibration, structured mesh. Results for GAMG CPU solver. Same format as in Table 4.

Threshold	0.0	0.01	0.02	0.03	0.04	0.05	0.06	0.07
$\text{It}_{\text{ellip, mean}}$	143.1	66.17	64.03	52.47	44.85	42.18	9.50	9.64
$T_{\text{ellip, mean}}$ (s)	4.4E−01	1.6E−01	1.7E−01	1.8E−01	1.8E−01	1.8E−01	6.0E−02	7.0E−02

6.1. Test 1 - AMG threshold calibration

In the first test, we have studied the behavior of the average solution time per timestep for the elliptic equation while varying the threshold parameters, i.e. the threshold for the GAMG preconditioner and the strong threshold for Hypre. These parameters act in the coarsening step of the algorithm and the best choices for them are problem dependent [30,31]. For each implementation of the preconditioner employed, we have performed tests with different threshold parameters on a single node, exploiting 4 CPUs or 4 GPUs for each test. The geometry considered is the truncated ellipsoid discretized by a $32 \times 32 \times 16$ hexahedral mesh. For each multigrid implementation we have reported the CG iterations and the solution times for the elliptic system. In Tables 4–6 we have reported the results for Hypre on GPU, Hypre on CPU and for GAMG on CPU, respectively. As expected, only CG iterations and solution times for the elliptic equation are affected for varying the threshold parameters. Optimal results are obtained with a strong threshold of 0.4 for Hypre on GPU, 0.5 for Hypre on CPU and 0.06 for GAMG. Different libraries for parallel matrix operations are responsible for the slightly different results between the GPU implementation and the CPU one.

6.2. Test 2 - strong scaling

We have performed here a strong scaling test, by fixing the global size of the problem while increasing the number of GPUs or CPUs. As in the previous test, we have studied the average solution time per timestep for the elliptic and parabolic equations and for the membrane model (TP06). We have also reported the average number of CG iterations per time step for both the parabolic and elliptic systems. Regarding the elliptic system, the setup of AMG is the one described in the previous section. We fix the global size of the problem by employing a $128 \times 128 \times 64$ mesh, leading to a total of 2 163 330 DOFs. The results reported in Figs. 1 and 2 have shown that all solvers are scalable in terms of CG iterations, which remain almost bounded when increasing the number of the CPUs or GPUs. The solution times on GPU are not scalable (where here scalability is intended in the sense that it presents adequate time reduction, despite not halving when the number of workers is doubled), whereas on CPU they are scalable up to 128/256 cores. However, the solution time for the elliptic system with the Hypre GPU solver is significantly lower than with the CPU Hypre and GAMG solvers. The results reported in Fig. 3 (left panel) show that the membrane model solver is scalable both on CPU and GPU. Moreover, the solution times on GPU are about two order of magnitude lower than on CPU. Fig. 3 (right panel) shows that, in terms of total solution time, the best performance is obtained for GPU. This performance cannot be achieved using CPU.

6.3. Test 3 - Weak scaling

We have performed here a weak scaling test, by fixing the local size of the problem while increasing the number of GPUs or CPUs. In Tables 7–9 we have reported the average solution time per timestep for the elliptic and parabolic equations and for the membrane model (TP06). We have studied also the average number of CG iterations per time step for both the parabolic and elliptic systems. Regarding the elliptic system, the setup of AMG is the same previously described. The local size of the problem is fixed such that each CPU/ GPU handles a local mesh of $16 \times 16 \times 16$ elements, for a total amount of 9826 DOFs per worker.

Data reported in Tables 7–9 have shown that in terms of CG iterations the solvers are scalable, with an argument similar to the previous section. Solution times for the elliptic equation with Hypre on GPU are not scalable, but the results seem to be affected by a synchronization overhead due to idle threads in the GPUs. Data relative to CPU, both for GAMG and Hypre show instead good weak scalability, with solution times which are overall comparable, increasing the number of cores, while keeping fixed the local size of the problem. Similar comments can be made regarding the parabolic equation for both CG iterations and solution time (see Fig. 4). We have also observed very good scalability with both CPUs and GPUs for the solution time of the membrane model.

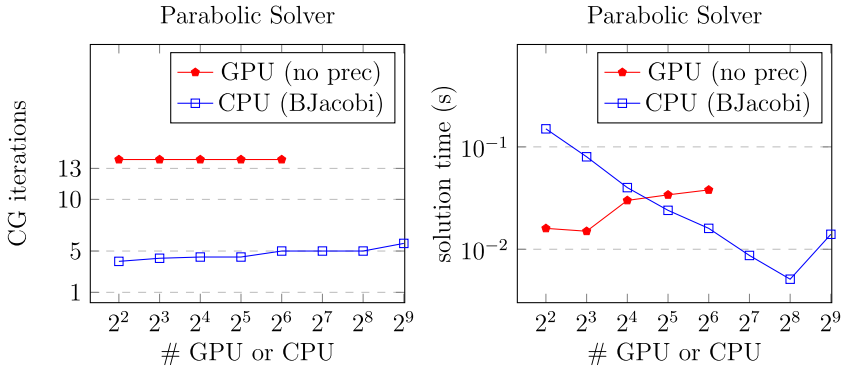

Fig. 1. Strong scaling, structured meshes. Comparison of CG iterations (left) and solution time (right) for the parabolic solver on GPU or CPU.

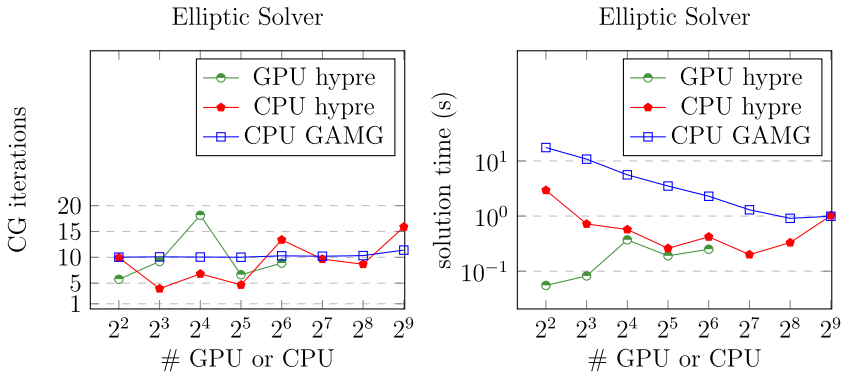

Fig. 2. Strong scaling, structured meshes. Comparison of CG iterations (left) and solution time (right) for the elliptic solver on GPU or CPU.

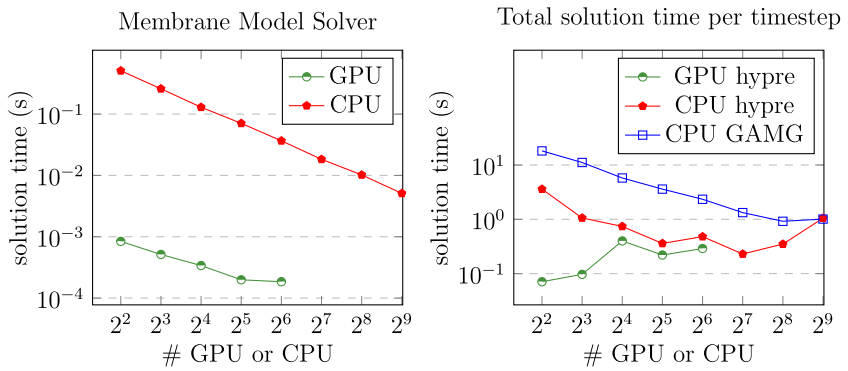

Fig. 3. Strong scaling, structured meshes. Comparison of solution time for the membrane model on GPU or CPU (left). Total solution time per timestep, given as the sum of parabolic, elliptic and membrane solution times on GPU or CPU (right).

6.4. Test 4 - simulation of the whole heartbeat

In this test, the simulation has been ran for a whole heartbeat, namely 500 ms, on the structured mesh.

In particular, we have considered the maximum number of DOFs that can be handled by the memory (i.e. 8 487 168, corresponding to a FEM mesh of 256 × 256 × 128 elements) when Hypre is employed as preconditioner and solved the problem exploiting 4 GPUs, namely all the GPUs available for a single node (see Table 10). Then, we have done the same test for 32 CPUs, all the processors available on a single node for a 128 × 128 × 64 mesh, yielding a total amount of 1 073 280 DOFs, namely the maximum number of DOFs that can be handled by the memory (240 GB) when using GAMG as preconditioner.

We have also performed other tests on a 128 × 128 × 64 mesh with 4 GPUs and 32 CPUs with Hypre in order to compare GAMG and Hypre performance on the same setup. In Figs. 6–8 we have reported for each simulation timestep the number of iterations for

Table 7
Weak scaling on GPU, structured meshes. CG preconditioned by Hypre BoomerAMG GPU as elliptic solver and unpreconditioned CG as parabolic solver. Membrane model solved on GPU. BoomerAMG Threshold = 0.4. $It_{parab,\ mean}$: average CG iterations per timestep for the parabolic solver. $It_{ellip,\ mean}$: average CG iterations per timestep for the elliptic solver. $T_{memb,\ mean}$: average solution time (in s) per timestep for the membrane model. $T_{parab,\ mean}$: average CG solution time (in s) per timestep for the parabolic solver. $T_{ellip,\ mean}$: average CG solution time (in s) per timestep for the elliptic solver.

Num GPU	DOFs	$It_{parab,\ mean}$	$It_{ellip,\ mean}$	$T_{memb,\ mean}$ (s)	$T_{parab,\ mean}$ (s)	$T_{ellip,\ mean}$ (s)
4	37,026	25.80	11.10	1.7E−04	1.4E−02	5.0E−02
8	71,874	23.04	16.32	2.3E−04	5.1E−02	3.4E−01
16	141,570	21.09	15.82	5.2E−04	9.3E−02	7.0E−01
32	278,850	17.21	7.29	1.0E−03	1.5E−01	6.8E−01
64	549,250	16.63	19.63	1.1E−03	1.5E−01	1.81

Table 8
Weak scaling on CPU, structured meshes. CG preconditioned by Hypre BoomerAMG CPU as elliptic solver and CG preconditioned by Block Jacobi as parabolic solver. Membrane model solved on GPU. BoomerAMG Threshold 0.5. Same format as in Table 7.

Num CPU	DOFs	$It_{parab,\ mean}$	$It_{ellip,\ mean}$	$T_{memb,\ mean}$ (s)	$T_{parab,\ mean}$ (s)	$T_{ellip,\ mean}$ (s)
4	37,026	3.00	7.45	1.7E−03	2.0E−03	2.7E−02
8	71,874	3.00	69.55	3.1E−03	2.1E−03	3.1E−01
16	141,570	3.00	36.93	6.7E−03	2.3E−03	3.2E−01
32	278,850	3.00	12.02	1.1E−02	2.8E−03	9.1E−02
64	549,250	4.90	13.84	1.4E−02	4.5E−03	1.3E−01
128	1,090,050	4.97	20.48	1.4E−02	5.2E−03	2.5E−01
256	2,163,330	5.00	10.59	1.4E−02	6.7E−03	2.2E−01
512	4,293,378	7.87	7.77	1.3E−02	1.1E−02	2.1E−01

Table 9
Weak scaling on CPU, structured meshes. CG preconditioned by PETSc GAMG CPU as elliptic solver and CG preconditioned by Block Jacobi as parabolic solver. Membrane model solved on GPU. GAMG Threshold 0.5. Same format as in Table 7.

Num CPU	DOFs	$It_{parab,\ mean}$	$It_{ellip,\ mean}$	$T_{memb,\ mean}$ (s)	$T_{parab,\ mean}$ (s)	$T_{ellip,\ mean}$ (s)
4	37,026	3.00	9.50	1.9E−04	2.1E−03	5.9E−02
8	71,874	3.00	10.16	7.8E−04	2.2E−03	1.8E−01
16	141,570	3.00	10.16	1.5E−03	2.3E−03	1.6E−01
32	278,850	3.00	10.27	2.9E−03	2.9E−03	1.6E−01
64	549,250	4.90	10.59	3.1E−03	4.2E−03	7.8E−01
128	1,090,050	4.97	10.36	3.5E−03	7.8E−03	7.7E−01
256	2,163,330	5.00	10.52	3.4E−03	1.1E−02	6.7E−01
512	4,293,378	7.87	13.81	3.2E−03	2.0E−02	4.02

Table 10
Mean times and iterations on a $256 \times 256 \times 128$ mesh for a whole heartbeat. Results obtained for Hypre on 4 GPUs.

Preconditioner	$It_{parab,\ mean}$	$It_{ellip,\ mean}$	$T_{memb,\ mean}$ (s)	$T_{parab,\ mean}$ (s)	$T_{ellip,\ mean}$ (s)
Hypre 4 GPU	26.00	43.62	4.7E−03	7.1E−02	0.80

Table 11
Mean times and iterations for Hypre and GAMG on a $128 \times 128 \times 64$ mesh for a whole heartbeat. On GPU the parabolic problem is not preconditioned, thus we have an higher number of iterations.

Preconditioner	$It_{parab,\ mean}$	$It_{ellip,\ mean}$	$T_{memb,\ mean}$ (s)	$T_{parab,\ mean}$ (s)	$T_{ellip,\ mean}$ (s)
GAMG 32 CPU	6.53	10.18	1.7E−03	3.5E−02	1.81
Hypre 32 CPU	6.53	4.44	4.7E−03	3.3E−02	2.4E−01
Hypre 4 GPU	26.98	15.22	7.7E−04	2.5E−02	1.1E−01

the elliptic problem and the corresponding solution time, while in Table 11 we have reported the same mean values benchmarked in the previous sections. We can observe that, at least for our implementation, using GPUs does not significantly change the profiling of the solution procedure, but only reduces the solution times.

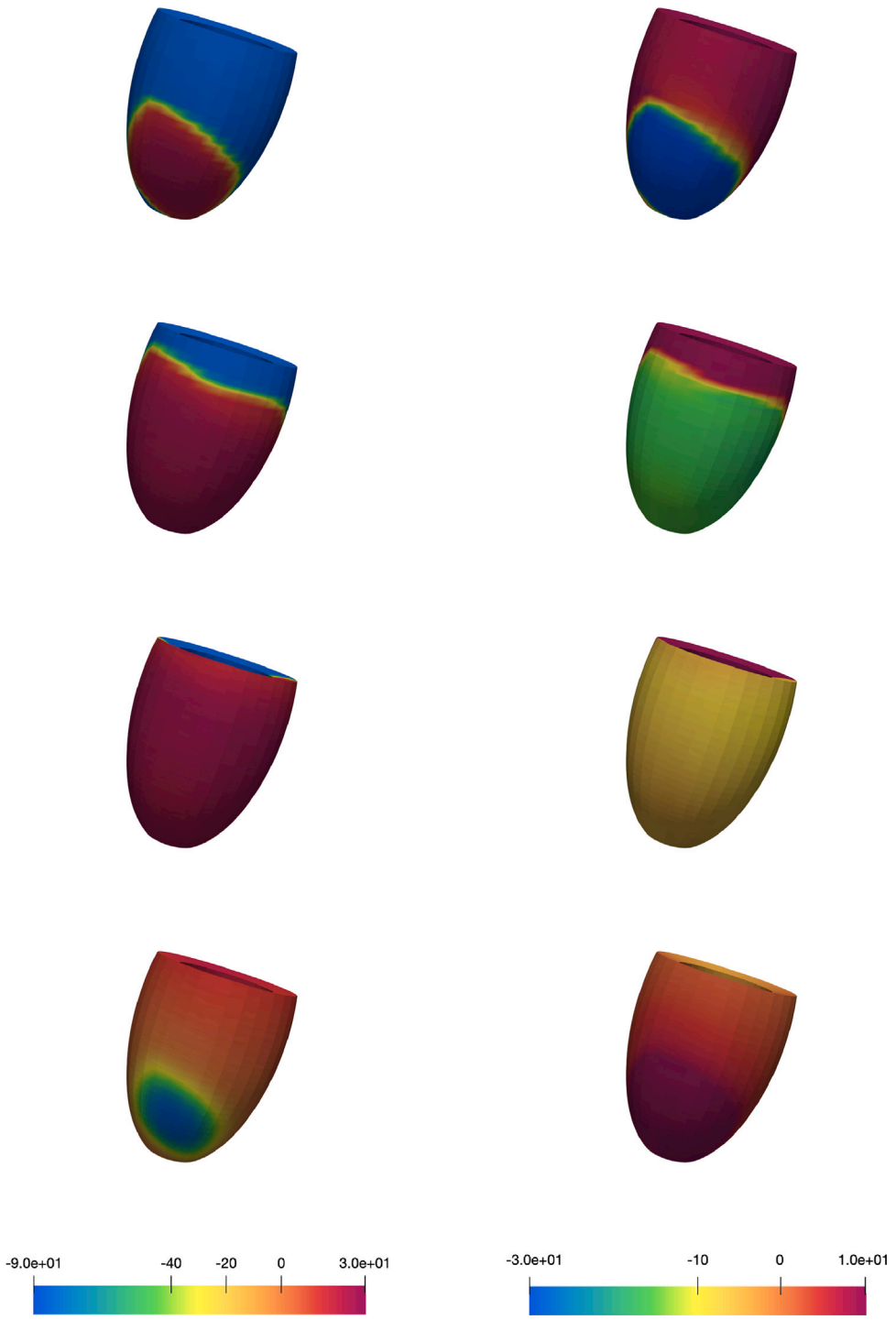

Fig. 4. Transmembrane potential (left) and extracellular potential (right) snapshots on the epicardial surface, represented by the structured mesh. The values of the displayed map are in mV.

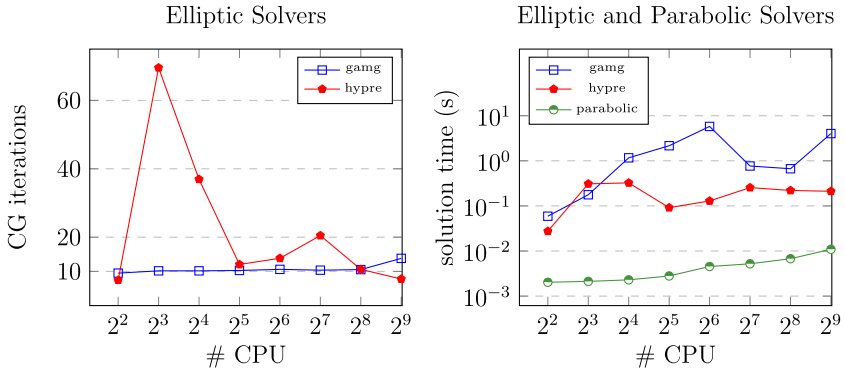

Fig. 5. Weak scaling, structured meshes. Comparison of CG iterations for the elliptic solvers (left) and solution time for the elliptic and parabolic solvers (right) on CPU.

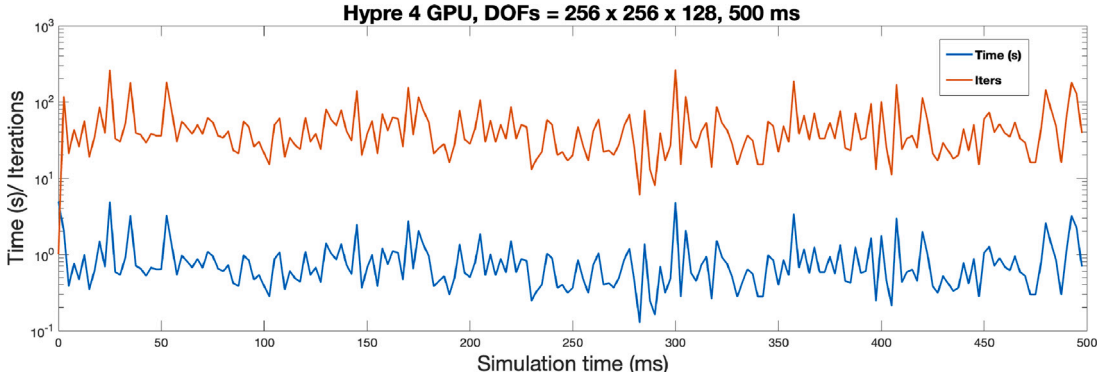

Fig. 6. Simulation of the whole heartbeat on the structured meshes. Solution time and CG iterations for the elliptic system vs. simulation time for Hypre on 4 GPUs with a mesh of 256 × 256 × 128 elements.

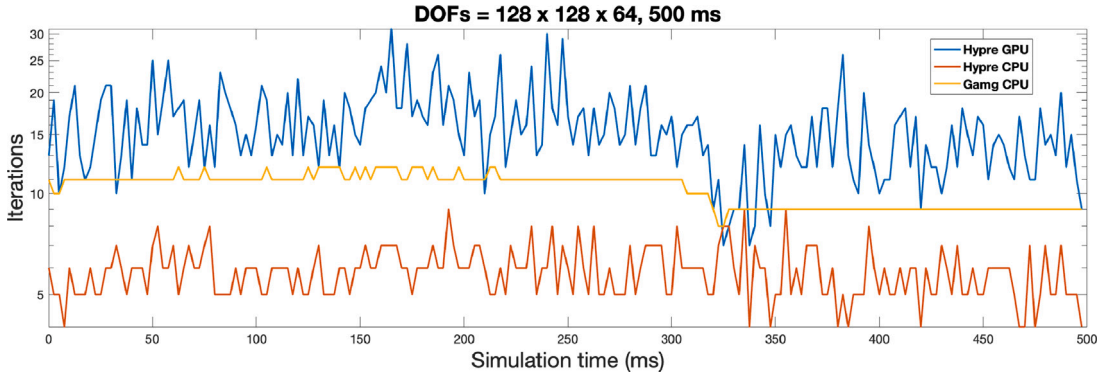

Fig. 7. Simulation of the whole heartbeat on the structured meshes. Iterations for the elliptic system on a whole heartbeat simulation with a mesh of 128 × 128 × 64 elements.

7. Tests on unstructured mesh

In this section we will comment the results of the parallel numerical tests performed using the unstructured mesh as geometry for solving the Bidomain cardiac model. Snapshots of the numerical solutions for the potentials on the epicardial surface are shown in Fig. 13.

First, in *Test 1* we have studied the threshold parameters for the multigrid preconditioner exploited for the elliptic equation. Then, in *Test 2* we have performed a scaling test on three refinements of the geometry. Since the meshes considered are unstructured, a precise subdomain decomposition with an equal number of DOFs per worker has not been possible and the local DOFs have been calculated automatically through the `PETSC_DECIDE` option.

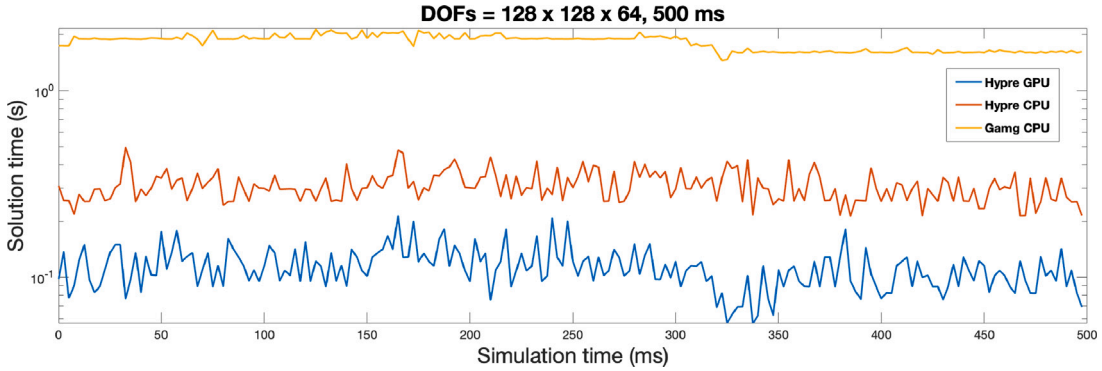

Fig. 8. Simulation of the whole heartbeat on the structured meshes. Time to solve the elliptic system on a whole heartbeat simulation with a mesh of $128 \times 128 \times 64$ elements.

Table 12
AMG threshold calibration, unstructured mesh (U-mesh 1, DOFs = 71 450). Results for Hypre GPU solver. $\text{It}_{\text{ellip, mean}}$: average CG iterations per timestep for the elliptic solver. $T_{\text{ellip, mean}}$: average CG solution time (in s) per timestep for the elliptic solver.

Threshold	0.25	0.3	0.4	0.5	0.6	0.7
$\text{It}_{\text{ellip, mean}}$	66.89	40.41	38.88	10.29	5.41	4.50
$T_{\text{ellip, mean}}$ (s)	3.05E−01	1.77E−01	1.89E−01	5.24E−02	4.46E−02	4.46E−02

Table 13
AMG threshold calibration, unstructured mesh (U-mesh 1, DOFs = 71 450). Results for Hypre CPU solver. Same format as in Table 12.

Threshold	0.25	0.3	0.4	0.5	0.6	0.7
$\text{It}_{\text{ellip, mean}}$	7.77	15.51	10.46	3.06	3.95	12.61
$T_{\text{ellip, mean}}$ (s)	5.7E−02	1.0E−01	7.5E−02	3.0E−02	3.2E−02	7.6E−02

Table 14
AMG threshold calibration, unstructured mesh (U-mesh 1, DOFs = 71 450). Results for GAMG CPU solver. Same format as in Table 12.

Threshold	0.0	0.01	0.02	0.03	0.04	0.05	0.06	0.07
$\text{It}_{\text{ellip, mean}}$	72.08	60.55	49.28	27.80	44.80	24.49	12.03	11.77
$T_{\text{ellip, mean}}$ (s)	1.3E−01	1.1E−01	1.5E−01	5.8E−02	1.0E−01	6.6E−02	4.0E−02	4.3E−02

7.1. Test 1 - AMG threshold calibration

For threshold tuning, we used the first mesh, U-mesh 1, for a total of 71 450 DOFs, solving the membrane model with the GPU. The best results were obtained around 0.06–0.07 per GAMG, while for hypre, values between 0.5 and 0.6 were considered of particular interest. Different libraries for parallel matrix operations are responsible for the slightly different results between the GPU implementation and the CPU one.

The tests in this section were performed on a single node, in particular, if GPUs were used, all those available on the node (4 GPUs) were used, the same procedure was applied using CPUs: in this case, each node had a maximum of 32 physical cores (see Tables 12–14).

7.2. Test 2 - Strong scaling

In this test we have performed a strong scaling test, solving the problem on each of the three refinements considered. We have performed tests on both CPU and GPU architectures, exploiting the algebraic multigrid implementations provided by PETSc for preconditioning the elliptic system. Again, on CPU we have preconditioned the parabolic system using a block Jacobi preconditioner, while on GPU the parabolic system is unpreconditioned. We notice in general good performance using GPUs instead of CPUs, with generally lower solution times especially with an higher number of DOFs, which consent to fully exploit the potential of the GPU acceleration, minimizing losses in time due mainly to synchronization overhead. We also noticed an out of memory error with U-mesh 3, the finest one and more than 32 GPUs. This is probably a technical issue related to the handling of the copies for the

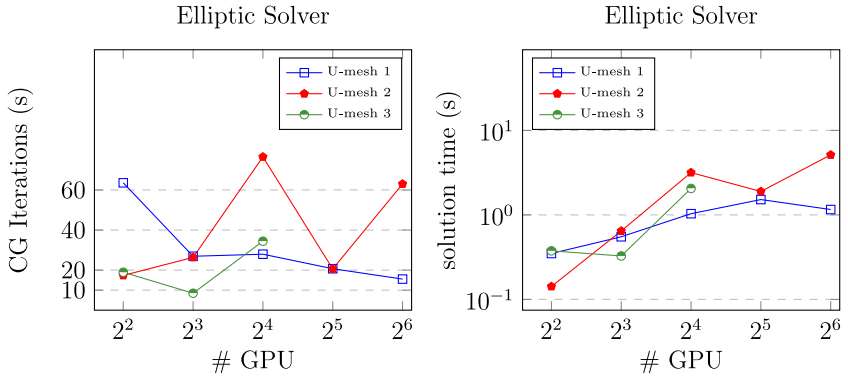

Fig. 9. Strong scaling, unstructured meshes. Comparison of CG iterations (left) and solution time (right) for the elliptic solver on GPU (Hypre BoomerAMG). U-mesh 3 goes out of memory with 32 and 64 GPUs.

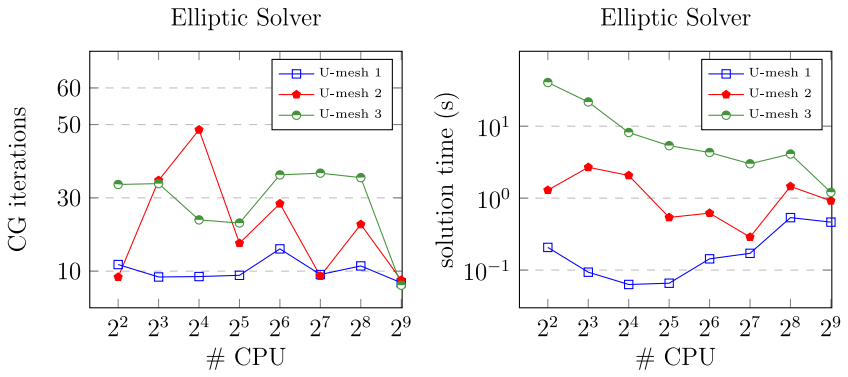

Fig. 10. Strong scaling, unstructured meshes. Comparison of CG iterations (left) and solution time (right) for the elliptic solver on CPU (Hypre BoomerAMG).

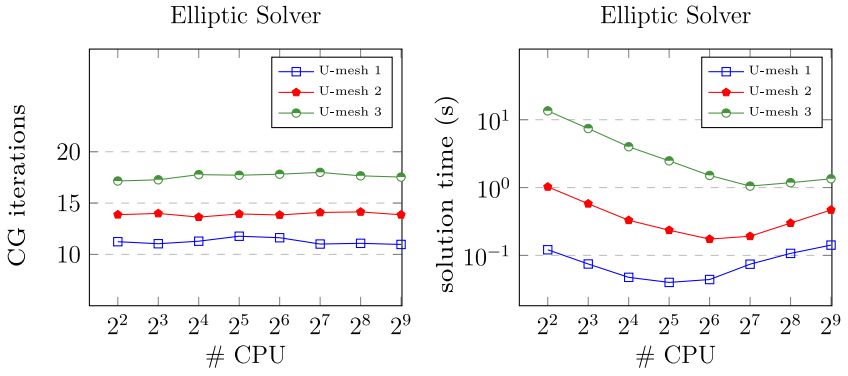

Fig. 11. Strong scaling, unstructured meshes. Comparison of CG iterations (left) and solution time (right) for the elliptic solver on CPU (GAMG).

acceleration device and it is still under investigation. However, up to 16 GPUs, performances are about one order of magnitude better than the CPU counterpart (see Fig. 9).

In Figs. 10 and 11 are reported the results of the strong scaling test performed on CPU. In Fig. 12 CG iterations and solution time for all the implementations are compared for the unstructured mesh. While in the structured mesh case results showed better performance for Hypre in comparison to GAMG, in this case the results are shifted. Also the robustness of GAMG for this case is another result to highlight: its iterations are basically constant.

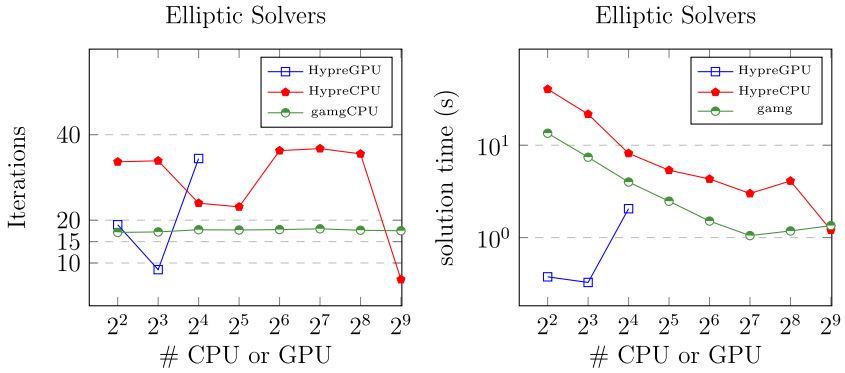

Fig. 12. Strong scaling, unstructured meshes (U-mesh3). Comparison of CG iterations (left) and solution time (right) for the elliptic solver on CPU (GAMG and Hypre BoomerAMG) and on GPU (Hypre BoomerAMG).

8. Conclusions

In this work we have performed scalability tests for the Bidomain cardiac model, focusing in particular on the performance of the algebraic multigrid implementations provided by PETSc (GAMG and Hypre) for preconditioning the linear system arising from the discretization of the elliptic system in the parabolic–elliptic formulation of the model.

Tests were performed on CPU and GPU on two different kind of meshes: a structured one and a more realistic unstructured one, representing a cardiac ventricle. Overall results have shown a general better scaling properties (especially strong scaling) when the problem is solved on CPU, even if the absolute best performance was obtained when solving the problem on GPU. In particular, in case of structured meshes we observed that the solution of the elliptic problem on GPU is 3.6 times faster than the CPU counterpart, whereas in case of unstructured meshes the solution of the elliptic problem is 3.2 times faster than on CPU. From the tests presented we have also confirmed the scalability properties of AMG used as preconditioner for a more complex problem than the one considered in [40] and we have provided an extension of the results in [19] with more GPUs, different numerical schemes and an updated version of the software employed.

CRediT authorship contribution statement

Edoardo Centofanti: Writing – review & editing, Writing – original draft, Software, Data curation, Conceptualization. **Simone Scacchi:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

We would like to thank Marco Fedele from Politecnico di Milano for providing us the patient-specific LV mesh and Luca F. Pavarino from the University of Pavia for many helpful discussions and suggestions.

This work was part of the MICROCARD project. This project has received funding from the European High-Performance Computing Joint Undertaking EuroHPC (JU) under grant agreement No 955495. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Italy, Germany, Austria, Norway, Switzerland. E. Centofanti and S. Scacchi have been supported by grants of INdAM–GNCS. S. Scacchi has been supported by MIUR grants PRIN 2017AXL54F_003 and PRIN 202232A8AN_003.

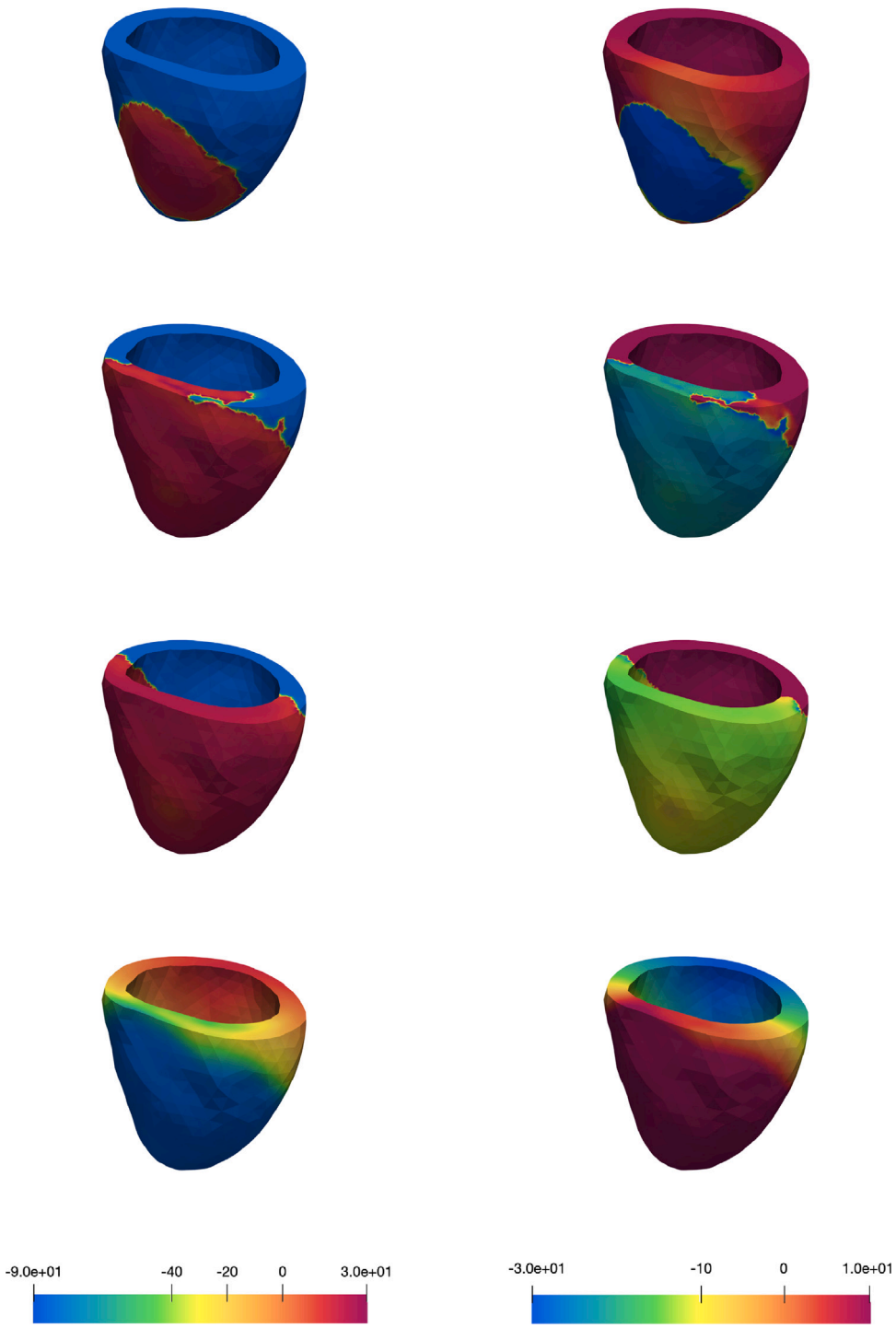

Fig. 13. Transmembrane potential (left) and extracellular potential (right) snapshots on the epicardial surface, represented by the U-mesh. The values of the displayed map are in mV.

Table B.15
Technical data for Leonardo.

Theoretical peak performance	CPU (nominal/peak freq.)	1680 GFlops
	GPU	73.2 TFlops
	Total	74.9 TFlops
Memory bandwidth (nominal/peak freq.)	600/807 GB/s	

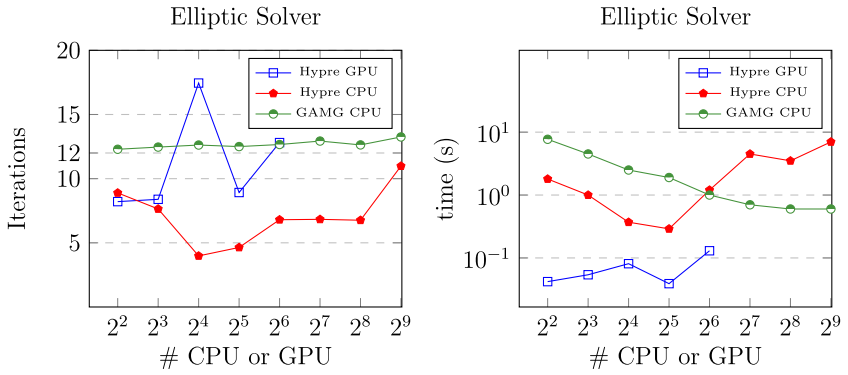

Fig. B.14. Strong scaling, structured meshes. Comparison of CG iterations (left) and solution time (right) for the elliptic solvers.

Appendix A. PETSc options

In this section we report the PETSc options set for our numerical parallel tests.

For Hypre BoomerAMG, we have considered default options except for the following:

```
-dm_mat_type $MAT_TYPE                  # mpiaij or aijcusparse
-dm_vec_type $VEC_TYPE                  # mpi or cuda
-pc_hypre_boomeramg_strong_threshold $THR # Threshold value
```

For GAMG instead, we have considered the following *non-default* options

```
-dm_mat_type $MAT_TYPE                  # mpiaij or aijcusparse
-dm_vec_type $VEC_TYPE                  # mpi or cuda
-pc_GAMG_type agg
-pc_GAMG_agg_nsmooths 1
-pc_GAMG_coarse_eq_limit 100
-pc_GAMG_reuse_interpolation
-pc_GAMG_square_graph 1
-pc_GAMG_threshold $THR                 # Threshold value
-mg_levels_ksp_max_it 2
-mg_levels_ksp_type chebyshev
-mg_levels_esteig_ksp_type cg
-mg_levels_esteig_ksp_max_it 10
-mg_levels_ksp_chebyshev_esteig 0,0.05,0,1.05
-mg_levels_pc_type jacobi
```

Appendix B. Strong scaling on Leonardo

Strong scaling tests have been reproduced also on LEONARDO machine, which technical details are reported in Table B.15.

On this machine latency using over than 32 CPU lead to higher times if compared to MARCONI100 results and generally a bad scaling. Also, load imbalance between the CPU/GPU workers brought to scaling issues, since we considered the same problem (both algorithmically and in terms of memory) on a theoretically more performing machine. Anyway, up to 32 CPU times are generally better than the ones recorded on Marconi100 and the scaling ratio is comparable between the two machines, as expected. Also iterations are comparable between the two machines and the stability in iteration of GAMG is confirmed.

In this case, we did not experience the out of memory error on GPU for the unstructured mesh. All the unstructured mesh tests have been performed using the U-mesh 3 (see Figs. B.14 and B.15).

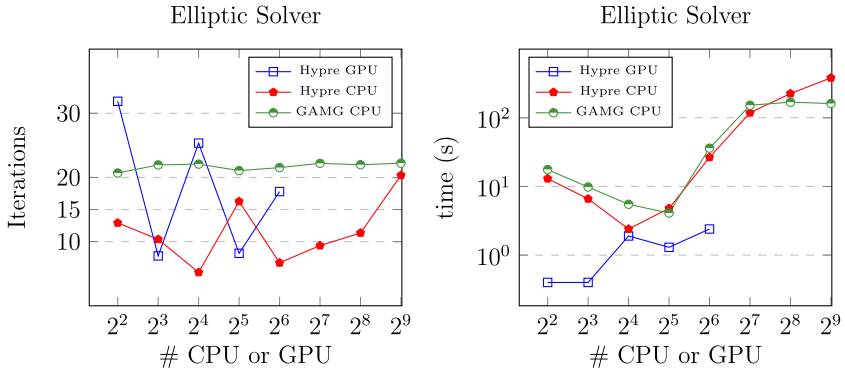

Fig. B.15. Strong scaling, unstructured meshes. Comparison of CG iterations (left) and solution time (right) for the elliptic solvers.

References

[1] M. Potse, B. Dubé, J. Richer, A. Vinet, R.M. Gulrajani, A comparison of monodomain and bidomain reaction–diffusion models for action potential propagation in the human heart, IEEE Trans. Biomed. Eng. 53 (12) (2006) 2425–2435.

[2] C.M. Augustin, A. Neic, M. Liebmann, A.J. Prassl, S.A. Niederer, G. Haase, G. Plank, Anatomically accurate high resolution modeling of human whole heart electromechanics: A strongly scalable algebraic multigrid solver method for nonlinear deformation, J. Comput. Phys. 305 (2016) 622–646.

[3] A. Quarteroni, T. Lassila, S. Rossi, R. Ruiz-Baier, Integrated heart—Coupling multiscale and multiphysics models for the simulation of the cardiac function, Comput. Methods Appl. Mech. Engrg. 314 (2017) 345–407.

[4] M. Fedele, R. Piersanti, F. Regazzoni, M. Salvador, P.C. Africa, M. Bucelli, A. Zingaro, L. Dedè, A. Quarteroni, A comprehensive and biophysically detailed computational model of the whole human heart electromechanics, Comput. Methods Appl. Mech. Engrg. 410 (2023) 115983.

[5] Yi Jiang, Rongliang Chen, Xiao-Chuan Cai, A highly parallel implicit domain decomposition method for the simulation of the left ventricle on unstructured meshes, Comput. Mech. 66 (2020) 1461–1475, Publisher: Springer.

[6] Sebastian Laudenschlager, Xiao-Chuan Cai, An inner-outer subcycling algorithm for parallel cardiac electrophysiology simulations, Int. J. Numer. Methods Biomed. Eng. 39 (3) (2023) e3677, Publisher: Wiley Online Library.

[7] J.W. Ruge, K. Stüben, Algebraic multigrid, in: Multigrid Methods, 1987, pp. 73–130, (Chapter 4).

[8] M. Pennacchio, V. Simoncini, Algebraic multigrid preconditioners for the bidomain reaction–diffusion system, Appl. Numer. Math. 59 (12) (2009) 3033–3050.

[9] M. Pennacchio, V. Simoncini, Fast structured AMG preconditioning for the bidomain model in electrocardiology, SIAM J. Sci. Comput. 33 (2) (2011) 721–745.

[10] G. Plank, M. Liebmann, R.W. dos Santos, E.J. Vigmond, G. Haase, Algebraic multigrid preconditioner for the cardiac bidomain model, IEEE Trans. Biomed. Eng. 54 (4) (2007) 585–596.

[11] J. Sundnes, G.T. Lines, K.A. Mardal, A. Tveito, Multigrid block preconditioning for a coupled system of partial differential equations modeling the electrical activity in the heart, Comput. Methods Biomech. Biomed. Engrg. 5 (6) (2002) 397–409.

[12] E.J. Vigmond, R.W. Dos Santos, A.J. Prassl, M. Deo, G. Plank, Solvers for the cardiac bidomain equations, Prog. Biophys. Mol. Biol. 96 (1–3) (2008) 3–18.

[13] L.F. Pavarino, S. Scacchi, Multilevel additive Schwarz preconditioners for the bidomain reaction–diffusion system, SIAM J. Sci. Comput. 31 (2008) 420–443.

[14] L.F. Pavarino, S. Scacchi, Parallel multilevel Schwarz and block preconditioners for the bidomain parabolic-parabolic and parabolic–elliptic formulations, SIAM J. Sci. Comput. 33 (2011) 1897–1919.

[15] S. Zampini, Balancing Neumann–Neumann methods for the cardiac bidomain model, Numer. Math. 123 (2013) 363–393.

[16] S. Zampini, Dual-primal methods for the cardiac bidomain model, Math. Models Methods Appl. Sci. 24 (2014) 667–696.

[17] N.M.M. Huynh, Newton-Krylov-BDDC deluxe solvers for non-symmetric fully implicit time discretizations of the bidomain model, Numer. Math. 152 (4) (2022) 841–879.

[18] N.M.M. Huynh, Luca F. Pavarino, Simone Scacchi, Parallel Newton–Krylov BDDC and FETI-DP deluxe solvers for implicit time discretizations of the cardiac bidomain equations, SIAM J. Sci. Comput. 44 (2) (2022) B224–B249.

[19] A. Neic, M. Liebmann, E. Hoetzl, L. Mitchell, E.J. Vigmond, G. Haase, G. Plank, Accelerating cardiac bidomain simulations using graphics processing units, IEEE Trans. Biomed. Eng. 59 (8) (2012) 2281–2290.

[20] P. Colli Franzone, L.F. Pavarino, G. Savaré, Computational electrocardiology: mathematical and numerical modeling, in: A. Quarteroni, L. Formaggia, A. Veneziani (Eds.), Complex Systems in Biomedicine, Springer, 2006, pp. 187–241.

[21] P. Colli Franzone, L.F. Pavarino, S. Scacchi, Mathematical Cardiac Electrophysiology, Springer, 2014.

[22] M. Veneroni, Reaction–diffusion systems for the macroscopic bidomain model of the cardiac electric field, Nonlinear Anal. Real World Appl. 10 (2) (2009) 849–868.

[23] M. Pennacchio, G. Savaré, P. Colli Franzone, Multiscale modeling for the bioelectric activity of the heart, SIAM J. Math. Anal. 37 (4) (2005) 1333–1370.

[24] K.H.W.J. ten Tusscher, A.V. Panfilov, Alternans and spiral breakup in a human ventricular tissue model, Am. J. Physiol. Heart Circ. Physiol. 291 (3) (2006) H1088–H1100.

[25] I.J. LeGrice, B.H. Smaill, L.Z. Chai, S.G. Edgar, J.B. Gavin, P.J. Hunter, Laminar structure of the heart: ventricular myocyte arrangement and connective tissue architecture in the dog, Am. J. Physiol. Heart Circ. Physiol. 269 (2 (Pt 2)) (1995) H571–H582.

[26] P.C. Africa, R. Piersanti, M. Fedele, L. Dedè, A. Quarteroni, lifex-fiber: an open tool for myofibers generation in cardiac computational models, BMC Bioinform. 24 (1) (2023) 143.

[27] R. Piersanti, P.C. Africa, M. Fedele, C. Vergara, L. Dedè, A.F. Corno, A. Quarteroni, Modeling cardiac muscle fibers in ventricular and atrial electrophysiology simulations, Comput. Methods Appl. Mech. Engrg. 373 (2021) 113468.

[28] K.H.W.J. ten Tusscher, D. Noble, P.J. Noble, A.V. Panfilov, A model for human ventricular tissue, Am. J. Physiol. Heart Circ. Physiol. 286 (4) (2004) H1573–1589.

[29] H. De Sterck, U.M. Yang, J.J. Heys, Reducing complexity in parallel algebraic multigrid preconditioners, SIAM J. Matrix Anal. Appl. 27 (4) (2006) 1019–1039.

[30] S. Balay, et al., PETSc/TAO Users Manual, Technical Report ANL-21/39 - Revision 3.20, Argonne National Laboratory, 2023.
[31] R.D. Falgout, J.E. Jones, U.M. Yang, The design and implementation of hypre, a library of parallel high performance preconditioners, in: A.M. Bruaset, A. Tveito (Eds.), Numerical Solution of Partial Differential Equations on Parallel Computers, Springer, 2006, pp. 267–294.
[32] V.E. Henson, U.M. Yang, BoomerAMG: A parallel algebraic multigrid solver and preconditioner, Appl. Numer. Math. 41 (1) (2002) 155–177.
[33] M. Griebel, B. Metsch, D. Oeltz, M.A. Schweitzer, Coarse grid classification: a parallel coarsening scheme for algebraic multigrid methods, Numer. Linear Algebra Appl. 13 (2–3) (2006) 193–214.
[34] M. Griebel, B. Metsch, M.A. Schweitzer, Coarse Grid Classification-Part II: Automatic Coarse Grid Agglomeration for Parallel AMG, Technical Report, University of Bonn, 2006, SFB 611.
[35] K. Stüben, Algebraic multigrid (AMG), An Introduction with Applications, GMD Forschungszentrum Informationstechnik, 1999.
[36] M.F. Adams, J. Demmel, A parallel maximal independent set algorithm, in: Proceedings 5th Copper Countain Conference on Iterative Methods, 1998.
[37] M.F. Adams, Algebraic multigrid methods for constrained linear systems with applications to contact problems in solid mechanics, Numer. Linear Algebra Appl. 11 (2–3) (2004) 141–153.
[38] H. De Sterck, U.M. Yang, J.J. Heys, Reducing complexity in parallel algebraic multigrid preconditioners, SIAM J. Matrix Anal. Appl. 27 (4) (2006) 1019–1039.
[39] L. Michael, A simple parallel algorithm for the maximal independent set problem, in: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, 1985, pp. 1–10.
[40] A.J. Cleary, R.D. Falgout, V.E. Henson, J.E. Jones, T.A. Manteuffel, S.F. McCormick, G.N. Miranda, J.W. Ruge, Robustness and scalability of algebraic multigrid, SIAM J. Sci. Comput. 21 (5) (2000) 1886–1908.