GPU accelerated solver for nonlinear reaction-diffusion systems. Application to the electrophysiology problem

Andres Mena[a], Jose M Ferrero[c], Jose F Rodriguez[b,d,*]

[a] *CIBER, Zaragoza, Spain*
[b] *Aragon Institute for Engineering Research, Universidad de Zaragoza, Zaragoza, Spain*
[c] *Instituto de Investigacion Interuniversitario en Bioingenieria y Tecnologia Orientada al Ser Humano (I3BH), Universidad Politecnica de Valencia, Valencia, Spain*
[d] *LabS, Department of Chemistry, Materials and Chemical Engineering "Giulio Natta", Politecnico di Milano, Milano, Italy*

Abstract

Solving the electric activity of the heart posses a big challenge, not only because of the structural complexities inherent to the heart tissue, but also because of the complex electric behaviour of the cardiac cells. The multi-scale nature of the electrophysiology problem makes difficult its numerical solution, requiring temporal and spatial resolutions of 0.1 ms and 0.2 mm respectively for accurate simulations, leading to models with millions degrees of freedom that need to be solved for thousand time steps. Solution of this problem requires the use of algorithms with higher level of parallelism in multi-core platforms. In this regard the newer programmable graphic processing units (GPU) has become a valid alternative due to their tremendous computational horsepower. This paper presents results obtained with a novel electrophysiology simulation software entirely developed in Compute Unified Device Architecture (CUDA). The software implements fully explicit and semi-implicit solvers for the monodomain model, using operator splitting. Performance is compared with classical multi-core MPI based solvers operating on dedicated high-performance computer clusters. Results obtained with the GPU based solver show enormous potential for this technology with accelerations over 50× for three-dimensional problems.

*Corresponding author

Email address: mena@unizar.es (Andres Mena), cferrero@gbio.i3bh.es (Jose M Ferrero), josefelix.rodriguezmatas@polimi.it (Jose F Rodriguez)

Keywords:
electrophysiology simulation, GPU simulation, Excitable media

1. Introduction

In the last decades, mathematical modelling and computer simulations have become a useful tool for tackling problems in like science and engineering, among which heart electrophysiology. In this regard, modelling the electric activity of the heart, under physiological and pathological conditions, has attracted the attention of a number of researchers [1] since ventricular tachycardia and fibrillation are among the major causes of sudden death [2]. Since direct measurements are many times limited to only surface signals, multi-scale numerical simulations where the electrical activity at the surface as well as in the myocardium can be related to the underlying electro-chemical behaviour of the cell, helps to gain further insights into the problem. In addition, mathematical modelling and computer simulations have become a useful tool for training and educational purposes. In this regard, near real-time GPU-based simulators give the possibility the user to interact and they have been successfully employed to train bachelor students [3].

The electric activity of the heart is usually studied using the well known bidomain model [4, 5]. It consists of an elliptic partial differential equation and a parabolic partial differential equation coupled to a system of stiff non-linear ordinary differential equations describing the ionic current through the cellular membrane. This model can be simplified to the so called anisotropic monodomain equation [4], a parabolic reaction-diffusion equation describing the propagation of the transmembrane potential coupled to a system of ordinary differential equations describing the cellular ionic model. The monodomain model represents a much less computationally expensive model for the electric activity of the heart, and has been extensively used [6, 7, 8, 9].

The high computational cost of the bidomain and monodomain models is due to the stiffness of the system of ordinary differential equation describing the transmembrane ionic current which introduces different space and time scales. The depolarization front is localised in a thin layer of less than a millimetre, requiring therefore discretizations of the order of tenth of a millimetre in order to accurately resolve the depolarization front, implying models with millions of degrees of freedom to simulate the heart. The time scale is another fundamental issue in cardiac simulations. The time constants

involved in the kinetics of cellular models range from 0.1 to 600 ms, requiring in some phases of the process the use of time steps of the order of a hundredth of a millisecond. Hence, solving a single heart beat requires thousands time steps.

A number of alternatives have been proposed to solve this problem. In this particular, the multi-length scale nature of the problem has inspired the development of adaptive techniques where the mesh is allowed to change with time coupled with adaptive time integration schemes to improve the computational performance [10, 11, 12, 13]. However, dynamic loading for these adaptive schemes is still cumbersome, limiting their application in massive parallel architectures. Recent efforts [14, 15, 16] suggest the use of multilevel meshes, fixed in time, along with adaptive time schemes which take advantage of the different kinetics of the ionic currents. This allows for reductions of up to two orders of magnitude in CPU time with respect to traditional explicit algorithms. However, these techniques use the fine mesh (lower level mesh) for solving the partial differential equations (responsible for the propagation of the depolarization front).

Despite of the efforts of designing more efficient schemes, the solution of the electrophysiology problem requires the use of algorithms with higher level of parallelism in multi-core platforms. In this regard, the next generation of high-performance computing (HPC) platforms promise to deliver better performance in the PetaFLOPS range. However, achieving high performance on these platforms relies on the fact that strong scalability can be achieved, something challenging due to the performance deterioration caused by the increasing communication cost between processors as the number of cores increases. That is, with increasing number of cores, the load assigned to each processor decreases, but the communication between different processors associated with the boundaries of a given partitioned domain increases. Therefore, when communication costs domain, no further benefits are obtained from adding additional processors. An alternative to the multi-core platforms is emerging in the newer programmable graphic processing units (GPU) which in recent years has become a highly parallel, multithreaded, many-core processor with tremendous computational horsepower [17, 18]. GPUs outperform multi-core CPUs architectures in terms of memory band width, but underperforms in terms of double precision floating point arithmetic. However, GPUs are built to schedule a large number of threads, thus, reducing latencies in their multi-core architecture.

Sanderson et al. [19] proposed a general purpose, graphics processing unit

(GP-GPU)-based approach for the solution of advection-reaction-diffusion models. They report an increase of performance of up to 27 times for an explicit solver when used on three dimensional problems. Regarding cardiac electrophysiology, previous studies have reported speedups by a factor of 32 for the monodomain model [20] using an explicit finite difference scheme with a rather simple transmembrane ionic model. In their study Sato et al. [20] have established the solution of the PDE equation as the bottle neck of the computation with GPU. However, in their studies older NVidia GT8800 and GT9800 GX2 cards only supporting single precision floating point operations were used, that greatly limited the computations of the parabolic system. Chai et al. [21] succesfully solved a 25 million nodes problem on a multi-GPU architecture using the mono domain model and a four state variables model. Bartocci et al. [22] have performed an implementation of a finite difference explicit solver for cardiac electrophysiology. They evaluate the effect of the ionic model size (number of state variables) on the performance in simulating 2D tissues, and compare single precision and double precision implementation. They provided acceleration with respecto to real time. For small ionic models and the single precision implementation they report simulations faster that real time for small problems, whereas for highly detail models, with larger number of state variables, they report simulation times between 35 and 70 times larger than real time. Rocha et al. [23] implemented and an implicit method on the GPU. Spatial discretization of the parabolic equation was performed by mean of the Finite Element Methods (FEM) keeping full stiffness matrices. Promising acceleration ratios were achieved with two dimensional (2D) bidomain tissue models using an unpreconditioned CG method. However, with unstructured three dimensional (3D) bidomain simulations, the number of iterations required for convergence became prohibitive. In a more recent work Neic et al. [26] showed that 25 processors were equivalent to a single GPU when computing the bidomain equations. This new capability to solve the governing equations on a relatively small GPU cluster makes it possible to one day introduce simulation using patient specific computer models into a clinical workflow. On a more recent work, Vigueras et al. [27] have port to the GPU a number of components of a parallel c-implemented cardiac solver. They report accelerations of 164 times of the ODE solver and up to 72 times for the PDE solver. They have also achieved accelerations of up to 44 times for the mechanics residual/Jacobian computation in electromechanical simulations.

This paper presents results obtained using a novel electrophysiology sim-

ulation software entirely developed in Compute Unified Device Architecture (CUDA). The software implements semi-implicit and explicit solvers in double precision for the monodomain model, using operator splitting and FEM for spatial discretization. The potentiality of the GPU code simulations is demonstrated by running strong scalability benchmarks in 1D, 2D, and 3D problems, including realistic geometries of human heart and atria. Performance results are compared with a multi-CPU based software [9].

The remainder of the paper is organised as follows. Section 2 describes the basic equations defining the electrophysiology problem, and the basic algorithm. Section 3 details the GPU implementation and data structure. Section 4 describes the scalability benchmark examples. Section 5 details the main results obtained, and section 6 concludes with some discussion and a summary of the most relevant contributions.

2. The numerical algorithm

This section describes the proposed numerical algorithm for solving the reaction-diffusion system.

2.1. Governing equations and basic algorithm

The reaction-diffusion system describing the monodomain model for cardiac electrophysiology is given by [4]

$$\boldsymbol{\nabla} \cdot (\mathbf{D}\boldsymbol{\nabla} V) = C_m \frac{\partial V}{\partial t} + J_{ion}(V, \mathbf{u}) + J_{stm}(t), \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f}(\mathbf{u}, V, t), \tag{2}$$

where, V is the transmembrane potential, C_m is the cell capacitance, $\mathbf{D}$ is the second order symmetric and positive definite conductivity tensor, $J_{stm}(t)$ is a time dependent stimulation current, $J_{ion}(V, \mathbf{u})$ the transmembrane ionic current, and $\mathbf{u}(\mathbf{x}, t)$ is a vector of state variables related to the ionic model (gates and concentrations), $\mathbf{f}$ is a vector valued function also dependent on the ionic model, and t refers to time.

Equation (1) is subjected to the following Neumann boundary conditions

$$\mathbf{n} \cdot (\mathbf{D}\boldsymbol{\nabla} V) = 0, \tag{3}$$

where $\mathbf{n}$ is the outward pointing unit normal to the computational domain.

An efficient way of solving Equations (1-3) is by applying the operator splitting technique [28]. The basic steps for a time t^k are summarised in the following.

- **Step I:** Using $V(t^k)$ as the initial condition to integrate equation

$$\begin{aligned} C_m \frac{\partial V}{\partial t} &= -J_{ion}(V, \mathbf{u}) - J_{stm}(t), \\ \frac{\partial \mathbf{u}}{\partial t} &= \mathbf{f}(\mathbf{u}, V, t), \end{aligned} \qquad \text{for } t \in [t^k, t^k + \Delta t] \tag{4}$$

- **Step II:** Use the result obtained in Step I as the initial conditions to integrate

$$C_m \frac{\partial V}{\partial t} = \nabla \cdot (\mathbf{D} \nabla V), \qquad \text{for } t \in [t^k, t^k + \Delta t] \tag{5}$$

2.2. Spatial-temporal discretisation

When performing Step II, the computational domain must be discretised in space by a mesh of either finite elements or finite differences to approximate the dependent variables of the problem, V and $\mathbf{u}$, which allows writing Eq. (5) as

$$\mathbf{M}\dot{\mathbf{V}} + \mathbf{K}\mathbf{V} = \mathbf{0}, \tag{6}$$

minimise where $\mathbf{M}$ and $\mathbf{K}$ are the mass and stiffness matrices respectively, obtained by assembling individual element matrices over the entire computational domain.

The most well known algorithms for integrating in time the semi-discrete system (6) are members of the generalised trapezoidal family of methods [29]. Let $\mathbf{V}^k$ and $\dot{\mathbf{V}}^k$ denote vectors of the transmembrane potential and its time derivative at each nodal point of the mesh at time t^k, where k is index of the time step, then at time t^{k+1} we can write

$$\mathbf{M}\dot{\mathbf{V}}^{k+1} + \mathbf{K}\mathbf{V}^{k+1} = \mathbf{0}, \tag{7}$$

$$\mathbf{V}^{k+1} = \mathbf{V}^k + \Delta t \dot{\mathbf{V}}^{k+\theta}, \tag{8}$$

$$\dot{\mathbf{V}}^{k+\theta} = (1-\theta)\dot{\mathbf{V}}^k + \theta \dot{\mathbf{V}}^{k+1}, \tag{9}$$

where $\theta \in [0, 1]$ is a scalar parameter. Equations (7) through (9) can be combined to obtained an algebraic system of equations to determine $\mathbf{V}^{k+1}$.

When using the operator splitting algorithm for solving the monodomain model, equations (1) and (2) are solved in two steps. First, the electrophysiological cellular model

$$\mathbf{V}^* = \mathbf{V}^k - \Delta t \, \left(J_{ion}(\mathbf{V}^k, \mathbf{u}) + J_{stm}(t) \right), \tag{10}$$

is solved at each mesh point to obtain an intermediate transmembrane potential vector $\mathbf{V}^*$ (Step I). Even though a forward Euler scheme has been used in (10), any other ODE solver can be used to calculate $\mathbf{V}^*$. With this intermediate solution at hand, along with equations (8) and (9) and $\mathbf{M}\dot{\mathbf{V}}^k = -\mathbf{K}\mathbf{V}^k$ from the previous converged time increment, equation (7) becomes

$$\mathbf{M}\frac{\mathbf{V}^{k+1} - \mathbf{V}^*}{\Delta t} = -\mathbf{K}\left(\theta\,\mathbf{V}^{k+1} + (1-\theta)\,\mathbf{V}^k\right), \tag{11}$$

or alternatively

$$\hat{\mathbf{K}}\mathbf{V}^{k+1} = \hat{\mathbf{b}}, \tag{12}$$

where $\hat{\mathbf{K}}$ is everything that multiplies onto $\mathbf{V}^{k+1}$, and $\hat{\mathbf{b}}$ contains the other terms in equation (11). Equation (12) is solved for the entire domain to obtained $\mathbf{V}^{k+1}$ (Step II).

Hence, the basic algorithm at time t^{k+1} can be summarised, as:

- **Step I:** Use $\mathbf{V}^k$ as the initial condition to integrate equation (10) to obtain $\mathbf{V}^*$
- **Step II:** Use the result obtained in Step I to solve (12) for $\mathbf{V}^{k+1}$

For different values of the parameter θ, different time integration schemes are obtained for integrating the discretisdiscretiseed homogeneous parabolic equation system (11):

$\theta = 0$	Forward Euler (conditionally stable).
$\theta = 0.5$	Crank-Nicolson scheme (unconditionally stable).
$\theta = \dfrac{2}{3}$	Galerkin Scheme (unconditionally stable).
$\theta = 1$	Backward Euler (unconditionally stable).

Crank-Nicolson scheme is second order accurate in time, whereas the other are first order accurate in time. However, for $\theta \geqslant 0.5$ integration schemes are unconditionally stable.

As mentioned before, Step I can be performed using a backward difference approximation in time (implicit integration), or a forward difference approximation in time (explicit integration). Implicit integration requires the solution of a nonlinear system of equations at each point of the mesh making it computationally costly. However, it ensures the stability of the numerical solution. On the contrary, explicit integration is computationally cheaper but imposes more stringent conditions on the size of the time step in order to avoid numerical instabilities.

2.3. Integration of the mass matrix

For the standard finite element formulation, the elemental mass matrix, $\mathbf{M}^e$ for equation (11), is given by [29]

$$M_{ij}^e = \int_{\Omega_e} N_i N_j d\mathbf{x},$$

where N_j is the shape function of node j of the element e, and $C_m = 1$ has been assumed, without loss of generality,

When the shape functions $\mathbf{N}$ used to compute $\mathbf{M}^e$ are the same used to approximate the potential V, the resulting non-diagonal matrix is known as the *consistent* mass matrix. Using a consistent mass matrix implies that a linear system of equations has to be solved for V^{k+1} when an explicit scheme is used in (11). In order to improve numerical efficiency, the proposed algorithm evaluates $\mathbf{M}^e$ using a mass preserving nodal quadrature [30]. Nodal quadrature is based on the use of different base functions to those used to approximate the transmembrane potential, V. This approximation with different shape functions is admissible since it satisfies the finite element criteria of integrability and completeness [30]. In the implementation, we have considered a nodal quadrature to evaluate $\mathbf{M}^e$ with $\mathbf{N}_i = J_i\mathbf{I}$, being J_i the element Jacobian evaluated at node i.

3. Parallel implementation in GPU

The CUDA implementation consists of two subprograms: i) a CPU part or *host subprogram*, and ii) a GPU part or *device subprogram* as shown in Fig. 1. The *host subprogram* prepares all structures require for the GPU execution, and then moving data from CPU main memory to the GPU memory. In addition, the *host subprogram* controls the execution and launches the device subprogram. The *device subprogram* is organised in kernels, with each

kernel executed in parallel by each GPU thread. In the discretised scheme, there are two main contributors to the computational cost: solving the system of ODEs (reaction term) at each mesh point (Step I), and solving the linear system of equations associated with the parabolic PDE (Step II). In order to maximise performance, all vectors and matrices associated with the system of equations and the ionic model reside in GPU memory. Only the transmembrane voltage vector, $\mathbf{V}$, is transferred back to the CPU memory when the data has to be saved on disk in order to minimise the communication time. In addition, in order to minimise memory storage, all data is stored using sparse matrix structures. The entire implementation of the solver i.e., ODE and PDE solvers, is performed in double precision, since previous studies indicate an important loss of accuracy [22] when using single precision to integrate the ionic model in Step I.

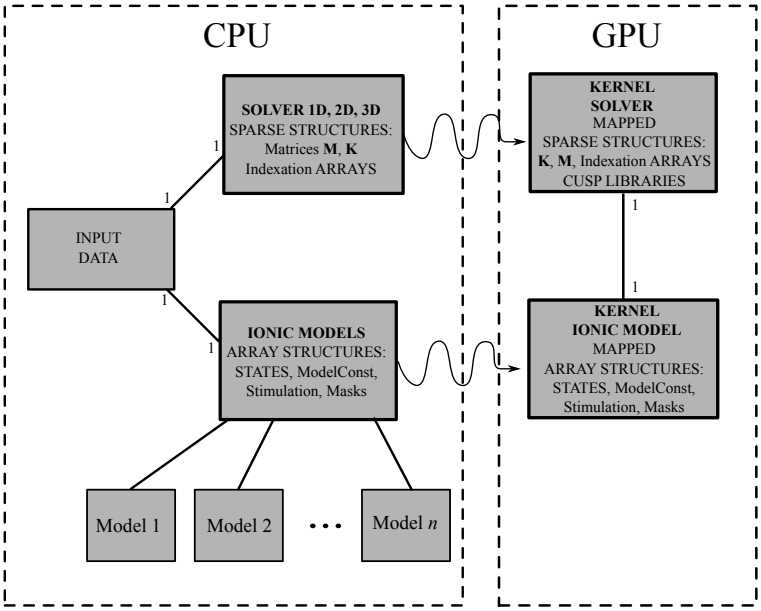


Figure 1: CUDA program model.

3.1. ODE solver.

The C/C++ code describing the ionic model for GPU implements the Rush-Larsen integration scheme for the gating variables [11] in order to improve accuracy. The exact same code was executed in CPU and GPU in order to validate the implementation.

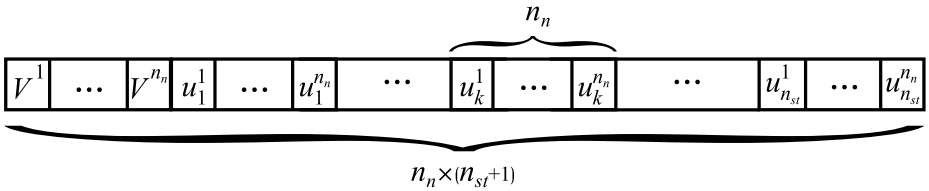


Figure 2: Structure of the vector of state variables for the ionic model, $\mathbf{u}$, and the transmembrane potential, V.

When executed in GPU, the transmembrane potential and the state variables (SV) for all nodes in the model are stored in a unique vector in the global GPU memory. Fig. 2 shows an example for the case of a discretised model with n_n nodes and an ionic model with n_{st} state variables. This data structure allows all GPU threads within a thread block to have access to continuous memory blocks during execution, guaranteeing data coalescence in memory and maximizing performance [17]. On the contrary, constants associated to the ionic model required to evaluate (4) are kept in the shared GPU memory.

In more complex cardiac models having a large number of SV, use of a single kernel to integrate the ionic model may not be possible because the number of registers available per thread is insufficient (this is the case in small cards as the Nvidia GTX295). In this case, the solution of the ionic model will require a sequence of multiple kernel invocations, with each kernel devoted to the solution of a group of SV. Because a kernel invocation may modify the input of the following kernel, it is necessary to resolve these dependencies by duplicating the variables that are common input among the kernels. Since every kernel invocation introduces an overhead in computing time, but also in memory requirements, an optimise implementation is required to find the best trade-off between kernel splitting, memory use, and kernel invocation overhead.

The influence of thread divergence was also considered in our implementation. On the GPU threads within a block run concurrently only if they execute the same instruction. In this regard, synchronisation between threads may be reduced in a conditional statement since the threads may follow different branches causing a reduction in parallelism. In the implemented models, conditional statements were substituted by Heaviside functions allowing mimicking the conditional as a product between a literal and the condition of the Heaviside function (implemented as a multiplication with a predicate).

3.2. Parabolic solver

The linear system given in (12) was solved on GPUs using the CUSP and Thrust libraries developed by Nvidia [31]. CUSP is implemented for a single GPU and natively supports a number of sparse matrix formats providing specific subroutines for an easy passage between different sparse matrix formats. The library includes highly optimised matrix-vector multiplication algorithms and iterative solvers. In addition, a variety of preconditioners based on algebraic multigrid (AMG) and approximate inverse operators are ready available in the library. In our implementation, mass and stiffness matrices, $\mathbf{M}$ and $\mathbf{K}$ respectively, are assembled in parallel in the GPU and stored in compressed sparse row (CSR) format at host memory. These matrices are then transformed to an efficient sparse matrix format when transferred in the GPU memory for computations.

The performance of the linear solver in CUDA is highly conditioned by the efficiency of the sparse matrix-vector multiplication, which depends on the sparse matrix format used to store the matrix during computations in the GPU memory. The CUSP library offers five different sparse matrix formats that cane used when solving a liner system of equations. Namely: i) Coordinate format (COO), a simple storage scheme for which the required storage is proportional to the number of non zeros in the matrix; ii) CSR format, a natural extension of the COO format with a simple compression scheme applied to the repeated row indices found in the COO format; iii) Diagonal format (DIA), is a not general purpose format, but represents an efficient storage scheme for matrices arising from structured meshes; iv) ELL format represents a storage scheme well suited for vector architectures. More general than DIA, ELL efficiently encodes semi-structured and unstructured meshes when the maximum number of nonzeros per row does not greatly differ from the average; v) Hybrid (HYB) format, is a hybrid ELL/COO format which stores the majority of matrix entries in ELL and the remaining entries in COO. This allow to explore the well-suited structure of ELL for vector architecture with the storage efficiency of COO. Our implementation has consider different sparse matrix format at GPU level in order to evaluate their performance on structured and non-structured meshes.

4. Benchmarking

Separate benchmarking tests were performed to evaluate the ODE solver and the parabolic solver. The speed-up and accuracy of the ODE solver was

evaluated on three models of action potential: i) A 4 SV model proposed by Bueno-Orovio et al. [32] (BCF); ii) the model for ventricular cardiomyocytes proposed by ten Tusscher and Panfilov (TP06) [8] comprising 19 SV; and iii) the model for atrial cardiomyocytes proposed by Maleckar et al. [33] (MLK) with 29 SV. The accuracy of the ODE solver was addressed by comparing the solution obtained using the GPU with the solution obtained with a single CPU. In this regard, the ionic model was stimulated every second for 2 ms during 100 s with as stimulation current, J_{stm}, of 24 μA/μF, recording the solution for the last second of simulation with a resolution of 0.2 ms. The parallel performance was evaluated as the speedup, S, i.e., the execution time of the GPU implementation with respect to a single CPU core as the number of nodes, or degrees of freedom, increases (see Fig. 3a).

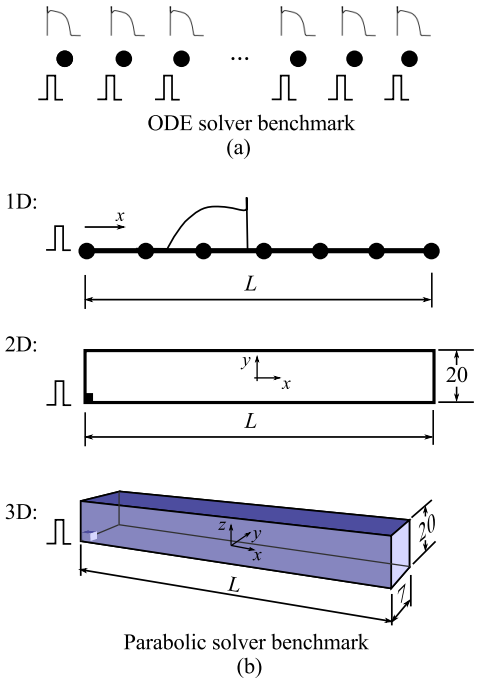


Figure 3: Model problems used to test the performance of the GPU algorithm. a) ODE solver, b) Parabolic solver.

The performance of the parabolic solver was tested on 1-, 2- and 3-dimensional problems. The model geometry was defined as a cable of length L, a rectangle with dimensions $L \times 20$ mm^2, and a cuboid with dimensions of $L \times 20 \times 7$ mm^3 as shown in Figure 3b. The dimension L was varied in order

to increase the number of nodes in the problem. The different geometries where meshed with structured (quadrilateral and hexahedral elements) and non-structured (triangular and tetrahedral elements) meshes with a characteristic element size of 0.1 mm. In addition, since ventricular cardiac muscle is anisotropic, the tissue was modelled as transversally isotropic with the fibre direction oriented along the x-axis. A propagating wave front was initiated at one of the corners of the model using a stimulation current pulse of 50 μA/(μFcm^2) strength and 2 ms duration. The electrical activity was simulated over 1000 ms, with a fixed time-step of 20 μs. The parallel performance was evaluated calculating the speedup, S.

The efficiency of the GPU is further demonstrated on the simulation of a heart beat in realistic models of a biventricular human heart and a human atria. The biventricular human heart was discretised with 1289000 hexahedral elements and 1434129 nodes (grid points). The TP06 model has been used to simulated the action potential model of the ventricular tissue. The human atria was used to demonstrate the efficiency of the code with non-structured meshes. The human atria was discretised with 1378054 elements and 266450 nodes. The MLK model of atrial action potential was used for simulating the electric activity. In both models, cardiac muscle anisotropy has been accounted for.

The speed-up of the GPU implementation is compared against a fortran implementation of the algorithm described in Section 2 using the Message Passing Interface (MPI) for parallel computations [34, 35]. Domain decomposition for the parallel solution has been carried out using the METIS library [36], where as the linear system of equations (12) has been solved using the Conjugate Gradient (CG) method with an ILU preconditioner from the PSBLAS library [37].

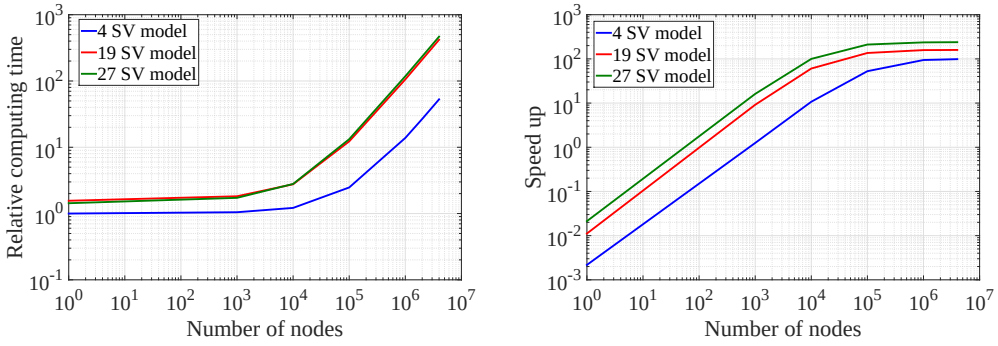
GPU simulations were run on a computer node with two Intel-Xeon Quad-Core CPUs E5620 clocked at 2.4GHz and 48GB DDR3 RAM. The node is equipped with four Nvidia Tesla E2090 GPUs, each with 6GB DDR5 RAM for a total of 24GB DDR5 RAM. All simulations were run in a single GPU. The CPU benchmark was run on a cluster with 8 nodes with two Intel-Xeon Quad-Core E5520 clocked at 2.26GHz and 24GB DDR3 RAM connected by a high speed infiniband network.

5. Results

The coefficient of determination, R^2 between the solution obtained with the GPU and CPU was calculated for each of the state variables of all three ionic models considered. The results show a $R^2 > 0.9$ for all state variables, indicating an excellent agreement between both implementations, in addition to demonstrating the capabilities of double precision arithmetic of the E2090 GPU. Table 1 shows the results for the TP06 model. Similar results were found for the BCF and MLK models.

State Variable	R^2	State Variable	R^2	State Variable	R^2
V	0.9974	$[Ca^{2+}]_i$	0.9993	$[Ca^{2+}]_{SR}$	0.9982
$[Ca^{2+}]_{SS}$	0.9388	$[Na^{+}]_i$	0.9691	$[K^{+}]_i$	0.9899
m	0.9981	h	0.9967	j	0.9961
x_s	1.0000	r	0.9971	s	0.9997
d	0.9976	f	1.0000	f_2	0.9998
f_{cass}	0.9973	R	0.9993	x_{r1}	0.9997
x_{r2}	0.9995				

Table 1: Coefficient of determination between the state variables corresponding to the CUDA implementation and the C++ implementation. The coefficient of determination has been calculated with the entire time course of each state variable during the last beat of the stimulation protocol.



(a) (b)

Figure 4: Performance of the ODE solver: a) Relative computing time for the GPU-ODE solver as the number of nodes increases for models with different number of state variables (times have been normalised with respect to the 4 SV model; b) Speed-up against a single CPU core for ionic models with different number of state variables.

The performance of the ODE solver for the three ionic models is shown in Fig. 4. The enormous parallel capabilities of the GPU is demonstrated in Fig. 4a where the relative computing time required by the GPU as the number of nodes in the model increases is depicted. The figure shows that the computing overhead increases with the number of SV in the model. Furthermore, the figure shows that the overhead increases monotonically with the number of nodes in the model until reaching a steady state value depending on the complexity of the ionic model. This figure also indicates the existing a threshold (depending on the number of SV) for which the GPU is able to integrate the ionic model simultaneously without experiencing a significant degradation in computing time. After this threshold is surpassed, the computing time increases almost linearly as the number of nodes in the problem increases. Figure 4b shows the speed-up, S, obtained with the GPU as the number of nodes in the problem increases. The figure shows that a single CPU is faster than a single GPU thread. However, the enormous computational horsepower of the GPU leads the GPU to overtake a single CPU as the number of nodes increases until attaining an asymptotic speedup, which may be up to 200$\times$ for the MLK model on problems with more than a one million nodes. Note, that the maximum speedup reduces as the number of SV in the ionic model decreases.

Another important aspect to account for in the implementation of the ODE solver is the effect of splitting the kernel in multiple kernel invocations do to the limitation in the number of registers per thread. Even though, the Tesla M2090 allows the implementation of all three studied models on a single thread, a multiple kernel implementation of the TP06 model was performed in order to allow execution (in double precision) on a Nvidia GTX295 with 895Mb of memory per device. A careful optimisation of the code using the CUDA SDK profiler yield an splitting of the ionic model into 22 kernels and duplicating 15 SV. Table 2 shows the computation overhead introduced by the multi kernel implementation as the number of nodes in the problem increases. The computational overhead on a single CPU is also depicted for completeness. As expected, the computing overhead remains constant for the CPU implementation. However, for the GPU, the computation overhead reduces asymptotically to cero as the number of nodes in the model increases.

Figure 5 shows the performance of the parabolic solver for a fully-explicit algorithm (Fig. 5a) and a semi-implicit algorithm (Fig. 5b) with structured meshes with the TP06 model. A CG iterative algorithm with ILU preconditioning and a hybrid sparse matrix format have been used for the computa-

N	Computation overhead (%)	
	GPU	CPU
10^0	464	38
10^3	410	30
10^4	249	30
10^5	57	30
10^6	5	33
2×10^6	2	30

Table 2: Computation overhead of the single kernel versus the multiple kernel implementation for the TP06 ionic model.

tions. No significant differences were observed on the benchmarks conducted with non-structured meshes (results not shown).

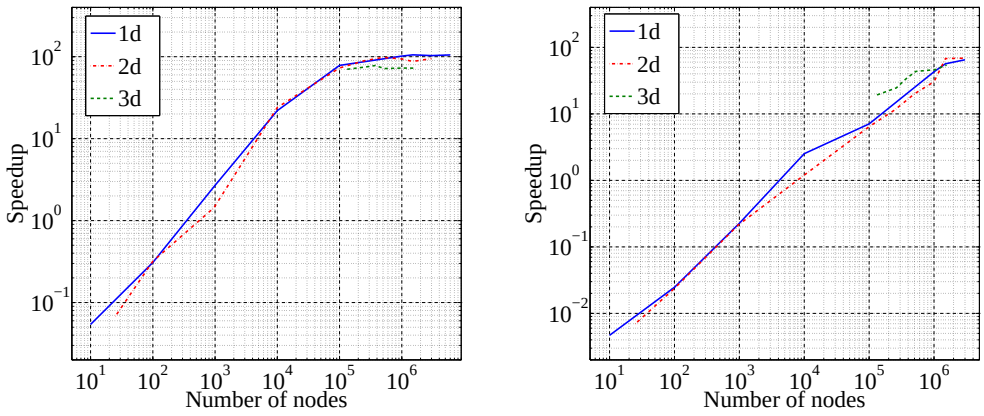
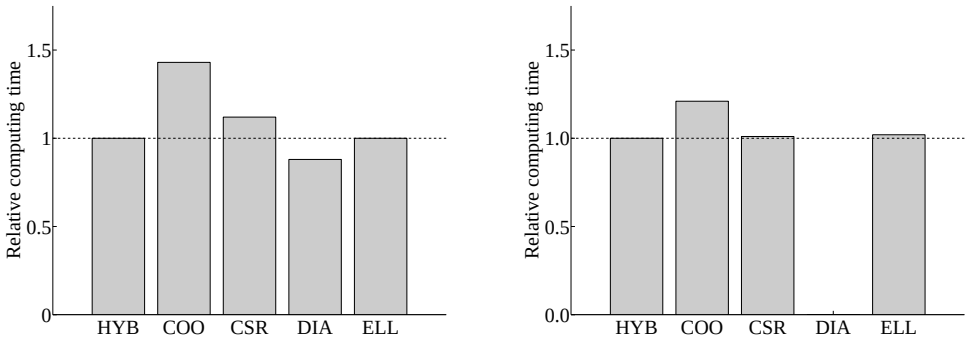


Figure 5: Performance of the parabolic solver: a) Explicit algorithm ($\theta = 0$); b) Semi-implicit algorithm ($\theta = 1$).

The speed-up obtained with the explicit scheme (see Figure 5a) follows the same trend obtained for the ODE solver shown in Fig. 4a. In fact, the results indicate that integrating the ionic model occupies more than 90% of the computing time. As observed with the ODE solver, for small problems the GPU underperforms the single CPU execution due to the lower clock-speed of the single GPU with respect to the single CPU, and the faster double precision arithmetic of the CPU. However, as the number of degrees

of freedom increases the GPU outperforms the CPU (a threshold was found in 800 nodes for the benchmark problems with the TP06 model) until reaching an asymptotic behaviour with a speedup of: $\sim 100\times$ for 1D-problems, $\sim 90\times$ for the 2D-problems, and $\sim 70\times$ for the 3D-problems. The reduction in acceleration is mainly due to the additional cost involved in the matrix multiplication that takes place during Step II. The cost of solving the linear system is more evident with the semi-implicit scheme (Fig. 5b). In this case, the speed-up curve does not reaches an asymptotic value as for the explicit algorithm, and the minimum problem size required for the GPU to outperform the CPU was found in 8000 nodes. The maximum speed-up for the semi-implicit scheme was found to be $\sim 65\times$ for 1D-problems, $\sim 60\times$ for 2D-problems, and $\sim 50\times$ for 3D-problems. It is important to point out that, the acceleration observed in Fig. 5 depends on the size and complexity of the ionic model used.



(a) Structure meshes

(b) Unstructure meshes

Figure 6: Average relative computing time of the implicit parabolic solver for different sparse matrix formats on structured meshes (a), and unstructured meshes (b). Note that the DIA sparse format is not suitable for unstructured meshes.

As mentioned in Section 3, the performance of the linear solver in CUDA is highly conditioned by the efficiency of the sparse matrix-vector multiplication, which depends on the sparse matrix format used to store the matrix $\hat{\mathbf{K}}$ in (12). Figure 6 shows the relative computing time with respect to the HYB format for the 3D benchmark problem for both structured and unstructured meshes. For the structured meshes DIA format gave the best performance with a reduction of a 10% in computing time with respect to the HYB and ELL formats. On the contrary, COO format gave the worst performance.

For unstructured meshes HYB gives the best performance although no significant differences were found for the CSR and ELL formats, whereas the DIA format is not suitable for this type of dicretisation. These results are in agreement with the benchmark results reported by Bell and Garland [31] for the sparse matrix-vector multiplication.

As part of the benchmark, the performance of the implicit-parabolic solver with different iterative schemes for solving Step II has also been evaluated. Namely: i) Conjugate Gradient (CG), Bi-Conjugate Gradient (BCG), Stabilised Bi-Conjugate Gradient (BCGSTAB), and iv) Generalised Minimum Residual (GEMRES). Figure 7 shows the relative computing time with respect the CG scheme for the 3D benchmark problems with structured meshes. The HYB sparse matrix and ILU preconditioning has been used for the computations.

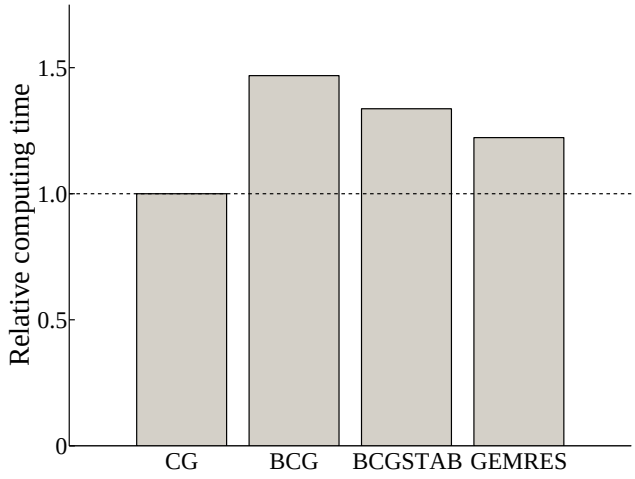
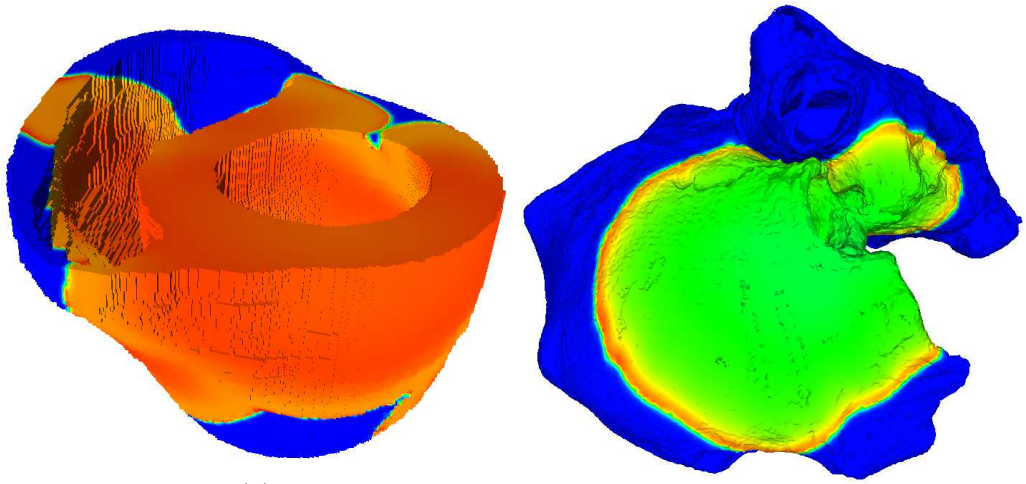


Figure 7: Average relative computing time of the implicit parabolic solver for different iterative schemes. HYB sparse solver and ILU preconditioning have been used for the computations.

Figure 7 shows that CG gives the best performance for the problem under consideration, whereas the BCG scheme is the most expensive in terms of computing time. The same trend was observed when using different preconditioning offered by the CUSP library, and for different sparse matrix formats. In general it was observed that CG required more iterations to converge to the same tolerance per time increment than the other schemes.

However, the lower cost per iteration in terms of matrix-vector multiplications involved in the CG scheme lead to a lower computation time. These findings are in agreement with results reported in [38, 39] for the solution of the electrophysiolgy problem using the mono domain model.



(a) (b)

Figure 8: Depolarization front after 20ms of activation. a) Human bi-ventricular model discretised with hexahedral elements; b) Human atria discretised with tetrahedral elements.

Figure 8 shows the depolarisation front in the voxelised human bi-ventricular model (Fig. 8a), and a human atria discretised with tetrahedral elements (Fig. 8b) . The speed-up, S, obtained on this realistic models with the semi-implicit scheme was $50\times$ for the human bi-ventricular heart and $40\times$ for the human atria with respect to a single CPU core, where as for the explicit scheme, the speed-up, S was $\sim 50\times$ for both problems. These results were in excellent agreement with those obtained in the benchmark (see Fig. 5) for both integration schemes. An additional test was conducted by running the human bi-ventricular heart on 42 cores in our computer cluster. In this case, the GPU still performed better by a factor of $1.8\times$, which implies a slightly better performance as compared to a single processor ($\sim 76\times$), due to the extra cost added by the communication between the different processes during the parallel execution.

In order to gain a better insight of the performance of the GPU solver with non-diagonally block matrices (matrices with large bandwidth), we have

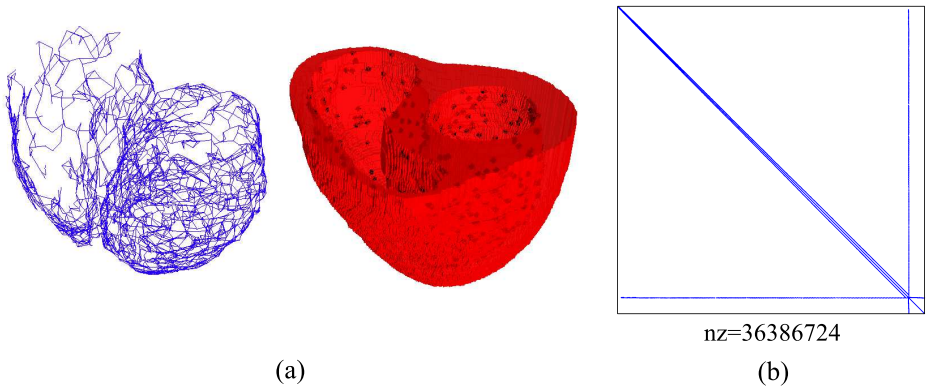


Figure 9: A coupled problem with large bandwidth. a) Detailed of the one-dimensional network representing the fast conduction system of the heart (left panel) coupled to the human bi-ventricular model (right panel). Coupling between the two meshes takes place at discrete points only (black markers in the right panel); b) Sparsity pattern of the resulting linear system.

solved a voxelised human bi-ventricular heart coupled with a network of one-dimensional linear elements representing the fast conduction system in the heart (see Fig. 9a). The resulting model involves 1.5 million degrees of freedom, and since the coupling between both meshes occurs only at a discrete number of points, the innervation points (see Fig. 9a, right panel), the resulting $\hat{\mathbf{K}}$ matrix in (12) has an sparsity pattern as shown in Fig. 9b with a bandwidth of 1418491 and a maximum number of zeros per row of 35. Solving 1600 ms of electric activity with a fixed time step of 0.02 ms took the GPU-solver 9000 s using the semi-implicit scheme and the HYB sparse format for $\hat{\mathbf{K}}$, implying an acceleration of 50$\times$ with respect a single CPU-core. This result is in agreement with the benchmark results demonstrating the efficiency of the GPU solver to handle ill sparsity patterns as those shown in Fig. 9b. In fact, when the sparsity pattern was optimised using a CutHill-MacKee algorithm which reduced the bandwidth of the matrix $\hat{\mathbf{K}}$ 80 fold, the performance was not affected in neither the GPU-solver nor the CPU-solver.

6. Discussion and conclusions

The potential of GPU implementation for solving reaction-diffusion systems describing cardiac electrophysiology has been investigated. Scalability tests were performed for fully explicit and semi-implicit numeric schemes in 1D-, 2D-, and 3D-model problems with structured and unstructured meshes

using novel software entirely developed in C++/CUDA. The performance of the method was ulteriorly demonstrated on realistic models of a biventricular human heart and human atria.

Previous studies [20] have reported speedups of 32× for the monodomain model using an explicit finite difference scheme along with the phase I Luo-Rudy ionic model (LR1) [40], a much less complex model (8 SV) than the ionic models used in this study (19 SV of the TP06 or 28 SV of the MLK action potential model of atria). In their study Sato et al [20] have established the solution of the PDE equation as the bottle neck of the computation with GPU, in part because they were forced to used single precision for their computations requiring a more stringent step size when solving the Step II than the required for integrating the action potential model in Step I. In addition, Sato et al. were using an adaptive time stepping when integrating the system of ODEs, which introduces latencies in the computations. In our implementation, we integrate with a constant time step size that guarantees both, the stability of the ionic model (most restrictive step size) and the stability of the PDE, similarly as has been proposed in order studies [26]. With this strategy, along with an efficient data structure for the ionic model and a sparse format for the matrices, we have been able to make full use of the GPU parallel potential by reducing latency during GPU execution leading to acceleration of up to 90× on 2D-problems of similar size as those considered in [20] with an explicit scheme. We must also stress that, Sato et al. used older NVidia GT 8800 and GT 9800 GX2 cards that are faster for single precision floating point operations than the Tesla GPUs used in our study, but do not support double precision calculations.

For the ODE solver, Lionetti et al. [24] performed 20 ms of a heart beat simulation on a domain that consisted of 42,240 grid points with no spatial integration (all the cells were decoupled) since their main interest was to optimise the ODE solver. They also used different optimization techniques for the different cell models considered. For the 18 SV Puglisi-Bers model [25], comparable to the TP06 model, they report a total simulation time of 5.98 s using a time step of 0.0008 ms and single precision on a NVIDIA GTX295. Our simulations with the TP06 model took 15 s on a the C2090 using double precision arithmetic. Taking into account the overhead due to double precision arithmetic, the performance is comparable. In a more recent work Neic et al. [26] report an execution time of 360 s to simulate 250 ms with the 27 SV proposed by Mahajan et al. [41] using a time step of 0.025 ms on a domain with approximately 800000 grid points. The same simulation with

the MLK model took in our case 178 s representing an effective speedup of 2. However, differences in the mathematical complexity of the two models could account for the difference in spite of having the same number of state variables. Vigueras et al. [27] has reported similar speedup values for the ODE solver with the TP06 model, whereas they report slight lower speedups for the three dimensional problem benchmark with 1.2 million grid points (30× against a 70×). The reason for this reduction in performance with respect to our implementation may be related to the implementation of the PDE solver. While we perform the solution of the linear system entirely in the GPU memory with the CUSP libraries, they perform make use of a hybrid scheme in which at each simulation step, the system of PDEs is solved by copying the required data from CPU memory to GPU memory, and then the result copied back to CPU memory. This operation introduces a considerable overhead that may reduce the overall performance of the solver.

In other recent work Neic et al.[26] have used multiple-GPU enhanced simulation code to accelerate cardiac bidomain simulations with the Mahajan 28 SV rabbit model. In their study Neic et. al. have reported an overall speed-up of 16× for the bi-domain problem, but only moderate speed-up between 8.1×-9.5× for the parabolic PDE equivalent to the monodomain solver. Even though our simulations with the Maleckar model (28 SV) on a 3D problem with 1.5 million grid points showed gave a speedup of the order of 50, a direct comparison is difficult, not only for the different problem size, but also because their implementation considers a computationally more expensive bidomain approximation. In addition, their multi-GPU implementation, communications between GPUs were performed through the host memory and not using a peer-to-peer communication between the GPU cards, which substantially increases communication time, and therefore a penalising overhead. Rocha et al. [23] report a simulation time of 1804 s with the TP06 model for 500 ms on a 2D grid size of 641×641 using single precision arithmetics and an implicit scheme for the PDE. Our implementation takes 867s in double precision, representing an speedup of 2. However, our computations were carried on an Tesla C2090 which is considerably faster than the GTX280 used in their study. The same problem was also solved by Bartocci et al. [22] but using an explicit algorithm and finite differences for the spatial discretisation. Bartocci et al. report a total computation time of 105.4 s on a Tesla C2070 using single precision arithmetic, while our explicit implementation takes 220 s on the same graphic card with double precision arithmetic. Considering that they report an overhead of 1.8 when

using double precision arithmetic with respect to their single precision results for the TP06 model, their computation time for this problem would be 198 s which is comparable to our results. It should also be accounted the fact that Bartocci et al., use a finite difference discretisation for the Laplacian term which implies a $\hat{\mathbf{K}}$ matrix with lower bandwidth than the one obtained with the finite element discretisation (bandwidth of 5 for finite differences against a bandwidth of 9 for finite elements), which implicates a reduction in the computing cost.

Results from Fig. 4 and Fig. 5 indicate that a minimum problem size is required in order to optimise GPU performance. This is due to the slower clock speed and limited double precision floating point arithmetic of the GPU cores as compare to CPU cores. This lower core performance is, however, compensated by the capability of the GPU to schedule a vast numbers of threads and efficiently reducing latency in this many core architecture. In this regard, for the E2090 our results indicate that for the ODE solver, the GPU is able to accommodate up to 8000 nodes without degradation of computer performance when working with a complex model like TP06. For the monodomain solver, working with the TP06 ionic model, this threshold was found to be at 800 nodes for the fully explicit scheme and at 8000 nodes for the semi-implicit scheme, independently of problem dimensionality and mesh structure (structured or unstructured mesh). In this regard, we must add that we have used highly optimised unstructured meshes which maximise the bandwidth of the resulting algebraic system of equations. After this threshold is surpassed the scalability of the GPU with the number of degrees of freedom approaches to linearity. Even though these results are dependent on the GPU card used, this threshold appeared to be independent of the problem dimensionality for the cases considered. We must remark that the bandwidth of the linear system did not change much between the structured and unstructured meshes. This explains the similarity between the results obtained for both benchmarks, as well as the speedup found for the ventricle and atria simulations. However, ulterior tests performed on non-diagonally dominant matrices show no-degradation on the computing time with respect to the benchmark problems.

Our study also shows the importance of an efficient sparse matrix format and adequate iterative solver for the computations. We found that using a DIA sparse format is optimal when working with purely structured meshes. However, this format is not suitable for unstructured or semi-structured meshes for which the ELL and HYB formats were found to be the best

in agreement with the study conducted by NVIDIA [31]. In this regard, our study points the HYB sparse matrix format as the best alternative in terms of storage flexibility and numerical performance. It is important to point out that complex models of cardiac electrophysiology include the presence of the specialised conduction system, i.e., Purkinje system, (see Fig. 9a) leading to models with mixed element meshes i.e., 3D elements connected to 1D elements, which result in an ill sparsity pattern as shown in Fig. 9b. Regarding the iterative solvers for solving the parabolic system, our benchmark confirms (see Fig. 7) the results from Potse et al. [39] which found that for the monodomain model, an ILU preconditioned in conjunction with a Conjugate Gradient iterative solver were optimum in terms of computing time.

This work confirms other studies that, a significant reduction on computing time can be achieved for solving the cardiac monodomain equations using GPUs. Despite the significant lower performance observed on a single GPU core with respect to a single CPU core, a single GPU card offered excellent performance with speedups of 70× for three-dimensional problems solved explicitly and near 50× for three-dimensional problems solved with a semi-implicit scheme when compared with a single CPU. However, we must mention that when an adaptive time step is used in the single CPU core (an alternative that is not valid for GPU due to the introduced latencies and dynamic load loss), the overall acceleration of the GPU over the CPU reduces to 42× for 1D problems, and 32× for 2D and 3D problems, implying a reduction of about a 40% with respect the case of constant time step. However, despite this reduction, the performance of the GPU is still outstanding with respect to the CPU. As a final remark, we have seen that a personal workstation is able to perform a simulation of the electric activity of a whole heart in reasonable times and at a reasonable cost due to the significant lower prize of a GPU card as compared to a computer cluster and its outstanding performance. This aspect, enables researchers to interact more easily with their simulations helping them to improve their understanding of the physiology and the synergy between the different scales involved in electrocardiographic simulations. However, working with GPUs poses additional programming challenges over traditional parallel CPU implementations, in particular in aspects related to efficient management of threads and memory, as well as data structure in order to obtains the maximum performance of the GPU.

Acknowledgments

This project was partially supported by the "VI Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica" from the Ministerio de Economía y Competitividad of Spain (grant numbers TIN2012-37546-C03-01 and TIN2012-37546-C03-03) and the European Commission (European Regional Development Funds - ERDF - FEDER). We also thanks Dr Gunnar Seemann at the Karlsruhe Institute of Technology for providing the tetrahedral model of the human atria.

References

[1] P. J. Hunter, A. J. Pullan, B. H. Smaill, Modeling total heart function, Annu. Rev. Biomed. Eng. 5 (1) (2003) 147–177.

[2] A. S. Go, D. Mozaffarian, V. L. Roger, ... , M. B. Turner, Heart disease and stroke statistics 2014 update: A report from the american heart association, Circulation 129 (3) (2014) e28–e292.

[3] E. Bartocci, R. Singh, F. B. von Stein, ... , F. H. Fenton, Teaching cardiac electrophysiology modeling to undergraduate students: Laboratory exercises and GPU programming for the study of arrhythmias and spiral wave dynamics. Adv. Physiol. Educ. 35 (4) (2011) 427–437.

[4] D. B. Geselowitz, W. T. Miller III, A bidomain model for anisotropic cardiac muscle, Ann. Biomed. Eng. 11 (3-4) (1983) 191–206.

[5] C. Henriquez, A brief history of tissue models for cardiac electrophysiology, IEEE T. Bio-Med. Eng. 61 (5) (2014) 1457–1465.

[6] P. Colli-Franzone, L. Pavarino, A parallel solver for reactiondiffusion systems in computational electrocardiology, Math. Models Methods Appl. Sci. 14 (06) (2004) 883–911.

[7] F. H. Fenton, E. M. Cherry, A. Karma, W.-J. Rappel, Modeling wave propagation in realistic heart geometries using the phase-field method, Chaos 15 (1) (2005) 013502.

[8] K. H. W. J. Ten Tusscher, A. V. Panfilov, Alternans and spiral breakup in a human ventricular tissue model, Am. J. Physiol-Heart C. 291 (3) (2006) 1088–1100.

[9] E. Heidenreich, J. M. Ferrero, M. Doblare, J. F. Rodriguez, Adaptive macro finite elements for the numerical solution of monodomain equations in cardiac electrophysiology, Ann. Biomed. Eng. 38 (7) (2010) 2331–2345.

[10] E. M. Cherry, H. S. Greenside, C. S. Henriquez, Efficient simulation of three-dimensional anisotropic cardiac tissue using an adaptive mesh refinement method, Chaos 13 (3) (2003) 853–865.

[11] P. Colli-Franzone, P. Deuflhard, B. Erdmann, J. Lang, L. F. Pavarino, Adaptivity in space and time for reaction-diffusion systems in electrocardiology, SIAM J. Sci. Comput. 28 (3) (2006) 942–962.

[12] J. P. Whiteley, Physiology driven adaptivity for the numerical solution of the bidomain equations, Ann. Biomed. Eng. 35 (9) (2007) 1510–1520.

[13] A. Cristoforetti, M. Mase, F. Ravelli, A fully adaptive multiresolution algorithm for atrial arrhythmia simulation on anatomically realistic unstructured meshes, IEEE T. Bio-Med. Eng. 60 (9) (2013) 2585–2593.

[14] J. Whiteley, An efficient technique for the numerical solution of the bidomain equations, Ann. Biomed. Eng. 36 (8) (2008) 1398–1408.

[15] M. O. Bernabeu, R. Bordas, P. Pathmanathan, J. Pitt-Francis, J. Cooper, A. Garny, D. J. Gavaghan, B. Rodriguez, J. A. Southern, J. P. Whiteley, Chaste: incorporating a novel multi-scale spatial and temporal algorithm into a large-scale open source library, Philos. T. Roy. Soc. A. 367 (1895) (2009) 1907–1930.

[16] M. Bendahmane, R. Bürger, R. Ruiz-Baier, A multiresolution space-time adaptive scheme for the bidomain model in electrocardiology, Numer. Methods Partial Differential Eq. 26 (6) (2010) 1377–1404.

[17] J. Nickolls, W. Dally, The gpu computing era, IEEE Micro 30 (2) (2010) 56–69.

[18] R. Buchty, V. Heuveline, W. Karl, J.-P. Weiss, A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators, Concurrency Computat.: Pract. Exper. 24 (7) (2012) 663–675.

[19] A. R. Sanderson, M. D. Meyer, R. M. Kirby, C. R. Johnson, A framework for exploring numerical solutions of advection-reaction-diffusion equations using a GPU-based approach, Comput. Visual Sci. 12 (4) (2009) 155-170.

[20] D. Sato, Y. Xie, J. N. Weiss, Z. Qu, A. Garfinkel, A. R. Sanderson, Acceleration of cardiac tissue simulation with graphic processing units, Med. Biol. Eng. Comput. 47 (9) (2009) 1011–5.

[21] J. Chai, M. Wen, N. Wu, D. Huang, J. Yang, X. Cai, C. Zhang, Q. Yang, Simulating cardiac electrophysiology in the era of GPU-cluster computing, IEICE Trans Inf Syst. E96-D (12) (2013) 2587–2595.

[22] E. Bartocci, E. M. Cherry, J. Glimm, R. Grosu, S. A. Smolka, F. H. Fenton, In *Proc. of CMSB 2011: the 9th ACM International Conference on Computational Methods in Systems Biology, Paris, France, September 21-23*, ACM (2011) 103–112 .

[23] B. M. Rocha, F. O. Campos, R. M. Amorim, G. Plank, R. W. d. Santos, M. Liebmann, G. Haase, Accelerating cardiac excitation spread simulations using graphics processing units, Concurrency Computat.: Pract. Exper. 23 (7) (2011) 708–720.

[24] F. V. Lionetti, 2010. GPU Accelerated Cardiac Electrophysiology. Master?s thesis (2010). University of San Diego, California.

[25] J. L. Puglisi, D. M. Bers. LabHEART: an interactive computer model of rabbit ventricular myocyte ion channels and Ca transport. Am. J. Physiol-Heart C. 281(6) (2001) C2049-2060.

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

[27] G. Vigueras, I. Roy, A. Cookson, J. Lee, N. Smith, D. Nordsletten, Toward GPGPU accelerated human electromechanical cardiac simulations, Int. J. Numer. Meth. Biomed. Engng. 30(1) (2014) 117–134.

[28] G. Strang, On the construction and comparison of difference schemes, SIAM J. Numer. Anal. 5 (3) (1968) 506–517.

[29] T. J. R. Hughes, The Finite Element Method: Linear Static and Dynamic Finite Element Analysis, Dover Publications, 2000.

[30] O. C. Zienkiewicz, R. L. L. Taylor, J. Z. Zhu, The Finite Element Method: Its Basis and Fundamentals, 7th Edition, Vol. 1, Butterworth-Heinemann, 2013.

[31] N. Bell, M. Garland, Cusp-library: Generic parallel algorithms for sparse matrix and graph computations, Tech. Rep. NVR-2008-008, NVIDIA corporation (Dec 2008).

[32] A. Bueno-Orovio, E. M. Cherry, F. H. Fenton, Minimal model for human ventricular action potentials in tissue, J. Theor. Biol. 253(3) (2008) 544–560

[33] M. M. Maleckar, J. L. Greenstein, W. R. Giles, N. A. Trayanova, K^+ current changes account for the rate dependence of the action potential in the human atrial myocyte, Am. J. Physiol-Heart C. 297 (2009) H1398–H1410.

[34] E. A. Heidenreich, F. J. Gaspar, J. M. Ferrero, J. F. Rodriguez, Compact schemes for anisotropic reaction-diffusion equations with adaptive time step, Int. J. Numer. Meth. Eng. 82 (8) (2010) 1022–1043.

[35] S. A. Niederer, E. Kerfoot, A. P. Benson, ... , N. P. Smith, Verification of cardiac tissue electrophysiology simulators using an n-version benchmark, Philos. T. Roy. Soc. A. 369 (1954) (2011) 4331–4351.

[36] K. G, V. Kumar, Metis. a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. (1998).

[37] S. Filippone, M. Colajanni, Psblas: A library for parallel linear algebra computation on sparse matrices, ACM Trans. Math. Softw. 26 (4) (2000) 527–550.

[38] M. Pennacchio, V. Simoncini, Efficient algebraic solution of reaction-diffusion systems for the cardiac excitation process, J. Comput. Appl. Math. 145 (1) (2002) 49–70.

[39] M. Potse, B. Dube, J. Richer, A. Vinet, R. Gulrajani, A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart, IEEE T. Bio-Med. Eng. 53 (12) (2006) 2425–2435.

[40] C. Luo, Y. Rudy, A model of the ventricular cardiac action potential. depolarization, repolarization, and their interaction, Circ. Res. 68 (6) (1991) 1501–1526.

[41] A. Mahajan, Y. Shiferaw, D. Sato, ... , J. N. Weiss, A rabbit ventricular action potential model replicating cardiac dynamics at rapid heart rates, Biophys. J. 94 (2008) 392–410.