Title
Acceleration of cardiac tissue simulation with graphic processing units

Journal
Medical & Biological Engineering & Computing, 47(9)

ISSN
1741-0444

Authors
Sato, Daisuke
Xie, Yuanfang
Weiss, James N.
et al.

Publication Date
2009-09-01

DOI
10.1007/s11517-009-0514-4

Peer reviewed

Med Biol Eng Comput (2009) 47:1011–1015
DOI 10.1007/s11517-009-0514-4

TECHNICAL NOTE

Acceleration of cardiac tissue simulation with graphic processing units

Daisuke Sato · Yuanfang Xie · James N. Weiss · Zhilin Qu · Alan Garfinkel · Allen R. Sanderson

Received: 9 February 2009 / Accepted: 25 June 2009 / Published online: 5 August 2009

Abstract In this technical note we show the promise of using graphic processing units (GPUs) to accelerate simulations of electrical wave propagation in cardiac tissue, one of the more demanding computational problems in cardiology. We have found that the computational speed of two-dimensional (2D) tissue simulations with a single commercially available GPU is about 30 times faster than with a single 2.0 GHz Advanced Micro Devices (AMD) Opteron processor. We have also simulated wave conduction in the three-dimensional (3D) anatomic heart with GPUs where we found the computational speed with a single GPU is 1.6 times slower than with a 32-central processing unit (CPU) Opteron cluster. However, a cluster with two or four GPUs is faster than the CPU-based cluster. These results demonstrate that a commodity personal computer is able to perform a whole heart simulation of electrical wave conduction within times that enable the investigators to interact more easily with their simulations.

Keywords General-purpose computing on graphics processing units · Whole heart simulation · Excitable media

D. Sato · Y. Xie · J. N. Weiss · Z. Qu
Cardiovascular Research Laboratory, Departments of Medicine (Cardiology), David Geffen School of Medicine at UCLA, Los Angeles, CA, USA
e-mail: dasato@mednet.ucla.edu

A. Garfinkel (✉)
Cardiovascular Research Laboratory, Departments of Medicine (Cardiology), Physiological Science, David Geffen School of Medicine at UCLA, Los Angeles, CA, USA
e-mail: agarfinkel@mednet.ucla.edu
URL: http://www.cardiology.med.ucla.edu/

A. R. Sanderson
Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, USA

1 Introduction

In the last few decades, computer simulation has become an important tool to investigate various phenomena in cardiac biology, including studies of single ion channel properties [9], action potentials of the myocyte [3, 5], dynamics of action potential propagation in tissue [2], subcellular calcium dynamics [7], etc. In spite of the advancement of computational technology, the simulation of action potential waves in three-dimensional (3D) cardiac tissue with a realistic geometry is still considered as a "large-scale simulation."

General-purpose computing on GPUs (GPGPU) is a recently emerging technology [1, 4, 8], which uses GPUs, instead of CPUs, to compute large simulations in parallel. GPUs are massively parallel single instruction multiple data processing units. Each GPU may contain 128–240 "stream processors" whereas today's CPUs contain 2, 4, or 8 cores. In this paper, we demonstrate that the GPU is about 30 ~ 40 times faster than the CPU, enabling it to perform whole heart electrophysiology simulations within practical time.

In this study, we chose the simulation of the propagation of the action potential in cardiac tissue, which is modeled as the propagation of a wave in an excitable medium. Therefore, this technique can be applied to a number of phenomena in physics, chemistry, and biology.

2 Methods

We used two test models. The first was a 2D homogeneous sheet, and the second was an anatomic rabbit ventricular

model with 'fiber rotation' [10], that is, an anisotropy that varies from point to point in the heart. Each model was simulated using both the GPUs and CPUs.

The GPU simulation was performed with a single NVIDIA Geforce 8800 GT 1GB Graphic random-access memory (RAM) and an NVIDIA Geforce 9800 GX2 1GB Graphic RAM. These graphic cards were installed into a system with a dual-core 2.0 GHz AMD Opteron processor and 4GB error correction code (ECC) RAM. The operating system is OpenSUSE 10.2. Our programs are written in C++. We used GNU C++ compiler version 4.1.2 and NVIDIA CUDA version 1.1.

The CPU simulation was performed with an 8-node high performance-computing (HPC) cluster. Each node has two dual-core 2.0 GHz AMD Opteron processors (i.e., 4 cores in each node) and 4GB ECC RAM. The operating system is Fedora Core 5. We used an Intel C++ compiler 10.1. In order to parallelize on this cluster, we used Message Passing Interface 1.0. The FORTRAN version of this code was used in some of our previous studies [10].

All 2D simulations, and all 3D simulations with one GPU, were performed with the NVIDIA Geforce 8800 GT. 3D simulations with two or four GPUs were performed with the NVIDIA Geforce 9800 GX2.

Because these GPUs support only single precision, all floating-point calculations were done using single precision across both GPU and CPU simulations.

The code for the GPU is called a "kernel." When the GPU kernel code is executed, it is similar to a CPU based parallel implementation accomplished through a series of threads, with each thread running independently in parallel. Similar to a CPU implementation, it was necessary to synchronize all threads after each ordinary differential equation (ODE) or partial differential equation (PDE) kernel execution. We can then thread these intra-GPU as they control the processing within a single GPU.

In addition to having to manage threads intra-GPU, it was also necessary to have inter-GPU threads to control each GPU. For instance, the NVIDIA Geforce 9800 GX2 graphics card has two GPUs on one card. In order to utilize each GPU there must be a corresponding thread created from the main program.

As with a CPU cluster with distributed memory, it is also necessary to manage the distributed GPU memory. However, unlike a CPU where data can be moved from one CPU to another, GPUs can and must communicate with the CPU memory, that is, data is transferred from one GPU to the other GPU via the main RAM; GPU1↔RAM↔GPU2.

The cardiac tissue was modeled using the following partial differential equation:

$$\frac{\partial V}{\partial t} = -\frac{I}{C_m} + \nabla \cdot D\nabla V,$$

where V is the transmembrane voltage, I is the total ionic current, C_{m} is the transmembrane capacitance, and D is the diffusion tensor. The cell model used in this study was phase I of the Luo–Rudy action potential model [3]. We solved this reaction-diffusion equation with the forward Euler method, using the technique of operator splitting [6]. The time step was adaptively varied between 0.01 and 0.1 ms and the space step was 0.015 cm. Details of the modeling of cardiac tissue are described in our previous study [10]. For each time step, the ODE part was solved once and the PDE part was solved four times for the 2D simulation and six times for the 3D simulation (Fig. 1).

To test the GPU code, we induced spiral waves in 2D and 3D tissue using 'cross-field' stimulation, that is, two successive perpendicular rectilinear wave fronts. In each case, we simulated 1 s of real world cardiac time (Fig. 2).

For the 2D tissue simulations, the benchmark protocol involved pacing the tissue from the corner for 3 s of simulated time at a pacing cycle length of 150 ms. Tissue size was varied from 100×100 (1.5 cm $\times$ 1.5 cm) to 800×800 (12 cm $\times$ 12 cm). For the 3D tissue simulations, the benchmark protocol consisted of pacing the whole heart from the apex for 3 s of simulated time, at a pacing cycle length of 150 ms.

Finally, we investigated where the computational 'bottlenecks' occurred. We split the program into three parts, the ODE calculation, the PDE calculation, and the data

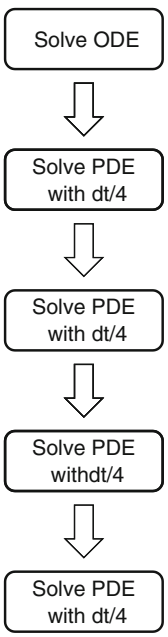

Fig. 1 Sample GPU code to solve the reaction-diffusion equation in 2D tissue. At each time step, the ODE part is called once and the PDE part is called four times. The ODE kernel code and the PDE kernel code are in the Appendix

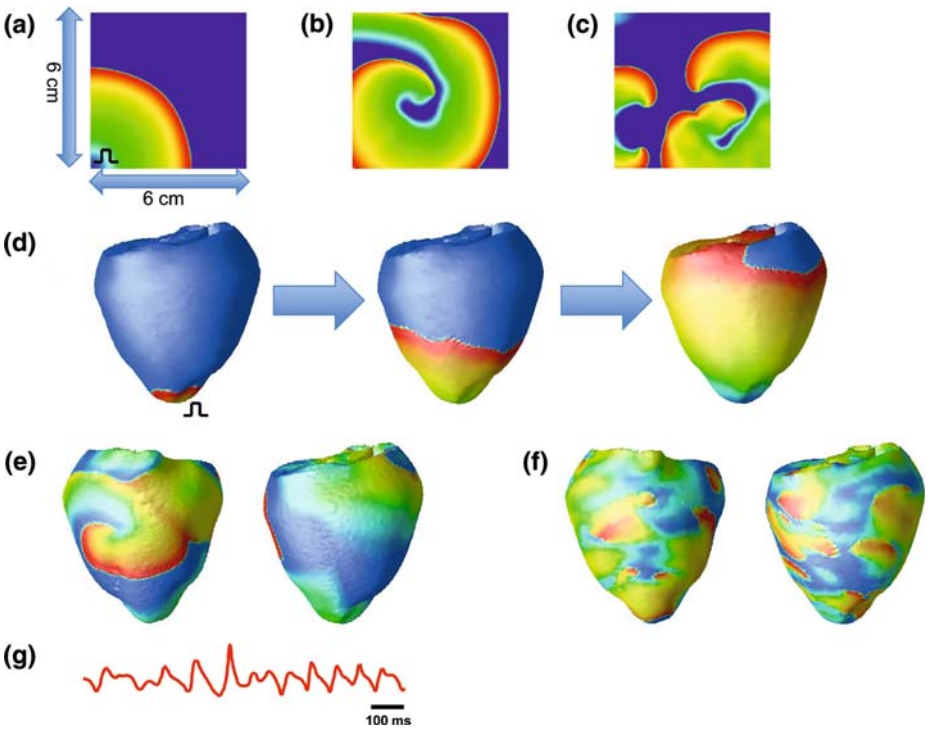

Fig. 2 Action potential propagation in 2D tissue and in the anatomic heart model. **a** Action potential propagation in 2D tissue. Tissue was placed from the corner. **b** a spiral wave in 2D tissue. The spiral wave was induced by cross-field stimulation. **c** Spiral wave breakup in 2D tissue. **d** Action potential propagation in the anatomic heart. **e** a spiral wave in the anatomic heart. **f** Spiral wave breakup. **g** Electrocardiogram from the anatomic heart simulation f

transfer. In order to measure the data transfer time, the ODE calculation and the PDE calculation were skipped, and the total time elapsed was then assigned to data transfer. Then, skipping the ODE calculation, we could measure the time for the PDE calculation plus the data transfer. The time for the PDE calculation was then estimated by subtracting the data transfer time from the (PDE + data transfer) time. The time for the ODE calculation was obtained by subtracting the (PDE + data transfer) time from the time for the whole simulation.

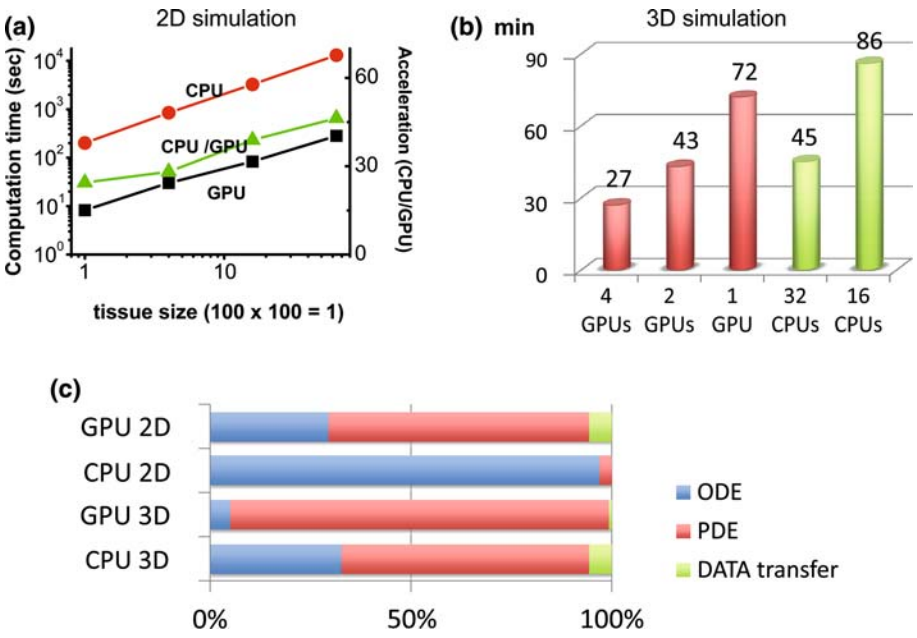

Fig. 3 Comparison between GPU and CPU. **a** Time to compute 3 s of simulation time in 2D tissue. *X*-axis is the tissue size. *Left Y*-axis is computation time. *Right Y*-axis is acceleration (i.e., computation time with CPU/computation time with GPU). **b** Time to simulate the whole heart for 1 s of the simulation time. **c** Computation ratio of the ODE, the PDE, and the data transfer for each simulation

3 Results

When tissue is homogeneous, parallel computation is very efficient. To compute 1 s of simulated time in 100 × 100 tissue, a single GPU took 8.2 s, whereas the CPU took 201 s. For larger tissue (800 × 800) the GPU took 283 s, while the CPU took 13,113 s. This is because as the tissue size becomes larger, the boundary/non-boundary ratio becomes smaller and the parallel computation becomes more efficient. In these cases, a single GPU is 24∼46 times faster than the single CPU (Fig. 3a).

To simulate the anatomic rabbit ventricular model [10] for 1 s, the HPC cluster with 32 CPUs (8 nodes) took 45 min. On the other hand, one GPU took about 72 min and two and four GPUs took about 43 and 27 min respectively for the same simulation (Fig. 3b).

The bottleneck of the computation with CPUs is mainly in the ODE part. On the other hand, the bottleneck of the computation with GPUs is mainly in the PDE part (Fig. 3c).

4 Conclusions

We demonstrate that GPUs are substantially faster than CPUs in the simulation of action potential propagation in cardiac tissue. A single GPU simulation of the whole heart is only 1.6 times slower than the simulation in an HPC cluster, and two or four GPUs are even faster than the HPC cluster, making the GPU a new tool for cardiac simulations. Utilizing GPUs poses additional programming requirements over that of traditional parallel CPU implementations. However, like parallel CPU implementations, management of threads and memory must be well thought out if maximum performance is to be achieved.

Appendix

The ODE kernel code used in 2D and 3D simulations is the following. “solve ODE(LR1)”, we solved phase I of the Luo-Rudy action potential model [3] using the forward Euler method.

```
__shared__ float vsm[256];
unsigned int tid = threadIdx.x;
unsigned int bid = blockIdx.x;
unsigned int bdim = blockDim.x;
unsigned int gdim = gridDim.x;
int step=bdim*gdim;
int num=X*Y;
for (int id=bid * bdim + tid;id<num;id+=step){
    //solve ODE(LR1)
}
```

The PDE kernel code used in 2D is the following.

```
unsigned int tid = threadIdx.x;
unsigned int bid = blockIdx.x;
unsigned int bdim = blockDim.x;
unsigned int gdim = gridDim.x;
int step=bdim*gdim;
int num=X*Y;
const float dt=0.1;
const float Dfu=0.0005;
const float dx=0.015;
const float Dfudtdx2=Dfu*dt/(dx*dx)/4;
for (int id=bid * bdim + tid;id<num;id+=step){
    if ((id)==0)
  v[id]=vold[id]+(vold[id+1]+vold[id+1]+vold[id+Y]+vold[id+Y]-
4*vold[id])*Dfudtdx2;
  else if ((id)==Y-1)
    v[id]=vold[id]+(vold[id-1]+vold[id-1]+vold[id+Y]+vold[id+Y]-
4*vold[id])*Dfudtdx2;
  else if ((id)==X*Y-Y)
    v[id]=vold[id]+(vold[id+1]+vold[id+1]+vold[id-Y]+vold[id-Y]-
4*vold[id])*Dfudtdx2;
  else if ((id)==X*Y-1)
    v[id]=vold[id]+(vold[id-1]+vold[id-1]+vold[id-Y]+vold[id-Y]-
4*vold[id])*Dfudtdx2;
  else if ((id)<Y)
    v[id]=vold[id]+(vold[id-1]+vold[id+1]+vold[id+Y]+vold[id+Y]-
4*vold[id])*Dfudtdx2;
  else if((id)>X*Y-Y)
    v[id]=vold[id]+(vold[id-1]+vold[id+1]+vold[id-Y]+vold[id-Y]-
4*vold[id])*Dfudtdx2;
  else if ((id)%Y==0)
    v[id]=vold[id]+(vold[id+1]+vold[id+1]+vold[id-Y]+vold[id+Y]-
4*vold[id])*Dfudtdx2;
  else if ((id)%Y==Y-1)
    v[id]=vold[id]+(vold[id-1]+vold[id-1]+vold[id-Y]+vold[id+Y]-
4*vold[id])*Dfudtdx2;
  else
    v[id]=vold[id]+(vold[id-1]+vold[id+1]+vold[id-Y]+vold[id+Y]-
4*vold[id])*Dfudtdx2;
}
```

References

1. Fan Z, Qiu F, Kaufman A, Yoakum-Stover S (2004) GPU cluster for high performance computing. (abstr) Proceedings of the 2004 ACM/IEEE conference on supercomputing. http://doi.ieeecomputersociety.org/10.1109/SC.2004.26
2. Garfinkel A, Kim YH, Voroshilovsky O, Qu Z, Kil JR, Lee MH, Karagueuzian HS, Weiss JN, Chen PS (2000) Preventing ventricular fibrillation by flattening cardiac restitution. Proc Natl Acad Sci USA 97:6061–6066
3. Luo CH, Rudy Y (1991) A model of the ventricular cardiac action potential. Depolarization, repolarization and their interaction. Circ Res 68:1501–1526
4. Macedonia M (2003) The GPU enters computing's mainstream. Computer 36(10):106–108
5. Mahajan A, Shiferaw Y, Sato D, Baher A, Olcese R, Xie LH, Yang MJ, Chen PS, Restrepo JG, Karma A, Garfinkel A, Qu Z, Weiss JN (2008) A rabbit ventricular action potential model replicating cardiac dynamics at rapid heart rates. Biophys J 94(2):392–410
6. Qu Z, Garfinkel A (1999) An advanced numerical algorithm for solving partial differential equation in cardiac conduction. IEEE Trans Biomed Eng 46(9):1166–1168
7. Restrepo JG, Weiss JN, Karma A (2008) Calsequestrin mediated mechanism for cellular calcium transient alternans. Biophys J 95(8):3767–3789
8. Sanderson AR, Meyer MD, Kirby RM, Johnson CR (2009) A framework for exploring numerical solutions of advection-reaction-diffusion equations using a GPU-based approach. Comput Vis Sci 12(4):155–170. doi:10.1007/s00791-008-0086-0
9. Stern MD, Song LS, Cheng H, Sham JS, Yang HT, Boheler KR, Rios E (1999) Local control models of cardiac excitation-contraction coupling. A possible role for allosteric interactions between ryanodine receptors. J Gen Physiol 113:469–489
10. Xie F, Qu Z, Yang J, Baher A, Weiss JN, Garfinkel A (2004) A simulation study of the effects of cardiac anatomy in ventricular fibrillation. J Clin Invest 113(5):686–693