# Parallel Computing Simulation of Electrical Excitation and Conduction in the 3D Human Heart

Di Yu, Dongping Du, Hui Yang*, Yicheng Tu

***Abstract*—A correctly beating heart is important to ensure adequate circulation of blood throughout the body. Normal heart rhythm is produced by the orchestrated conduction of electrical signals throughout the heart. Cardiac electrical activity is the resulted function of a series of complex biochemical-mechanical reactions, which involves transportation and bio-distribution of ionic flows through a variety of biological ion channels. Cardiac arrhythmias are caused by the direct alteration of ion channel activity that results in changes in the AP waveform. In this work, we developed a whole-heart simulation model with the use of massive parallel computing with GPGPU and OpenGL. The simulation algorithm was implemented under several different versions for the purpose of comparisons, including one conventional CPU version and two GPU versions based on Nvidia CUDA platform. OpenGL was utilized for the visualization / interaction platform because it is open source, light weight and universally supported by various operating systems. The experimental results show that the GPU-based simulation outperforms the conventional CPU-based approach and significantly improves the speed of simulation. By adopting modern computer architecture, this present investigation enables real-time simulation and visualization of electrical excitation and conduction in the large and complicated 3D geometry of a real-world human heart.**

***Keywords*:** cardiac electrical dynamics, parallel computing, cellular automata, whole-heart modeling.

## I. INTRODUCTION

Computer simulation of human heart is receiving increasing attention because it empowers scientists to advance cardiac research and battle against heart disease. The heart beat is the result of a series of complex biochemical-mechanical reactions, which involves transportation and concentration distribution of a dozen of ionic and molecular species in heart tissue through a various different biological channels [1, 2]. These biochemical movements of cardiac ions (i.e., $Na^+$, $Ca^{2+}$, $K^+$) create a local electrical potential variation, which in turn cause the contraction of heart cells. The 4 phases of electrical potential of cardiac myocytes (i.e., action potential – AP) are as shown in the Figure 1.

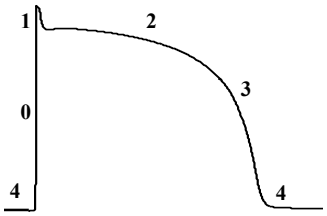

Fig. 1. Four phases in the cardiac action potential [1, 2]

This work is supported in part by the National Science Foundation (CMMI-1266331 and IOS-1146882).

***Corresponding Author***: Hui Yang is an assistant professor with the Department of Industrial and Management Systems Engineering at the University of South Florida, Tampa, FL, 33620 USA. (e-mail: huiyang@usf.edu, voice: (813) 974-5579; Fax: (813) 974-5953).

This electrical potential fluctuation will propagate through every cell of the entire heart to complete one cycle of heart beat. An ideal comprehensive simulation model will take into consideration of every aspect of this complicated biochemical and mechanical interaction and simulate the process on a full scale of 3D model. However, because of the complexity of cardiac system, the unknown/unclear aspects of many details involved in this biological process, and the limitation of modern computational power; currently available simulation models usually simplify certain aspects of the computation and generate an approximate result of the real-world cardiac process. If the bio-chemical interaction is carefully monitored and various ionic and molecular species are simulated, the model will only be able to simulate the activities of 2D tissue. With currently available (2012) computational power, several models were developed under such scheme and they are all only capable of performing calculation for simplified geometry such as 1D or 2D tissues, as opposed to real-time whole-heart modeling [3].

Our approach follows a different methodology, which focus on the macro-scale behavior of the entire heart. To do such a large-scale simulation, we greatly simplify the micro-scale simulation by only computing the propagation of the electrical potential and ignore interactions of any other bio-chemical species involved in this biological process. This approach allowed us to observe and analyze the excitation pattern of a whole heart in real-time condition, which has not been well-established before. To facilitate this demanding task, a massive parallel computation model is developed based on the modern GPGPU (i.e., General-purpose computing on graphics processing units) architecture.

The organization of this paper is as follows: Section II presents research methodology of the model. The algorithm and implementation are described in Section III in details. In Section IV, we compared the performance of our algorithm under conventional single CPU architecture and under modern GPGPU architecture. Section V concludes this present study.

## II. RESEARCH METHODOLOGY

### *A. Cellular automata model*

Two major methodologies coexist in the area of cardiac modeling. The first is based on a macro/minimal approach, such as cellular automata model, fitzHugh-Nagumo model [4], and Barkley Model [5]. The second approach is based on micro/maximal models, e.g., Hodgkin-Huxley model, Luo-Rudy model, Rasmusson model and Nygren-Lindblad model [6, 7]. The macro/minimal models simplify simulation of ion physics, but they are easier to be incorporated into a

large-scale system and they can be computed much faster. This present study is based on the first type of the simulation approaches. We assumed heart tissue cells are uniformly distributed excitable media and the excitation and propagation of cardiac electrical potentials are governed by the potential of the directly adjacent neighboring cells. The cell will stay in excited condition for an experimentally predetermined period, and then drop back to resting condition and waiting for the next excitation. Based on clinical experimental data of heart tissue and only considering the measured potential of the cells, we free ourselves from the time consuming calculation of the underlying bio-chemical process, and avoid the uncertainties and inaccuracy introduced by unreliable and /or unavailable constants and parameters in the biological transportation model of ions and molecular species in various channels of heart tissue. With the modern GPGPU architecture, we are able to develop a fast algorithm that can handle large and complicated 3D geometry of a real human heart.

*B. GPGPU acceleration*

Our model can be categorized as one type of cellular automata models. Cellular automata models have been widely used in computational simulation research on biological processes, excitable media modeling, disease and disaster propagation, macroeconomic fluctuations in human society, and Conway's mathematical game of life [8]. In this project, to achieve real-time simulation, we employed modern GPGPU computing architecture to accelerate the massive computation in the simulation. The cellular automata model is suitable for deployment on a parallel computing framework. Each unit in such a system is only determined by the state of its neighboring units. To determine the overall state of the system, each unit's state can be calculated independently. Such a problem is a typical one among many computation bound problems that are perfectly suitable for harnessing the computational capabilities of GPGPU processors [9]. The modern GPGPU devices are increasingly gaining popularity in the areas of scientific research as well as industrial applications because of the better affordability and cost effectiveness.

*C. Algorithm*

Because we only consider the potential of the cell and the potential can be determined by time alone in our simplified model, we can model the excitation of a single cell if we keep track of the duration when the cell has entered excitation state. For example, if a cell is excited at time zero (t=0), it will stay in the excited state for 200 ms (t=200) and then returns to resting state. A cell can only be excited when a neighboring cell is in the excited state. It would be ideal if each cell in the heart can be represented by one point in the 3D heart model; however that will generate an overwhelmingly huge 3D model that cannot be handled by current computers. Therefore, a 3D heart model with limited resolution has to be used and the distance between two neighboring points will cover a space between many heart cells. In this work, we used the 3D anatomic heart model digitally reconstructed from a series of 2D images of many parallel cross sections of a real human heart. This distinguishes our model from conventional cellular automata models. Our model need to calculate the distance between any two neighboring points and estimate how much time it will take for the excitation to propagate from one point to the other. The whole algorithm is provided below:

The 3D heart model consists of two files:

1) A list of all the points representing mass of a heart. The information for each point includes the index of this point, and xyz coordinates of this point.
2) A list of all the tetrahedral formed by any 4 conjunction points. Each tetrahedral is represented by index of 4 points.

The data preparation steps:

1) For each point, find out its neighboring points and then generate a linked list of these neighboring points. This is done by traversing the tetrahedral file and adding a link to the linked list of each point if this point is found in current tetrahedral data.
2) Calculate the distance from each point to every neighbor that this point has a conjunction relationship.

After the data preparation step, we then have an array of all the points and each element of the array is a linked list recording every neighboring point and mutual distances. Therefore, each point can be represented by the index of the point→neighboring point 1 (distance), neighboring point 2 (distance), neighboring point 3 (distance), etc. An example of this data structure is shown below:

0→1(8),3(8),4(6.9282),5(6.9282),7(8),8(6.9282),249(6.9282),2692(7.64532),2695(6.9282),2696(7.7316),2698(7.32724),2713(8.57508),2717(5.65685),2718(6.63325),2721(6.9282),2740(5.65685),2741(6.63325),2742(6.63325),2759(4),2760(6.63325),2762(5.65685),2764(5.65685),2767(6.9282),2769(8.30697),2771(5.65685),2772(8.34392),2775(6.9282),2803(6.63325),2808(4),2809(5.65685),2810(5.65685),2824(6.63325),2825(4),2826(5.65685),2830(5.65685),7789(5.65685)total neighbor36

1→0(8),2(8),4(6.9282),5(6.9282),15(8),17(8),18(6.9282),19(6.9282),27(6.9282),37(6.9282),2720(6.63325),2721(6.9282),2762(5.65685),2767(6.9282),2806(6.63325),2810(5.65685),2863(6.63325),2865(5.65685),2922(6.63325),2926(4),2927(5.65685),2928(5.65685),7745(6.63325),7747(5.65685),7748(6.63325),7789(5.65685),7790(6.63325),7792(4),7793(6.63325),7797(5.65685),7801(5.65685)total neighbor31

2→1(8),3(8),4(6.9282),5(6.9282),9(8),11(6.9282),12(6.9282),26(8),27(6.9282),36(6.9282),37(6.9282),38(6.9282),2811(6.63325),2817(6.63325),2821(5.65685),2925(6.63325),2927(5.65685),2931(6.63325),2934(4),2935(5.65685),2936(5.65685),7795(6.63325),7797(5.65685),7798(6.63325),7813(5.65685),7814(6.63325),7816(4),7817(6.63325),7820(5.65685),7823(5.65685)total neighbor30

The computation steps:

1) Make an array of timer for each point. The value of timer indicates the current state of the point. If the timer value is in between 0 and 200, the point is in excited state. If the

timer value is greater than 200, the point is in resting state. If the timer value is negative, the point has an excited neighboring point and will be excited in a later time when timer increases to 0. Initialize all the timer value to a large positive number such as 5000, which indicating the whole heart is at resting condition.

2) Select one particular point, and set its timer value to 0 as making this point the starting of the excitation.
3) For each point that the timer value is 0, look up the neighboring points of this point. We will call this point as starting point, and the neighboring points as ending points. Depends on the timer value, we will do one of the three possible tasks to each ending point's timer:
   a) if the ending point is already excited (timer value is within 0 to 200 range) do nothing
   b) if the ending point is in resting state (timer value > 200), set its timer value to a negative integer, which equals to floor($C*d$), here $C$ is a negative constant that represents the propagation speed of such excitation within heart tissue, and $d$ is the distance between starting point and the ending point, and the floor($C*d$) function takes the maximum integer that's less than $C*d$.
   c) if the ending point's timer already has a negative value, which means some other starting point has already set the timer of this ending point, compare its current timer value to floor($C*d$) and set the timer value to the one with less absolute value. Smaller absolute value of the negative timer indicates that excitation reaches the ending point quicker.
4) Increment every point's timer by 1.
5) Repeat step 3 and 4 and we will observe excitation propagation by visualizing the 3D heart according to the timer value of each point.

### *D. Algorithm Discussion*

Under the framework of our current model, we have specifically considered the following points:

1) Since the model use discrete time to calculate the state of the points, we have to assign integer value to the timer of each point, and thus the floor function is adopted to round off digits after point. The floor function will introduce certain error into the calculated results. However, we do not expecting such error will be significant since we are modeling the process in a very fine resolution of 1ms.
2) The excitation propagation speed constant, $C$, should be determined with experiment and might varies under different situation.
3) The excitation duration from 0 to 200 ms is based on experimental results and might vary from case to case.
4) The location and number of initially excited point(s) can be picked arbitrarily
5) Spiral waves are simulated under certain circumstances. For e.g., excitation time span is too small or multiple excitation starting points are selected. This indicates irregular cardiac arrhythmias in the form of spiral excitation and reentry phenomena in an excitable media.

## III. Materials and Experimental Design

### *A. Implementation Platform:*

We implemented our simulation application on a workstation with a Intel dual core i3-2100 CPU @ 3.10GHz and 16G of DDR3 memory. The graphic card is an Nvidia Telsa C2075 with 6GB global memory, for fast rendering and simulation computation. The operating system is 64bit Window 7. The simulation application consists of two major components, the simulation component and the visualization component. OpenGL was selected as the visualization / interaction platform because it is open source, light weight and universally supported by various operating systems. The application is developed with C++ under Microsoft Visual Studio 2010. The source code of the application should be relatively easy to transfer to other operating system since no special dependency to Microsoft VS IDE exists. OpenGL provided the basic rendering loop iteration and GUI mechanism. The simulation algorithm of application was implemented under several different versions, one conventional CPU version and several GPU versions based on NVidia CUDA platform.

### *B. OpenGL User Interface:*

The *glew* and *freeglut* libraries are used for common OpenGL rendering and UI event commands [10]. OpenGL automatically adjusts the drawing frame rate to accommodate the simulation. The regular frame rates OpenGL supports are 60fps, 30 fps, 20 fps, 15fps, and 10 fps and below. To achieve better interactive visualization results, the frame rate of 30fps and above is preferred. In each rendering loop, if the simulation takes less than 1/60 second, the OpenGL will wait until 1/60 second is reached and draw one frame, thus resulting in a 60fps rendering rate. If the simulation takes longer than 1/60 second but shorter than 1/30 second, the OpenGL will then wait until 1/30 second is reached and then draw one frame and thus resulting in a 30 fps rendering rate. Therefore, the ultimate goal of simulation is to efficiently calculate the status of all points in shortest possible time.

Several functions were introduced to facilitate real-time user interaction with the 3D model. They are listed below:

1) Restart simulation at any time
2) Manually select any excitation point and restart simulation at any time
3) Introduce new excitation at any time during a simulation process
4) Freeze the simulation at any time and observe simulation evolves step by step at any time.
5) Change the excitation time range and excitation propagation constant at any time
6) Slice the heart open and into pieces and observe real-time or frozen excitation propagation within the heart
7) Observe the simulation under point cloud mode or regular rending mode

All these functionalities are realized under OpenGL events and can be controlled by either keyboard or mouse.

### C. CPU Simulation Implementation

A straightforward implementation under CPU architecture was carried out. The algorithm traverses through all the 728321 points in the 3D model to finish one round of iteration of the simulation calculation and determine the status (timer value) of every point in the 3D heart. It then calls the OpenGL to render the whole heart according to the timer value of every point. A gradient color scheme was adapted for rendering the heart. The white color represents the exact moment of excitation of the point (t=0), the red color represents the waiting and resting status and (t<0 and t>200). The color transitions from white to yellow, orange, and red when the timer value increases from 0 to 200.

### D. GPU Simulation Implementation – Case 1

The original algorithm was modified to a parallel version to adapt to the NVidia GPU architecture. Instead of iterating through every point, we issue every point a thread, which is responsible for incrementing the timer of the thread, looking up the neighboring points, and calculating and storing the negative timer value for the point if applicable. We thus launch 728321 threads to the GPU multicore processors and run these threads in parallel. The status of the timer for each point is stored in an array located in the global video memory. The neighboring points and distance data were stored in arrays located in the global video memory as well. Each thread will access the global video memory when it needs to, and thus made the application memory-bandwidth bound instead of computation bound. Once all the threads finished their work to update the timer value, a second GPU kernel is lunched, again one thread per point to update a color buffer of the points, which is stored in global memory of the video card. The color is calculated according to the timer value of each point. This step is again a memory-bandwidth bound step. After the color buffer is updated, CUDA hand OpenGL a pointer to this color buffer so that OpenGL can perform rendering based on the color data. No extra copy of the data in the color buffer array is needed. This technique is called OpenGL/CUDA coop, which will significantly reduce the time needed for rendering.

### E. GPU Simulation Implementation –Case 2

Because the algorithm is a memory-bandwidth bound one, it is possible to improve the performance by reducing the memory access. To do that, we considered moving our data into faster shared memory and accessing them from there. Since CUDA copy memory in a batched manner, instead of individually, when a particular start point need to access the first neighboring point data, all its neighboring points data will be copied to shared memory at the same time. Therefore, we can only copy once and utilize the neighboring data multiple times. To utilize that, we now lunch a thread for each neighboring points. Because the irregular data pattern, some points have up to more than 200 neighbors. Therefore, we lunch 256 threads for each starting point. Each of these 256 threads will work on one neighbor point of that starting point. The first thread will be responsible to copy the entire data of the neighboring points into shared memory. Once that data is accessible in shared memory, all the thread will find a corresponding neighboring point to work with. The task performed is increment timer array that's located in global memory, calculate *C*d* if applicable and update timer value accordingly. If the thread amount is greater than the total neighboring points, extra threads will be idle and wasted. We perform such actions for all the 728321 points and finish one round of iteration of simulation computation. After this kernel is finished, a second kernel is lunched to update the color buffer that's stored in the global memory. The second kernel operates in the same manner as the first implementation of GPU.

### F. Performance Comparison

Three implementations were executed under the identical hardware and software environment. The average time required computing the status of all the points in the 3D heart model was recorded and compared.

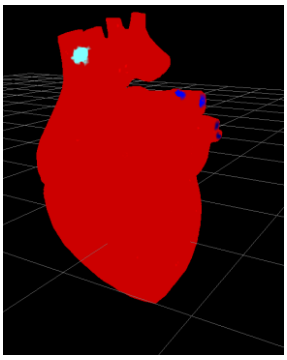

Fig. 2. 3D heart model

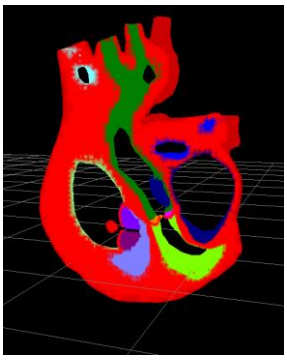

Fig. 3. Sectional view of 3D heart

## IV. RESULTS

### A. Visualization and User Interface

The heart 3D model is color coded according to its anatomic structure. The slicing function can be applied conveniently to show the inner condition of the heart. Totally 6 slice plane can be applied to divide the 3D model into any desirable configuration as shown in Figure 2 and 3.

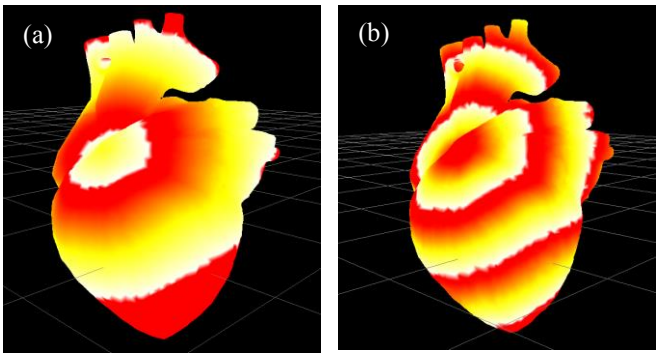


Fig. 4. Different excitation time span generates different excitation band width. (a) Time span = 200, (b) Time span = 50.

The simulation process can be viewed continuously. The progress of the propagation of the heart excitation wave can be paused at any moment. The excitation and propagation speed can be adjusted manually as shown in Figure 4 (a-b). When cells are firstly excited and the electrical potential reaches its peak value, the color turns white. As its status gets

back to resting condition, the potential decreases and the color turns yellow, orange and eventually original red color. The excitation origin can be handpicked by mouse. The excitation can start from a single point, several points or a continuous region.

Spontaneous spiral waves are observed under certain conditions, such as repeatedly exciting a certain area of the heart as shown in the Figure 5. This is more prone to happen when the excitation band width is relatively narrow, and the excitation wave front has better chance to penetrate the wave thickness and reaches its own wave end. The slicing functionality helps visualize the formation and evolvement of spiral waves inside the heart.

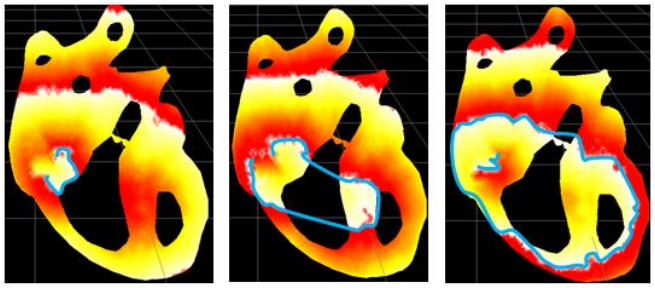

Fig. 5. The sectional views of heart show the formation of spiral waves during electrical conduction. Spiral waves are initiated on the spot located at the middle left site inside the heart, which is made visible when the heart is sliced. The excited front of spiral waves is marked with blue lines to show its progression and self-regeneration.

### *B. Performance comparison*

Even though the appearance of three different implementations is the same, the performances are drastically different. Figure 6 shows the average time of calculating the status for all the points in the heart.

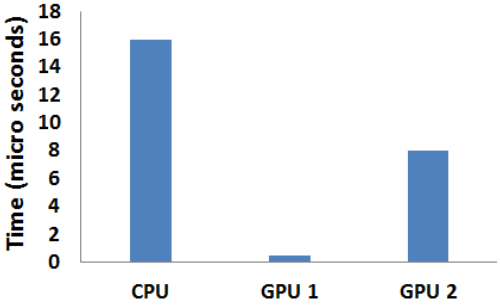


Fig. 6. Performance comparison of three implementations

This simulation process for CPU implementation is relatively slow and can barely satisfy the real-time rendering requirement. The total time elapsed for the simulation to finish is approximately 16ms on our machine. The GPU 1 implementation lunched one thread for each point and utilized OpenGL/CUDA coop to save data transferring time. It had the best performance of an average of 0.5ms, which is 30 times faster than purely based on CPU. The GPU 2 implementation attempts to make use of the fast data transferring within shared memory. The overall performance of this implementation is not as fast as the first implementation of GPU, but still better than CPU implementation. An average of 8 ms was reported for this implementation. Shared memory did not accelerate the overall simulation process, possibly because most points has only approximately 30 neighboring points, and we launch 256 just to cover a few extremely crowed points. Therefore most of the points experience significant waste when the rest of the 200+ threads were not utilized. This behavior reduced possible benefit and prevented significant speed gain.

## V. CONCLUSION

With modern massive parallel computation power of GPGPU, we have developed an effective 3D model of an anatomically realistic human heart to perform realtime simulation of cardiac electrical activities. Our model is useful to demonstrate the nature formation of spiral wave, which indicates that such pattern of excitation behavior is an intrinsic character of the excitable media. It may be noted that this present work did not incorporate the micro-scale simulation of the ionic and molecular species. However, it is very possible to include such low level of modeling into the developed simulation platform. Given the rapid development in the field of GPGPU architecture, real-time, multi-scale, and comprehensive simulation of a 3D human heart with unprecedented details and accuracy will be explored in our future study.

## REFERENCES


[1] D. Du, H. Yang, S.A. Norring, E.S. Bennett, "Multi-scale modeling of glycosylation modulation dynamics in cardiac electrical signaling." Proceedings of 2011 IEEE Engineering in Medicine and Biology Society Conference, pp. 104-107, September 2, 2011, Boston, MA.

[2] D. Du, H. Yang, S. Norring, and E. Bennett, "In-silico modeling of glycosylation modulation dynamics in hERG channels and cardiac electrical signaling," Vol. 18, No. 1, p205-214, 2013

[3] E. Bartocci, E. M. Cherry, J. Glimm, R. Grosu, S. A. Smolka, S. A. Smolka, and F. H. Fenton, "Toward real-time simulation of cardiac dynamics," Proceedings of the 9th International Conference on Computational Methods in Systems Biology, pp. 103-112, 2011

[4] R. FitzHugh, Mathematical models of excitation and propagation in nerve, pp. 1–85, H.P. Schwan, ed., Biological Engineering, McGraw-Hill Book Co., N.Y., 1966

[5] D. Barkley, "A model for fast computer simulation of waves in excitable media". Physica D: Nonlinear Phenomena, vol. 49, Issues 1–2, pp. 61–70, 1991.

[6] J. R. Silva and Y. Rudy, "Multi-scale electrophysiology modeling: from atom to organ," The Journal of General Physiology, vol. 135, pp. 575-581, June 01, 2010.

[7] E. V. Bondarenko and L. R. Rasmusson, "Simulations of propagated mouse ventricular action potentials: effects of molecular heterogeneity," Am J Physiol Heart Circ Physiol, vol. 293, pp. H1816-H1832, 2007.

[8] M. Gardner, Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life", pp. 120–123. ISBN 0-89454-001-7, 1970

[9] GPU Computing Gems Emerald Edition, 1st Edition from Wen-mei Hwu. ISBN-9780123849885, Printbook , Release Date: 2011.

[10] OpenGL Programming Guide: The Official Guide to Learning OpenGL(R), Dave Shreiner, Addison Wesley 7th edition.