Multi-scale Cardiac Electrophysiological Simulation: A GPU-based System by Computational Steering

Shanzhuo Zhang, Songjun Xie, Kuanquan Wang, Yongfeng Yuan
School of Computer Science and Technology,
Harbin Institute of Technology,
Harbin, China
E-mail: shanzhuo.zhang@gmail.com, wangkq@hit.edu.cn

***Abstract*—A cardiac electrophysiological simulation tool is essential for the study of the modern virtual heart. However, there is not yet an ideal tool that can satisfy the requirements of rapidity, flexibility and interactivity. In this paper, we propose a simulation system to cope with this problem, which is (i) based on GPU for parallel computing, and very fast; (ii) flexible to embed various cell models and simulate multi-scale cardiac tissue; (iii) interactive and easy to use by computational steering. We designed a particular simulation case to show the convenience of computational steering and another test to compare the speed of our system to the traditional CPU-based method. These results suggest that our simulation system can be an ideal tool for the research of the virtual heart.**

Keywords—computational steering; multi-scale cardiac simulation; cardiac electrophysiology

I. INTRODUCTION

The heart is a vital organ in a human body, and the heart disease has been one of the four most serious threats to the human health. However, due to the complexity of the human heart, the research of clinical diagnosis, treatment and drug design in cardiology costs very long time and lots of money. In recent years, the development of virtual heart [1,2] has provided a new practical solution for these problems. Integrating computing physiology, mathematical modeling, and virtual reality technologies, heart modeling and simulation have gained widespread attentions and become increasingly important in the research of modern cardiology.

The study of virtual heart needs complex simulation. When using traditional simulation methods, we often have to face such an frustrating situation: after a series of time-consuming processes of complex heart modeling, partial differential equation discretization of the mathematical model, and defining the initial and boundary conditions, we finally find the results are unreasonable or meaningless, then we need to revise parameters and redo the whole process. In addition, sometimes we not only want to get the final results of the modeling, but also want to be able to observe its intermediate states, helping us understand the mechanisms of cardiac signaling and interaction. But the traditional simulation platform cannot provide this functionality. Thus, we need an effective tool for the research of virtual heart urgently.

Computational steering can just solve the problems above. It is a concept appears from the development of scientific visualization research, it can be defined as the interactive control over a computational process during execution [3]. Broadly speaking, computational steering includes all computational patterns that are characterized by intersections between manipulations and computation, such as debugging of source codes, interactive scientific visualization, and computer games. In the field of scientific computing, computational steering is used in medical, quantum-mechanical, fluid dynamical, and other simulations which need large-scale and interactive computation. Scientific investigation processes rely heavily on answers to a range of "What if?" questions. Computational steering allows back traces in the computation, so these questions can be answered more efficiently.

In the cardiac electrophysiological simulation area, some systems with computational steering have been developed, such as the gViz [4,5] and the SCIRun/BioPSE [6]. gViz has been used in multi-scale simulation of heart tissue [7]. SCIRun/BioPSE has been developed for a long time, which is more mature, and applied to solve the forward and inverse problems of the surface electrocardiograph [8] (ECG) and cardiac electrical activity simulation [9]. However, these systems were developed nearly ten years ago, and are not fast enough because of their CPU-based characteristic.

In our opinion, an ideal research tool for cardiology must have several features. The basic one the is the high performance that can satisfy the requirement of three-dimensional (3D) simulation. Second, the tool must be flexible enough to combine various cell models and be applied to multiple physical scales. Finally, the tool has to be a framework that is functionally suitable for iterative and interactive computation. Aiming at these demands, we developed a multi-scale cardiac electrophysiological simulation system, the main features of which are as follows:

- It's fast because of its GPU-based parallel computing characteristic.
- It's flexible in functionality, means that it can be combined with various cell models and applied to 1D, 2D, and 3D simulations.

- It is scalable, we can improve the computing speed and simulation resolution by adding hardware.
- It has the feature of computational steering.

II. Framework Details

A. Framework overview

For the aim of achieving computational steering, we used an architecture borrowed from van Liere et al [10]. The whole architecture is shown in Fig. 1, we adding a Data Manager as middleware between the traditional Client/Server architecture. The purpose of this is threefold. First, This architecture allows us to separate the complex and simple parts of tasks. So we can assign the highly time-consuming simulation tasks to a high-performance computer (the Server which may have multiple video cards, big RAM and strong CPUs), and other light-weight tasks to a normal desktop (the Client). Second, through a Data Manager, we can easily handle the data transmission between the Client and Server, since the cardiac tissue specification and simulation results can be very large. Finally, the Data Manage can also simplify the computational steering logic by using a kind of specific directory structure. More details will be discussed in the following sections.

The workflow of our system is roughly as follows:

- Some preparatory work is done. The cardiac tissue specification file is written to the Data Manager, and dynamic link libraries of cell models are added to the Computational Module.
- The Client creates a new simulation project, and adds commands to the Task Queue, then starts simulation.
- By observing the results of the simulation, users can determine whether the results meet the requirements, then choose to change parameters or backtrack by the Steering Controller. By doing this repeatedly to get satisfied final results.

B. Computational module

The major difference between our server-side computing module and traditional simulation module is the presence of a pre-processing and a post-processing phase. First, in order to achieve GPU-based parallel computing and the decoupling of the computational process and cell models, we add a pre-processing step before the simulation phase. Second, in order to get computational steering, we need a post-processing phase to dump the results elsewhere. The workflow of the Computational Module is shown in Fig. 2.

1) Pre-processing phase

At the beginning of this phase, the Computational Module needs to read the task specification, which includes the heart tissue specification and the command from the Client, and establish an abstract geometric representation of the tissue in the video memory. In the process of the establishment of this representation, we do not distinguish between dimensions of the input, but construct an abstract

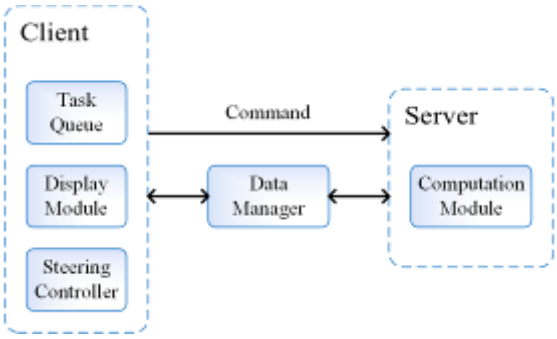


Fig. 1. The Client/Server Architecture of our system. The Client assumes the managing and low-costing work, and the Server does highly time-consuming computation. The Data Manager can be deloyed as a middleware or integrated in the Server.

data structure in accordance with the definition of cell coordinates, allowing us to achieve multi-scale simulation easily.

Then we load cell models into the Computational Module. A cell model is a pre-compiled dynamic link library written in C++, and can be dynamically loaded during the run. A cell model should include intermediate variable definitions and their data types which are used during the simulation, the cell state initialization function, and functions translated from differential equations which representing the ion current updating. It should be noted that, by using this dynamically loading method, we can load any number of types of cell models into the system, and we gain the flexibility to change models between different definitions of the cardiac tissue in the simulation process.

In the third step, we need to allocate computing resources provided by GPUs. As mentioned earlier, our system is scalable, which means the system is able to cope with the increased number of GPUs. We insert a dynamic planning algorithm in the pre-processing to calculate the amount of computation that each GPU should bear. For example, if there are four types of cells in the ventricular tissue and 2 GPUs, and the number of each type of cells is 3000, 2000, 1000, and 500, respectively. After the allocation, we want to assign 3000+500 cells to GPU1, and 2000+1000 cells to GPU2, thus guarantee the difference of the amount of computation among GPUs will not be too much, and ensure the efficiency of parallel computing.

2) Simulation phase

In this phase, the Computational Module has got all the prerequisites, can begin the simulation. The procedure of

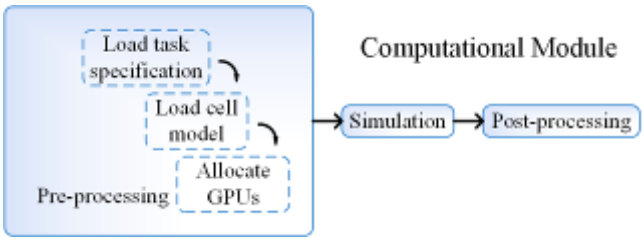


Fig. 2. The workflow of the Computational Module.

simulation follows the general cardiac electrophysiological simulation steps, mainly includes the iterative calculation over time. Each time step of the calculation is divided into two steps.

First, according to the current state of each cell (i.e., the membrane potential and other variables of a cell, the initial state if it is the first iteration), solve the differential equations describing the electrical behavior of the cell, and update each state of it. A typical cell model includes the equations describing the electrical activities of ion channels on the membrane, intracellular Ca^{2+} transient, etc. The change in membrane potential is mainly generated by the current through the membrane. A typical differential equation of a K^+ current is as (1).

$$\begin{cases} I = Gx^2(V-E) \\ \dfrac{dx}{dt} = \dfrac{x_\infty - x}{\tau} \end{cases} \tag{1}$$

Where I is the K^+ current, G is the channel conductivity, t is time, V is the membrane potential, E is the reversal potential of K^+, x_∞ and τ are functions of V.

Second, according to the intercellular coupling equation, calculate the updated voltage of each cell at current time step. The intercellular coupling equation is as (2).

$$\frac{\partial V_m}{\partial t} = \nabla \cdot (\mathrm{D}\nabla V_m) - \frac{I_{ion}}{C_m} \tag{2}$$

Where V_m is the membrane potential, t is time, C_m is membrane capacitance, ∇ is a gradient operator, D is the diffusion tensor, I_{ion} is the sum of all transmembrane ion currents.

Membrane potentials of all cells will be updated each time step, and after a pre-defined number of iterations, the data representing these potentials will be transmitted from the video memory to the RAM. Moreover, after each iteration, we also need to check the command from the Client to determine whether to make a snapshot, if so, we will have to dump all the intermediate variables of the calculation into the RAM then send to the Data Manager.

3) Finishing phase

At this phase, the system produces the final simulation results. All the snapshot files and results of all iterations that have been transferred from video memory to the RAM will be reorganized, then sent to the data manager, and written to the disk by the Data Manager.

C. Data Manager and Steering Controller

Data Manager is the key component to achieve the property of computational steering. Conceptually, back-tracking of the calculation process will form a computational trace like a tree, if we see the initial state as the root of the tree, creating a snapshot will generate a child below the root, if the snapshot is then abandoned and we restart the simulation from the beginning, generate another snapshot, then this new snapshot will be a sibling of the previous one.

We use a special directory structure to manage the trace. Creating a new project will generate a corresponding project directory, where we assume its name is ProjectRoot. Under the ProjectRoot, there are the tissue specification file and a directory called Cases holds the snapshots and result files. Each time we start a simulation, there will be a new folder created, including this simulation's parameters, all snapshots and simulation results. And in the Steering Controller, we show these directories through a tree structure, and can switch between different branches.

D. Task Queue

The Task Queue is an extension to the functionality of the system. Given that many simulation tasks require a lot of computing time, through the establishment of a Task Queue at the Client, users can run their simulation tasks in sequence and improve the equipment utilization. Task Queue and Server coordinate the task scheduling by a pre-defined TCP protocol.

E. Visualization Module

The Client provides a graphic user interface (GUI), including a series of control buttons and a display window for visualization. The whole interface is written with the QT UI framework, and the display window is written by the open source visualization toolkit called VTK. The actual window is shown in Fig. 3.

III. Simulation Results and Discussion

All the simulation results below were achieved by using a computer with an Intel Core i7-3930K @ 3.2GHz, 62.9GB RAM, and a NVIDIA GTX TITAN Z video card with 5760 stream processors. The operating system is Linux Mint 17.1 64-bit and the version of CUDA is 6.5.

A. Simulation and Computational Steering Results

Fig. 4 shows the 3D simulation model of the human left and right ventricles. The resolution of volume data is 328*328*428. The actual number of cells is about 7 million. In Fig. 4, we exhibit the procedure of how to make an attempt to get a split wave. Fig. 4(a) is the visualization of ventricular surface potential at 476ms of a simulation, and we take a snapshot here. The first attempt, we added a

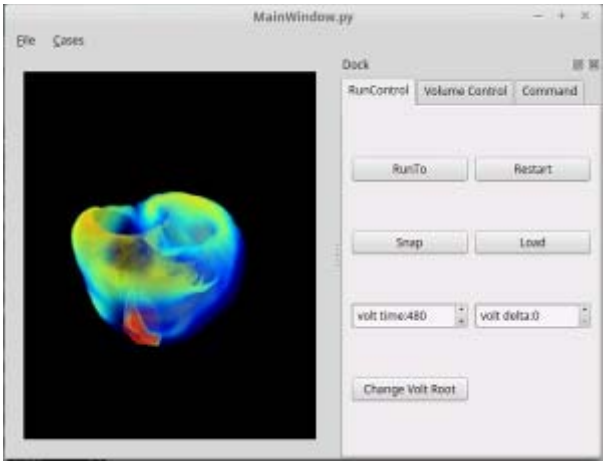
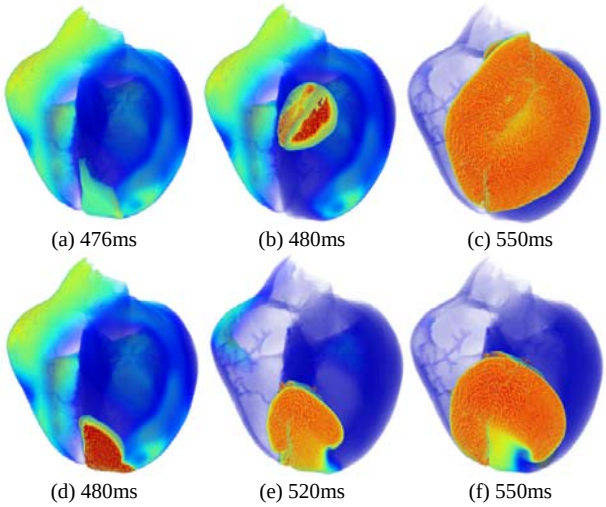


Fig. 3. A screen shot of the GUI of the Client.



(a) 476ms (b) 480ms (c) 550ms

(d) 480ms (e) 520ms (f) 550ms

Fig. 4. Trying to get a split wave by computational steering. Colors in the graph represent the different potentials. Red means a high potential and blue otherwise. See the text for details.

stimulus on the side as shown in Fig. 4(b), the red area, at 480ms. We continued the simulation to 550ms, and found no wave split. Then we backtracked to the snapshot point 476ms, tried to stimulate at the red area as shown in Fig. 4(d). The excitation wave gradually split from 520ms to 550ms in Fig. 4(e) and Fig. 4(f). We managed to save a lot of simulation time, because we didn't need to start from the beginning at the second attempt. There are 476 iterations from 1ms to 476ms, thus, we saved 2380 seconds (if one iteration costs 5 seconds), about 40 minutes.

B. *Simulation Speed Comparison Between GPU-based and CPU-based Computation*

The simulation speed depends primarily on the number of cells in the tissue. In our speed comparison, we used a two-dimensional rectangular tissue for the calculation. We compared the time consumption of the CPU-based method to our GPU-based method. As shown in TABLE I. We changed the tissue size from 250 thousand to 8 million cells, and recorded the simulation time cost for simulating 1ms by the two methods. As shown in the table, the GPU-based method only took one out of dozens, even hundreds time than the CPU-based method.

TABLE I. SPEED COMPARISON BETWEEN CPU- AND GPU-BASED METHODS

	Size of the tissue					
	1k*250	***1k*500***	***1k*1k***	***1k*2k***	***1k*4k***	***1k*8k***
CPU	29.7s	58.2s	117s	269s	536s	1073s
GPU	0.4s	0.4s	1.1s	1.5s	2.7s	5.4s

C. *Discussion*

Compared with existing systems SCIRun or gViz, the biggest advantage of our system is the fast speed of computation. A CPU-based method costs more than 10 minutes to simulate only 1ms result, if we want to get the results of one cardiac cycle of 1000ms, the total simulation time will be tens of hours. On the contrary, our GPU-based method will only cost an hour, which greatly improves the efficiency. On the other hand, our system is functionally flexible, it allows the user to embed a variety of cell types, and can be used for 1D, 2D, 3D simulation; our system is also scalable to extend, such as adding more video cards to improve the performance. Finally, we achieve a computational steering simulation framework that has the ability to backtrack or modify parameters during simulation, which can also help researchers save a lot of time.

IV. CONCLUSIONS AND FUTURE WORK

The simulation tool plays a very important role in the virtual heart study. By using a proper tool, researchers can save lots of time in the simulation, result visualization and analyses. In this paper, we present a multi-scale cardiac electrophysiological simulation system which is fast, flexible, and easy to use. This system is an ideal research tool for cardiologists, and has been used practically and produced many valuable results.

In the future, we plan to further strengthen the performance of our system. Now we use the VTK visualization toolkit for the visualization in the Client. Next, we can adopt a real-time visualization method, which is an unpublished work of our team, to improve the interactivity of the system.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61179009, No. 61173086.

References

[1] N. A. Trayanova, "Your personal virtual heart," *Spectrum, IEEE*, vol. 51, no. 11, pp. 34-59, 2014.

[2] D. Noble, "From the Hodgkin–Huxley axon to the virtual heart," *J Physiol*, vol. 580, no. 1, pp. 15-22, 2007.

[3] J. D. Mulder, J. J. van Wijk, and R. van Liere, "A survey of computational steering environments," *Future Gener Comp Sy*, vol. 15, no. 1, pp. 119-129, April, 1999.

[4] K. Brodlie, J. Wood, D. Duce, and M. Sagar, "gviz: Visualization and computational steering on the grid," in Proceedings of the UK e-Science All Hands Meeting, 2004, pp. 54-60.

[5] J. Wood, K. Brodlie, and J. Walton, "gViz–visualization and steering for the Grid," in Proceedings of e-Science All Hands Meeting, 2003.

[6] S. G. Parker, and C. R. Johnson, "SCIRun: a scientific programming environment for computational steering," in Proceedings of the 1995 ACM/IEEE conference on Supercomputing, San Diego, California, USA, 1995, pp. 52.

[7] O. Aslanidi, K. Brodlie, R. Clayton, J. Handley, A. Holden, and J. Wood, "Remote visualization and computational steering of cardiac virtual tissues using gViz," in Proceedings of the 4th UK e-Science All Hands Meeting (AHM'05), 2005.

[8] R. S. MacLeod, D. M. Weinstein, J. Davison de St Germain, D. H. Brooks, C. R. Johnson, and S. G. Parker, "SCIRun/BioPSE: integrated problem solving environment for bioelectric field problems and visualization," in IEEE International Symposium on Biomedical Imaging: Nano to Macro, 2004, pp. 640-643.

[9] D. M. Weinstein, J. V. Tranquillo, C. S. Henriquez, and C. R. Johnson, "BioPSE Case Study: Modeling, Simulation, and Visualization of Three Dimensional Mouse Heart Propagation," *Int. J. Bioelectromagnetism*, vol. 5, no. 1, pp. 316-317, 2003.

[10] R. van Liere, J. D. Mulder, and J. J. van Wijk, "Computational steering," *Future Gener Comp Sy*, vol. 12, no. 5, pp. 441-450, April, 1997.