Toward Real-time Simulation of Cardiac Dynamics

Ezio Bartocci
Dept. of Appl. Math and Stat.
Stony Brook University
Stony Brook, NY, USA
eziobart@ams.sunysb.edu

Elizabeth M. Cherry
School of Mathematical Sciences
Rochester Institute of Technology
Rochester, NY, USA
elizabeth.cherry@rit.edu

James Glimm
Dept. of Appl. Math and Stat.
Stony Brook University
Stony Brook, NY, USA
glimm@ams.sunysb.edu

Radu Grosu
Department of Computer Science
Stony Brook University
Stony Brook, NY, USA
grosu@cs.sunysb.edu

Scott A. Smolka
Department of Computer Science
Stony Brook University
Stony Brook, NY, USA
sas@cs.sunysb.edu

Flavio H. Fenton
Department of Biomedical Sciences
Cornell University
Ithaca, NY, USA
flavio.h.fenton@cornell.edu

ABSTRACT

We show that through careful and model-specific optimizations of their GPU implementations, simulations of realistic, detailed cardiac-cell models now can be performed in 2D and 3D in times that are close to real time using a desktop computer. Previously, large-scale simulations of detailed mathematical models of cardiac cells were possible only using supercomputers. In our study, we consider five different models of cardiac electrophysiology that span a broad range of computational complexity: the two-variable Karma model, the four-variable Bueno-Orovio-Cherry-Fenton (BCF) model, the eight-variable Beeler-Reuter (BR) model, the 19-variable Ten Tusscher-Panfilov (TP) model, and the 67-variable Iyer-Mazhari-Winslow(IMW) model. For each of these models, we treat both their single- and double-precision versions and demonstrate linear or even sub-linear growth in simulation times with an increase in the size of the grid used to model cardiac tissue. We also show that our GPU implementations of these models can increase simulation speeds to near real-time for simulations of complex spatial patterns indicative of cardiac arrhythmic disorders, including spiral waves and spiral wave breakup. The achievement of real-time applications without the need for supercomputers may facilitate the adoption of modeling-based clinical diagnostics and treatment planning, including patient-specific electrophysiological studies, in the near future.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]: *Applications;* G.1.8 [**Partial Differential Equations**]: *Finite Differences*

General Terms

Performance; Theory

Keywords

GPU computing, High-performance computational systems biology, cardiac models.

1. INTRODUCTION

Cardiac arrhythmia, such as atrial fibrillation (AF) and ventricular fibrillation (VF), is a disruption of the normal excitation process in cardiac tissue due to faulty processes at the cellular level, at the single ion-channel level, or at the level of cell-to-cell communication. The clinical manifestation is a rhythm with altered frequency (tachycardia or bradycardia, see Fig. 1(a)) or the appearance of multiple frequencies (polymorphic ventricular tachycardia), with subsequent deterioration to a chaotic signal (VF, see Fig. 1(b)) [3, 14, 15, 41].

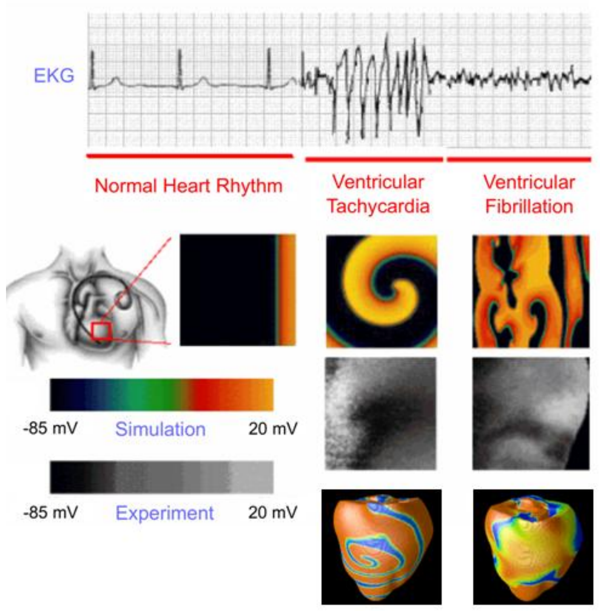


Figure 1. Schematic of the transition from normal heart rhythm to ventricular tachycardia and ventricular fibrillation.

Cardiac tissue is typically modeled mathematically as a reaction-diffusion system involving partial differential equations (PDEs) for diffusing species (typically only the transmembrane potential) and a system of nonlinear differential equations describing all other state variables that describe the flux of ions across the cell membrane along with ion concentrations. Detailed models of cardiac cells can include more than 80 state variables and hundreds of fitted parameters [8].

Because of the computational demands such models place on normal CPU-based and multi-core-based workstations, most studies of the electrical activity of the heart traditionally have focused on using either complex cell models in single-cell formations or simplified cell models in more realistic 3D heart structures. A few efforts have used supercomputers to integrate models of intermediate complexity with 3D structures [18, 29, 38].

Real-time simulation refers to the ability to execute a computer model of a physical system at the same rate as the actual physical system. Recently, the advantages of GPU over CPU processing have been established for many areas of science, including systems biology, where the highly parallel and multi-core capabilities of GPUs allow the implementation of extremely fast simulations of complex models in 2D and 3D using CUDA (Compute Unified Device Architecture) from NVIDIA. If such large-scale realistic models can be simulated in *near* real-time, many more applications, including patient-specific treatment strategies for cardiac-rhythm disorders, become feasible without the need for supercomputers. To date, such efforts have come up short.

Below, we show that it is possible to perform simulations of models of cardiac cells ranging in complexity from two to 67 variables in near real time for realistic problem sizes through careful GPU implementations. To maximize the performance gain, model-specific optimization techniques, including partitioning of the model equations among multiple CUDA kernels as appropriate and judicious use of the different types of memory available to GPUs, are incorporated.

2. CUDA PROGRAMMING MODEL

CUDA is a general-purpose parallel-computing architecture and programming model that leverages the parallel compute engine in NVIDIA GPUs. Optimal programming of GPUs requires a thorough understanding of the CUDA architecture and the underlying GPU hardware, including the concepts of threads, processors, and kernels, as well as the different levels of memory available. As illustrated in Figure 2, the GPU architecture is built around a scalable array of multithreaded Streaming Multiprocessors (SMs), made up of 8 or 32 Scalar Processor (SP) cores. SP cores contain a fused add-multiply unit capable of both single- and double-precision arithmetic and share a common local memory.

The CUDA parallel computing model uses tens of thousands of lightweight threads assembled into one- to three-dimensional *thread blocks*. A thread executes a function called the *kernel* that contains the computations to be run in parallel; each thread uses different parameters. Threads located in the same thread block can work together in several ways. They can insert a synchronization point into the kernel, which requires all threads in the block to reach that point before execution can continue. They also can share data during execution. In contrast, threads located in different thread blocks cannot communicate in such ways and essentially operate independently. Although a small number of threads or blocks can be used to execute a kernel, this arrangement would not fully exploit the computing potential of the GPU. To utilize the GPU most efficiently, the underlying problem should be separated into independent blocks that can be further divided into cooperative threads; see Figure 3. Problems that cannot be implemented in this manner will benefit significantly less from implementation using GPUs.

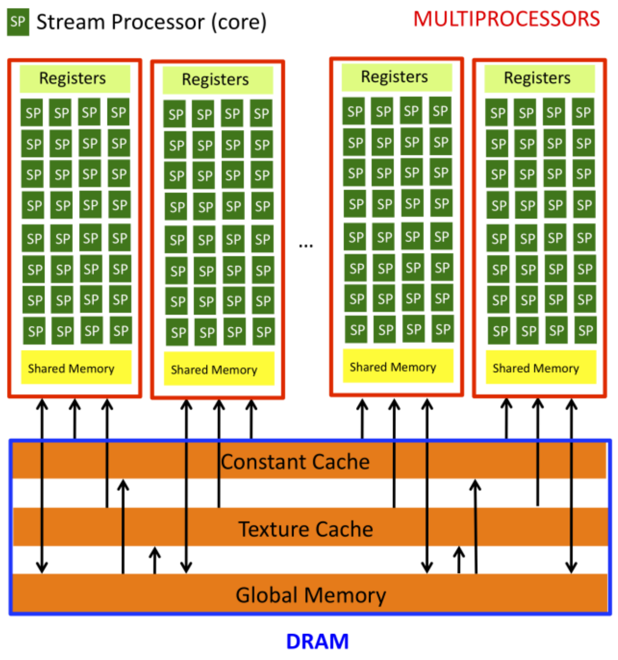


Figure 2. GPU architecture.

Different types of memory are available for use in CUDA, and their judicious use is key to performance. The most general is global memory, to which all threads have read/write access. The generality of global memory makes its performance less optimized overall, so it is important that access to it be coalesced into a single memory transaction of 32, 64, or 128bytes [28]. Constant memory is a cached, read-only memory intended for storing constant values that are not updated during execution. All instances of a kernel may access these values regardless of location. Texture memory is another cached, read-only memory that is designed to improve access to data with spatial locality in up to three dimensions. For example, texture memory is a natural choice for storing an array that also will require retrieval of neighboring values whenever a single entry is retrieved. Linear interpolation is provided with texture memory. Finally, local memory is invoked when a thread runs out of available registers. CUDA library functions in the host code running on the CPU administer such tasks as kernel execution and memory management.

Additional, significantly faster levels of memory are available within an SM, including 16KB or 32KB of registers partioned among all threads. As such, using a large number of registers within a CUDA kernel will limit the number of threads that can run concurrently. In addition, each SM has a shared memory region (16KB). This level of memory, which can be accessed nearly as quickly as the registers, facilitates communication between threads and also can be used as a memory cache that can be controlled by the individual programmer [22]. Shared memory is divided into 16 banks; for optimal performance, threads executed concurrently should access different banks to prevent bank conflicts [28].

The computing resources of CUDA-capable video cards are characterized by their compute capability. Devices with compute capability1.0 and 1.1 make up the first generation of CUDA devices, based on the G80 GPU, whereas those with compute capability 1.2 and 1.3 are based on the more advanced GT200 GPU. Only cards with compute capability greater than or equal to 1.3 allow double-precision floating-point operations. Recently NVIDIA introduced a new family of cards called the Fermi-based GF100 GPU with compute capability 2.0 that supports object-oriented programming.

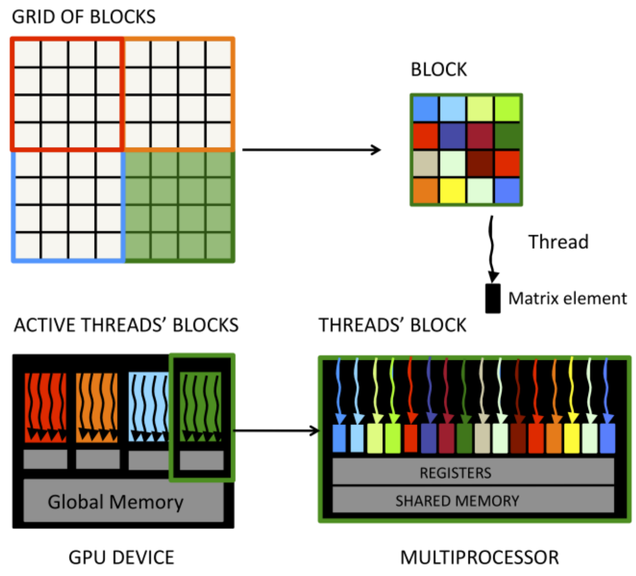


Figure 3. The parallel execution of a kernel is divided into independent blocks of cooperative threads.

Our GPU testbed is an NVIDIA Tesla C2070 processor containing 448 scalar cores organized as 14 SMs, and having 6GB of DRAM. The processor core clock is 1.15 GHz and the maximum memory-access bandwidth is 144 GB/sec. The C2070 can perform 1030 Gigaflops using single-precision arithmetic or 515 Gigaflops using double-precision arithmetic. To compare our results with other papers, we have also used a GPU Tesla C1060 with 240 SP cores divided equally among 30 SMs, and 4GB of DRAM. The SP core clock is 1.29 GHz and the maximum bandwidth of memory access is 102 GB/sec. The C1060 is able to perform 933 Gigaflops using single-precision arithmetic or 78 Gigaflops using double-precision arithmetic.

3. CARDIAC MODELS

Modeling cardiac tissue consists of implementing a reaction-diffusion equation of the transmembrane potential coupled to one or more ordinary differential equations. The diffusion term represents intercellular coupling and may contain information about tissue architecture that affects wave propagation, including the local muscle fiber orientation, which indicates the fastest direction of wave propagation and describes tissue anisotropy. The reaction term is nonlinear and incorporates additional variables.

The exact form of the reaction term varies depending on the level of detail and complexity of the electrophysiology model. Most models of cardiac cell electrophysiology have their origins in the Hodgkin-Huxley model [17] of a neuron and the Noble model [27], which was the first to apply the same modeling principles to cardiac cells. This type of model describes the reaction term as the sum of currents of different ion species (for cardiac cells, primarily Na^+, K^+, and Ca^{2+}) crossing the cell membrane through ion channels. These currents operate in a specific fashion to make up a cellular action potential, an excursion from a negative resting membrane potential (around -85 mV) to a positive potential (around 20 mV in tissue) and back to the resting membrane potential. The initial increase in potential occurs quickly (on the order of a few ms) via a large inward current carried by Na^+ ions. Over the rest of the action potential, which lasts hundreds of ms in large mammals, the membrane potential is determined primarily by a balance between inward Ca^{2+} currents and outward K^+ currents. The range of voltages and times over which each ion current is active is determined by one or more factors, including the transmembrane potential, time-dependent gating variables that modulate the permeability of ion channels, and the concentrations of ions inside and outside the cells. Gating variables, ion concentrations, and other terms often are state variables of a model and evolve according to their own differential equations and can have different time scales.

The precise details of the electrophysiology models can be represented at different levels. Biophysically detailed models take into account a large number of currents and may have anywhere from a few to dozens of state variables and differential equations in addition to the transmembrane potential. One of the earliest models of cardiac cells is the Beeler-Reuter (BR) model [1], which represents the electrophysiology of ventricular cells (located in the lower chambers of the heart). This model includes eight state variables and has been widely used for several decades; we include it as one of the models we implement in CUDA. More recent models make use of subsequent biophysical discoveries to represent cardiac cell electrophysiology in more detail. We implement two such models, both of human ventricular electrophysiology: the 19-variable Ten Tusscher-Panfilov model (TP) [39] and the 67-variable Iyer-Mazhari-Winslow (IMW) model [19].

The wealth of biophysical detail in complex electrophysiology models can obscure the physical phenomena underlying their behavior. For this reason, less complicated models also have been developed that can represent the behavior of cardiac cells and tissue in a way that facilitates analysis of their dynamics and of the role of different parameters in determining their behavior. One such model is the Karma model, which is a simplification of the Noble model. Another model is the Bueno-Orovio-Cherry-Fenton (BCF) model [2] (also called the minimal model [4, 7]). This model uses four variables and three ion currents that represent summary Na^+, K^+, and Ca^{2+} currents and incorporates the minimal level of complexity necessary to reproduce accurate action potential shapes and rate-dependent properties. We also implement both of these models in CUDA.

Therefore, we implement a total of five different models containing from two to 67 variables. By considering models that vary over a broad range of biophysical detail and computational complexity, we are able to identify specific issues that arise in CUDA programming depending on the model size. This information should be important in determining how best to implement particular types of models to maximize performance in CUDA.

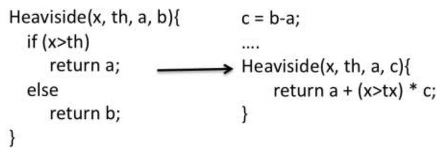
4. REACTION TERM

The reaction term in cardiac models consists of anywhere from one to dozens of additional differential equations that provide simple or detailed descriptions of the electrophysiology of cardiac cells. Appropriate implementation of the reaction term is vital to optimizing the performance of cardiac electrophysiology simulations on GPUs.

Performance is increased significantly by tabulating nonlinear functions of one variable in lookup tables that are accessed through the texture memory. This provides two main advantages: it reduces the latency of global memory access, and the hardware provides a built-in linear interpolation capability. To remove singularities that occur in some functions for values that make the denominator of a fraction zero, we calculate the limit of the functions at the relevant value using l'Hopital's rule.

We also improve performance in several other ways. All divisions that do not involve variables are replaced with equivalent multiplications. Also, some equations are implemented using semi-implicit methods that allow the use of larger integration time steps.

In a number of cases, the reaction term in cardiac models uses biological switching functions in the form of Heaviside functions. The *Heaviside function* is a discontinuous function whose value is zero for negative arguments and one for positive arguments. Heaviside functions are usually represented by an *if* statement that is penalized by the GPU, because it leads to thread divergence during parallel execution. In our simulations, we have used an alternative implementation in which the *if* statement is replaced by multiplication with a predicate, as Fig. 4 shows.



```
Heaviside(x, th, a, b){
  if (x>th)
     return a;
  else
     return b;
}
```

```
c = b-a;
....
Heaviside(x, th, a, c){
     return a + (x>tx) * c;
}
```

Figure 4. Heaviside function implementations by *if* statement (left) or multiplication with a predicate (right).

A central concern in the implementation of the reaction term is the number of registers used per thread. The total number of threads per block and the number of registers per thread should be chosen to best utilize the available computing resources. The relation among these quantities, as given by the CUDA Programming Guide [28], is

$$\frac{R}{B \times ceil(T, 32)},$$

where R is the total number of registers per multiprocessor (a device-specific quantity), B is the number of active blocks per multiprocessor, T is the number of threads per block, and ceil(T,32) is T rounded up to the closest multiple of 32. Having multiple active blocks for each multiprocessor ensures that the multiprocessor will not be idle during thread synchronization or device memory access. By overlapping execution of blocks that wait and blocks that can run, the multiprocessor is able to hide better the latency of communication.

In simple cardiac models with only a small number of variables (two or four), it is possible and in fact is advisable to implement the solution of the reaction term as a single kernel. In this case, the number of registers used per thread is usually less than 32, so that two or more active thread blocks of 256 threads can be executed by the same multiprocessor (with a device equipped with 16KB of registers for each multiprocessor). In more complex cardiac models having more than four variables, use of a single kernel to solve the reaction term is not recommended and often is not possible because the number of registers available per thread is insufficient. In this case, the solution of the reaction term is implemented as a sequence of multiple kernel invocations, with each kernel devoted to solving a group of related variables. Because a kernel invocation may modify the input of the following kernel, it is necessary to resolve these dependencies by buffering the variables that are common input among the kernels. Every kernel invocation introduces an overhead. To optimize the performance of our implementation with multiple kernels, we used the visual profiler provided by the recent CUDA SDK to find the best trade-off between kernel splitting, resource utilization, and the kernel invocation overhead.

5. DIFFUSION TERM

Cardiac models also include a diffusion term that couples the main variable (transmembrane potential) spatially. Solving the diffusion term essentially consists of calculating the Laplacian operator on all of the grid points. This operation requires frequent access to values of neighboring cells. The use of the global memory is not desiderable for this operation, because the specific memory access pattern that the threads should follow in order to read from the neighbor cells is not coalesced [28], which reduces performance considerably. To solve this problem more efficiently, we consider two solutions, one using shared memory and the other using texture memory, proposed in the literature [25, 32].

In the shared memory approach, the grid points can easily be subdivided into smaller overlapping parts (see Fig. 5), which then can be assigned to the threads' blocks. The values at neighboring elements are then read using shared memory within a block. This operation is performed by all the threads of the block, which control both the yellow and the red elements shown in Fig. 5. After synchronizing among the threads belonging to the same block, the threads controlling the red cells read the neighborhood collected in the shared memory and write in their cell the updated value of the Laplacian. This solution can be used with both single- and double-precision implementations, but the drawback is that it needs to use more threads than the number of elements of the matrix.

An alternative approach that we have considered is to use the texture memory, which provides a cache that is optimized for 1D, 2D or 3D spatial locality, so that threads that read texture addresses that are close together will achieve the best performance. Currently, it is not possible to bind the texture to double precision data, so the use of the texture memory for implementing the diffusion term is restricted only to single-precision implementations.

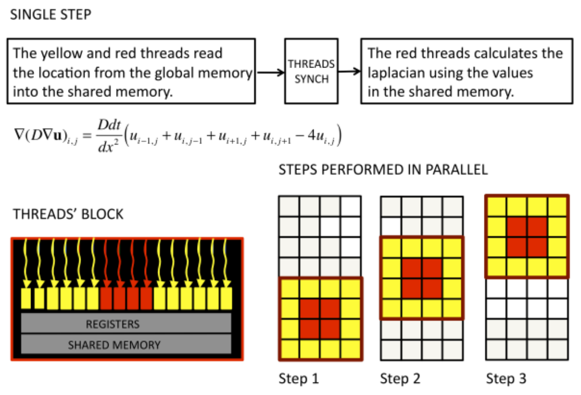

SINGLE STEP
The yellow and red threads read the location from the global memory into the shared memory.
THREADS SYNCH
The red threads calculates the laplacian using the values in the shared memory.
$\nabla(D\nabla \mathbf{u})_{i,j} = \frac{Ddt}{dx^2}\left(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1} - 4u_{i,j}\right)$
STEPS PERFORMED IN PARALLEL
THREADS' BLOCK
REGISTERS
SHARED MEMORY
Step 1
Step 2
Step 3

Figure 5. Calculating the diffusion term using shared memory.

6. MODELS

In this section we perform 2D simulations of the five models of interest and analyze their performance. Four square grid sizes are used to assess how the performance scales with the number of nodes. Although the larger grid sizes are physiologically unrealistic for 2D human cardiac surfaces, the numbers of nodes they contain are similar to what would be required for some 3D implementations. Note that because of the necessity of representing information from neighboring thread blocks in shared memory implementations, our 16 x 16 thread blocks are effectively 14 x 14 for the shared memory implementations. Therefore, the grid sizes in the shared memory implementations (512, 1024, 1536, and 2048) are slightly different than those in the texture memory implementations (520, 1038, 1556, and 2074) in all cases.

6.1 Karma Model (2 Variables)

The Karma model is a simplified model of cardiac electrophysiology that reproduces some basic features of cardiac dynamics, including wavelength oscillations, which can be seen in Fig. 6. To quantify GPU performance, we initiated a spiral wave [12] using the Karma model in square grids with each side consisting of 512, 1024, 1536, or 2048 elements (corresponding to 2^{18}, 2^{20}, $1.125x2^{21}$, and 2^{22} grid points, respectively), as shown in Fig. 6. Note that the wavelength of a spiral wave in this model is smaller than that of the human ventricular models (compare Figs. 9 and 12). We used three different implementations: double precision, single precision using shared memory to calculate the diffusion term, and single precision with the texture memory used for the diffusion term. The double precision simulation required just over twice as much time as the corresponding single precision simulation. For the single precision simulations, use of the texture memory for the diffusion term improved performance. For the smallest grid size, which was similar in size to the surface area (epicardium) of a human ventricle, the simulation times were almost real time for the shared memory implementations, and the simulation was faster than real-time for single precision using the texture memory.

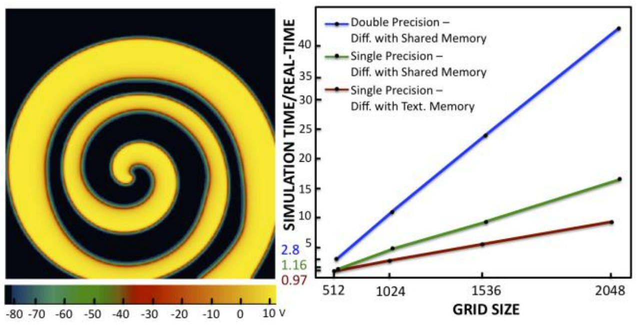


Figure 6. Left: Spiral wave using the Karma model in a 512 x 512 tissue (12.8 cm x 12.8 cm; resolution 0.0262 cm). Right: Simulation time normalized to real time for computing 1 s using different grid sizes.

6.2 BCF Model (4 Variables)

The BCF model is a minimal model of cardiac electrophysiology that reproduces many properties of cardiac tissue and can be parameterized [2, 4] in many cases to reproduce the dynamics of more complex models as well as experimental data. As with the Karma model, we initiated a spiral wave using the BCF model for square grids with each side consisting of 512, 1024, 1536, or 2048 elements and used the same three implementations: double precision, single precision with shared memory, and single precision with texture memory, as shown in Fig. 7. For the BCF model, the double precision simulation required almost three times as much time as the corresponding single precision simulation. For the single precision simulations, use of the texture memory for the diffusion term improved performance, but not by as large a factor as for the Karma model. For the smallest grid size, the simulation times were between a factor of 2 and 5 times longer than real time for all three implementations.

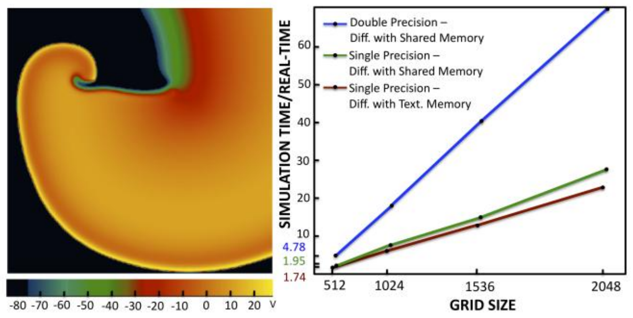


Figure 7. Left: Spiral wave using the BCF model in a 512 x 512 tissue (12.8 cm x 12.8 cm; resolution 0.025 cm). Right: Simulation time normalized to real time for computing 1 s using different grid sizes.

6.3 BR Model (8 Variables)

The BR model is an 8-variable model of cardiac electrophysiology that was the first detailed model of mammalian ventricular cell electrophysiology. As with the Karma and BCF models, a spiral wave was initiated using the BR model for square grids with each side consisting of 512, 1024, 1536, or 2048 elements and the performance of the same three implementations (double precision, single precision with shared memory, and single precision with texture memory) was quantified, as shown in Fig. 8. For the BR model, the double precision simulation was about two times slower than the corresponding single precision simulation. As with the Karma model, use of the texture memory for calculation of the diffusion term improved performance significantly for the single precision case. For the smallest grid size, the simulation times for the three implementations were between a factor of 10 and 25 times longer than real time.

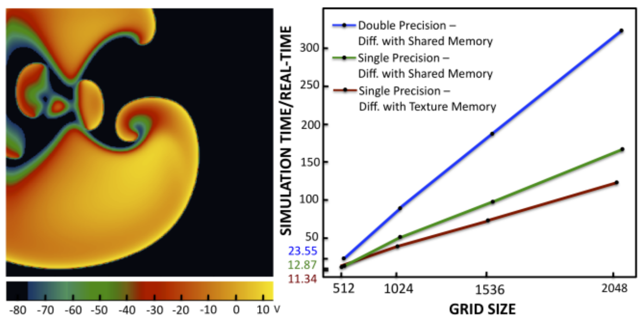


Figure 8. Left: Spiral wave using the BR model in a 512 x 512 tissue (10.24 cm x 10.24 cm; resolution 0.02 cm). Right: Simulation time normalized to real time for computing 1 s using different grid sizes.

6.4 TP Model (19 Variables)

The TP model is a 19-variable model that describes the electrophysiology of human ventricular cells. As with the previous models, a spiral wave was initiated using the TP model for square grids with each side consisting of 512, 1024, 1536, or 2048 elements and the performance of the same three implementations was quantified, as shown in Fig. 9. For the TP model, the double precision simulation was about two to three times slower than the corresponding single precision simulation. Use of the texture memory for calculation of the diffusion term resulted in a substantial performance improvement: for the largest grid size, the texture memory simulation required only half as much time as the corresponding shared memory simulation. At the smallest grid size, the simulation times for the three implementations were between a factor of 35 and 70 times longer than real time.

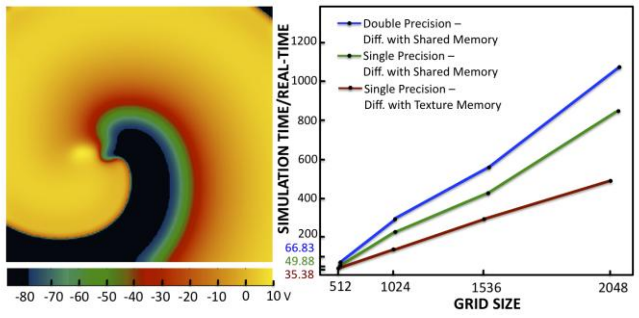


Figure 9. Left: Spiral wave using the TP model in a 512 x 512 tissue (10.24 cm x 10.24 cm; resolution 0.02 cm). Right: Simulation time normalized to real time for computing 1 s using different grid sizes.

For the other models discussed so far, no significant differences were observed between the single and double precision simulations. However, we knew that for some more biophysically detailed models, including the TP model, single precision is not sufficient to represent small but important changes in the intracellular K^+ and intracellular Na^+ concentrations over the course of each action potential. Thus, in the single precision simulations of the TP model, very small changes in concentration were represented as zeros, which produced non-smooth time traces of these concentrations within a single action potential. Although the concentration differences between single and double precision over one action potential were slight, the difference accumulated over time and changed not only the value of the concentration but also the trend of the concentration over time, especially for the K^+ concentration, as shown in Fig. 10. The differences in concentrations affect the time progression of spiral waves generated using single and double precision. Fig. 11 shows snapshots of spiral waves obtained after 600 s (10 min) of simulation time and indicates that the waves are at different points in their rotation paths.

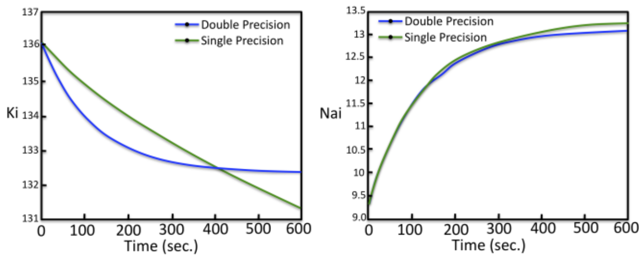
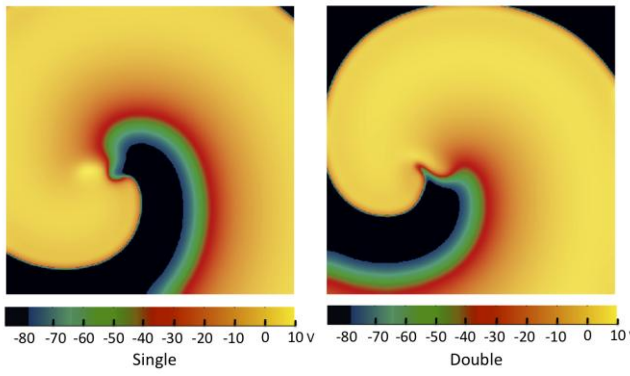


Figure 10. Time evolution of the intracellular K^+ (left) and Na^+ (right) concentrations observed at a representative grid point for the TP model with single and double precision.



-80 -70 -60 -50 -40 -30 -20 -10 0 10 V
Single
-80 -70 -60 -50 -40 -30 -20 -10 0 10 V
Double

Figure 11. Spiral waves generated for the TP model with single precision (left) and double precision (right) after 10 min.

6.5 IMW Model (67 Variables)

The IMW model is a 67-variable model that describes the electrophysiology of human ventricular cells in more detail than the TP model. As with the previous models, a spiral wave was initiated for square grids with each side consisting of 512, 1024, 1536, or 2048 elements and the performance of the same three implementations was quantified, as shown in Fig. 12. For the IMW model, the double precision simulation was about twice as slow as the corresponding single precision simulation. As with the Karma, BR, and TP models, use of the texture memory for calculation of the diffusion term improved performance significantly. At the smallest grid size, the simulation times for the three implementations ranged from 680 to 1300 times longer than real time. As with the TP model, double precision is necessary for adequate representation of ion concentrations for the IMW model.

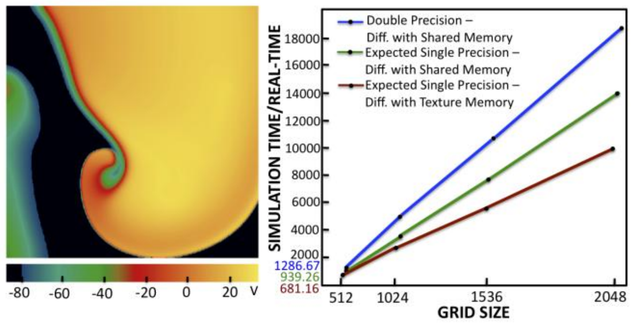


Figure 12. Left: Spiral wave using the IMW model in a 512 x 512 tissue (10.24 cm x 10.24 cm; resolution 0.02 cm). Right: Simulation time normalized to real time for computing 1 s using different grid sizes.

7. PERFORMANCE

Figure 13 shows the performance for the different grid sizes as a function of the number of model variables. For the models with four, eight, and 19 variables, the simulation time scales linearly with the number of variables. For the IMW model (67 variables), the departure from linear scaling can be explained by several factors. For one, it is necessary to split the solutions of the ordinary different equations into 21 kernels calls. As a result, for any variable needed by more than one kernel, it is necessary to duplicate calculation of that variable within each such kernel to avoid communication between kernels. This duplication results in increased overhead for every integration step computed. In addition, the IMW code was not optimized as fully as the codes for the other models were (in terms of lookup tables, division eliminations, etc.).

In the future, we expect to obtain better performance for all the models by using other integration methods for the diffusion term, such as the alternating direction implicit scheme, that can allow the use of larger integration time steps [9].

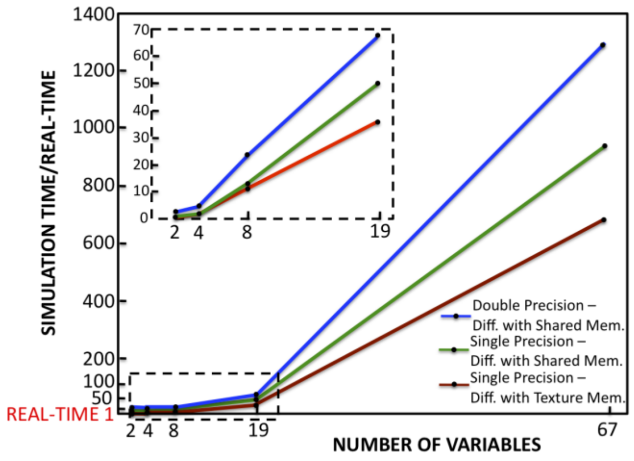


Figure 13. Simulation time normalized to real time as a function of the number of model variables.

8. RELATED WORK

Over the last five years, GPU performance has exceeded that of the CPU. As this trend in outperformance continues [28] many areas of research that require large-scale simulation, such as systems biology [6], have turned to GPU implementations to gain that acceleration in computing, and cardiac electrical dynamics is no exception [5]. Simulations of human heart dynamics at currently feasible spatial resolutions require the solution of between 2^{24} and 2^{27} nodal points or "cells" [18, 29]. Each cell, turn, involves a separate implementation of mathematical equations describing its electrophysiology, the description of which can be as simple as two [20] or as complicated as 67 [19] or 87 [11] ordinary differential equations. Even with a simple cell model, 0.6 seconds of simulation requires about 2 days using 32 CPU processors [29]; for a more complex model, the same simulation time uses about 10 hours with 6144 CPUs [18]. However, accelerations to near real time soon will become possible on single desktops by taking advantage of GPU processing capabilities. The electrophysiological equations of cardiac cells are of the reaction-diffusion type, for which GPUs have been shown to be superior to CPUs in both 2 and 3 dimensions with typical acceleration values between 5 and 40 depending on the algorithms used [25, 33]. The first simulation of cardiac arrhythmias using GPUs actually was performed on an Xbox 360 [35] using the BCF model [2] (Fig. 7). The eight-variable Luo-Rudy I (LRI) model was the first simulated using a realistic rabbit ventricular structure on a GPU [34], with 1 s of simulation taking 45 minutes using a cluster with 32 CPUs and 72 minutes on a single GPU. Since then other studies have emerged comparing the speeds between CPUs and GPUs for different cardiac cell models. The 27-variable Mahajan et al. model [24] was reported to run 9 to 17 times faster (depending on tissue size) on GPUs [40]. More recently, Rocha et al. [31] reported a gain of up to 20 times for a single GPU implementation compared to a parallel CPU implementation running with 4 threads on a quad–core machine, with parts of the code accelerated by a factor of 180 for the 8-variable LRI model [23] and the 19-variable TP model [39]. Lionetti et al. [21, 22] showed how different implementations are needed for different cell models (two-variable FitzHugh-Nagumo [10], eight-variable BR [1], 18-variable Puglisi-Bers (PB) [30], 42-variable Grandi et al. [13], and 87-variable Flaim et al. [11]) in order to optimize each one and obtained a speedup of 6.7 for the 87-variable model. GPUs also have been used for intracellular calcium dynamics within a single cell using Monte Carlo simulations, where a factor of 15,000 reduction in time compared to previous studies was found [16]. In addition to electrophysiological dynamics, GPUs have been used to accelerate heart manipulations to enhance intervention simulations such as catheter positioning [43], surgical deformation [26], simple contractions [42, 44], and ECG generation [36, 37].

In this manuscript, we do not focus on comparing CPU vs GPU performance, as this has been amply demonstrated already using many different cell models. Instead, we focus on developing optimal implementations to obtain the maximum accelerations and get as close as possible to real time simulations. Sato et al. report 1 s of simulation in the 8-variable LR1 model in an 800x800 domain taking 283 s; in contrast, our simulations in the 8-variable-BR model (the two models are mathematically almost equivalent and share more than 90% of the same equations) take 11.34 s on a 512 x 512 domain and 39.2 s on a 1024 x 1024 domain (rescaling our times to the 800 x 800 domain results in a comparable speedup of a factor of 11). Vigmond et al. [40] report that 1 s of simulation time on 5 million nodes using the 27-variable Mahajan et al. model takes about 16 ksec (~4.5 h), whereas our 19-variable TP implementation in a 2048 x 2048 domain (close to 5 million nodes) takes about 8.2 min. However, a direct comparison is difficult to make as there is not only a difference of eight ODEs, but their simulations utilize a computationally more expensive bidomain approximation (used during simulations of defibrillation, where a Poisson equation needs to be solved at each time step). Lionetti et al [21, 22] performed 300 ms of a heart beat simulation on a domain that consisted of 42,240 "cell" points to represent a ventricular section. Their main interest was to optimize the ODE part of the reaction diffusion system, so no spatial integration was performed and all the cells were decoupled; therefore, their integration times did not included the spatial integrations. However, they used different optimizations for the different cell models tested and showed how different models benefit from different types of optimizations. For the two-variable FHN model, 300 ms of simulation in their 42,240 required 5.91 s; for the eight-variable BR model, 22.64 s; for the 18-variable PB model, 49.87 s; and for the 87-variable Flaim et al. model, 119.29 s. To compare with our simulations, in which the smallest domain consisted of 512 x 512 grid points (a domain about 6.2 times larger), and for 1 s of simulation time, we need to multiply their timing results by 20.5. Therefore, 1 s of simulation of the two-variable Karma model (with the same complexity as the FHN model) took 0.97 s vs. 121 s, the eight-variable BR model took 11.34 s vs. 464 s, the 19-variable TP model took 35.4 s vs. the 18-variable PB model 1022 s, and the 67-variable IMW 681 s vs. the 87-variable Flaim et al. model 2445 s. It is important to recall that the simulations by Lionetti et al. do not include the spatial integration component, making our timing results even more impressive in comparison.

Rocha et al. report simulations of the eight-variable LR1 and the 19-variable TP models for 500 ms for different 2D grid sizes (the largest of which was 640 x 640) using a higher spatial resolution of 0.01 cm. To compare with their results, we performed 500 ms simulations using the same domain size and spatial resolution. They report a simulation time of 11.4 minutes and 2.8 hours for the LR1 and the TP models, whereas we obtain for the BR and TP models 23.05 s and 285.56 sec on a C1060 card similar to theirs and 13.9 s and 105.4 s on a C2070 (Fermi-based) card. It is important to note that the times reported by Rocha et al. includes outputting data at unspecified intervals; for comparison, our times include outputting a byte representation of the voltage at all nodes every 1 ms.

9. CONCLUSION

In summary, we have shown that we can achieve near real-time performance of simulated cardiac dynamics in tissues of realistic sizes by using GPU architectures. To achieve the maximum gains in computational efficiency, it is necessary to consider model-specific aspects of the implementation, including appropriate division of the model among multiple kernels and careful use of the available levels of memory. The significant performance gains should facilitate implementation of novel applications of simulation, including possible use in diagnosing cardiac disease or developing patient-specific treatment strategies.

10. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation through grants. No. 0926190 and 1028261 (EMC and FHF).

11. REFERENCES

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

[2] Bueno-Orovio, A. et al. 2008. Minimal model for human ventricular action potentials in tissue. *Journal of Theoretical Biology*. 253, 3 (Aug. 2008), 544-60.

[3] Cherry, E.M. and Fenton, F.H. 2008. Visualization of spiral and scroll waves in simulated and experimental cardiac tissue. *New Journal of Physics*. 10, 12 (2008), 125016.

[4] Cherry, E.M. et al. 2007. Pulmonary vein reentry--properties and size matter: insights from a computational analysis. *Heart Rhythm: The Official Journal of the Heart Rhythm Society*. 4, 12 (Dec. 2007), 1553-62.

[5] Clayton, R.H. et al. 2011. Models of cardiac tissue electrophysiology: progress, challenges and open questions. *Progress in Biophysics and Molecular Biology*. 104, 1-3 (2011), 22-48.

[6] Dematté, L. and Prandi, D. 2010. GPU computing for systems biology. *Briefings in Bioinformatics*. 11, 3 (May. 2010), 323-333.

[7] Fenton, F.H. 1999. *Theoretical investigation of spiral and scroll wave instabilities underlying cardiac fibrillation*. Doctoral Thesis. Northeastern University.

[8] Fenton, F.H. and Cherry, E.M. 2008. Models of cardiac cell. *Scholarpedia*. 3, 8 (2008), 1868.

[9] Fenton, F. and Karma, A. 1998. Vortex dynamics in three-dimensional continuous myocardium with fiber rotation: Filament instability and fibrillation. *Chaos*. 8, (1998), 20-47.

[10] Fitzhugh, R. 1961. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*. 1, 6 (Jul. 1961), 445-466.

[11] Flaim, S.N. et al. 2006. Contributions of sustained INa and IKv43 to transmural heterogeneity of early repolarization and arrhythmogenesis in canine left ventricular myocytes. *American Journal of Physiology. Heart and Circulatory Physiology*. 291, 6 (Dec. 2006), H2617-2629.

[12] Frazier, D.W. et al. 1989. Stimulus-induced critical point. Mechanism for electrical initiation of reentry in normal canine myocardium. *The Journal of Clinical Investigation*. 83, 3 (Mar. 1989), 1039-52.

[13] Grandi, E. et al. 2010. A novel computational model of the human ventricular action potential and Ca transient. *Journal of Molecular and Cellular Cardiology*. 48, 1 (2010), 112-121.

[14] Gray, R.A. et al. 1995. Mechanisms of cardiac fibrillation. *Science (New York, N.Y.)*. 270, 5239 (Nov. 1995), 1222-3; author reply 1224-5.

[15] Gray, R.A. et al. 1998. Spatial and temporal organization during cardiac fibrillation. *Nature*. 392, 6671 (Mar. 1998), 75-8.

[16] Hoang-Trong, T.M. et al. 2011. GPU-enabled stochastic spatiotemporal model of rat ventricular myocyte calcium dynamics. *Biophysical Journal*. 100, (Feb. 2011), 557.

[17] Hodgkin, L. and Huxley, A.F. 1952. A quantitative description of membrane currents and its application to conduction and excitation in nerve. *Journal of Physiology*. 117, (1952), 500-544.

[18] Hosoi, A. et al. 2010. A multi-scale heart simulation on massively parallel computers. *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (Washington, DC, USA, 2010), 1–11.

[19] Iyer, V. et al. 2004. A computational model of the human left-ventricular epicardial myocyte. *Biophysical Journal*. 87, 3 (Sep. 2004), 1507-1525.

[20] Karma, A. 1994. Electrical alternans and spiral wave breakup in cardiac tissue. *Chaos*. 4, 3 (Sep. 1994), 461-472.

[21] Lionetti, F.V. 2010. *GPU Accelerated Cardiac Electrophysiology*. Masters Thesis. University of California, San Diego.

[22] Lionetti, F.V. et al. 2010. Source-to-Source Optimization of CUDA C for GPU Accelerated Cardiac Cell Modeling. *Euro-Par 2010 - Parallel Processing*. P. D'Ambra et al., eds. Springer Berlin Heidelberg. 38-49.

[23] Luo, C.H. and Rudy, Y. 1991. A model of the ventricular cardiac action potential. Depolarization, repolarization, and their interaction. *Circulation Research*. 68, 6 (Jun. 1991), 1501-1526.

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

[25] Molnár Jr., F. et al. Simulation of reaction-diffusion processes in three dimensions using CUDA. *Chemometrics and Intelligent Laboratory Systems*. In Press, Corrected Proof.

[26] Mosegaard, J. et al. 2005. A GPU accelerated spring mass system for surgical simulation. *Medicine Meets Virtual Reality 13: The Magical Next Becomes the Medical Now*. IOS Press. 342-348.

[27] Noble, D. 1962. A modification of the Hodgkin--Huxley equations applicable to Purkinje fibre action and pace-maker potentials. *The Journal of Physiology*. 160, (Feb. 1962), 317-352.

[28] NVIDIA CUDA Programming Guide v. 3.0: *http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf*.

[29] Potse, M. et al. 2006. A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart. *IEEE Transactions on Bio-Medical Engineering*. 53, 12 Pt 1 (Dec. 2006), 2425-2435.

[30] Puglisi, J.L. and Bers, D.M. 2001. LabHEART: an interactive computer model of rabbit ventricular myocyte ion channels and Ca transport. *American Journal of Physiology. Cell Physiology*. 281, 6 (Dec. 2001), C2049-2060.

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

[32] Sanders, J. and Kandrot, E. 2010. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley.

[33] Sanderson, A.R. et al. 2008. A framework for exploring numerical solutions of advection–reaction–diffusion equations using a GPU-based approach. *Computing and Visualization in Science*. 12, 4 (Mar. 2008), 155-170.

[34] Sato, D. et al. 2009. Acceleration of cardiac tissue simulation with graphic processing units. *Medical & Biological Engineering & Computing*. 47, 9 (Sep. 2009), 1011-1015.

[35] Scarle, S. 2009. Implications of the Turing completeness of reaction-diffusion models, informed by GPGPU simulations on an XBox 360: cardiac arrhythmias, re-entry and the Halting problem. *Computational Biology and Chemistry*. 33, 4 (Aug. 2009), 253-260.

[36] Shen, W. et al. 2009. GPU-based parallelization for computer simulation of electrocardiogram. *Computer and Information Technology, International Conference on* (Los Alamitos, CA, USA, 2009), 280-284.

[37] Shen, W. et al. 2010. Parallelized computation for computer simulation of electrocardiograms using personal computers with multi-core CPU and general-purpose GPU. *Computer Methods and Programs in Biomedicine*. 100, 1 (Oct. 2010), 87-96.

[38] Trayanova, N.A. 2011. Whole-heart modeling: applications to cardiac electrophysiology and electromechanics. *Circulation Research*. 108, 1 (Jan. 2011), 113-128.

[39] ten Tusscher, K.H.W.J. and Panfilov, A.V. 2006. Alternans and spiral breakup in a human ventricular tissue model. *American Journal of Physiology. Heart and Circulatory Physiology*. 291, 3 (Sep. 2006), H1088-1100.

[40] Vigmond, E.J. et al. 2009. Near-real-time simulations of biolelectric activity in small mammalian hearts using graphical processing units. *Conference Proceedings: Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*. 2009, (2009), 3290-3293.

[41] Witkowski, F.X. et al. 1998. Spatiotemporal evolution of ventricular fibrillation. *Nature*. 392, 6671 (Mar. 1998), 78-82.

[42] Yu, R. et al. 2009. A framework for GPU-accelerated virtual cardiac intervention. *The International Journal of Virtual Reality*. 8, 1 (2009), 37-41.

[43] Yu, R. et al. 2010. Real-time and realistic simulation for cardiac intervention with GPU. 3, (Jan. 2010), 68-72.

[44] Yu, R. et al. 2010. GPU accelerated simulation of cardiac activities. *Journal of Computers*. 5, 11 (Nov. 2010).