INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN BIOMEDICAL ENGINEERING
*Int. J. Numer. Meth. Biomed. Engng.* 2014; **30**:117–134
Published online 20 September 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cnm.2593

SPECIAL ISSUE PAPER - NUMERICAL METHODS AND APPLICATIONS OF MULTI-PHYSICS IN BIOMECHANICAL MODELING

# Toward GPGPU accelerated human electromechanical cardiac simulations

Guillermo Vigueras, Ishani Roy, Andrew Cookson, Jack Lee, Nicolas Smith and David Nordsletten*,†

*Department of Biomedical Engineering, King's College London, UK*

SUMMARY

In this paper, we look at the acceleration of weakly coupled electromechanics using the graphics processing unit (GPU). Specifically, we port to the GPU a number of components of $\mathcal{C}$Heart—a CPU-based finite element code developed for simulating multi-physics problems. On the basis of a criterion of computational cost, we implemented on the GPU the ODE and PDE solution steps for the electrophysiology problem and the Jacobian and residual evaluation for the mechanics problem. Performance of the GPU implementation is then compared with single core CPU (SC) execution as well as multi-core CPU (MC) computations with equivalent theoretical performance. Results show that for a human scale left ventricle mesh, GPU acceleration of the electrophysiology problem provided speedups of 164× compared with SC and 5.5 times compared with MC for the solution of the ODE model. Speedup of up to 72× compared with SC and 2.6× compared with MC was also observed for the PDE solve. Using the same human geometry, the GPU implementation of mechanics residual/Jacobian computation provided speedups of up to 44× compared with SC and 2.0× compared with MC. © 2013 The Authors. *International Journal for Numerical Methods in Biomedical Engineering* published by John Wiley & Sons, Ltd.

Received 21 December 2012; Revised 17 July 2013; Accepted 1 August 2013

KEY WORDS: GPU; cardiac electrophysiology; tissue mechanics; electromechanics

## 1. INTRODUCTION

The ability to predict the electromechanical behavior of the heart from imaging and other physiological data is one of the compelling, yet still only partially fulfilled, goals of the personalized healthcare [1–3]. The challenge that is central to bringing electromechanical modeling into the clinic is the process of patient-specific tailoring of the model as well as *in silico* treatment evaluation, both of which are processes requiring many electromechanical simulations. Patient-specific tailoring of models requires coupling patient data and model parameters using parameter estimation algorithms, which require the iterative solution of the model for a varied range of parameters [4, 5]. With a parameterized model, numerous simulations may be run to examine different potential treatment strategies. Although a number of authors have developed effective models and tools for simulating electromechanics, their use in diagnosis or treatment planning requires model analysis to conclude in clinically relevant time-scales, mandating continued improvement of simulation technologies.

The GPU architecture is a highly promising hardware with significant potential to accelerate cardiac electromechanics simulations. Toward this goal, a number of previous studies have already investigated the acceleration of the electrophysiology problem through GPUs [6–10]. Building on

*Correspondence to: David Nordsletten, Department of Biomedical Engineering, Kings College London, UK.

†E-mail: david.nordsletten@gmail.com

this work, in this paper, we propose the acceleration of the human scale electrical activation simulation and a novel GPU-based implementation of cardiac mechanics, which constitutes the first implementation of *weakly coupled* electromechanics on this platform. We analyze these parallel implementations by quantifying the computational gain of function, show the potential of this technology, to broaden the application of these types of Virtual Physiological Human (VPH) models.

The rest of the paper is organized as follows. In Section 2, previous studies of electrophysiology and electromechanics problems are reviewed. In order to understand architectural and programmability aspects of the GPU, Section 3 analyzes the main features of this parallel platform. Section 4 describes electrophysiology and mechanics models and numerical methods used in our CPU and GPU implementations. Section 5 presents CPU and GPU implementations for accelerating cardiac electromechanics simulations. A performance comparison between CPU and GPU versions is shown in Section 6, and their results are discussed in the conclusion (Section 7).

## 2. RELATED WORK

In order to tackle the computational barrier to the clinical translation of cardiac human models, some approaches have already been proposed that exploit parallel clusters facilities for simulating electrical activity [11, 12]. Although these works propose efficient High Performance Computing (HPC) implementations, the use of such large-scale computational facilities results in high cost in terms of price and power consumption and is less accessible in most clinical environments.

As the GPU has emerged as an efficient platform providing a good power/performance ratio, a number of groups have investigated the use of GPUs for accelerating cardiac electrophysiology simulations. Bartocci *et al.* [6] have proposed the implementation of the ODE solver on the GPU and evaluated the approach using 2D tissues. Another approach introduced by Vigmond *et al.* [7] has aimed at facilitating the acceleration of the ODE solver through the application of GPUs, demonstrating its efficacy in small mammalian hearts. A further extension has been recently proposed by Rocha *et al.* [8], who used the single-precision GPU to solve the system of PDEs and ODEs present in the Monodomain model to solve 2D tissue simulations. Plank *et al.* [9] recently developed a solution proposing a multi-GPU implementation for performing cardiac simulations using a rabbit model, showing significant speedups with respect to their parallel CPU code CARP.

Building on these efforts, in this paper, we look to simulate both electrophysiology and mechanics on the GPU. Although some recent works have simulated mechanics using low-order refined meshes [13, 14], most mechanical models of the heart use incompressible quasi-static finite elasticity solved on high-order curvilinear hexahedral elements [15–18]. Beyond the change in interpolation scheme, the inherent nonlinearity of cardiac mechanics and structure of the linearized system poses significantly different challenges to those faced with parallelization of electrophysiology. In this paper, we focus on the initial acceleration of mechanics computations by porting algorithms for Jacobian matrix and residual evaluations. In this context, we show the benefits provided by GPUs for simulating both the electrical activity and mechanical deformation in the human heart.

## 3. GPU ARCHITECTURE DESCRIPTION

The appearance of Compute Unified Device Architecture (CUDA) [19] has enabled the use of GPUs as powerful computing platforms and enabled their recent extension to general-purpose computing. The CUDA model is a hardware and software architecture to perform computations on the GPU as a data-parallel computing device, without the need of using a graphics API [19].

Figure 1 illustrates the hardware interface of CUDA for the Nvidia GPU G80. This parallel single instruction multiple data (SIMD) architecture is endowed with up to 128 cores, where thousands of threads run in parallel. These cores are organized into 16 multiprocessors (SMs), each one having a set of 32-bit registers, constants and texture caches, and 16 KB of on-chip shared memory as fast as local registers (one cycle latency). At any given cycle, each core executes the same instruction on different data (SIMD), and communication between multiprocessors is performed through global memory.



DOI: 10.1002/cnm

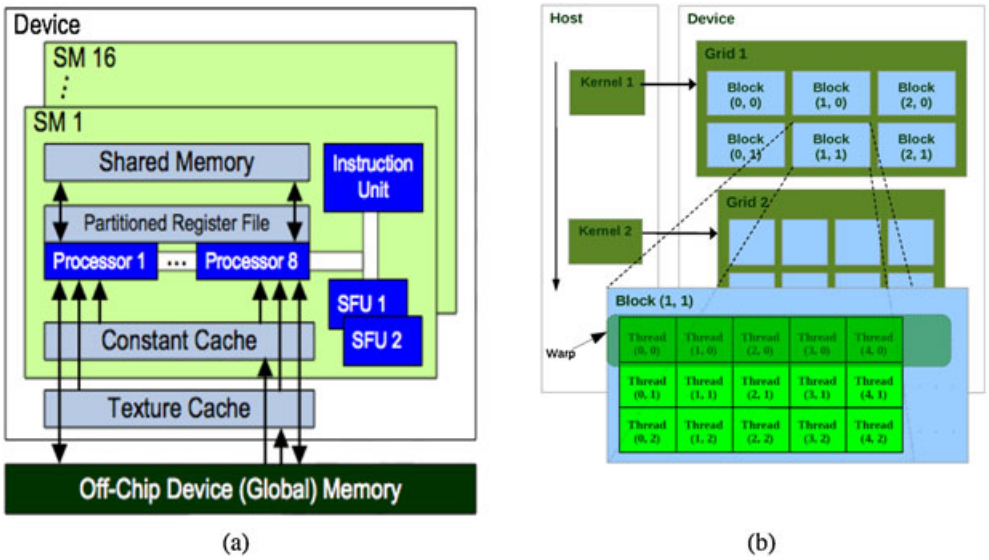


Figure 1. (a) Compute Unified Device Architecture (CUDA) hardware interface for the Nvidia GPU G80 (b) CUDA programming model [19].

Figure 1 outlines the CUDA programming model. CUDA consists of a set of C language library functions, which the programmer uses to specify the structure of a CUDA program. A CUDA program consists of two subprograms as follows: the CPU part (*host subprogram*) and the GPU part (*device subprogram*). The host subprogram prepares the GPU execution, moving data from CPU main memory to the GPU memory. Also, the host subprogram is in charge of setting up all the parameters involved in the execution and launching the device subprogram. In its turn, the device code is organized in functions or *kernels*. Each kernel is executed in parallel by each GPU thread.

A kernel execution is decomposed into blocks that run logically in parallel (physically if there are resources available on the GPU). Assembled by the developer, a block consists of a group of threads that is mapped to a single multiprocessor, where threads can share up to 16 KB of memory and also synchronize through barrier primitives. However, communication among threads of different blocks is only achieved through global memory, and they are synchronized by ending a kernel.

All the threads within a block are grouped into *warps*. A warp is a collection of threads that can actually run concurrently (with no time-sharing) on a given multiprocessor. The developer can decide the number of threads to be executed (up to a limit intrinsic to CUDA), but if there are more threads than the warp size, they are executed with time-sharing on the available hardware resources.

In the CUDA model, threads can access the whole GPU global memory, but there is a performance boost when threads access data stored in shared memory, which is explicitly managed. In order to make the most efficient usage of the GPU's computational resources, large data structures are stored in global memory, and the shared memory should be prioritized for storing strategic, often-used data structures. These hardware characteristics can have a big impact for accelerating cardiac electromechanical simulations through GPUs.

## 4. MODEL DESCRIPTIONS AND NUMERICAL METHODS

In this section, we introduce the electrophysiology and mechanical models used for modeling weakly coupled electromechanics. In the case of electrophysiology, we introduce the monodomain model and its solution using second-order Strang splitting (see section 4.1). This description is followed by an outline of the quasi-static finite elasticity equations applied to cardiac mechanics and its solution using finite elements (see section 4.2).



DOI: 10.1002/cnm

### *4.1. Electrophysiology problem*

Modeling electrophysiology in the heart is typically accomplished using the monodomain [20,21] or bidomain [22–26] equations, which simulate the spread of membrane potential or intra/extracellular potential, respectively. In this paper, we focus on modeling the electrophysiology in the heart, denoted by the domain $\Omega \subset \mathbb{R}^3$ (with boundary $\delta\Omega$, using the monodomain model). Here, we seek a membrane potential $u : \Omega \times I \to \mathbb{R}$ and the $m-$cell model variables $\boldsymbol{v} : \Omega \times I \to \mathbb{R}^m$ over some time interval $I = [0, T]$ satisfying [27],

$$C_m \frac{\partial u}{\partial t} - \nabla \cdot (\mathbf{D}\nabla u) - I_{ion}(u, \boldsymbol{v}) - I_{ext} = 0, \quad \text{on } \Omega \times I, \tag{1}$$

$$\frac{d\boldsymbol{v}}{dt} - \boldsymbol{f}(t, u, \boldsymbol{v}) = 0, \quad \text{on } \Omega \times I, \tag{2}$$

$$(\mathbf{D}\nabla u) \cdot \boldsymbol{n} = 0, \quad \text{on } \delta\Omega \times I, \tag{3}$$

$$u = u_0, \quad \boldsymbol{v} = \boldsymbol{v}_0, \quad \text{on } \Omega \times [0] \tag{4}$$

where $\mathbf{D} : \Omega \to \mathbb{R}^{3\times3}$ is the diffusion tensor related to the gap junctions between cells and membrane capacitance. $I_{ion}(u, v)$ is the total ionic current (which is a function of the voltage $u$, the gating variables and ion concentrations), $I_{ext} : \Omega \times I \to \mathbb{R}$ is the stimulus current, $\boldsymbol{f}$ is a function governing rate-of-change in the $m-$cell model variables, and $\boldsymbol{n}$ is the normal to the surface of the boundary $\delta\Omega$. The diffusion tensor $\mathbf{D}$ is of the form $\frac{\sigma}{\chi C_m}$, where $\sigma$ is the conductivity, $C_m$ is the membrane capacitance, and $\chi$ is the cell surface to volume ratio. In this paper, we have defined $\sigma$ using $\sigma = \sigma_i\sigma_e(\sigma_e + s\sigma_i)^{-1}$, where the intra-longitudinal, intra-transversal, extra-longitudinal, and extra-transversal conductivity values are 0.17, 0.019, 0.62, and 0.24 S/m, respectively. In our simulations, the value for membrane capacitance $C_m$ was 0.185 $\mu F$ and $\chi$ was 140 $mm^{-1}$. In this model, an external stimulus current $I_{ext}$of 35 mV/ms is applied at a time between 0 and 2 ms.

A wide variety of mathematical methods have been applied to solve the monodomain equations, including finite difference methods [28], FEMs [29–31], and finite volume methods [32]. Here, we solve the monodomain equations using the FEM, seeking solutions $u \in U$ and $\boldsymbol{v} \in \boldsymbol{s}V$,

$$U = \left\{ y \in L^\infty[I;\, L^2(\Omega)] \cap L^2[I;\, H^1(\Omega)] \;\middle|\; y = u_0, \quad \text{on} \quad \Omega \times [0] \right\},$$

$$\boldsymbol{V} = \{ \boldsymbol{y} \in L^\infty(I; \mathbf{L}^\infty(\Omega)) \mid \boldsymbol{y} = \boldsymbol{v}_0, \quad \text{on} \quad \Omega \times [0] \},$$

which satisfy the weak formulation of Equations (1)–(4) derived by the standard Galerkin procedure [33], that is,

$$\int_\Omega C_m \frac{\partial u}{\partial t} \cdot y + \int_\Omega (\mathbf{D}\nabla u) \cdot \nabla y - \int_\Omega (I_{ion}(u, \boldsymbol{v}) + I_{ext}) \cdot y = 0, \quad \forall y \in U \tag{5}$$

$$\frac{d\boldsymbol{v}}{dt} - f(t, u, \boldsymbol{v}) = 0, \tag{6}$$

*4.1.1. Discrete electrophysiology problem and solution.* In this paper, we focus on the solution of the monodomain problem on tetrahedral and hexahedral grids. Here, an approximation $\Omega_h$ of $\Omega$ is constructed by merging finitely many, non-overlapping elements, $\tau$, which assemble to form the mesh, $T_h(\Omega)$ (see Figure 2), that is,

$$\Omega_h = \bigcup_{\tau \in T_h(\Omega_h)} \tau, \quad T_h(\Omega) = \{\tau_1, \ldots \tau_N\}, \quad h = \max_{\tau \in T_h(\Omega)} diam(\tau).$$

The time domain, $I$, is first divided into $N_I$ non-overlapping intervals $(t^{n-1}, t^n)$, $t^{n-1} < t^n$, $t^0 = 0$ and $t^N = T$, which denote the time stepping sequence for the PDE (Equation (5)). However,



DOI: 10.1002/cnm

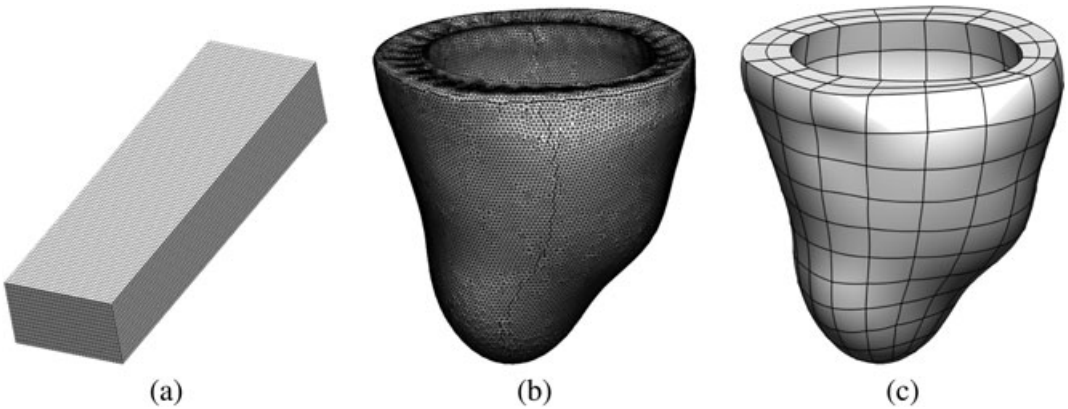


Figure 2. (a) Benchmark problem mesh. For this mesh, we have used the following resolutions: 0.2 mm (∼58 K DOFs), 0.1 mm (∼443 K DOFs), and 0.05 mm (∼3.5 M DOFs); (b) LV mesh. For this mesh, we have used the following resolutions: 0.5 mm (∼2.5 M DOFs) and the second mesh a resolution of 0.2 mm (∼19 M DOFs); (c) mechanics mesh—with 352 quadratic hexahedral elements and 555 nodes (3605 DOFs).

as the kinetics of the cell model have characteristic behavior that vary in space and time, the stepping sequence may be further subdivided into $r$ substeps, which are applied adaptively in the ODE [34], that is,

$$(t^n, t^{n+1}] = \bigcup_{k=1}^{r} (t^{n+(k-1)/r}, t^{n+k/r}].$$

Over each time interval or subinterval, the membrane potential and cell model variables are taken as constants in time, respectively. As a result, the solution to the PDE system at each time step is approximated in

$$U^h := \left\{ y_h \in C(\bar{\Omega}_h) \mid \quad y_h|_\tau \in \mathbb{P}^1, \quad \tau \in T_h(\Omega) \right\}$$

Letting $\{\phi_1, \ldots \phi_{K_u}\} = \boldsymbol{\phi}$ denote the basis of $U^h$ (where $K_u =$ span $U^h$), each PDE solution step may be expressed as the weighted sum $u_h = U \cdot \boldsymbol{\phi}$. In general, the approximation of cell model variables, $\boldsymbol{v}_h$, in the discrete setting may be handled a number of ways. In some cases, cell variables have been approximated at all quadrature points in $\Omega_h$, whereas others approximate cell variables at mesh vertices (see [30] for more details). In either case, the solution at each substep of the ODE system is solved at distinct points $\boldsymbol{P} = \{\boldsymbol{p}_k\}$, that is,

$$\boldsymbol{V}^h := \left\{ \boldsymbol{y}_h \in \boldsymbol{V} \mid \quad y_h|_{\boldsymbol{p}} \in \mathbb{R}^m, \quad \text{for some } \boldsymbol{p} \in \boldsymbol{P} \right\}.$$

In this case, the ODE model system is then solved independently at each discrete point (letting $V$ denote the total vector of ODE state variables), and its values interpolated between points (if necessary). Finally, using a backward Euler discretization of the time derivatives in Equation (5) and adaptive forward Euler in Equation (6), we may pose the discrete finite element weak form system as

$$\boldsymbol{M}U^{n+1} + \delta \boldsymbol{K}U^{n+1} = \boldsymbol{M}U^n + \delta R(U^\theta, V^\theta) \tag{7}$$

$$V^{n+1} = V^n + \delta \sum_{k=0}^{r-1} F(t^{n+k/r}, U^\theta, V^{n+k/r}) \tag{8}$$



DOI: 10.1002/cnm

where

$$\begin{aligned}(\boldsymbol{M})_{ij} &:= \int_{\Omega_h} C_m \phi_j \cdot \phi_i, \\ (\boldsymbol{K})_{ij} &:= \int_{\Omega_h} (\mathbf{D}\nabla\phi_j)\cdot\nabla\phi_i, \\ (R(U^\theta, V^\theta))_i &:= \int_{\Omega_h} (I_{ion}(u_h^\theta, \boldsymbol{v}_h^\theta) + I_{ext})\cdot\phi_i,\end{aligned}$$

and $\theta \in [n, n+1]$. Note that $\theta = n$ corresponds to a semi-implicit scheme, whereas $\theta = n+1$ represents a fully implicit scheme. The vector function $F$ denotes the application of $\boldsymbol{f}$ for each discrete point $\boldsymbol{P}$, detailing the dynamics of the cell. These cellular dynamics may be described using models such as the Luo Rudy [35] or ten Tusscher and Panfilov 2006 [36]. In this paper, the simulations were performed with the cell model described by the commonly used ten Tusscher and Panfilov 2006 model, resulting in a system of 19 variables at each $\boldsymbol{p} \in \boldsymbol{P}$. We note that in Equation (8), $r$ is selected adaptively both in space (that is, for each $\boldsymbol{p} \in \boldsymbol{P}$) and time based on the rate of change of membrane potential $\frac{\partial u}{\partial t}$ [34]. This allows the numerical solver to take a small time during the fast upstroke of the cardiac action potential and bigger time steps at other times.

An alternative approach—which is followed in this paper—providing improved computational efficiency is so-called Strang splitting for the monodomain problem [37]. The crux of this approach is splitting the discrete operator into linear PDE and nonlinear ODE parts [38] as shown in Equations (9)–(12).

$$\tilde{U}^{n+1/2} = \left(\boldsymbol{M} + \frac{\delta}{2}\boldsymbol{K}\right)^{-1} \boldsymbol{M} U^n \tag{9}$$

$$V^{n+1} = V^n + \delta \sum_{k=0}^{r-1} F(t^{n+k/r}, \tilde{U}^{n+1/2}, V^{n+k/r}) \tag{10}$$

$$\tilde{U}^{n+1} = \tilde{U}^{n+1/2} + \delta \boldsymbol{M}^{-1} R(\tilde{U}^{n+1/2}, V^{n+1}) \tag{11}$$

$$U^{n+1} = \left(\boldsymbol{M} + \frac{\delta}{2}\boldsymbol{K}\right)^{-1} \boldsymbol{M} \tilde{U}^{n+1} \tag{12}$$

Note that by choosing the points $\boldsymbol{P}$ to correspond to nodes of $T_h(\Omega)$ and approximating $I_{ion}$ and $I_{ext}$ linearly over each element, the matrix solve in Equation (11) may be eliminated. This has been shown to improve efficiency while preserving the accuracy of the method subject to reasonable limits on time step [30, 34, 39–41].

### *4.2. Cardiac mechanics problem*

The cardiac myocardium is typically modeled as a hyperelastic material and solved using quasi-static finite elasticity theory [2, 15]. The aim of simulating cardiac tissue mechanics is to find a displacement field $\boldsymbol{u} : \Omega \times I \to \mathbb{R}^3$ giving the deformed position,

$$\boldsymbol{x}(t) = \boldsymbol{u}(t) + \boldsymbol{X},$$

for every point in $\boldsymbol{X} \in \Omega$ and time $t \in I$. In the case of incompressible mixed formulation, we also solve for the hydrostatic pressure $p : \Omega \times I \to \mathbb{R}$, providing the force to constrain volume change. The displacement and pressure are then found by considering the saddle point of the quasi-static Helmholtz potential, $\Pi : \boldsymbol{U} \times W \times I \to \mathbb{R}$ at each $t \in I$, that is,

$$\Pi(\boldsymbol{u}, p, t) := \inf\{\sup\{\Pi(\boldsymbol{y}, q, t),\ q \in W\},\ \boldsymbol{y} \in \boldsymbol{U}\} \quad \Pi(\boldsymbol{y}, t) := \int_\Omega \Psi(\boldsymbol{y}, q, t) - \int_{\delta\Omega} \mathbf{t}(\mathrm{t})\cdot\boldsymbol{y}.$$



DOI: 10.1002/cnm

Here, $\Pi$ represents the balance of internal strain energy (given by $\Psi$) and the applied external energy. The solution is sought in $(\boldsymbol{u}, p) \in L^\infty(I; \boldsymbol{U}) \times L^\infty(I, W)$ with

$$\boldsymbol{U} := \{\boldsymbol{y} \in [C^1(\Omega)]^3 |\ \det \nabla \boldsymbol{y} + \boldsymbol{I} = 1\}, \quad W := L^2(\Omega),$$

ensuring that deformed body $\Omega(t) = \{\boldsymbol{x} \in \mathbb{R}^3 |\ \boldsymbol{x} = \boldsymbol{u}(t) + \boldsymbol{X},\ \text{for some } \boldsymbol{X} \in \Omega\}$ is a well-posed.

The resultant saddle point at each point in time is then sought by finding the zeros of $D_u\Pi(\boldsymbol{u}, p, t)(\boldsymbol{y})$ and $D_p\Pi(\boldsymbol{u}, p, t)(q)$. That is, by finding the point in $\boldsymbol{U} \times W$ for which the derivative in the direction of any function in the space $\boldsymbol{U}$ or $W$ is zero [42], that is,

$$D_u\Pi(\boldsymbol{u}, p, t)(\boldsymbol{y}) = \lim_{\epsilon \to 0} \frac{\Pi(\boldsymbol{u} + \epsilon \boldsymbol{y}, p, t) - \Pi(\boldsymbol{u}, p, t)}{\epsilon} = 0, \quad \forall \boldsymbol{y} \in \boldsymbol{U}, \tag{13}$$

$$D_p\Pi(\boldsymbol{u}, p, t)(q) = \lim_{\epsilon \to 0} \frac{\Pi(\boldsymbol{u}, p + \epsilon q, t) - \Pi(\boldsymbol{u}, p, t)}{\epsilon} = 0, \quad \forall q \in W. \tag{14}$$

The internal strain energy, $\Psi : \boldsymbol{U} \times W \times I \to \mathbb{R}$, in cardiac mechanics is typically defined in terms of the deformation gradient $\boldsymbol{F} = \nabla \boldsymbol{u} + \boldsymbol{I}$, right Cauchy Green tensor $\boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F}$ or Green strain $\boldsymbol{E} = \frac{1}{2}(\boldsymbol{F}^T\boldsymbol{F} - \boldsymbol{I})$ [42–44]. Thus, to apply the directional derivative in Equation (13), we must apply the chain rule. Considering the case where $\Psi$ is a function of the Green strain, the weak form equation may be stated as follows: find $(\boldsymbol{u}, p)(t) \in \boldsymbol{U} \times W$ such that

$$\int_\Omega \frac{\partial \Psi(\boldsymbol{E}, p, t)}{\partial \boldsymbol{E}} : \boldsymbol{F}^T \nabla \boldsymbol{y} + \int_\Omega \frac{\partial \Psi(\boldsymbol{E}, p, t)}{\partial p} q - \int_{\delta\Omega} \mathbf{t}(\mathbf{t}) \cdot \boldsymbol{y} = 0, \quad \forall (\boldsymbol{y}, q) \in \boldsymbol{U} \times \mathbf{W}. \tag{15}$$

*4.2.1. Cardiac constitutive law and boundary conditions.* In this paper, we modeled the myocardium using the anisotropic Costa law [45] combined with the active contraction law in [46,47]. Anisotropy was modeled using three vectors $\{\mathbf{q_1}, \mathbf{q_2}\mathbf{q_3}\}$ denoting the fiber, sheet, and sheet normal directions, respectively. Defined on $\Omega$, these vectors are mutually orthogonal and of unit length at all points in space, thus forming a basis oriented in the local microstructural directions [48].

$$\Psi(\boldsymbol{E}, p, t) = \frac{C}{2}(e^Q - 1) + p(J - 1) + (\boldsymbol{E} : \mathbf{q_1} \otimes \mathbf{q_1})T_a. \tag{16}$$

The first term in Equation (16) details the passive components, where $Q$ is defined to be

$$Q = (\boldsymbol{\alpha} \circ \boldsymbol{E}_F) : \boldsymbol{E}_F = (\boldsymbol{\alpha})_{ij}(\boldsymbol{E}_F)^2_{ij}, \quad (\boldsymbol{E}_F)_{ij} = \boldsymbol{E} : \mathbf{q}_i \otimes \mathbf{q}_j, \tag{17}$$

and $\boldsymbol{\alpha}$ is symmetric tensor of coefficients, which scale strain with respect to local microstructural directions. The resistance to volume change is provided by the second term of Equation (16), which adds internal energy if $J - 1 \neq 0$. Last, the active contraction in the tissue was generated using the Niederer contraction model [47]. This six-parameter model captures the length dependent rates of tension development, along with peak tension.[‡] In the model, active tension, $T_a$, was defined as

$$T_a = \begin{cases} T_0\, \upsilon \tanh\left(\frac{t}{t_r}\right)^2 \tanh\left(\frac{t_{max} - t}{t_d}\right)^2 & 0 < t - t_{act} < t_{max} \\ 0 & \text{else} \end{cases}, \tag{18}$$

$$\upsilon = \tanh\left(a_1\left(\sqrt{2E_{11}^F + 1} - a_2\right)\right), \quad t_r = t_{r0} + a_3(1 - \upsilon), \tag{19}$$

where $a_1$ corresponds to the degree of length dependence, $a_2$ is the length at which no tension is generated, $a_3$ is a scalar of length dependent activation, $t_{act}$ is the time of cellular activation computed from the electrophysiology model, $t_{r0}$ is the baseline activation time constant, $t_d$ is the relaxation time constant, $t_{max}$ is the duration of tension generation, and $T_0$ is the peak isometric

[‡]Note that following [46,47] the length dependence of $\boldsymbol{T}_a$ is not considered when taking $\partial/\partial \boldsymbol{E}$.



DOI: 10.1002/cnm

tension. The function $t_r$ regulates the rise time of the tension transient, whereas $\upsilon$ is a nonlinear length dependent function.

Simulating the heart cycle, the heart model was coupled to the Shi Windkessel model [49, 50], representing the dynamic load imposed by the systemic and pulmonary vascular compartments. Coupling was enforced using an additional Lagrange multiplier (scalar multiplied by the unit normal) applied on the endocardial boundary, which was used to impose endocardial volume change. The computed multiplier, denoting pressure, was then passed to the Windkessel model, which was used to compute the associated volume change (within a fixed point iteration).

*4.2.2. Discrete mechanics problem and solution.* As in Section 4.1, the solution to Equation (15) is approximated using the FEM. Constructing the solid mechanical mesh $S_h(\Omega)$ (see Figure 2) and approximation spaces,

$$\begin{aligned} \boldsymbol{U}^h &:= \left\{ \boldsymbol{y}_h \in [C(\bar{\Omega}_h)]^3 \mid \quad \boldsymbol{y}_h|_\tau \in \mathbb{P}^2, \quad \tau \in S_h(\Omega) \right\}, \\ W^h &:= \left\{ q_h \in C(\bar{\Omega}_h) \mid \quad q_h|_\tau \in \mathbb{P}^1, \quad \tau \in S_h(\Omega) \right\} \end{aligned}$$

Similarly, the time domain, $I$, is first divided into $N_I$ non-overlapping intervals $(t^{n-1}, t^n)$, $t^{n-1} < t^n$, $t^0 = 0$, and $t^N = T$. Letting $\{\Phi_1, \ldots \Phi_{K_{\boldsymbol{u}}}\} = \boldsymbol{\Phi}$ and $\{\varphi_1, \varphi_1, \ldots \varphi_{K_p}\} = \boldsymbol{\varphi}$ denote the basis of $\boldsymbol{U}^h$ and $W^h$, respectively, (where $K_{\boldsymbol{u}} =$ span $\boldsymbol{U}^h$ and $K_p =$ span $W^h$), then the solution at the $n^{th}$ time step may be written as $\boldsymbol{u}_h^n = \mathcal{U}^n \cdot \boldsymbol{\Phi}$ and $p_h^n = P^n \cdot \boldsymbol{\varphi}$. The resulting weak form may then be written as

$$\mathcal{R}_h^n = \mathbf{0}, \tag{20}$$

where

$$(\mathcal{R}_h^n)_k := \begin{cases} \int_{\Omega_h} \frac{\partial \Psi(\boldsymbol{E}_h^n, p_h, t^n)}{\partial \boldsymbol{E}} : (\boldsymbol{F}_h^n)^T \nabla \Phi_k - \int_{\delta\Omega_h} \mathbf{t}(t^n) \cdot \Phi_{\mathbf{k}}, & 1 \leqslant k \leqslant K_{\boldsymbol{u}} \\ \int_{\Omega_h} \frac{\partial \Psi(\boldsymbol{E}_h^n, p_h^n, t^n)}{\partial p} \varphi_k, & K_{\boldsymbol{u}} + 1 \leqslant k \leqslant K_{\boldsymbol{u}} + K_p \end{cases} \tag{21}$$

represents the residual function. The nonlinear mechanics system is subsequently solved using the Newton–Raphson scheme with line search and Jacobian reuse outlined in [51].

## 5. PARALLEL IMPLEMENTATIONS

This section describes both the GPU implementation and the CPU multi-physics software providing the infrastructure into which the GPU code is integrated. CPU and GPU codes have been developed in different languages (FORTRAN2003 and CUDA). Both CPU and GPU parts have been integrated, defining data structures to act as input and output interfaces. These interfaces are updated in the following way. During each simulation cycle, the CPU code updates the input data associated to the GPU functionality (ODE solve, PDE solve or mechanics) before running it. Once the GPU code finishes, output data is copied back to CPU memory, and the CPU side proceeds with the simulation. The following subsections explain different parallelization strategies of CPU and GPU parts.

### *5.1. CPU implementation*

All CPU simulations were run in the finite element code $\mathcal{C}$Heart. Developed for modeling multi-physics fluid-structure interaction in the heart [52–56], $\mathcal{C}$Heart has been further developed to incorporate additional physical systems and provide flexible multi-physics integration. Support for many finite element discretization schemes, physics, and coupling along with domain partition and parallelization are some of the core features of $\mathcal{C}$Heart. The automatic domain partition is carried out using the widely available open source software ParMetis [57]. Partitioning is computed in parallel, using an element-based partition, in which each subdomain is uniquely assigned to an individual core. Subsequently, all FEM-based procedures are computed over elements on a core and requisite computations passed between ranks to form global residuals and matrices using MPI. In this



DOI: 10.1002/cnm

way, the original mesh is partitioned by minimizing communication surface between subdomains and maximizing load balance. Parallelization is carried out at a low level, so, if properly coded, each of the coupled problem retains a good scalability. For solving the linear systems resulting from the different multi-physics problems, *C*Heart uses a number of established libraries including PETSc [58], MUMPS [59], and SuperLU [60]. For the monodomain problem, the algebraic system of equations is solved using Jacobi-preconditioned CG within PETSc. For the mechanics problem, MUMPS (a direct parallel solver) was used for solving the system of PDEs. Although direct methods are known to exhibit suboptimal scalability (with system size as well as the number of cores), they are particularly efficient for the mechanical system considered in this study.

### *5.2. GPU implementation*

*5.2.1. Cardiac electrical activation.* We have implemented on the GPU the solution process to Equations (1) and (2) of the electrical activation problem, as two different parts. The first part performs the solution of the system of ODEs, and the second one performs the solution of the system of PDEs present in the monodomain equation.

The integration of the state variables of each cell model is a trivially parallel task ideally suited for SIMD processing. This is due to the decoupling of each ODE model in space, which involves an update of each state variable that has no implicit dependence on membrane potential or implicit/explicit dependence on state variables at other nodes. For this reason, each GPU thread updates the cell model state variables in parallel and integrates the computed values. To further capitalize on the GPU architecture, the ODE model was coded to use fast on-chip memory, loading cell model parameters onto the shared memory of each GPU block. In this way, threads within the same block can share these values, reducing the memory latency.

Another factor that limits the GPU performance is the per thread register bench usage. The use of this resource is based on the private (local) data and number of instructions in a GPU kernel (function). The number of state variables of the implemented cell model and the set of equations involved results in high register usage values. For this reason, a number of automatic transformations are performed within the cell model code. The initial C language code was obtained from the publicly available repository CellML [61]. This code was then automatically transformed by reducing the number of temporal variables required. In addition, operations in the cell model equations involving constant values were evaluated and collapsed to a single value. These transformations allowed us to obtain a more efficient usage of the private register bench of each GPU thread.

Performance of the ODE code was further improved by reducing the GPU thread divergence. On the GPU, as opposed to the CPU case, threads within a block run concurrently only if they execute the same instruction. However, threads might follow different branches in a conditional statement, reducing the synchronization between threads and thus reducing the parallelism. To avoid this issue with conditionals (which are often used within cell models), we instead employ Heaviside functions. In this way, we can mimic the conditional as a product between a literal and the condition of the Heaviside function. In this way, results of the function are calculated, whereas conditional statements are avoided.

The other part of the monodomain problem ported to the GPU is the solution step for the system of PDEs. We have implemented the Jacobi preconditioned conjugate gradient (CG) method [62]. For the PDE implementation, a hybrid approach has been adopted. In this hybrid version, the CPU controls the code flow of the CG method (i.e., evaluates conditions such as termination criteria, etc.), and the GPU performs in parallel vector–vector operations present in the CG method. In order to optimize productivity while maintaining the efficiency of the GPU code, vector–vector operations have been implemented using two CUDA libraries (CUSPARSE[§] and CUBLAS[¶]). At each EP simulation step, the system of PDEs is solved by copying the required data from CPU memory to GPU memory, the system is then solved using the GPU-based Jacobi–CG implementation and the result copied back to CPU memory.

[§] http://developer.nvidia.com/cuda/cusparse
[¶] http://developer.nvidia.com/cuda/cublas



DOI: 10.1002/cnm

*5.2.2. Cardiac mechanics.* To accelerate the simulation of cardiac mechanics, we examined the first-order effects, which influence the compute time of the whole cycle. Because of the system matrix reuse strategy which significantly improves compute times [51] by reducing the number of matrix builds and factorizations, residual evaluations consume most of the compute time. As a result, our initial focus was to port both residual evaluations and Jacobian computations to the GPU. The Jacobian was computed and the residual evaluated locally for each element and later added to both the global Jacobian and residual. These per element Jacobian and residual computations have been parallelized on the GPU.

There are some calculations common to both the Jacobian computation and the residual evaluation. Specifically, these are related to the computation of the tensors and terms involved in the mechanics equations as well as the stress computation according to the constitutive law. Because each of these terms are evaluated at gauss points, their computation has also been parallelized on the GPU by assigning each gauss point computations to a GPU block. In this way, tensor operations for each gauss point are executed in parallel by threads within a GPU block.

Using the different mechanics term computations, the Jacobian of each element is calculated by means of a central finite difference method, which perturbs the displacement solution and re-evaluates all the mechanics terms and the residual. This perturbation method iterates for each element over the number of nodes, $n$, and each dimension of the displacement variable, $d$. Thus, the number of iterations is $n * d$. The central perturbation method has been parallelized by assigning to each GPU block computations of each dimension at each node (i.e., the number of blocks launched is $n * d$).

After the Jacobian is computed, the residual is then evaluated at each node for both the displacement and pressure variables. For this reason, this task has been parallelized by assigning to each GPU block computations of each node (i.e., the number of blocks launched is equal to the number of displacement nodes plus number of pressure nodes).

## 6. RESULTS

This section reports on performance improvements provided by the GPU for both the electrical and mechanical components of the cardiac simulations and the impact, in terms of execution times, that such improvements can have within the clinical context. Regarding the electrophysiology problem, we have run a range of different monodomain simulations with different mesh sizes. In order to check the performance when the mesh resolution decreases, we have used a recently established benchmark for the simulation of electrical activation [47]. To be consistent with the previous benchmark study, we have used a PDE time step of 0.01 ms and an ODE time step of 0.0005 ms for different resolutions (see Figure 2(a)); field variables within all of these meshes were interpolated using linear basis functions.

We have also obtained results using a realistic mesh of the human left ventricle (LV) at two different resolutions (see Figure 2(b)), both meshes again used linear basis functions. We have obtained activation time values by simulating 300 ms of electrical activity, setting the PDE time step to 0.01 ms and ODE time step to 0.005 ms. Adaptive stepping [34] allows us to use this small step size of 0.005 ms only during upstroke and alter the ODE step to a higher step size of 1/33 ms rest of the time. For the finest resolution meshes, these time step values were shown previously to be sufficient for numerical convergence [47]. Electrical activation was solved on the deformed mesh at end-diastole, reflecting the geometry at which activation typically occurs. Further, as the generation of contraction in the normal heart typically occurs on a longer time scale and the physiological significance of deformation on electrical conductivity remains debated, the mesh was assumed static.

Figure 3(a,b) shows the propagation of the membrane potential in the benchmark mesh with resolution 0.2 mm at two different stages when using the GPU. Figure 3 shows the activation times for the same resolution, represented by a color map and contour bands. Figure 4(a,b) shows the propagation of the membrane potential in the human left ventricular mesh at different stages when using the GPU. Figure 4 shows the activation times for the same mesh, represented by a color map and contour bands.



DOI: 10.1002/cnm

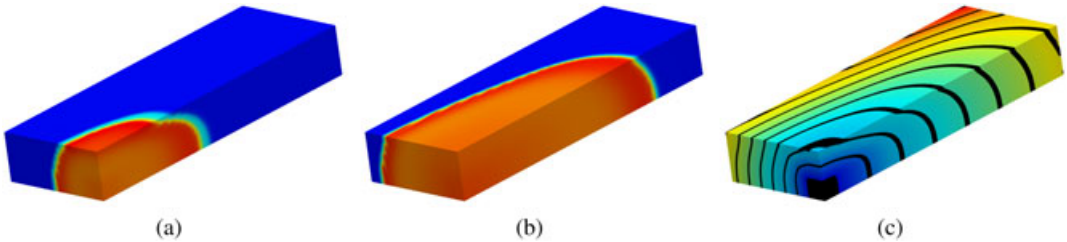


Figure 3. (a) and (b), Electrical activity propagation in the benchmark mesh at different simulation stages represented by a color map from dark blue (−86) to red (35); (c) activation times represented by a color map from dark blue (0) to red (57) and contour bands.

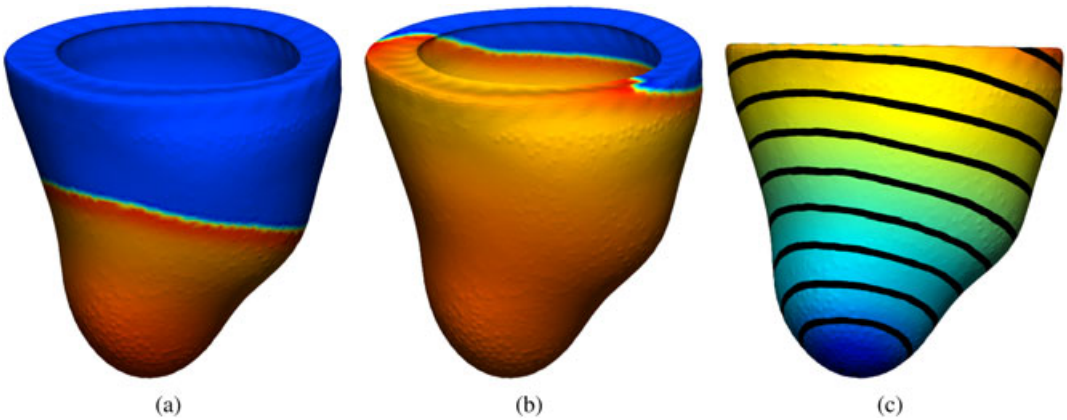


Figure 4. (a) and (b) Electrical activity propagation in a human left ventricular mesh at different simulation stages represented by a color map from dark blue (−86) to red (35); (c) activation times represented by a color map from dark blue (0) to red (75) and contour bands.

For mechanics problem, we have simulated the model described in Section 4.2, and we have used the same LV human geometry as for the electrophysiology problem. However, in this case, it has been discretized on the basis of a coarser mesh (see Figure 2(c)), reflecting the type of meshes often observed in cardiac mechanics (although the results illustrated are expected to be consistent with larger cardiac mechanics meshes). To solve the mechanics problem on this mesh, we have mapped the activation time from the fine grid electrophysiology mesh onto the mechanics mesh. Using these activation time values, we simulated the cardiac cycle, comparing performance over a single beat (with a duration of 1 s and a time step of 0.001 ms). Figure 5 shows displacement values during a cycle simulation. Figure 6 shows the principal strain vectors and fibers at end diastole and mid systole steps.

The different implementations have been compared using GPU, SC CPU, and MC CPU platform configurations. We enumerate the specifications of the different platforms in order to do a performance/price/power consumption comparison. For the CPU simulations, we used a machine with 32-core AMD Opterons @ 2.0 GHz and 128 GB of RAM shared among all cores. The processor used in the CPU tests has a power consumption range of ∼800/520 W and a price of ∼£4100. The theoretical performance of this platform per core is 17.75 GFlops. For our GPU simulations, we have used up to four Tesla C2070 processors, each one with 448 SPs and 6 GB of device memory. The theoretical performance of the GPU processor is 515 GFlops, with a power consumption of 238 W and a price of ∼£1600. Previous works comparing CPU/GPU implementations of biomedical problems do a single core CPU/single GPU comparison without taking into account the theoretical performance of each architecture. For this reason, we compare the performance of the GPU implementation executed on a Tesla C2070 processor with up to 32 CPU cores (with a theoretical peak performance of 568 GFlops). In this way, we are able to analyze the performance provided by the single core, multi-core, and GPU implementations. The accuracy of GPU implementations has been



DOI: 10.1002/cnm

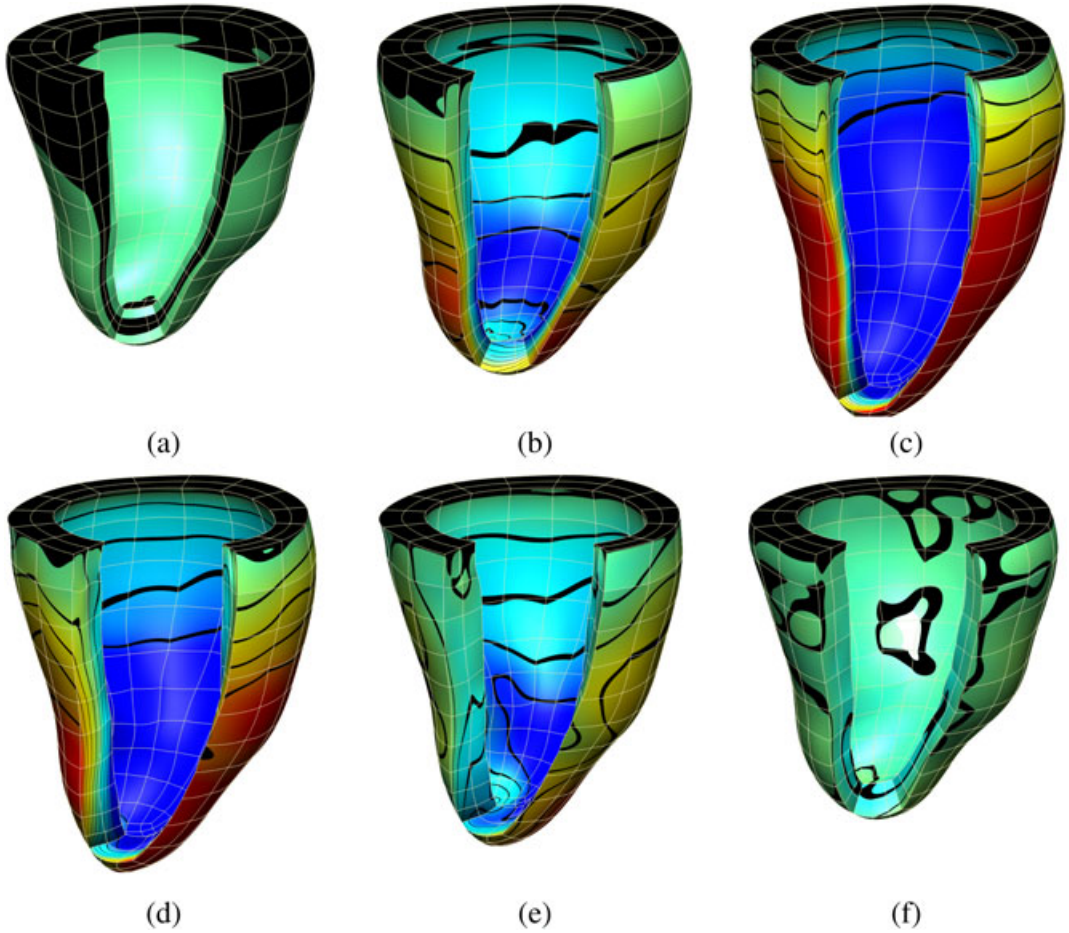


Figure 5. (a), (b), and (c) show the displacement values in the fiber direction at three time steps during diastole; (d), (e), and (f) at three time steps during systole. Displacement values are represented by a color map from dark blue (−5.0) to red (5.9) and contour bands.

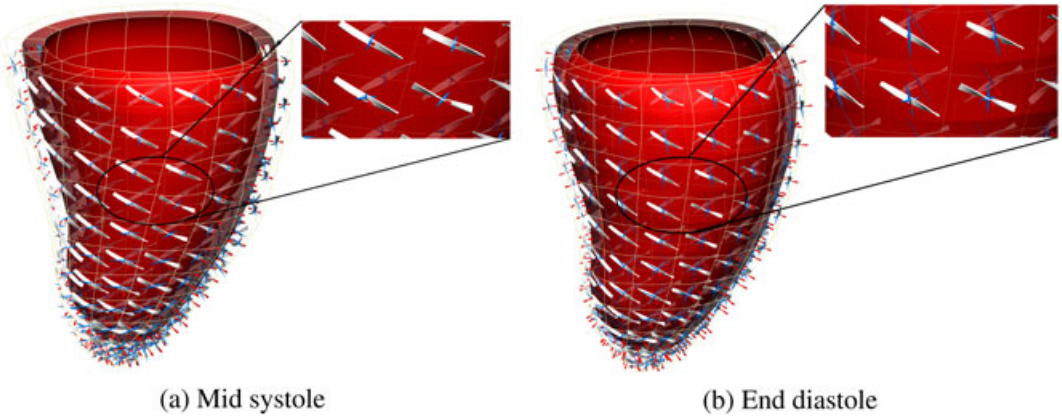


Figure 6. Detail of fibers and strain tensor (a) at mid systole (b) at end of diastole. Glyphs represent principle components of strain as follows: blue (stretch) and red (compression).



DOI: 10.1002/cnm

determined by comparing results provided by GPU and CPU simulations. For the monodomain problem, we compared membrane potential values obtaining a maximum difference of 1.0E-13. For the mechanics problem, we compared displacement, pressure, and fiber field values obtaining a maximum difference of 1.0E-15.

Figure 7 shows the speedup obtained by the different parallel platforms when solving the ODE problem. These results clearly demonstrate that the GPU outperforms both the sequential and parallel CPU versions. They also demonstrate that the performance is further improved when the problem size increases mainly because of the high memory bandwidth available on the GPU. On the other hand, Figure 7 shows that the GPU version is always faster than the multi-core CPU despite the fact that the CPU platform has a slightly higher theoretical performance. Specifically, the GPU achieves a 5.5× speedup compared with the MC CPU for the biggest LV mesh with respect to the multi-core CPU. This speedup allows to reduce the ODE run time from 93 h when using the MC CPU down to 17 h when using the GPU for the largest LV mesh.

Figure 8 shows the speedup obtained when comparing a single GPU version with respect to two and four GPUs, demonstrating the scalability of the ODE problem when using several GPUs. Results show that almost linear scalability is obtained for the ODE problem. Furthermore, when comparing the results grouped by mesh type, it can be seen that the performance is further increased with the problem size evidenced by the acceleration for the bigger LV mesh which is higher in comparison with the smaller LV mesh. These results demonstrate the potential of the GPU architecture for accelerating the ODE component of the solution procedure, specially for large-scale geometries.

Figure 9 shows the speedup obtained by the different parallel platforms considered when solving the PDE problem. Speedup values for the CPU versions have been obtained using the execution times provided by PETSc, which is the library used for solving the PDE. This comparison again demonstrates that the GPU outperforms both the SC and MC CPU versions. Performance is also further improved when the problem size increases. Unlike the ODE problem, the PDE algorithm provides lower acceleration rates between the GPU and the different CPU platforms. This is mainly because the CG algorithm requires more synchronization among GPU threads, resulting in a

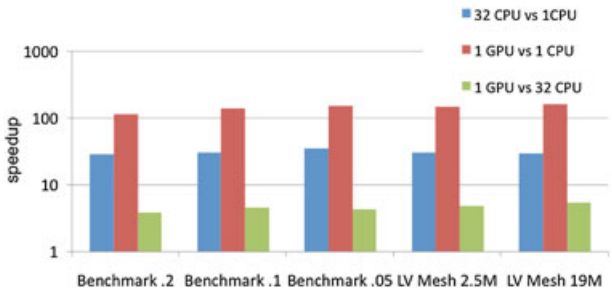


Figure 7. Speedup obtained by the different parallel platforms for the ODE problem.

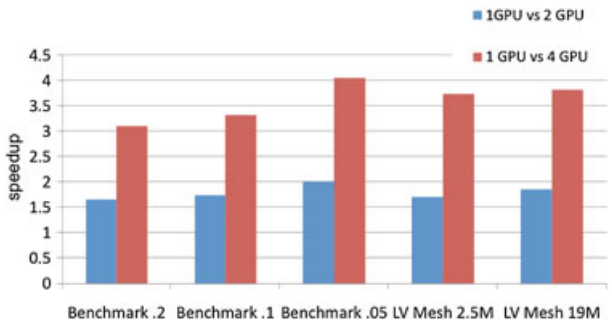


Figure 8. Scalability of the ODE problem when using multiple GPUs.



DOI: 10.1002/cnm

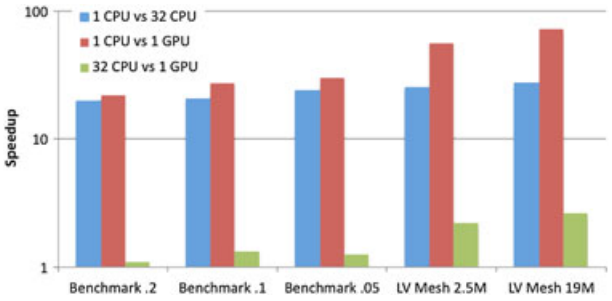


Figure 9. Speedup obtained by the different parallel platforms for the PDE problem.

performance degradation. Nevertheless, the GPU version always outperforms the MC CPU achieving up to 2.6× speedup for the largest LV mesh. This speedup allows to reduce the PDE run time from 53 h when using the MC CPU down to 20 h when using the GPU for the largest LV mesh.

Tables I and II show the performance provided by the different platforms when running a whole cycle mechanics simulation. Table I shows the total run time (in seconds) for the GPU and CPU versions and the percentage of the total run time that the three main tasks of the simulation take (i.e., Jacobian computation, residual evaluation, and PDE solve). We report results for these three tasks because for the single CPU version, they take most of the simulation run time (28146.5 s). When the mechanics simulation is parallelized on the multi-core CPU, the total run time is significantly reduced (1481.4 s). For the 32-core CPU version, the percentage of the total run time required by the Jacobian computation and the residual evaluation are reduced, but for the PDE solution, the step is increased (10%), meaning that the latter task provides a lower scalability. Looking at the GPU results, the percentage of time required by the Jacobian and residual computations is further decreased.

For heart problem, the MC CPU required 1585 s for 1 ms of simulated time on the largest LV mesh, whereas the GPU required only 400 s, resulting in a significant improvement in computation time. The performance was nearly equivalent for the solid mechanics problem, with both the MC CPU/1 GPU requiring 1.5 s for 1 ms of simulated time. However, this is due to the non-parallelized PDE solve, which is 3.7× faster in the MC CPU simulation (see Table II). Accounting for this time, we see that significant improvement in the MC CPU computation time.

Table II shows the speedup of the parallel CPU and GPU versions with respect to the sequential CPU version as well as the speedup of the GPU with respect to the parallel CPU version. This speedup evaluation between the different platforms has been performed by comparing the execution

Table I. Total run time split in mechanics cycle simulation.

| Stage | $1_{CPU}$ | $32_{CPU}$ | $1_{GPU}$ |
|---|---|---|---|
| Jacobian | 19% | 15% | 8% |
| Residual | 79% | 73% | 43% |
| PDE solve | 2% | 10% | 39% |
| Total time | 28146.5 s | 1481.4 s | 1481.6 s |

Table II. Speedup of the parallel CPU and GPU implementations with respect to the sequential CPU version.

| Stage | $1_{CPU}/32_{CPU}$ | $1_{CPU}/1_{GPU}$ | $32_{CPU}/1_{GPU}$ |
|---|---|---|---|
| Jacobian | 22.3× | 44.5× | 2.0× |
| Residual | 20.4× | 34.8× | 1.7× |
| PDE solve | 3.7× | 1.0× | 0.3× |



DOI: 10.1002/cnm

time of the three tasks of the simulation: the PDE solve time and the two parts of mechanics implemented on the GPU (i.e., Jacobian computation and residual evaluation). Looking at these results, it can be seen that tasks implemented on the GPU outperform both the sequential and parallel CPU versions of the same tasks. Comparing the sequential CPU and GPU versions, the following acceleration factors are obtained: 44.5× (Jacobian computation) and 34.8× (residual evaluation). When comparing the parallel CPU and GPU versions, the following acceleration factors are obtained: 2.0× (Jacobian computation) and 1.7× (residual evaluation). However, the PDE solve task only runs in parallel for the 32-core version and runs sequentially for the single CPU and GPU simulations. The acceleration factors provided by the parallel CPU and GPU versions enable the simulation of one cycle in around 25 min. However, if we consider the same PDE solve time for the GPU simulation as for the 32-core simulation, the GPU total run time is decreased to 15 min. It should be noticed that the GPU can improve the computational performance of electromechanical simulations while the price ratio MC CPU/GPU is around 2.5 (i.e., the GPU is 2.5× cheaper) and the power consumption ratio MC CPU/GPU is 3.36–2.18 (i.e., the GPU consumes less energy).

## 7. CONCLUSIONS

The application of electromechanical models within time sensitive environments such as the clinic, requires significant advancement of the computational software used to solve both cardiac electrical activation and mechanics. Previous efforts have addressed this problem by efficiently exploiting the computational capabilities of HPC based on clusters of CPU processors. Although significant speedups were obtained, these platforms have the disadvantage of a high cost in terms of price and power consumption. For tackling these problems, the GPU has arisen as an efficient platform providing a good power/performance ratio. Previous works have proposed the use of GPUs for solving the cardiac electrical activation problem. Building on these works, we have shown the potential utility of GPUs for simulating both electrical activation and mechanics within the human heart.

Specifically, we have developed a GPU-based scheme to enable the acceleration of a human scale electrical activation problem and a novel implementation of cardiac mechanics on the GPU. To evaluate the effectiveness of our implementation, we have focused on performing a comparison between a GPU and a multi-core CPU with similar theoretical performance. The GPU implementations were developed to take advantage of the features of this parallel platform and allowed to significantly accelerate the different problems simulated for human scale models. Concretely, for the human LV mesh (∼19 M. DOFs) speedups of 5.5× and 2.6× were achieved for the ODE solve and PDE solution steps, respectively. Regarding mechanics, for the same human geometry, speedups of 1.7× and 2.0× were obtained for the residual evaluation time and Jacobian computation time, respectively. The fact that this performance comparison was performed using a GPU and a multi-core CPU with similar theoretical performance provides an unbiased assessment of the capacity for the GPU platform to accelerate computations focused on electromechanical coupling. In addition, the GPU is more efficient offering a price ratio MC CPU/GPU of 2.5 (i.e., the GPU is 2.5× cheaper) and a power consumption ratio MC CPU/GPU of 3.36–2.18 (i.e., the GPU requires less energy).

Although the results presented in this paper show the benefits that the GPU architecture can provide to simulate VPH cardiac models, some improvements and extensions in functionality are left as future work. In this study, the GPU electrophysiology implementation only uses one cell model. The development of additional cell models for the GPU requires significant technical skill in comparison with coding the same model for the CPU. In order to generalize the use of the GPU platform to the VPH community, it would thus be desirable to develop a tool for automatically generating GPU code and add this functionality to existent cell model repositories [61]. Furthermore, this study proposes the implementation of the PDE solution step in the electrophysiology problem on a single GPU. Some previous studies have solved the system of PDEs using multiple GPUs [9]. In this approach, the communication between GPUs is handled by the CPU through MPI. However, CUDA has recently released a new peer-to-peer communication method where GPUs, within the same node, communicate directly through the Peripheral Component Interconnect (PCI) bus. In this way, a hierarchical method could be adopted where GPUs hosted in different nodes communicate through MPI, and GPUs within the same node communicate using the PCI bus interface.



DOI: 10.1002/cnm

Although the GPU mechanics implementation presented in this study provides a performance improvement with respect to MC CPU, there remains significant further potential for exploiting the GPU capabilities. In the GPU mechanics code, Jacobian and residual per element computations are performed in parallel on the GPU. Nevertheless, the mechanics code could be further accelerated by performing multiple elements computations in parallel. Computing multiple elements in parallel results in a higher consumption of GPU memory. However, the acceleration obtained justifies the higher memory required. This has been already observed in our mechanics implementation for the Jacobian computations. The Jacobian matrix is built using a perturbation method, which iterates over the number of DOFs of the mechanics problem. Because this loop is parallelized on the GPU, a higher speedup factor is provided for the Jacobian computation (2.0×) with respect to the residual evaluation (1.7×). Furthermore, because mechanics run time is dominated by residual computations (see Table I), the reduction of residual time should result in a significant acceleration of the total run time. As presented in the results section, the PDE solve step in the mechanics simulation was run sequentially on the CPU. This step can be also parallelized implementing on the GPU a direct solver or a preconditioned iterative solver. This parallelization of the PDE solve step would lead to a reduction of the mechanics total time.

## ACKNOWLEDGEMENTS

The authors would like to acknowledge funding from the EPSRC (EP/G007527/2), the King's College Medical Engineering Centre funded by the Wellcome Trust and EPSRC (WT088641/2/09/2), the Virtual Physiological Rat project (NIH IPG50GM094503-01), VPH-Share funded by the European Commission and the British Heart Foundation (NH/11/5/29058). We also acknowledge the support from the Wellcome Trust-EPSRC Centre of Excellence in Medical Engineering (WT 088641/Z/09/Z) and the NIHR Biomedical Research Centre at Guy's and St.Thomas' NHS Foundation Trust and KCL. The views expressed are those of the authors and not necessarily those of the NHS, the NIHR, or the DoH.

## REFERENCES

1. Smith N, de Vecchi A, McCormick M, Nordsletten D, Camara O, Frangi AF, Delingette H, Sermesant M, Relan J, Ayache N, Krueger MW, Schulze WHW, Hose R, Valverde I, Beerbaum P, Staicu C, Siebes M, Spaan J, Hunter P, Weese J, Lehmann H, Chapelle D, Rezavi R. euheart: personalized and integrated cardiac care using patient-specific cardiovascular modelling. *Interface Focus* 2011; **1**(3):349–364.
2. Nordsletten D, Niederer S, Nash M, Hunter P, Smith N. Coupling multi-physics models to cardiac mechanics. *Progress in Biophysics and Molecular Biology* 2011; **104**(1-3):77–88.
3. Niederer SA, Smith NP. At the heart of computational modelling. *Journal of Physiology*; **590**(2012):1331–1338.
4. Xi J, Lamata P, Niederer S, Land S, Shi W, Zhuang X, Ourselin S, Duckett S, Shetty A, Rinald C, Rueckert D, Razavi R, Smith N. The estimation of patient-specific cardiac diastolic functions from clinical measurements. *Medical Image Analysis* 2013; **17**:133–146.
5. Niederer SA, Plank G, Chinchapatnam P, Ginks M, Lamata P, Rhode KS, Rinaldi CA, Razavi R, Smith NP. Length-dependent tension in the failing heart and the efficacy of cardiac resynchronization therapy. *Cardiovascular Research* 2011; **89**:336–343.
6. Bartocci E, Cherry EM, Glimm J, Grosu R, Smolka SA, Fenton FH. Toward real-time simulation of cardiac dynamics. In *Proceedings of the 9th International Conference on Computational Methods in Systems Biology*, CMSB '11. ACM: New York, NY, USA, 2011; 103–112.
7. Vigmond EJ, Boyle PM, Leon L, Plank G. Near-real-time simulations of biolelectric activity in small mammalian hearts using graphical processing units. *Conf Proc IEEE Eng Med Biol Soc* 2009; **2009**:3290–3.
8. Rocha BM, Campos FO, Amorim RM, Plank G, Santos RWd, Liebmann M, Haase G. Accelerating cardiac excitation spread simulations using graphics processing units. *Concurr. Comput. : Pract. Exper.* 2011; **23**(7):708–720.
9. Neic A, Liebmann M, Hoetzl E, Mitchell L, Vigmond E, Haase G, Plank G. Accelerating cardiac bidomain simulations using graphics processing units. *IEEE Transactions on Biomedical Engineering* 2012; **59**:2281–2290.
10. Sato D, Xie Y, Weiss J, Qu Z, Garfinkel A, Sanderson A. Acceleration of cardiac tissue simulation with graphic processing units. *Medical & Biological Engineering & Computing* 2009; **47**(9):1011–1015.
11. Niederer S, Mitchell L, Smith N, Plank G. Simulating human cardiac electrophysiology on clinical time-scales. *Frontiers in Physiology* 2011; **2**(14):1–7.
12. Reumann M, Fitch BG, Rayshubskiy A, Keller DUJ, Seemann G, Dossel O, Pitman MC, Rice JJ. Strong scaling and speedup to 16,384 processors in cardiac electro-mechanical simulations. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, 2009; 2795–2798, DOI: 10.1109/IEMBS.2009.5333802.



DOI: 10.1002/cnm

13. Hosoi A, Washio T, Okada J, Kadooka Y, Nakajima K, Hisada T. A multi-scale heart simulation on massively parallel computers. *International Conference on High Performance Computing, Networking, Storage and Analysis*, 2010; 1–11.
14. Lafortune P, ArÃs R, Vazquez M, Houzeaux G. Coupled electromechanical model of the heart: parallel finite element formulation. *International Journal for Numerical Methods in Biomedical Engineering* 2012; **28**(1): 72–86.
15. Nash M, Hunter P. Computational mechanics of the heart. *Journal of Elasticity* 2000; **61**:113–141.
16. Sainte-Marie J, Chapelle D, Cimrman R, Sorine M. Modeling and estimation of the cardiac electromechanical activity. *Computers & Structures* 2006; **84**:1743–1759.
17. Stevens C, Remme E, LeGrice I, Hunter P. Ventricular mechanics in diastole: material parameter sensitivity. *Journal of Biomechanics* 2003; **36**(5):737–748.
18. Göktepe S, Kuhl E. Electromechanics of the heart: a unified approach to the strongly coupled excitation-contraction problem. *Computational Mechanics* 2010; **45**:227–243.
19. NVIDIA, 2012. NVIDIA CUDA Programming Guide 4.2.
20. Clayton RH, O OB, Cherry E, Dierckx H, Fenton F, Mirabella L, Panfilov A, Sachse F, G GS, Zhang H. Models of cardiac tissue electrophysiology: progress, challenges and open questions. *Progress in Biophysics & Molecular Biology* 2011; **104**:22–48.
21. Keener J, Sneyd J. *Mathematical Physiology*. Springer: New York, NY, USA, 2004.
22. Henriquez C. Simulating the electrical behavior of cardiac tissue using the bidomain model. *Critical Reviews in Biomedical Engineering* 1993; **21**(1):1–77.
23. Muler A, Markin V. Electrical properties of anisotropic nerve-muscle syncytia-I. Distribution of the electrotonic potential. *Biofizika* 1977; **2**(22):307–312.
24. Muler A, Markin V. Electrical properties of anisotropic nerve-muscle syncytia-II. Spread of flat front of excitation. *Biofizika* 1977; **3**(22):518–522.
25. Muler A, Markin V. Electrical properties of anisotropic nerve-muscle syncytia-III. Steady form of the excitation front. *Biofizika* 1977; **4**(22):671–675.
26. Gulrajani R. *Bioelectricity and Biomagnetism*. Wiley, 1998.
27. Leon LJ, Horacek BM. Computer model of excitation and recovery in the anisotropic myocardium. I. Rectangular and cubic arrays of excitable elements. *Journal of Electrocardiology* 1991; **24**(1):1–15.
28. Bueno-Orovio A, Cherry EM, Fenton FH. Minimal model for human ventricular action potentials in tissue. *JournalAnnals ofTheoretical Biology* 2008; **253**(3):544–560.
29. Dal H, Goktepe S, Kaliske M, Kuhl E. A fully implicit finite element method for bidomain models of cardiac electrophysiology. *Computer Methods in Biomechanics and Biomedical Engineering* 2012; **15**:645–656. DOI: 10.1080/10255842.2011.554410.
30. Pathmanathan P, Bernabeu MO, Bordas R, Cooper J, Garny A, Pitt-Francis JM, Whiteley JP, Gavaghan DJ. A numerical guide to the solution of the bidomain equations of cardiac electrophysiology. *Progress in Biophysics & Molecular Biology* 2010; **102**:136–155.
31. Rocha B, Kickinger F, Prassl A, Haase G, Vigmond E, dos Santos R, Zaglmayr S, Plank G. A macro finite-element formulation for cardiac electrophysiology simulations using hybrid unstructured grids. *IEEE Transactions on Biomedical Engineering* 2011; **58**:1055–65.
32. Trew M, Le Grice I, Smaill B, Pullan A. A finite volume method for modeling discontinuous electrical activation in cardiac tissue. *Annals of Biomedical Engineering* 2005; **33**:590–602.
33. Brenner S, Scott RL. *The Mathematical Theory of Finite Element Methods*. Springer: New York, NY, USA, 2005.
34. Land S, Niederer SA, Smith NP. Efficient computational methods for strongly coupled cardiac electromechanics. *IEEE Transactions on Biomedical Engineering* 2012; **59**(5):1219–1228.
35. hsing Luo C, Rudy Y. A model of the ventricular cardiac action potential - depolarisation, repolarisation and their interaction. *Circulation Research* 1991; **68**:1501–1526.
36. ten Tusscher KH, Panfilov AV. Alternans and spiral breakup in a human ventricular tissue model. *Am. J. Physiol Heart Circ. Physiol.* 2006; **291**:H1088–H1100.
37. Qu Z, Garfinkel A. An advanced algorithm for solving partial differential equation in cardiac conduction. *IEEE Transactions on Biomedical Engineering* 1999; **46**:1166–1168.
38. Hundsdorfer W, Verwer J. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*. Springer: New York, NY, USA, 2010.
39. Vigmond E, Hughes M, Plank G, Leon L. Computational tools for modeling electrical activity in cardiac tissue. *Journal of Electrocardiology* 2003; **36**:69–74.
40. Mardal KA, Skavhaug O, Lines GT, Staff GA, Odegard A. Using python to solve partial differential equations. *Computing in Science and Engineering* 2007; **9**(3):48–51.
41. Heidenreich EA, Ferrero JM, Doblare M, Rodriguez JF. Adaptive macro finite elements for the numerical solution of monodomain equations in cardiac electrophysiology. *Annals of Biomedical Engineering*; **38**(7): 2331–2345.
42. Bonet J, Wood R. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press: Cambridge, UK, 1997.
43. Malvern L. *Introduction to the Mechanics of Continuous Medium*. Prentice-Hall: Upper Saddle River, NJ, USA, 1969.



DOI: 10.1002/cnm

44. Wang C, Truesdell C. *Introduction to Rational Elasticity (Mechanics of Continua)*. Springer: New York, NY, USA, 1973.
45. Costa K, Holmes J, McCulloch A. Modeling cardiac mechanical properties in three dimensions. *Philisophical Transactions of the Royal Society* 2001; **359**:1233–1250.
46. Kerckhoffs R, Bovendeerd P, Prinzen F, Smits K, Arts T. Intra-and interventricular asynchrony of electromechanics in the ventricularly paced heart. *Journal of Engineering Mathematics* 2003; **47**:201–16.
47. Niederer SA, Kerfoot E, Benson AP, Bernabeu MO, Bernus O, Bradley C, Cherry EM, Clayton R, Fenton FH, Garny A, Heidenreich E, Land S, Maleckar M, Pathmanathan P, Plank G, RodrÃguez JF, Roy I, Sachse FB, Seemann G, Skavhaug O, Smith NP. Verification of cardiac tissue electrophysiology simulators using an N-Version benchmark. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 2011; **369**(1954):4331–4351.
48. Nielson P, Grice IL, Smaill B, Hunter P. Mathematical model of geometry and fibrous structure of the heart. *American Journal of Physiology* 1991; **260**:H1365–H1378.
49. Shi Y, Korakianitis T. Numerical simulation of cardiovascular dynamics with left heart failure and in–series pulatile ventricular assist device. *Artificial Organs* 2006; **30**:929–948.
50. Korakianitis T, Shi Y. A concentrated parameter model for the human cardiovascular system including heart valve dynamics and atrioventricular interaction. *Medical Engineering and Physics* 2006; **28**:613–628.
51. McCormick M, Nordsletten D, Kay D, Smith N. Simulating left ventricular fluid-solid mechanics through the cardiac cycle under lvad support. *Journal of Computational Physics* 2013; **244**:80–96. DOI: 10.1016/j.jcp.2012.08.008.
52. Lee J, Niederer S, Nordsletten D, Grice IL, Smaill B, Kay D, Smith N. Coupling contraction, excitation, ventricular and coronary blood flow across scale and physics in the heart. *Philisophical Transactions of the Royal Society A* 2009; **367**:2311–2331.
53. Nordsletten D, Kay D, Smith N. A non–conforming monolithic finite element method for problems of coupled mechanics. *Journal of Computational Physics* 2010; **20**:7571–7593.
54. Nordsletten D, McCormick M, Kilner P, Kay D, Smith N. Fluid-solid coupling for the investigation of diastolic and systolic human left ventricular function. *International Journal for Numerical Methods in Biomedical Engineering* 2011; **27**:1017–39.
55. McCormick M, Nordsletten D, Kay D, Smith N. Modelling left ventricular function under assist device support. *International Journal for Numerical Methods in Biomedical Engineering* 2011; **27**:1073–1095.
56. de Vecchi A, Nordsletten D, Remme E, Bellsham-Revell H, Greil G, Simpson J, Razavi R, Smith N. Inflow typology and ventricular geometry determine efficiency of filling in the hypoplastic left heart. *Annals of Thoracic Surgery* 2012; **94**:1562–1569.
57. Schloegel K, Karypis G, Kumar V. Multilevel diffusion schemes for repartitioning of adaptive meshes. *Journal of Parallel and Distributed Computing* 1997; **47**:109–124.
58. Balay S, Gropp WD, McInnes LC, Smith BF. Efficient management of parallelism in object oriented numerical software libraries. In *Modern Software Tools in Scientific Computing*, Arge E, Bruaset AM, Langtangen HP (eds). Birkhäuser Press: Basel, Switzerland, 1997; 163–202.
59. Amestoy PR, Duff IS, L'Excellent JY, Koster J. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications* 2001; **23**(1):15–41.
60. Li XS, Demmel JW. SuperLU_DIST: a scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software* 2003; **29**(2):110–140.
61. Cuellar AA, Lloyd CM, Nielsen PF, Bullivant DP, Nickerson DP, Hunter PJ. An Overview of CellML 1.1, a Biological Model Description Language. *SIMULATION: Transactions of the Society for Modeling and Simulation International* 2003; **79**(12):740–747.
62. Golub GH, Van Loan CF. *Matrix Computations*, 3rd Edition. The Johns Hopkins University Press: Baltimore, 1996.



DOI: 10.1002/cnm