A Survey of GPU Computing Methods for Real Time Simulation of Cardiac Dynamics

Eduardo C. Vasconcellos[a 1], Esteban W. G. Clua[a], Flavio H. Fenton[b] and Marcelo Zamith[c]

[a]Fluminense Federal University, Niteroi, RJ, Brazil
[b]Georgia Institute of Technology, Atlanta, Georgia, USA
[c]Universidade Federal Rural do Rio de Janeiro, Seropédica, RJ, Brazil

Abstract

The World Health Organization points cardiac diseases as a major cause of death in the world. There are many computational models for helping the diagnosis and treatment of cardiac conditions. Implementing them as simulation processes may increase the power of clinical understanding of particular dysfunctions. In this sense, it is required to optimize and parallelize such computational models, focusing on real time and interactive responses. This models are mathematical representations of ionic currents in cardiac tissue and can use from two up to hundreds of variables and parameters to represent the current at a given time. Even for low complexity models with small number of state variables, numerical simulations, depending on the domain size, can take several minutes or even hours. Therefore, in this work we present a survey of most relevant approaches that are focused on heterogeneous computation using both CPU and GPU to accelerate such simulations. We present an analysis of each application in the present heterogeneous computation scenery, discussing the important features of each one and the challenges that remain open. GPUs are low cost/maintainability devices that can deliver high throughput. This makes them as great tools to make cardiac electrophysiology simulations faster enough to clinical use.

Keywords: GPU Computing, Cardiac electrophysiology models, Parallel cardiac dynamics simulations.

1. Introduction

According the World Health Organization, cardiac diseases and dysfunctions are one of the main causes of death nowadays. Many models that simulate cardiac dynamics can be found in the literature. They helps to simulate

[1]E-mail Corresponding Author: evasconcellos@id.uff.br

cardiac conditions and its response to some kind of treatment. Although, depending on the level of details represented by the model, this simulations can spend several hours to run in a single CPU [15].

In this paper we will present a review of five different published works that aim to accelerate cardiac electrical dynamic simulations through parallel computing in GPUs. In §2 we present a brief overview of the mathematical equations used to model the cardiac electrical dynamics. In §3 we summarize each one of the 5 reviewed works, highlighting their key features concerning their GPU implementations. In §4 we discuss the different strategies adopted by the authors to achieve faster parallel simulations on a GPU. In §5 we conclude and try to point some directions to future works.

2. Cardiac Electrophysiology Models

In this survey we will discuss accelerated simulations of cardiac electrodynamics. This kind of simulation are performed by solving a differential equation that describes the time evolution of the voltage across the cell membrane, called the voltage or membrane potential V, along with one or more other variables. The differential that describe cardiac electrical dynamics is called reaction-diffusion equation and has the following form:

$$\frac{\partial V}{\partial t} = \nabla \cdot D\nabla V - \frac{I_{ion}}{C_m} \ . \tag{1}$$

The first term on the right side represents the diffusion term, where D is a diffusion tensor. The second term on the right side represents the reaction component, where I_{ion} represents the total current across the cell membrane and C_m the constant cell membrane capacitance. The diffusion tensor D describes how cells are coupled together and may contains some information about tissue architecture like the local fiber orientation. The information contained in D affects the electrical wave propagation and it is important to determine the wave velocity in the tissue.

The reaction term are compound by a nonlinear system of ordinary differential equations of the form

$$\frac{d\mathbf{y}}{dt} = \mathbf{F}(\mathbf{y}, V(\mathbf{y}, t), t) \ , \tag{2}$$

but its exact form depends on the level of complexity of the electrophysiology model. For each additional variable $\mathbf{y}$, $\mathbf{F}(\mathbf{y}, V, t)$ is a nonlinear function. Clayton & Panfilov gives a good review about cardiac electrical activity modelling in [8].

3. GPU Accelerated Cardiac Simulations

Bartocci et al., in [1], made a study with CUDA implementations of 5 two-dimensional (2D) cardiac electrophysiology models with different complexities (two-variable Karma model [10], four-variable Bueno-Orovio-Cherry-Fenton model [4], eight-variable Beeler-Reuter model [2], 19-variable ten Tusscher-Panfilov model [19], 67-variable Iyer-Mazhari-Winslow model [9]). They try different approaches to accelerate the computation of reaction and diffusion terms of the reaction-diffusion equation (Equation 1). For the reaction term computation, they use look-up tables stored in texture memory for tabulating some nonlinear functions of one variable. Then they substitute the divisions that do not involve any variables by equivalent multiplications and implement the *Heaviside function* with a multiplication instead of an `if` statement. To compute the diffusion term the domain grid are split in small grids that are assigned to different thread blocks. However, the computation of the diffusion term with a numerical method can demand some neighborhood knowledge. In one approach the authors store all necessary data needed by a thread block in the shared memory, but to do so, the block will use extra threads designed only to load neighbour data from global memory into shared memory. In a second approach, the authors access the domain grid in texture memory. According their results this second approach gives better results, but it are limited to single precision. When compare the shared memory approach with single and double precision, the double precision simulation was about two or three times slower than the corresponding single precision simulation. They also find that for models with 4, 8, and 19 variables, the simulation time scales linearly with the number of variables. For models with 2 and 4 variables and a 2D domain grid with 2^{18} points the simulation runs in time scales close to real-time. All experiments are executed in a GPU NVIDIA Tesla C2070.

In [15] Rocha et al. (2011) implement and evaluate 2D simulations of cardiac electrical dynamics in a GPU using two different cell models: the Luo-Rudy cell model and the ten Tusscher-Panfilov cell model. The simulations performances were evaluated in a GPU NVIDIA GTX 280. In their GPU implementation a single kernel is launched to solve the reaction term and another kernel to solve the diffusion term (see Section 2). Both kernels were launched with 256 threads per block in all experiments. For the simulations, all model variables were stored in a linear memory structure. Their one-dimensional (1D) array of variables have $M * N_{eq}$ positions where M is the number of cells in the domain and N_{eq} is the number of equations, or

state variables, in the model. According with the author, by using this strategy they avoid extra memory transactions between CPU and GPU and were able to reduce the non-coalescing memory access. To make a discrete representation of the domain, the author use the finite element method (FEM). They evaluated the performance of the simulations when storing arrays at two different data structures: the compressed sparse row (CSR) and the ELLPACK format [18]. ELLPACK uses two matrices to store data, being each one stored in memory as a 1D array in a column-major order. The solution of the diffusion term in Equation 1 involves to solve a linear system. The authors use the preconditioned conjugate gradient method (PCG) to solve the linear system, and to accelerate all matrix vector multiplications involved ($y = Ax$). Besides that, they used texture cache to access the vectors x during the multiplication. The authors implementation in GPU takes 500 msec for simulating a FEM discretization with 410,881 nodes, 409,600 elements and 3,690,241 non-zero entries in the sparse matrix: 648.78 sec, using Luo-Rudy cell model; 8394.53 sec, using ten Tusscher-Panfilov cell model.

In [13], Nimmaggada et al. describes a three-dimensional (3D) cardiac electrical dynamics simulations with GPUs. The authors use the ten Tusscher-Panfilov cell model. They test three searching strategies for a better occupancy of the GPU. The first strategy uses 4 kernels to calculate a single time step of the numerical method; The second uses 5 kernels and the third uses 6 kernels. The authors points that the third solution had better performance. They evaluate the performance with different blocks configurations, finding that 68 threads per block yields the best result. They also use a linearization form of the domain grid points to store all necessary data in GPU global memory. By using this linear array as their data structure instead of a 3D array they intend to reduce non-coalescent access of the memory by enabling threads within the same warp access neighbour data in memory. They also implement a multi-GPU solver for the reaction-diffusion equation 1. When using multi-GPUs the inter-GPU data dependency poses a major issue, since splitting the domain creates boundaries that must be shared among different GPUs. To address this dependency, the authors evaluated various data sharing techniques. They choose to copy only the boundary cells back to the CPU after each Jacobi iteration. Then the CPU update voltage in each boundary cell and transfer it back to their respective GPUs. The authors results points that the solution of diffusion term in equation 1 is the most expensive part taking 62% of the execution time. Their GPU/CUDA implementation in a single GPU NVIDIA Tesla C1060 takes 6800 seconds to simulate 350 msec of real time. The multi-GPU im-

plementations takes, with 4 Tesla C1060, 1850 seconds to simulate the same 350 msec in a domain with 2^{24} cells.

Xia et al. discuss, in [20], parallel strategies and their achieved performances when applied to process a cardiac electrical dynamics simulation on a GPU. They strategy is based on the monodomain 3D sheep atrial model [5]. The authors use approaches to reduce and optimize memory access patterns, in order to optmize this bottleneck. Their first step consists on analyzing the model, searching for variables and parameters that need not be stored in memory. They find that 40 variables need to be stores in memory and other 20 variables can be calculated in each time step. Their second step consists on storing all variables in a series of 1D arrays to reduce/eliminate non-coalescent memory accesses. However, due to the electrical heterogeneity of cells, some of them could represent empty caves and they will not participate in the computation. This can cause some waste of resources with one or more threads in a warp that stays idle. According with the authors, they solve this issue indexing threads in such way that no thread will try to access data in a position of an empty cell. The author also states that thread structure per thread block is different in the solution of reaction and diffusion terms of Equation 1. To solve the reaction term they use a 1D thread structure per block, while to solve the diffusion term they use a 3D thread structure. According with the authors, the 3D thread structure in diffusion term solution is more convenient because the computation of a cell is related to 26 adjacent cells. Is this case, a mapping table from 1 dimension to 3 dimensions needs to be constructed. With all this strategies, their GPU implementation takes $1,687$ sec to simulate 0.6 sec in a grid with approximately 7M points. The simulations were performed in GPU NVIDIDA K40.

Campos et. al present in [6] an implementation of the lattice Boltzmann method [7, 3] (LBM) for simulating cardiac electrical activity in a 3D discrete domain. They implemented the method for CPU and GPU. In both implementations they used a sparse data structure [3] to make more efficient memory utilization when dealing with irregular geometries. They stored each group of homogeneous points in a different 1D array and create an additional data structure (connectivity matrix) to store the list of all positions, within each one of the 1D arrays, of its neighboring points. The numerical solution of the system of ODEs was performed using the Rush-Larsen method [17]. With the optimizations, their GPU implementation achieved speedups between 230 and 480 for simulations. The authors evaluated simulations using three cell models with different levels of complexity: the Luo-Rudy model, the ten Tusscher-Panfilov model and the Mahajan-

Shiferaw model [11]. Depending of the complexity of the cell model and the number of points in the domain the GPU code takes from 3 to 1520 seconds to simulate 1s of electrical activity of the heart. For a regular domain grid with 224^3 nodes and using thread blocks with 256 threads and single precision, the simulation with the Luo-Rudy model takes 519 sec, while with the ten Tusscher-Panfilov model takes $1,520$ sec. For a irregular (anisotropic) domain grid with $3,760,560$ nodes, the Mahajan-Shiferaw model takes 635 sec. All simulations were performed in a GPU NVIDIA Tesla M2050.

4. Discussion

The main target of the presented works are related to achieving computation high performance using efficient GPU implementation. Table 1 synthesizes the main results, remarking relevant characteristics of similar experiments conduced by the authors in their papers.

Author	Bartocci et al.	Rocha et al.	Nimmagadda et al.	Xia et al.	Campos et al.
Ref	[1]	[15]	[13]	[20]	[6]
Cell model	tTP	tTP	tTP	3D Sheep atria model	tTP
Num. of state variables	19	19	19	40	19
Num. of points in the domain	$4,194,304$	$3,690,241$	$16,777,216$	$7,048,812$	$11,239,424$
precision	single	single	not informed	double	single
Real-time simulated	1.00 sec	0.50 sec	0.35 sec	0.60 sec	1.00 sec
Time spent in the simulation	$\sim$ 500 sec	$8,395$ sec	$1,850$ sec	$1,687$ sec	$1,520$ sec
GPU (NVIDIA)	Tesla C2070	GTX 280	Tesla C1060	K40	Tesla M2050
Num. of GPUs used in simulation	1	1	4	1	1

Table 1 - Efficiency comparison of most similar experiments presented in the literature The model tTP is the ten Tusscher-Panfilov cell model. It was adopted in almost all implementations, except in [20] that used a sheep atria model described in [5].

Models capable to explore all resources of the GPU can be hard to achieve. Two important challenges in GPU programming are improvements

on efficient memory access and thread distribution. They are key factors to fully exploit the computational power of a GPU [14].

GPUs have a memory hierarchical composed by main memory, two levels of cache, shared and texture memories. They present different storage capacity, lifetime, scope and latency. The memory pattern access and correct use of memory hierarchical affect the performance of CUDA application. We summarize the characteristics of memories in according to: *i*) **Global Memory** or **DRAM**, which is the slowest memory and with lifetime defined by CPU which is responsible for its management. It resides off chip and all threads can access any position of global memory. Global memory can storage few bytes or many gigabytes relying on its model. *ii*) **Cache L2** is shared by all GPU processors, having a smaller latency and size than the global memory. It is also off chip and its lifetime is given by processing kernels. After ending up a kernel, the Cache L2 data are lost. *iii*) **Shared memory** has access time close to cores register, it is on-chip and its lifetime is given by elapsed time to processing a block on SM. Shared memory provides a communication way among threads of the same block. *iv*) **Cache L1** is only used by **constant memory** and **Texture memory** in the newer architectures. The former works with a low latency and allows all threads to read data in this memory. This memory is cached and its lifetime is the same of Cache L2. Texture memory has the scope and lifetime similar to the global memory, but with smaller latency due its read only pattern. However, GPUs constrain the size of allocated the texture memory, being more commont to be used at the graphic context.

As the global memory takes several machine cycles to transfer data to the SMs, the works presented in this survey try to deal with this problem with some kind of linear memory structure. The linear format was chosen because allows threads within the same warp process data stored in adjacent positions on global memory, enabling the hardware to coalesce groups of reads or writes of multiple data items into one operation. This approach reduces the number of memory operations allowing better performances, as demonstrated by [20]. However, data structures at the global memory level are not the only memory optimization evaluated; [1] demonstrate that by using different memory hierarchies it is also possible to improve the performance.

Another major aspect to achieve high performance with GPUs are the thread/block structure. A carefully planed thread/block structure can avoid thread divergence and optimize GPU occupancy hiding memory transactions latency by switching between warps performing computational tasks and warps performing memory operations. [1] propose an specific thread/block

structure to solve the adjacent cells data dependency when computing the diffusion term solution. [6] and [20] show how their implementations performs with different number of threads per block.

As all the authors agree, the high number of state variables and parameters that need to be computed at each time step in a cardiac electrical dynamics simulation pose a great challenge when developing a parallel simulator to GPU. We see that in general the authors follow the path of using some 1D arrays to store data in GPU global memory. Some of them ([1] and [16]) use texture memory to make their simulations faster. Indeed, the intelligent usage of GPU memory hierarchy can enable achieve higher performances. As a reference, Mei et al. published, in [12], a nice benchmark analysis of GPU memory hierarchies that can helps future works.

Another relevant issue, as address by [1] and [20], is the data dependency of adjacent cells when computing the numerical solution of the diffusion term. For example, in [1] to update a cell the numerical method demands data from another 4 adjacent cells. This data dependency elevates the number of memory operations and it is a major source of non-coalescent memory access and thread divergence. [1] try to reduce this last two problems by launching extra threads per block exclusively to read data from block neighbor cells. This solution can reduce the divergence problem, but without a proper memory structure the non-coalescent memory access problem remains.

5. Conclusion

All the works presented here try to optimize memory access to make their simulation faster. The cited authors try 1D arrays to make threads within the same warp process data storage in adjacent memory positions. However, this should fail to threads in the block edge when solving the diffusion term. We believe that a data structure that store all data needed by a thread block in adjacent positions combined with a carefully planed thread index make possible avoid non-coalesced memory access, thread divergence and resources waste, when process cells in block edge. Also, some improvements in the new GPU architecture can help to accelerate global memory access. According NVIDIA, the 3D memory in Pascal GPUs have higher throughput and efficiency, letting more quick access to memory.

We hope this survey can help future developers to improve and invest some effort in the task of accelerate cardiac dynamics simulations toward real time. This is an important field that can help physicians and researchers to better understand cardiac diseases and develop efficient treatments.

References

[1] Bartocci, E., et al. Toward real-time simulation of cardiac dynamics. In *Proceedings of the 9th International Conference on Computational Methods in Systems Biology* (New York, NY, USA, 2011), CMSB '11, ACM, pp. 103–112.

[2] Beeler, G. W., and Reuter, H. Reconstruction of the action potential of ventricular myocardial fibres. *the Journal of Physiology 268*, 1 (1977), 177–210.

[3] Bernaschi, M., et al. A flexible high-performance lattice boltzmann gpu code for the simulations of fluid flows in complex geometries. *Concurrency and Computation: Practice and Experience 22*, 1 (2010), 1–14.

[4] Bueno-Orovio, A., Cherry, E. M., and Fenton, F. H. Minimal model for human ventricular action potentials in tissue. *Journal of Theoretical Biology 253*, 3 (2008), 544–560.

[5] Butters, T. D., et al. A novel computational sheep atria model for the study of atrial fibrillation. *Interface Focus 3*, 2 (2013).

[6] Campos, J. O., et al. Lattice boltzmann method for parallel simulations of cardiac electrophysiology using {GPUs}. *Journal of Computational and Applied Mathematics 295* (2016), 70–82. {VIII} Pan-American Workshop in Applied and Computational Mathematics.

[7] Chen, S., and Doolen, G. D. Lattice boltzmann method for fluid flows. *Annual Review of Fluid Mechanics 30*, 1 (1998), 329–364.

[8] Clayton, R., and Panfilov, A. A guide to modelling cardiac electrical activity in anatomically detailed ventricles. *Progress in Biophysics and Molecular Biology 96*, 1-3 (2008), 19–43. Cardiovascular Physiome.

[9] Iyer, V., Mazhari, R., and Winslow, R. L. A computational model of the human left-ventricular epicardial myocyte. *Biophysical Journal 87*, 3 (2004), 1507–1525.

[10] Karma, A. Spiral breakup in model equations of action potential propagation in cardiac tissue. *Phys. Rev. Lett. 71* (Aug 1993), 1103–1106.

[11] Mahajan, A., et al. A rabbit ventricular action potential model replicating cardiac dynamics at rapid heart rates. *Biophysical Journal 94*, 2 (2008), 392 – 410.

[12] Mei, X., et al. *Benchmarking the Memory Hierarchy of Modern GPUs*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 144–156.

[13] Nimmagadda, V. K., et al. Cardiac simulation on multi-gpu platform. *The Journal of Supercomputing 59*, 3 (2012), 1360–1378.

[14] NVIDIA. CUDA C programming guide. `http://docs.nvidia.com/cuda/cuda-c-programming-guide/`, 2016. [Online; accessed 25-September-2016].

[15] Rocha, B. M., et al. Accelerating cardiac excitation spread simulations using graphics processing units. *Concurrency and Computation: Practice and Experience 23*, 7 (2011), 708–720.

[16] Rocha, B. M., et al. A macro finite-element formulation for cardiac electrophysiology simulations using hybrid unstructured grids. *IEEE Transactions on Biomedical Engineering 58*, 4 (April 2011), 1055–1065.

[17] Rush, S., and Larsen, H. A practical algorithm for solving dynamic membrane equations. *IEEE Transactions on Biomedical Engineering BME-25*, 4 (July 1978), 389–392.

[18] Saad, Y. *Iterative methods for sparse linear systems*. Siam, 2003.

[19] ten Tusscher, K. H. W. J., and Panfilov, A. V. Alternans and spiral breakup in a human ventricular tissue model. *American Journal of Physiology - Heart and Circulatory Physiology 291*, 3 (2006), H1088–H1100.

[20] Xia, Y., Wang, K., and Zhang, H. Parallel optimization of 3d cardiac electrophysiological model using gpu. *Computational and Mathematical Methods in Medicine 2015* (2015), 1–10.