Bio-Medical Materials and Engineering 26 (2015) S739–S746
DOI 10.3233/BME-151365
IOS Press

# Accelerating sino-atrium computer simulations with graphic processing units

Hong Zhang[a,*], Zheng Xiao[a] and Shien-fong Lin[b]
[a]*School of Electrical Engineering, Xi'an Jiaotong University, 710049, Xi'an, China*
[b]*Institute of Biomedical Engineering, National Chiao-Tung University, Hsinchu, Taiwan*

**Abstract.** Sino-atrial node cells (SANCs) play a significant role in rhythmic firing. To investigate their role in arrhythmia and interactions with the atrium, computer simulations based on cellular dynamic mathematical models are generally used. However, the large-scale computation usually makes research difficult, given the limited computational power of Central Processing Units (CPUs). In this paper, an accelerating approach with Graphic Processing Units (GPUs) is proposed in a simulation consisting of the SAN tissue and the adjoining atrium. By using the operator splitting method, the computational task was made parallel. Three parallelization strategies were then put forward. The strategy with the shortest running time was further optimized by considering block size, data transfer and partition. The results showed that for a simulation with 500 SANCs and 30 atrial cells, the execution time taken by the non-optimized program decreased 62% with respect to a serial program running on CPU. The execution time decreased by 80% after the program was optimized. The larger the tissue was, the more significant the acceleration became. The results demonstrated the effectiveness of the proposed GPU-accelerating methods and their promising applications in more complicated biological simulations.

Keywords: Sino-atrial node, atrium, graphic processing units, optimization, computer simulations, dynamic model

## 1. Introduction

Sino-atrial node cells (SANCs) are the primary pacemakers of the heart. Investigations of the interactions between sino-atrial node (SAN) and the atrium can greatly help us reveal the mechanisms of SAN automaticity and arrhythmia [1]. As an important quantitative method, simulation of the cellular mathematical model has been widely used for clinical and research purposes [2, 3]. However, due to the complex and dynamic natures of cardiac tissue [4], excessive computational loads have to be implemented given the limited computational power of Central Processing Units (CPUs).

Recently, the utilization of Graphic Processing Units (GPUs) instead of CPUs in cardiac simulations is being paid more and more attention [5, 6]. Each GPU may contain hundreds of stream multiprocessors. The inherent data-parallel nature of the cardiac computer simulation makes GPU an ideal platform. In the present study, the GPU system was nVIDIA GeForce GTX550Ti. CUDA (the Compute Unified Device Architecture) was used as the developing platform. In CUDA [6], the GPU is viewed as a computing device suitable for parallel data applications. It has its own device random access memory and may run a very high number of threads in parallel. Threads are grouped in blocks

[*]Address for correspondence: Hong Zhang, School of Electrical Engineering, Xi'an Jiaotong University, No. 28 Xianning West Road, Xi'an, 710049, China. Tel.: +86 29 82668630; Fax: 82668630; E-mail: mhzhang@mail.xjtu.edu.cn

and many blocks may run in a grid of blocks. Threads of the same block share data through the fast shared on-chip memory and can be synchronized through synchronization points.

GPU computation generally uses a CPU and GPU together in a heterogeneous co-processing computing model. The sequential part of the application runs on the CPU while the computationally intensive part is accelerated by GPU [7]. In order to enhance the performance of GPUs, a number of aspects have to be taken into account by a researcher, including the size of the thread block, the communication between the CPU host and the GPU device, as well as the data partition across various kinds of memories on GPUs. Although GPUs have been evaluated in a simple ventricular model with 8 state variables [8], detailed information about parallelization and optimization methods was not mentioned in the report. In addition, the GPU's application in the modern and much more complicated cardiac models is still worth discussing, since a different number of equations and state variables imply different arrangements of memories, registers and threads on GPU. Parallelization and optimization strategies usually need to be reconsidered in order to maximize performance.

In this paper, based on two types of single cellular models, SAN and atrium, an inhomogeneous one-dimensional sino-atrium tissue model was developed. The strategies to accelerate simulation with GPUs were proposed and evaluated. The effectiveness of GPUs in simulations with two different types of cellular models was demonstrated to show their promising future in more complicated simulations. The technique can also be applied to other biological phenomena modelled as wave propagation in an excitable medium.

## 2. Methods

### *2.1. Algorithm and parallelization strategies*

An inhomogeneous fiber was developed by coupling SANCs [9] to atrial cells [10] through gap junctions. A monodomain system in Eq. (1) was used to describe the electrical behaviors of the tissue, in which electrical excitation was considered to propagate along an idealized cable [11]:

$$\nabla \bullet (\sigma V) = A_m (C_m \frac{\partial V}{\partial t} + I_{ion}) \tag{1}$$

where $V$ is the membrane potential, $\sigma$ denotes the conductivity of the gap junction, $C_m$ is the membrane capacitance, $t$ is the time, $I_{\text{ion}}$ represents the total ionic current and $A_{\text{m}}$ is the surface-to-volume ratio.

Generally, the current leaving a cardiac surface is considered to be zero in terms of the Neumann boundary condition; therefore, at the two ends of the tissue, no-flux boundary conditions were used:

$$(\nabla V) \big|_{x=0} = \nabla V \big|_{x=ls+la} = 0 \tag{2}$$

where $x$ is the spatial coordinate in the string, and $l_s$ and $l_a$ represent the length of the SAN and the atrium, respectively.

Since the conductivity of the gap junction is discontinuous at the SAN-atrial border, a conservation of the flux condition given in Eq. (3) was imposed at $x = l_s$:

$$\sigma_s \nabla V \;|_{(x-ls)\to 0^-} = \sigma_a \nabla V \,|_{(x-la)\to 0^+} \tag{3}$$

where $\sigma_s$ and $\sigma_a$ represent conductivities in the region of the SAN and atrium, respectively.

To address regional differences between central and peripheral SANCs, the conductances of the gap junction, capacitance and the size of SANCs were changed via an exponential form [12]. The operator splitting technique [13] was used to split Eq. (1) into Eq. (4) and Eq. (5), in which the ordinary differential equation (ODE) describing the behavior of each single cell was solved separately from the partial differential equation (PDE) describing the electrical diffusion along the fiber. As shown in Eq. (4), $I_{\mathrm{ion}}$ was the function of $V$ and gating variable $Y$. Since $Y$ indicated the open probability of a specified channel and was also described by an ODE, a set of ODEs was required to solve for the intermediate value of $V$. In simulations, the Euler method was used to solve these ODEs. After computation, $V$ was used as an initial value for solving coupled PDE.

$$\frac{dV}{dt} = -\frac{1}{C_m}(I_{ion}(V, Y)) \tag{4}$$

$$\frac{\partial V}{\partial t} = \frac{1}{A_m C_m}(\nabla \bullet (\sigma V)) \tag{5}$$

The operator splitting method allowed us to calculate the ODEs separately from the PDE. For each time step, the first task was to compute the ODEs of each cell on the tissue one after another. After getting the membrane voltage of each single cell, the electrical diffusion was computed. Obviously, with more cells, the bottleneck of the computation was mainly in the ODE part. Therefore, in parallel strategy 1, the ODEs were assigned to GPU for implementation, in which each thread was responsible for a single cell, so different threads could be launched at the same time for different cell calculations. The computation of Eq. (4) was partitioned into segments, and a kernel for each segment was created. In strategy 1, five kernels were designed and launched one after another to initialize the variables and parameters of the SANCs and atrial cells in Kernel 1 and Kernel 2, calculate gating variables, ionic currents and transmembrane potentials of SANCs in Kernel 3 and Kernel 4, respectively. The computation of the atrial cells was carried out in Kernel 5.

In parallel strategy 2, the PDE solver and boundary conditions (2) and (3) were moved into GPU to compute, creating a new kernel for the computation of the electrical diffusion. In addition, Kernel 3 and 4 in strategy 1 were combined to compute the tissue of SANC. The initializations of the SANC and atrial variables were also merged in strategy 2 and implemented by one kernel. Therefore, in strategy 2, a total of 4 kernels were designed and launched one after another to initialize the variables and parameters of the SANCs and atrial cells in Kernel 1, calculate gating variables, ionic currents and transmembrane potentials of the SANCs and atrial cells in Kernel 2 and Kernel 3, respectively. The computation of the electrical diffusion was carried out in Kernel 4.

In strategy 3, the number of kernels was further decreased by combining Kernels 2 to 4 in strategy 2. In other words, except for variable initializations, all the other executions were carried out in one large kernel. During calculations, through the cudaMalloc function, the host first allocated memory for the parameters used on GPU. Under the control of instructions, the host then copied the values from the main memory to the global memory on GPU. After that, Kernel 1 was launched to initialize the SAN and atrial cell variables. Then, a cycle was executed on GPU by launching the second kernel, in which gating variables, ionic currents, transmembrane potentials and electrical diffusion were computed. Calculation results were copied back to the main memory for storage and visualization. The cycle continued until the end of the simulation. Finally, the memory space was released.

### *2.2. Program optimization*

The strategy with the shortest execution time was selected as the starting point for optimization. In order to improve its performance, the entire CUDA program was first analyzed through the NVIDIA Visual Profiler, after being compiled with VS2010. The GPU system and CUDA in this study had a computing capability 2.0; therefore, each multiprocessor could accommodate up to 1536 threads, 8 blocks and 48 warps. In order to make the workload of the threads occupy the entire multiprocessor, the number of threads should be greater than Threads Per Multiprocessor divided by Thread Block, that is, 1536/8 = 192. Each thread required 32 registers, and each multiprocessor had 32768 registers; therefore, each multiprocessor could have 1024 threads. In the present simulation, two blocks with 512 threads in each block were used.

Additionally, a large amount of time would be consumed if the data were copied to the memory on GPU and then back to the main memory during each operation. To reduce the data exchange, after a large amount of calculations on GPU, a data transfer between the host and the device was launched. This allowed more data to be copied back to the main memory during one exchange operation. Due to the limited storage capability of the shared memory on GPU, there was an up limit for data storage. In the present study, a data exchange was arranged for every 500 time step computation on GPU.

The data partition across various memories should also be considered in order to reduce the memory access time. In each strategy, Kernel 1 was used to initialize the variables of the single cell, so these initial data had to be placed in the global memory in order to be accessed by all threads. Since the read/write speed of the global memory was slow, shared memory and registers were also utilized to store frequently used data to reduce the access time.

## 3. Results and discussion

### *3.1. The efficacy of parallelization*

The NVIDIA Visual Profiler was used to measure the running time of each designed strategy to test its efficacy. Figure 1(a) shows a 1.6s simulation of real world cardiac time on tissue made up of 200 SANCs and 30 atrial cells. The result illustrated that strategy 3 was the fastest, and strategy 1 was the slowest. Compared with strategy 2, the PDE solver in strategy 1 was calculated on CPU. This made the parallelization degree low, while the data transfer between the host and the device frequent, thus resulting in a long running time. For strategy 3, due to combination of three kernels in strategy 2, the calls among multiple kernels and the access to memories were reduced, making the whole execution time short. The result suggested that to improve efficiency, the program must be well-organized to maximize the parallelization degree and reduce the calls among different tasks.

Figure 1(b) displays the time taken at different stages in strategy 3. As noticed, the computation time of gating variables, ionic currents, voltages and electrical diffusion counted for 96.3% of the total computation time, while the execution time on CPU took 2.1% of the time. The variable initialization

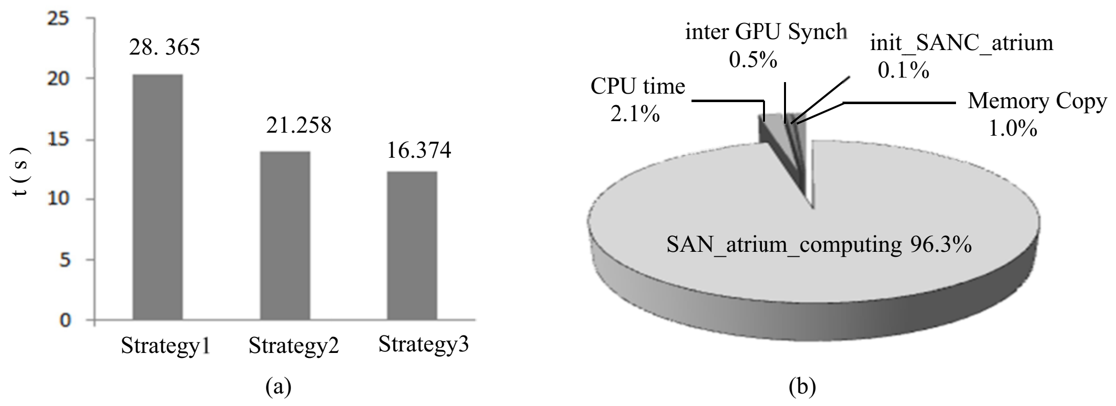

Fig. 1. The execution time for 3 strategies (a) and the time taken at different stages in strategy 3 (b).

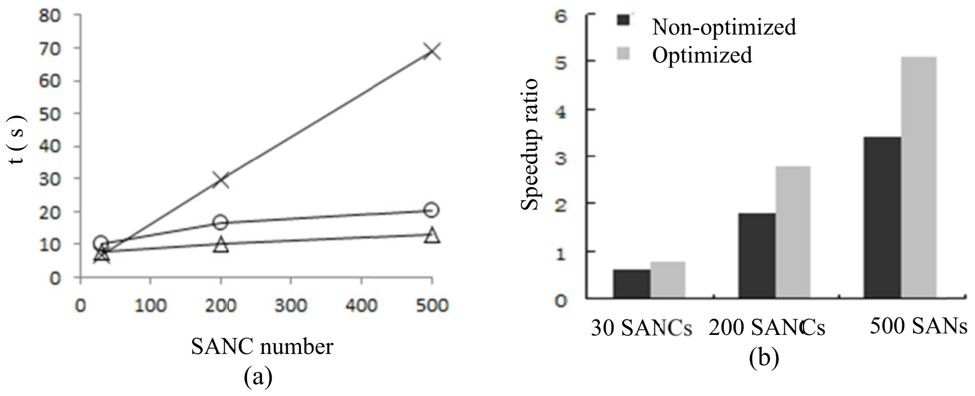

Fig. 2. Execution time and speedup ratio for different tissue sizes. In all situations, the number of atrial cells is 30. (a): the execution time for strategy 3 before (labeled with circle) and after (triangle) optimization, and the time taken by the serial program running on CPU (cross). (b): the speedup ratio of the CUDA program relative to the serial program before and after optimization for strategy 3.

was 0.1%. The data exchange through the memory copy occupied 1.0%. The GPU synchronization was 0.5%. Obviously, the computation on GPU dominated the whole running time, as the host CPU was only responsible for the calls of the kernel functions, memory allocation and data exchange.

The execution time taken by strategy 3 before and after optimization and the time taken by a serial program running on CPU are shown in Figure 2(a). The speedup ratio for the non-optimized and optimized program with respect to the serial program is illustrated in Figure 2(b). As noticed in Figure 2(a), for simulations on the fiber, including either 200 or 500 SANCs with adjoining 30 atrial cells, the execution time of the CUDA program was much shorter than that of the program written in serial code. The larger the tissue was, the more significant the acceleration became. GPU efficiency could be further improved by optimizations. For a simulation with 500 SANCs, the execution time taken by the non-optimized CUDA program decreased by 62% relative to the serial program. The time decreased by 80% after the CUDA program was optimized. However, for a simulation with only 30 SANCs, the execution time for the non-optimized program was 10.36s. After being optimized, the time was reduced to 8.922s. In this situation, the CUDA program took more time than the serial program. In the CUDA program, the data needed to be transferred between CPU and GPU before and after computation. This process required a certain amount of time. If the tissue is not large enough, then with respect to the time saved by GPU, the data transfer time might be too large to be neglected. Therefore, for very small-scale simulations, GPU showed less significance.

The results in Figure 2(b) illustrated that the speedup ratio could be elevated by optimizing the CUDA program. With larger tissues, the optimization became more effective.

Figure 3 displays the time taken at different stages in strategy 3 after the program was optimized. Compared with the non-optimized results in Figure 1(b), the computation time on GPU increased by 2.7%, while the running time on CPU decreased by 76.2%. The time taken by the data copy between CPU and GPU reduced by 60%.

## *3.2. The effects of atrium on SAN*

Figure 4 presents the electrical propagations along the developed tissue with and without the atrium. As noticed, compared with the peripheral cells, the central SANCs were characterized by decreased upstroke velocity, and prolonged action potential duration and cycle length. Despite the different characteristics in action potentials, SAN was able to synchronize their firing rate by electrotonic interactions among adjoining cells via gap junctions. With the presence of the atrium in Figure 4(a), the action potential propagated from the upper central SANCs to the lower periphery and drove the atrial cells to fire. Thus, the leading pacemaker site is located at the upper central SANCs. Without the

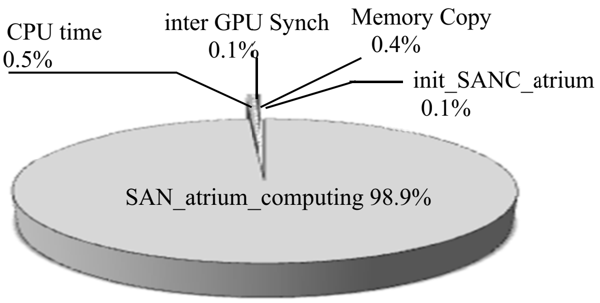

Fig. 3. The time taken at different stages in strategy 3 after optimization.

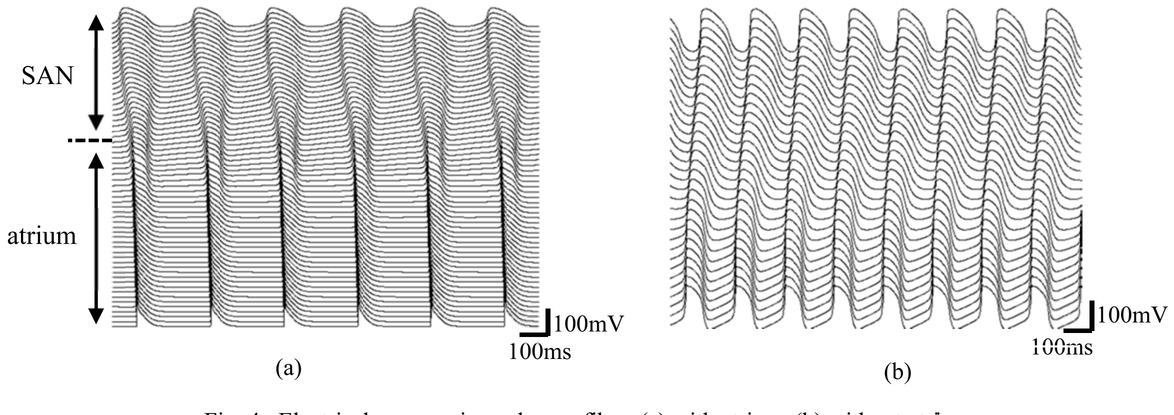

Fig. 4. Electrical propagations along a fiber. (a) with atrium. (b) without atrium.

presence of the atrium, the electrical wave propagated in the reverse direction (Figure 4(b)). A shift of the leading pacemaker site from the center to the periphery occurred. Additionally, the pacemaker rate became rapid compared to Figure 4(a). The characteristics of the action potential and the significant effect of atrial load on the activity of SANCs were all in good agreement with other reports [12, 14], thus demonstrating the correctness of our computation with GPU.

## 4. Conclusions

Detailed accelerating strategies for sino-atrium simulations on a GPU platform were discussed and presented in this paper. It was suggested that the iterative and inherent data-parallel natures of the tissue model make NVIDIA GPU an ideal architecture for computation, even if different cellular types were involved in the simulation. By evenly distributing the workload and balancing the data partitioning among threads to maximize the parallel degree while optimizing access to memory and reducing the frequency of data exchange, an optimal performance from GPU was achieved. The results indicated the correctness and effectiveness of the proposed strategies.

Acceleration of tissue simulation including both the SAN and the atrium with GPU was first examined in this paper. Although the developed tissue model was simple, it proved that with more cells, the accelerating efficiency from GPU would become more significant. Therefore, the proposed parallelization and optimization strategies provide a promising approach for more complicated biological simulations.

## Acknowledgments

This work is supported by the National Natural Science Foundation of China (Nos. 81271661, 30870659), the Health Foundation of Shaanxi province in China (08D23), the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry (SRF for ROCS), and the Fundamental Research Funds for the Central Universities.

## References

[1] P.S. Chen, B. Joung and T. Shinohara, The initiation of the heart beat, Circulation **74** (2010), 221–225.

[2] H. Zhang, B. Joung, T. Shinohara, X. Mei, P.S. Chen and S.F. Lin, Synergistic dual automaticity in sinoatrial node cell and tissue models, Circulation Journal **74** (2010), 2079–2088.

[3] H. Zhang, S.F. Lin and Z. Yang, Vulnerability during short-term memory induced response in canine ventricle, Bio-medical Materials and Engineering **24** (2014), 893–899.

[4] D.C. Michaels, D.R. Chialvo, E.P. Matyas and J. Jalife, Chaotic activity in a mathematical model of the vagally driven sinoatrial node, Circulation Research **65** (1989), 1350–1360.

[5] R.D. Yu, Y. Zhang, S.Y. Zhang, P. Chiang, Y.Y. Cai, J.M. Zheng and K.H. Mak, A framework for GPU-accelerated virtual cardiac intervention, International Journal of Virtual Reality **8** (2009), 37–41.

[6] R.D. Yu, Y. Zhang, S.Y. Zhang, P. Chiang, Y.Y. Cai, J.M. Zheng and K.H. Mak, GPU accelerated simulation of cardiac activities, Journal of Computing **5** (2010), 1700–1705.

[7] G. Jayshree and P. Jitendra, GPGPU processing in CUDA architecture, International Journal of Advanced Computing **3** (2012), 105–105.

[8] D. Sato, Y.F. Xie, J.N. Weiss, Z.L. Qu, A. Garfinkel and A.R. Sanderson, Acceleration of cardiac tissue simulation with graphic processing units, Medical & Biological Engineering & Computing **47** (2009), 101–1015.

[9] H. Zhang, A.V. Holden and I. Kodama, Mathematical models of action potentials in the periphery and center of the rabbit sinoatrial node, American Journal of Physiology - Heart and Circulatory Physiology **279** (2000), 397–421.

[10] D.W. Hilgemann and D.Noble, Excitation-contraction coupling and extracellular calcium transients in rabbit atrium: Reconstruction of basic cellular mechanisms, Proceedings of the Royal Society of London **230** (1987), 163–205.

[11] M. Potse, B. Dube and J. Richer, A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart, IEEE Transactions on Biomedical Engineering **53** (2006), 2425–2435.

[12] A. Garny, P. Kohl, P.J. Hunter, M.R. Boyett and D. Noble, One-dimensional rabbit sinoatrial node models: Benefits and limitations, Journal of Cardiovascular Electrophysiology **14** (2003), S121–S132.

[13] J. Geiser, Stabilization theory of iterative operator-splitting methods, International Journal of Computer Mathematics **87** (2010), 1857–1871.

[14] B. Joung, L. Tang and M. Maruyama, Intracellular calcium dynamics and acceleration of sinus rhythm by beta-adrenergic stimulation, Circulation **119** (2009), 788–796.