Accelerating cardiac excitation spread simulations using graphics processing units

B. M. Rocha[*], F. O. Campos[†‡], R. M. Amorim[‡], G. Plank[†§], R. W. dos Santos[‡], M. Liebmann[¶], G. Haase[¶‖]

Abstract

The modeling of the electrical activity of the heart is of great medical and scientific interest, because it provides a way to get a better understanding of the related biophysical phenomena, allows the development of new techniques for diagnoses and serves as a platform for drug tests. The cardiac electrophysiology may be simulated by solving a partial differential equation coupled to a system of ordinary differential equations describing the electrical behavior of the cell membrane. The numerical solution is, however, computationally demanding because of the fine temporal and spatial sampling required. The demand for real time high definition 3D graphics made the new graphic processing units (GPUs) a highly parallel, multithreaded, many-core processor with tremendous computational horsepower. It makes the use of GPUs a promising alternative to simulate the electrical activity in the heart. The aim of this work is to study the performance of GPUs for solving the equations underlying the electrical activity in a simple cardiac tissue. In tests on 2D cardiac tissues with different cell models it is shown that the GPU implementation runs 20 times faster than a parallel CPU implementation running with 4 threads on a quad–core machine, parts of the code are even accelerated by a factor of 180.

Keywords: Cardiac electrophysiology, graphic processing units, high performance computing

1 INTRODUCTION

The phenomenon of electrical propagation in the heart comprises a set of complex nonlinear biophysical processes. Its multi-scale nature spans from nanometer processes such as ionic movements and protein dynamic conformation, to

[*]National Laboratory of Scientific Computing, Av. Getúlio Vargas, 333, 25651-075 Petropolis, Brazil

[†]Medical University of Graz, Institute of Biophysics, Harrachgasse 21, 8010 Graz, Austria

[‡]Federal University of Juiz de Fora, Department of Computer Science and Master Program in Computational Modeling, Martelos, 36036-330 Juiz de Fora,Brazil

[§]Oxford e–Research Centre, University of Oxford, UK

[¶]Karl–Franzens–University of Graz, Institute for Mathematics and Scientific Computing, Heinrichstrasse 36, 8010 Graz, Austria

[‖]The final publication can be found at http://onlinelibrary.wiley.com/doi/10.1002/cpe.1683/pdf

centimeter phenomena such as whole heart contraction [1]. Computer models have become valuable tools for the study and comprehension of such complex phenomena, as they allow different information acquired from different physical scales and experiments to be combined in order to generate a better picture of the whole system functionality.

There are two components that contribute to the modeling of cardiac electrical propagation [2]. The first is a model of cellular membrane dynamics, describing the flow of ions across the cell membrane. Such models are usually formulated as a system of nonlinear ordinary differential equations (ODEs) describing processes occurring on a wide range of time scales. These models are being continually developed to give an increasingly detailed and accurate description of cellular physiology. However, this development also tends to increase the complexity of the models making them substantially more costly to solve numerically [3, 4].

The second is an electrical model of the tissue that describes how currents from one region of a cell membrane interact with other regions and with neighboring cells. When these two components are put together, the ODEs are coupled to a set of partial differential equations (PDEs) giving rise to the bidomain model [5].

Despite being currently the most complete description of cardiac electrical activity, the numerical solution of the bidomain equations are very computationally demanding [6, 7, 8, 9]. A simpler approximation can be obtained from the model formulation in [5] by considering the extracellular potential to be constant; or the tissue conductivity to be isotropic; or the intracellular and extracellular conductivities to have equal anisotropy ratios [10]. If one of these assumptions is made, the equations are reduced to the monodomain model [11], which has been recently demonstrated to yield essentially to equivalent results as the bidomain model [12].

Nevertheless, the numerical solution of these tissue models involves the discretization in space and time of PDEs as well as the integration of nonlinear systems of ODEs. Using an appropriate decomposition for the time discretization, the nonlinearity of the system may be isolated, i.e. for each node of the spatial mesh a system of ODEs, that comes from the cellular model, may be solved independently. To perform realistic simulations of cardiac tissue, a large number of ODE systems must be solved at each time step, which contributes substantially to the total computational work. Therefore, it is necessary to pursuit new efficient ways of solving the large linear algebraic system that arises from the discretization of the PDEs as well as the systems of ODEs associated with the cell models.

The advent of multi-core CPUs and many-core GPUs means that mainstream processor chips are now parallel systems. Remarkably, the newer programmable GPUs have become a highly parallel, multithreaded, many-core processors with tremendous computational horsepower and very high memory bandwidth. Nevertheless, to develop applications that can efficiently exploit the increasing number of processor cores of this technology is still a challenge. The NVIDIA CUDA technology [13] is a general purpose parallel program-

ming model and software environment designed to overcome this challenge while maintaining a low learning curve for programmers familiar with standard programming languages such as C.

Many efforts have been made to solve efficiently the equations modeling the cardiac electrical activity on a single CPU [3, 6, 14] as well as in cluster of computers [7, 9]. A recent study has also reported the use of GPUs to parallelize the monodomain model [15]. However, a rigorous analysis of the parallel solvers is not shown and only one simple cellular model is employed in the simulations.

The aim of this work is to evaluate the performance of CPU and GPU solvers for the monodomain model developed in standard C (parallelized with the OpenMP library [16]) and extended to the NVIDIA CUDA parallel environment. Differently to what was presented in [15], we analyze the performance of the ODE solver to different and more complex ionic models and also the performance of iterative solvers with sparse matrices.

2 MATHEMATICAL FORMULATION

2.1 Monodomain model

In cardiac tissue, the excitation wave spreads through the tissue because the cardiac cells are electrically coupled via special proteins called gap junctions. The monodomain model describes the propagation of the excitation wave:

$$\nabla \cdot (\boldsymbol{\sigma} \nabla V_m) = \beta I_m \tag{1}$$

where β is the surface-to-volume ratio of the cardiac cells, V_m is the transmembrane voltage, $\boldsymbol{\sigma}$ is the conductivity tensor and I_m is the ionic current across the membrane.

In order to accurately simulate the propagation pattern of the electrical wave front it is necessary to include the orientation of the muscle fibers in the model. The heart muscle is strongly anisotropic since it consists of fibers and the conductivity is higher in the direction of the fibers than in the cross–fiber direction. The muscle fibers are organized in sheets, which defines three directions for the conductive values of the tissue: parallel to the fibers, orthogonal to the fibers but parallel to the sheet, and perpendicular to the sheet. At any point $\mathbf{x}$ we have three unit vectors $\mathbf{a_l}(\mathbf{x}), \mathbf{a_t}(\mathbf{x}), \mathbf{a_n}(\mathbf{x})$, with $\mathbf{a_l}(\mathbf{x})$ parallel to the local fiber direction, $\mathbf{a_t}(\mathbf{x})$ orthogonal to $\mathbf{a_l}(\mathbf{x})$ but in the sheet plane, and $\mathbf{a_n}(\mathbf{x})$ normal to the sheet plane. Using these vectors, an entry (i, j) of the intracellular conductivity tensor $\boldsymbol{\sigma}$ can be expressed as

$$\sigma^{ij} = \sigma_l a_l^i a_l^j + \sigma_t a_t^i a_t^j + \sigma_n a_n^i a_n^j \tag{2}$$

where σ_l, σ_t and σ_n are the intracellular conductivities defined by fiber direction, across the fiber direction and normal to the sheet plane, respectively.

2.2 Ionic models

The dynamics of the processes that underlie the action potential (AP) in a cardiac cell are typically described by a set of ODEs modeling the total ionic current I_m (predominantly Na^+, K^+, Ca^{2+} and Cl^-) through protein channels in the cell membrane [2]:

$$I_m = C_m \frac{\partial V_m}{\partial t} + I_{ion}\left(V_m, \eta_i\right) - I_{stim} \tag{3}$$

$$\frac{d\eta_i}{dt} = f(t, \eta_i) \tag{4}$$

where C_m is the membrane capacitance, I_{ion} is the sum of ionic transmembrane currents present in the cardiac cell, f is a vector–valued function, η are variables that contribute to the modeling of I_{ion} and I_{stim} is a stimulus current.

In this work two different cardiac cell models with different levels of complexity and suitability for performing large-scale spatial simulations were considered to simulate the kinetics of I_{ion} in Eqs. (3)-(4): the classical LR-I model [17] describing the electrical activity in a general mammalian ventricular cell; the TNNP human ventricular model [18] designed to provide a good trade-off between a considerable level of physiological detail and computational efficiency [19].

2.2.1 The LR-I model

In LR-I model, I_{ion} is defined as:

$$I_{ion} = I_{Na} + I_{si} + I_K + I_{K1} + I_{Kp} + I_b \tag{5}$$

where I_{Na} is the fast sodium current, I_{si} is the slow inward current, I_K is the time-dependent potassium current, I_{K1} is the time-independent potassium current, I_{Kp} is the plateau potassium current and I_b is time-independent background current.

Four of the 6 ionic currents in Eq. (5) are controlled by gating variables described by ODEs following the the Hodgkin-Huxley formalism [20] which are of the form

$$\frac{dn}{dt} = \frac{n_\infty - n}{\tau_n} \tag{6}$$

where n represents the gating variable, and the terms n_∞ and τ_n are defined as

$$n_\infty = \frac{\alpha}{\alpha + \beta}, \qquad \tau_n = \frac{1}{\alpha + \beta}. \tag{7}$$

with α and β being functions of V_m whose complete description can be found in [17].

In addition to the ODEs for the gating parameters, the model includes an ODE for the intracellular calcium concentration $[Ca^{2+}]_i$:

$$\frac{d[Ca^{2+}]_i}{dt} = -10^{-4} I_{si} + 0.07\left(10^{-4} - [Ca^{2+}]_i\right) \tag{8}$$

In summary, the model is based on a set of 8 ODEs describing ionic currents and intracellular calcium concentration. For a full specification of the ionic currents see [17].

2.2.2 The TNNP model

In the TNNP human ventricular model, the term I_{ion} is the sum of all transmembrane ionic currents and given by:

$$\begin{aligned} I_{ion} = & I_{Na} + I_{K1} + I_{to} + I_{Kr} + I_{Ks} + I_{CaL} \\ & + I_{NaCa} + I_{NaK} + I_{pCa} + I_{pK} + I_{bCa} + I_{bNa} \end{aligned} \tag{9}$$

where I_{Na} is the fast Na^+ current, I_{K1} the inward rectifier K^+ current, I_{to} the transient outward K^+ current, I_{Kr} and I_{Ks} are the rapid and the slow delayed rectifier K^+ currents respectively, I_{CaL} the L-type Ca^{2+} current, I_{NaCa} is the Na^+/Ca^{2+} exchanger, I_{NaK} is Na^+/K^+ pump, I_{pCa} and IpK are plateau Ca^{2+} and K^+ currents, and I_{bCa} and I_{bK} are background Ca^{2+} and K^+ currents.

Likewise the LR-I model, every ionic current is described by a formalism similar to that presented by Eqs. (6)-(7). Intracellular Na^+ and K^+ concentrations as well as a more extensive description of intracellular Ca^{+2} handling are also included in the model.

In summary, the model is based on a set of 19 ODEs that describes the ionic currents and the intracellular concentration of Na^+, K^+ and Ca^{+2}. The reader is referred to [18] for a full specification of the model.

3 NUMERICAL SCHEMES

In order to obtain a numerical solution for Eqs. (1)-(4), the Godunov operator splitting [21] was employed. The coupled system is then reduced to a two numerical step scheme that involves the solutions of a parabolic PDE and a nonlinear system of ODEs at each time step. This splitting leads to the following two steps algorithm:

1. Solve the systems of ODEs

$$\frac{\partial V_m}{\partial t} = \frac{1}{C_m}\left[-I_{ion}(V_m, \eta_i) + I_{stim}\right] \tag{10}$$

$$\frac{\partial \eta_i}{\partial t} = f(V_m, \eta_i) \tag{11}$$

2. Solve the parabolic problem

$$\frac{\partial V_m}{\partial t} = \frac{1}{\beta C_m}\left[\nabla \cdot (\boldsymbol{\sigma} \nabla V_m)\right] \tag{12}$$

The finite element method (FEM) is used for the spatial discretization of Eq. (12) and, the second–order Crank-Nicolson method is used to advance the solution in time. Applying the FEM for the spatial discretization results in a linear system of equations that needs to be solved at each time step and has the following form:

$$(M + \frac{C}{2}K)v^{k+1} = (M - \frac{C}{2}K)v^{k} \tag{13}$$

where v discretizes V_m at time $k\Delta t$; M and K are the mass and stiffness matrices from the FEM discretization, respectively; and the constant C is $\frac{\Delta t}{\beta C_m}$. The preconditioned conjugate gradient method (PCG) was used to solve this linear system using the Jacobi preconditioner.

Due to the discretization nature of the FEM, the matrices K and M are sparse, i.e., matrices where the number of non-zero entries is only a small fraction of the total. Realistic anatomic models of the heart may easily exceed 1 million nodes [22], which after spatial discretization using the FEM lead to large sparse matrices. Therefore, it makes sense to look for sparse matrix representations that store only the non-zero elements of the matrix. In the present work, specific data structures were used to store the FEM matrices in an efficient manner.

Since in this work we used only 2D structured square meshes the resulting matrices are structured and therefore the following two sparse matrix formats were used: the *compressed sparse row* (CSR) format and the ELLPACK format [23]. Let N and Nz be the number of rows of the matrix and the total number of non-zero entries of the matrix, respectively. Then, the CSR format stores a sparse matrix in three arrays: *vals*, *cols* which are of size Nz and hold, respectively, the non-zero entries and the columns indexes of these entries and finally, a third array *ptrs* of size $N + 1$ that stores the pointers to the beginning of each row in the arrays *vals* and *cols*, as illustrated in Figure 1. This CSR format is the natural choice when working with matrices originating from FEM discretizations of unstructured meshes and therefore we wanted to investigate this storage scheme.

The ELLPACK format stores the sparse matrix in two arrays of dimensions $N \times MaxRow$, where $MaxRow$ is the maximum number of non-zeros entries per row in the matrix. One array, *vals*, saves the non-zero entries of the matrix and the other, *cols*, the indexes of the columns of every entry, as illustrated in Figure 1. Since this format has a regular data structure, like a dense matrix, it is suitable for operations on vector architectures [23]. On the other hand, operations with this format lose performance when it is used to store matrices obtained from unstructured meshes where the number of entries per row is irregular.

The system of ODEs in Eqs. (10)-(11) was integrated using the explicit Euler method. Although it is well known that explicit numerical methods have strong limitations because of stability and accuracy restrictions [3], they are widely used due to their simplicity of implementation [6, 9].

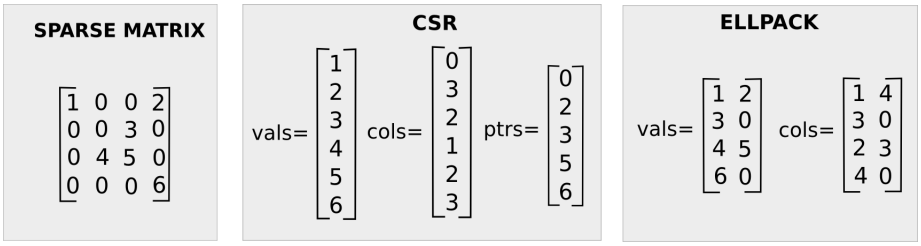

Figure 1: Example of storage of a sparse matrix using the CSR and ELLPACK formats.

Table 1: Dimensions of the *in-silico* tissue preparation, number of nodes, elements and non-zero entries of the sparse matrix.

Tissue	Nodes	Elements	Non-zero
$1.6\,\text{cm} \times 1.6\,\text{cm}$	25921	25600	231361
$3.2\,\text{cm} \times 3.2\,\text{cm}$	103041	102400	923521
$6.4\,\text{cm} \times 6.4\,\text{cm}$	410881	409600	3690241

4 METHODS

We have developed three different *in-silico* tissue preparations for our investigations. Each preparations is a two dimensional square piece of tissue, which was spatially discretized through the FEM with bilinear quadrangular elements, resulting in diagonal structured matrices with a maximum of 9 non-zero entries per row. Table 1 presents the physical dimensions of the simulated tissues, properties of the FEM meshes as well as the number of non-zero entries stored in the sparse matrix formats.

Boundary conditions were modeled by imposing homogeneous Neumann conditions on V_m in all simulations. Spatial discretization was set to $\Delta x = 100\,\mu\,m$. Numerical tests using the forward Euler method to solve the system of ODEs for the LR-I and TNNP models revealed that a $\Delta t = 0.01\,ms$ and $\Delta t = 0.001\,ms$, respectively, are the largest allowable time steps for stability constraints. The absolute tolerance of the PCG method was set to 10^{-6}.

The parameters of the monodomain model were taken from the literature [1, 24]: $C_m = 1\,\mu F/cm^2$; e$\beta = 0.14\,\mu m^{-1}$. Tissue conductivity was considered homogeneous and anisotropic with conductivity values given by $\sigma_l = 0.0003\,mS/\mu m$ and $\sigma_t = 0.0001\,mS/\mu m$ (also taken from the literature [25]). Cardiac fibers were also homogeneously oriented to 45 degrees.

0.6

We simulated $500\,ms$ of electrical propagation in these cardiac tissues after applying a stimulus in the central area of the preparations. The stimulus was made by setting V_m in this area to a value above the threshold required to

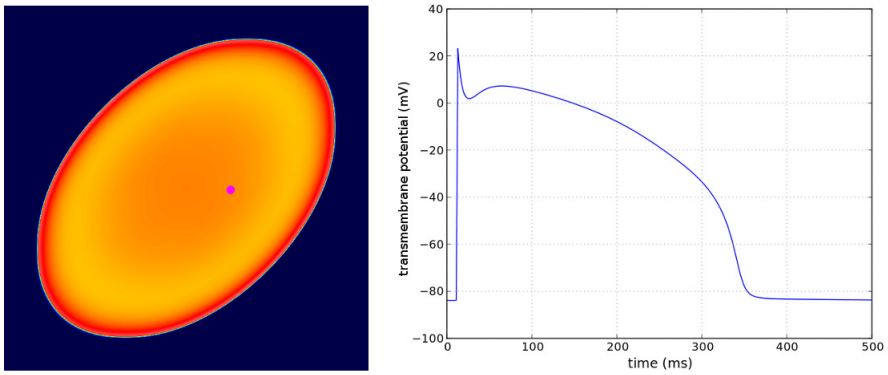

Figure 2: Simulation results. A) Spatial distribution of V_m in a 6.4 cm $\times$ 6.4 cm tissue simulation using the LR-I model 50 ms after the stimulus onset. B) The AP related to the cell highlighted in A).

generate an AP. Figure 2 (A) illustrates the spatial distribution of the transmenbrane potential, V_m, for a simulated tissue of dimensions 6.4 cm $\times$ 6.4 cm using the LR-I model at 50ms following the stimulus onset; and (B) the time course of the AP of the cell highlighted in (A).

4.1 Implementation details

In order to obtain high performance using GPU programming some requisites should be kept in mind. The first is the challenge inherent to the development of applications for parallel systems, which should balance the workload and scale it over a number of processors. In addition, differently than in the CPU, the global memory space is not cached in the GPU. Finally, it is also very important to follow the right access pattern to get maximum memory bandwidth [13].

We implemented the numerical solution of the monodomain problem described by Eqs. (1)-(4), in which I_{ion} is given by either Eq. (5) or Eq. (9). The program for the CPU version was written in C and compiled with GNU C compiler version 4.3.3. In order to exploit the power of the available multi-core processors we parallelized the CPU code using OpenMP [16]. The code for the GPU was extended from the CPU code to the NVIDIA CUDA parallel environment. Thus, specific CUDA kernels were developed to solve the system of ODEs as well as the parabolic problem. Figure 3 illustrates the solution algorithm of the monodomain model using the GPU, describing memory allocations, computations and data transfers betwenn CPU and GPU.

All CUDA kernels in this work were launched with 256 threads per block and the number of blocks per grid was set accordingly to the size of the problem, i.e., accordingly to the number of nodes in the FEM mesh. Additionally, we used single precision for all floating point operations.

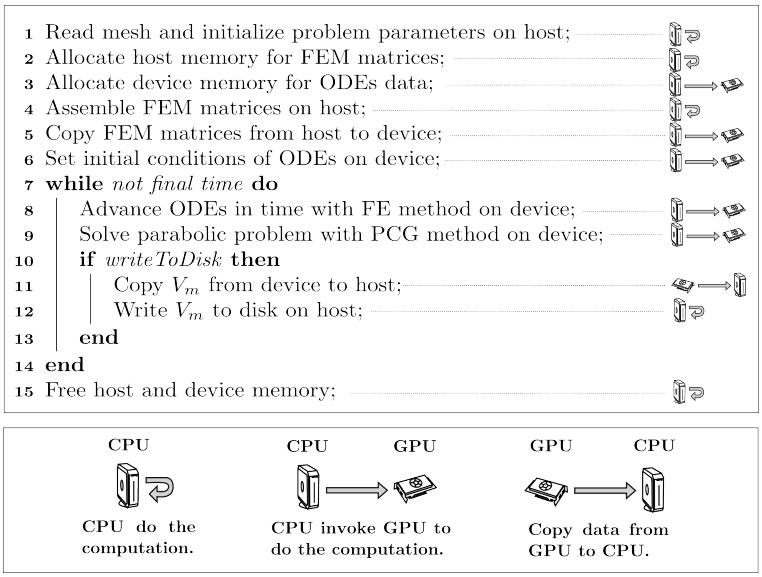

Figure 3: Solution algorithm of the monodomain model on the GPU.

4.2 The ODE systems

We have developed two kernels to solve each of the systems of ODEs related to Eqs. (3) and (4). The first one refers to the setting of initial conditions to the systems of ODEs, whereas the second kernel integrates the systems of ODEs at each time step.

The state variables of M cells were stored in an array SV of size $M * N_{eq}$, where N_{eq} is the number of differential equations of the ionic model (in this work $N_{eq} = 8$ or $N_{eq} = 19$ for the LR-I or TNNP model, respectively). The SV array was organized in such way that the first M entries correspond to the first state variable, followed by M entries of the next state variable, and so on. Moreover, for all ionic models the first M entries of the SV array correspond to the transmembrane potential V_m. During the solution, after the integration of the ODEs systems, the transmembrane potential of each node should be passed to the PCG solver. Due to the memory organization of the SV array this is straightforward, since the M first entries of the array correspond to the V_m of each node, which avoids extra memory transactions between CPU and GPU.

The global memory of the SV array was created using the `cudaMallocPitch` routine from the CUDA API [13] that allocates a pitch linear memory and may pad the allocation to get best performance of a given piece of hardware by meeting alignment requirements for memory coalescing [13]. A strict coalescing requires that thread j out of n threads has to access data $u[j]$ if $u[0]$ is accessed by thread 0, i.e., each thread should perform data access by stride n. Otherwise the performance decreases by a factor of 10 for older GPUs as the 8800 GTX and a factor of 2 on the GTX 280. Therefore, in the kernel to set the initial

conditions each thread sets the values of all its state variables. The kernel that solves the system of ODEs operates similarly, i.e., each thread computes and updates its state variables writing to the right position in memory that correspond to their variables.

4.3 The parabolic problem

The solution of the parabolic problem using the PCG method essentially consists of basic linear algebra operations such as: vector addition, multiplication by a scalar, vector scalar product and sparse matrix vector multiplication (SpMV). The implementation of linear algebra operations such as vector addition, multiplication by a scalar and the SAXPY operation ($y = ax + y$) in CUDA is a straightforward task since each thread is assigned to do the computation of one position of the array. On the other hand, the implementation of the scalar product is not straightforward since it requires a parallel reduction operation. In the current work, this kernel was implemented doing a partial sum of the vectors on each multiprocessor and then a reduction on shared memory.

The efficiency of iterative solvers such as the PCG greatly depends on the performance of SpMV operations ($y = Ax$), since these methods usually require several matrix-vector products to reach convergence. Therefore, the implementation of this operation has to be carefully designed to obtain satisfactory results. The SpMV using the CSR format can be easily parallelized since the product of one row of the matrix and the vector is independent of other rows. We implemented it assigning one thread to compute one dot product of the matrix row and the vector. Although, there are more efficient variants of this algorithm such as the *block compressed row storage* (BCRS) [26] or the *CSR vector* kernel [27], we have only tested the CSR kernel as described before.

In our implementation of the ELLPACK format the arrays *vals* and *cols* were stored as one dimensional arrays using a column–major order. This ensures that threads within the same warp access memory contiguously [27]. The ELLPACK format uses one thread per matrix row, thus each thread computes the dot product between the corresponding matrix row and the x vector. Furthermore, the implementation of this format used padding to align the *vals* and *cols* arrays of the ELLPACK format appropriately, which ensures full coalesced accesses to memory. A warp is a group of threads executed physically in parallel on one of the 30 streaming multi–processors (SM) of a GTX280, each SM consists of 8 execution cores.

All SpMV kernels of this work used the *texture cache* of the CUDA–enabled devices to access the x vector during the matrix vector multiplication in order to improve performance. In these kernels the read operation of the x vector were replaced with a texture fetch instruction using the `tex1Dfetch` function.

5 RESULTS

In this section we present the numerical results obtained for the monodomain simulations using the LR-1 and TNNP cell models on graphics processing units

Table 2: Environment specifications.

	Memory	Mem. Bandwidth.	Peak Performance.
GPU - NVIDIA GeForce GTX 280	1 GB	141.7 GB/s	933.1 GFLOPS
CPU - AMD Phenom 9950 Quad-Core	8 GB	17.1 GB/s	12.5 GFLOPS

through NVIDIA CUDA environment and on a CPU using OpenMP. We used the NVIDIA GeForce GTX 280 with a total of 240 CUDA cores and 1GB of memory. A description of the hardware used in our numerical experiments is given in Table 2. The code was compiled with NVIDIA CUDA 2.0 and GCC 4.3.3, Using this environment, simulations were performed on a Linux 2.6.28-18 x86-64, using OpenMP with 4 cores and compared against simulations running on the GPU.

It is important to point out that the following optimization flags were used: `-O3 -ffast-math` to compile the CPU code with GCC and `-O3 -use_fast_math` to compile the GPU code with the NVIDIA compiler nvcc. Although the flags `-ffast-math` for CPU code and `-use_fast_math` for GPU code seems to be similar, they actually optimize the code differently. In spite of this fact, they were used in order to optimize each code as much as possible with respect to mathematical operations.

0.6

5.1 Multi-core CPU

Initially, we studied the performance of the OpenMP implementation with a different number of cores. Simulations as previously described were performed using the 641×641 tissue preparation using 1, 2 and 4 processors cores. Figure 4 presents the elapsed time to solve the ODE and the parabolic problems. When using 4 cores a parallel speedup of about 3.98 was observed for the ODE problem, whereas a speedup of 2.35 was achieved for the parabolic problem.

5.2 GPU *vs* Multi-core CPU

We compared the CPU and GPU implementations by means of elapsed time required to integrate the ODE systems and to solve the parabolic system. Table 3 shows the execution times for the ODE solvers. Execution times for the two different sparse matrices formats were not compared since it is irrelevant for the ODE solver. According to Table 3, the GPU achieved much better performance than the CPU in all tissue simulations. Speedup factors between 132-fold and 180-fold were achieved for the LR-I, whereas 65-fold and 75-fold were obtained for the TNNP model.

Furthermore, we measured execution times for the parabolic problem which consists of a SpMV operation and a call to the PCG solver. The execution

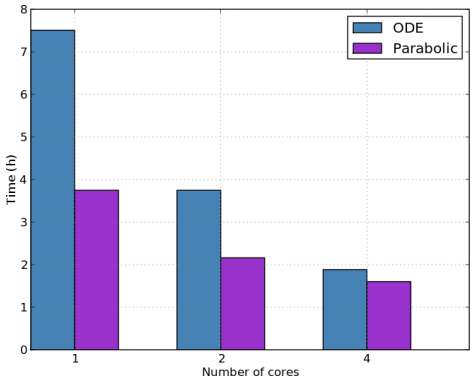

Figure 4: Execution times of the ODE and parabolic problems given in hours for the LR-I model in the 641 $\times$ 641 testcase using OpenMP with 1, 2 and 4 cores.

Table 3: Comparison of execution times in seconds between CPU using 4 cores and GPU solvers for the ODE problem.

	LR-I			TNNP		
Tissue (cm)	CPU	GPU	Speedup	CPU	GPU	Speedup
1.6 $\times$ 1.6	509.80	3.86	132.07	8586.43	130.30	65.90
3.2 $\times$ 3.2	1728.64	10.48	164.95	34370.58	463.38	74.17
6.4 $\times$ 6.4	6776.98	37.64	180.05	136453.63	1804.67	75.61

times for the parabolic problem with different sparse matrix formats are shown in Table 4, where CPU denotes the execution time on the CPU with 4 cores using the CSR format, GPU_{CSR}, GPU_{ELL}, $\text{Speedup}_{\text{CSR}}$ and $\text{Speedup}_{\text{ELL}}$ are the execution times on the GPU with the CSR and ELLPACK sparse matrix representations and their respective speedups, respectively. Table 4 shows that the GPU implementation obtained speedup factors between 1.5-2.3 and between 2-9 using the CSR and ELLPACK formats, respectively. We focused in comparisons between the CSR and ELLPACK sparse matrix representations for electrophysiological simulations on the GPU only, since it is not the aim of this work to compare different formats in single- or multi-core environments. Therefore, for the CPU simulations we used CSR format because the FEM matrices are naturally assembled in this format in our code. In Table 4 we also observe that the type of cell model, LR-I or TNNP, only marginally affects the execution times of the parabolic problem.

0.8

0.8

Finally, we compared the total execution time of each simulation, which includes not only the time to solve the systems of ODEs and the parabolic problem

Table 4: Comparison of execution times in seconds between CPU using 4 cores and GPU solvers for the parabolic problem.

	Tissue (cm)	CPU	GPU_{CSR}	GPU_{ELL}	$Speedup_{CSR}$	$Speedup_{ELL}$
LR-I	1.6 × 1.6	350.75	236.28	121.02	1.48	2.90
	3.2 × 3.2	1350.36	689.38	245.54	1.96	5.50
	6.4 × 6.4	5761.30	2427.48	648.78	2.37	8.88
TNNP	1.6 × 1.6	4338.50	3029.73	1607.76	1.43	2.70
	3.2 × 3.2	17834.58	8984.83	3217.05	1.98	5.54
	6.4 × 6.4	77654.12	31406.05	8394.53	2.47	9.25

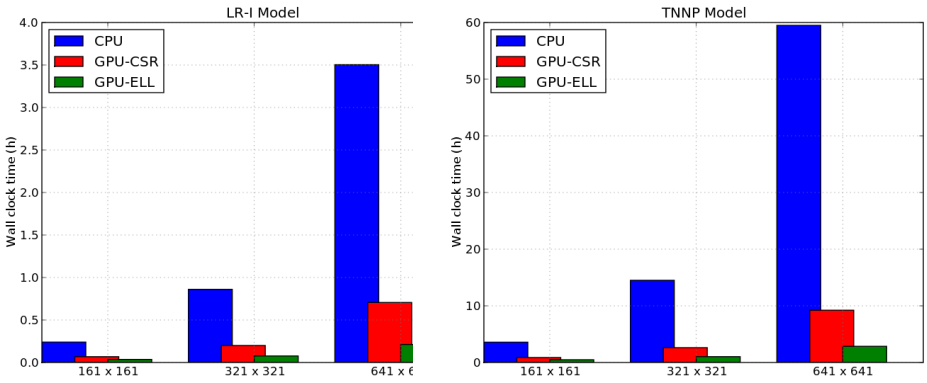

Figure 5: Total execution times given in hours for the LR-I and TNNP models.

but also the time spent assembling the FEM matrices and writing data to disk. Figure 5 presents the total time of the simulations on CPU and on GPU implementations for the LR-I and TNNP model on the left and right, respectively. Clearly, the GPU implementations were faster than the CPU version for both ionic models. In the largest tissue simulation using the TNNP model, the GPU with ELLPACK was up to 20 times faster than the CPU with 4 cores, which took about 59 hours to execute, whereas on the GPU it took about 2 hours to complete. Furthermore, it shows that the GPU implementation with the ELLPACK format was faster than GPU version with the CSR format. Comparing both GPU versions, we noticed that the ELLPACK format was 3 times faster than the CSR format for the *6.4 cm x 6.4 cm* case and at least 2 times faster for the other tissue sizes.

In order to check for possible deviations in the numerical solutions due to the use of single precision in the arithmetic operations, the V_m obtained using single

precision was compared against one computed using double precision through the relative root-mean-square norm $\text{RRMS} = 100\sqrt{\sum_t \sum_{i,j} (v_{i,j}^t - \nu_{i,j}^t)^2} / \sqrt{\sum_t \sum_{i,j} (v_{i,j}^t)^2}$ where v is the solution computed using double precision, ν is the solution computed using single precision and t and i, j are indexes in time and space, respectively. The RRMS error using the 1.6cm $\times$ 1.6cm mesh revealed that an error of $1.4950 \times 10^{-3}\%$ and $1.5383 \times 10^{-3}\%$ for the LR–I model using the CPU and GPU, respectively. Similarly, the RRMS error for the TNNP model was $3.7669 \times 10^{-1}\%$ and $3.7598 \times 10^{-1}\%$.

6 DISCUSSION

We have evaluated the performance of CPU and GPU solvers for the monodomain model developed in standard C with OpenMP and with NVIDIA CUDA parallel environment. Three different *in-silico* tissue preparations were used in this work for the performance tests. In all cases, the GPU performed much better than the CPU during the integration of the ODE systems, where a speedup of about 75-fold was obtained for the TNNP ionic model. An even better speedup of about 180-fold was achieved for the LR-I model. By comparing these two ionic models we observed that the mathematical equations of the right-hand side equations of the LR-I model were more complex than those found in the TNNP model. This has lead to a higher arithmetic intensity (operations/ words transferred) for the LR-I and, as a consequence, to better speedups obtained by the GPU implementation. The LR-I model had an arithmetic intensity of 44.1 and the TNNP model had an arithmetic intensity of 31.8. Another reason for speedup difference is the amount of registers needed by the kernels. The LR-I model kernel needed less registers than the maximum allowed by the block size choosen, whereas the TNNP model kernel needed more than the maximum and had to use the maximum allowed for the configuration.

Moreover, a modest speedup of about 2.5-fold was achieved during the solutions of the parabolic problem using the CSR format in the GPU. This speedup increased to 9-fold when the GPU implementation adopted the ELLPACK format. The performance of the parabolic solver is strongly dependent on the storage format for sparse matrix associated to linear system since during its solutions several SpMV are performed. We have employed the well known CSR format which is used to store general sparse matrices, and the ELLPACK format which is more suitable for matrices arising from structured meshes. Although the CSR format is general and easy to implement, from the point of view of parallelization using the CUDA environment, the CSR format has a significant drawback. In this format, the threads do not access the CSR arrays *cols* and *data* contiguously resulting in uncoalesced accesses to the global memory. On the other hand, since the matrices resulting from the spatial discretization are diagonal structured the use of the ELLPACK format for the SpMV operations delivered better performance for the parabolic problem since the global memory accesses are fully coalesced.

The performance gain of the GPU in comparison to using all 4 cores of

the quad–core CPU reaches a factor of 9 for the parabolic problem solver and an excellent acceleration of 180 in case of the ODE solver. Furthermore, we noticed that the single precision used did not influence the accuracy of numerical solutions. The presented results show that GPUs are a promising alternative to clusters of CPUs for simulating the electrical activity in the heart. The GPU allows to increase complexity and dimensions of the problem that can be modeled, and also represents a parallel system with high performance computing capabilities for relatively low costs in comparison to a cluster of CPUs needed to achieve the same performance.

6.1 Limitations and Future Work

We have compared the performance of GPU and Multi-core implementations using the forward Euler method for the solution of the cell models. Although the Euler method is the most commonly used method in the area, more efficient methods have been proposed and employed for the solution of the ODEs, such as the Rush-Larsen [28] method and implicit solvers, which are difficult to implement. In addition, our experiments were carried out with structured meshes, whereas whole heart simulations would require the use of unstructured meshes to precisely represent the cardiac tissue and its boundaries. Therefore, further studies on different sparse matrix formats and on efficient preconditioners for the CG solver as well as other iterative solvers are required. Among the several sparse matrix representation available the Hybrid format [27] is a promising alternative for unstructured matrices. Several preconditioners such as incomplete LU factorization, SSOR and AMG have been tested for the bidomain model [8, 7] on clusters of CPU. The performance of these preconditioners should be evaluated for the case of cardiac excitation simulations running on graphic processors units.

6.2 Related Work

Recently, Sato et al. have presented in [15] the performance results of an implementation of the Monodomain equations on GPUs. The implementation was based on the Nvidia CUDA language. Both system of ODEs and parabolic problem were solved via the explicit forward Euler method and the cell model used in this study was phase I of the LuoRudy action potential model [17]. The reported GPU implementation was 30 times faster than a CPU implementation. Although the numerical methods and cell models are very different from those used in this work we have obtained a similar speedup of 20 for the Monodomain model. We used the second order Crank-Nicolson method for the solution of the parabolic problem which requires the solution of a linear system at each time step and in addition we also used the ten–Tusscher model.

In [29], Vigmond et al. have presented a GPU implementation of the linear system of ODEs as part of the simulations of the Monodomain equations. The implementation was based on the Nvidia CUDA. The reported results indicated

that the GPU implementation was around 10 times faster than the CPU implementation. The rabbit ventricular cell model published by Mahajan et al. [30] was chosen to describe the cellular dynamics.

Recently we have reported in [31] an automatic tool that converts cell models described by the CellML language [32] to CUDA-based code for GPU. Four different cell models were tested (LuoRudy, ten-Tusscher, Mahajan and Bondarenko [33]) as linear systems of ODEs in simulations based on the Monodomain equations. Speedup factors between 75 and 290 were obtained by the GPU implementations when compared to the CPU ones.

ACKNOWLEDGEMENTS

The work of B. M. Rocha was supported by CAPES and LNCC. F. O. Campos and G. Plank were supported by the grant P19993-N15 and SFB grant F3210-N18 from the Austrian Science Fund (FWF). The work of R. M. Amorim and R. W. dos Santos was supported by FAPEMIG, CNPq and CAPES. The work of G. Haase and M. Liebmann was supported by the BMWF via the Austrian Grid 2 project and by SFB grant F3210-N18 from FWF.

References

[1] G Plank, L Zhou, J L Greenstein, S Cortassa, R L Winslow, B O'Rourke, N A Trayanova. From mitochondrial ion channels to arrhythmias in the heart: computational techniques to bridge the spatio-temporal scales. *Philos Transact A Math Phys Eng Sci* 2008; **366**:3381–409.

[2] R Plonsey, R C Barr. Mathematical modeling of electrical activity of the heart. *Electrocardiol* 1987; **20**(3):219–26.

[3] M C Maclachlan, J Sundnes, R J Spiteri. A comparison of non-standard solvers for odes describing cellular reactions in the heart. *Comput Methods Biomech Biomed Engin* 2007; **10**:317–326.

[4] R J Spiteri, M MacLachlan. An efficient non-standard finite difference scheme for an ionic model of cardiac action potentials. *Journal of Difference Equations and Applications* 2003; **9**(12):1069–1081.

[5] D B Geselowitz, W T Miller. A bidomain model for anisotropic cardiac muscle. *Ann Biomed Eng* 1983; **11**:191–206.

[6] E J Vigmond, F Aguel, N Trayanova. Computational techniques for solving the bidomain equations in three dimensions. *IEEE Trans. Biomed. Eng.* 2002; **49**(11):1260–1269.

[7] R W Santos, G Plank, S Bauer, E J Vigmond. Parallel multigrid preconditioner for the cardiac bidomain model. *IEEE Trans Biomed Eng* 2004; **51**(11):1960–1968.

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

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

[10] J Sundnes, B F Nielsen, K A Mardal, X Cai, G T Lines, A Tveito. On the computational complexity of the bidomain and the monodomain models of electrophysiology. *Ann. of Bio. Eng.* 2006; **34**(7):1088–1097.

[11] J Clark, R Plonsey. A mathematical evaluation of the core conductor model. *Biophys J* 1966; **6**(1):95–112.

[12] M Potse, B Dubé, A Vinet, R Cardinal. A comparison of monodomain and bidomain propagation models for the human heart. *IEEE Trans. Biomed. Eng.* 2006; **53**:2425–2435.

[13] NVIDIA Corporation. NVIDIA CUDA Programming Guide 2009.

[14] J Sundnes, G Lines, A Tveito. Efficient solution of ordinary differential equations modeling electrical activity in cardiac cells. *Mathematical Biosciences* 2001; **172**:55–72.

[15] 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* 2009; **47**:1011–1015.

[16] The OpenMP API specification for parallel programming. OpenMP. www.openmp.org 2010.

[17] C Luo, Yoram Rudy. A model of the ventricular cardiac action potential. depolarization, repolarization, and their interaction. *Circ Res* 1991; **68**(6):1501–26.

[18] K H ten Tusscher, A V Panfilov. Alternans and spiral breakup in a human ventricular tissue model. *Am J Physiol Heart Circ Physiol* 2006; **291**(3):H1088–100.

[19] K H ten Tusscher, O Bernus, R Hren, A V Panfilov. Comparison of electrophysiological models for human ventricular cells and tissues. *Prog Biophys Mol Biol* 2006; **90**(1-3):326–45.

[20] A L Hodgkin, A F Huxley. A quantitative description of membrane current and its application to conduction and excitation. *The Journal of Physiology* 1952; **117**:500–557.

[21] J Sundnes, G T Lines, A Tveito. An operator splitting method for solving the bidomain equations coupled to a volume conductor model for the torso. *Mathematical Biosciences* 2005; **194**:233–248.

[22] A J Prassl, F Kickinger, H Ahammer, V Grau, J E Schneider, E Hofer, E J Vigmond, N A Trayanova, G Plank. Automatically Generated, Anatomically Accurate Meshes for Cardiac Electrophysiology Problems. *IEEE Trans Biomed Eng* 2008; **56**(5):1318–1330.

[23] Y Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.

[24] B J Roth. Electrical conductivity values used with the bidomain model of cardiac tissue. *IEEE Trans. Biomed. Eng.* 1997; **44**(4):326–328.

[25] J Sundnes, G T Lines, X Cai, B F Nielsen, K A Mardal, A Tveito. *Computing the Electrical Activity in the Heart*. Springer, 2006.

[26] L Buatois, G Caumon, B Lévy. Concurrent number cruncher: an efficient sparse linear solver on the gpu. *LNCC* 2008; **4782**:358–371.

[27] N Bell, M Garland. Efficient Sparse Matrix-Vector Multiplication on CUDA. *Technical Report*, NVidia Corporation 2008.

[28] S Rush, H Larsen. A Practical Algorithm for Solving Dynamic Membrane Equations. *IEEE Transactions on Biomedical Engineering* 1978; **28**(4):389–392.

[29] Vigmond EJ, Boyle PM, Leon LJ, Plank G. Near-real-time simulations of bioelectric activity in small mammalian hearts using graphical processing units. *31st Annual International Conference of the IEEE EBMS*, 2009.

[30] Mahajan A, Shiferaw Y, Sato D, Baher A, Olcese R, Yie LH, Yang MJ, Chen PS, Restrepo JG, Karma A, *et al.*. A Rabbit Ventricular Action Potential Model Replicating Cardiac Dynamics at Rapid Heart Rates. *Biophysical Journal* 2008; **94**:392–410.

[31] Amorim RM, Rocha BM, Campos FO, dos Santos RW. Automatic code generation for solvers of cardiac cellular membrane dynamics in GPUs. *Engineering in Medicine and Biology Society*, 2010. EMBC 2010 Annual International Conference of the IEEE. Submitted, 2010.

[32] Garny A, Nickerson D, Cooper J, dos Santos RW, McKeever S, Nielsen P, Hunter P. Cellml and associated tools and techniques. *Philosophical Transactions of the Royal Society A* 2008; **366**:3017–3043.

[33] Bondarenko VE, Szigeti GP, Bett GCL, Kim SJ, Rasmusson RL. Computer model of action potential of mouse ventricular myocytes. *Am J Physiol Heart Circ Physiol* 2004; **287**:1378–1403.