GPU Acceleration of Transmural Electrophysiological Imaging

M Corraine[1], S Lopez[1], L Wang[2]

[1]Department of Computer Engineering, [2]Computing and Information Sciences,
Rochester Institute of Technology, Rochester, NY, USA

Abstract

Tranmural electrophysiological imaging (TEPI) is becoming a possibility with the aid of 3D in silico *cardiac EP models and the statistical estimation theory. By* quasi *Monte-Carlo (MC) simulation of the 3D EP models on the subject-specific anatomical model, complex and physiologically meaningful spatiotemporal priors are produced to achieve the 2D-to-3D transition of EP data, an inverse problem that otherwise has no unique solutions. While it is acknowledged that macroscopic phenomenological models are more suited for the purpose of inverse problems, the nature of these algorithms still incurs tremendous computational cost that hinders their clinical translation, particularly caused by the MC simulation of high-dimensional, nonlinear EP models and large-matrix operations involved in probabilistic estimation. In this paper, we explore the use of Graphic Processing Units (GPU) to accelerate TEPI because of its high parallelism and large bandwidth. While initial steps have been taken towards GPU acceleration of whole-heart EP simulation using complex, ionic models, few effort is reported on GPU acceleration of subject-specific, data-driven EP imaging. In this study we will show how we take advantage of the high level of parallelism available in the hardware resources in GPUs and achieve a preliminary but important* $16\times$ *speedups compared to the most high-end CPU version of TEPI. We also present benchmarking on 3 different GPU devices, point out the bottlenecks that limit the performance, and give guidelines on balancing the cost* versus *performance tradeoffs for clinical and researcher environments.*

1. Introduction

The need of noninvasive electrophysiological (EP) imaging of the heart motivated many computational research that use *body-surface electrocardiographic data* and *image-derived anatomic data* to compute subject-specific cardiac EP details [1–5]. The most widely-applied methods compute EP signals on the pericardium that surrounds the heart [1,3] or on the ventricular surface [2,5]. In consequence, EP or substrate information is not yet available along the depth of the myocardium. By considering current density within the myocardium, transmural activation isochrones were reconstructed in healthy hearts [4]. However, repolarization phenomena are not considered, which constitute the primary features for evaluating arrhythmia vulnerability and localizing EP substrates [6, 7].

In the context of probabilistic estimation theory, we exploited the power of computational models in providing physiological meaningful spatiotemporal priors for the Bayesian estimation, and developed a unique technique of *noninvasive transmural electrophysiological imaging* (TEPI) that computes subject-specific action potential dynamics not only on heart surfaces but deep into the 3D myocardium of the ventricles [8]. Feasibility of TEPI was preliminarily verified in several *in-vivo* experiments on large-animal models (healthy and with chronic infarction) [9] and human subjects (with prior myocardial infarction) [10, 11], with reference data in forms of *in-vivo* surface EP mapping and/or DCE imaging.

Despite the initial success, the clinical translation of TEPI is hindered by several critical obstacles, one of which lies in the tremendous computational cost that underlies the algorithm. Though employing the 2-variable *Alieve-Panfilov* model [12] ($\sim$2 minutes per simulation), an averaged computational time of 250 hours on a regular desktop was reported [9]. In this paper, we explore the use of Graphic Processing Units (GPU) to accelerate TEPI. On one hand, the type of algorithm being considered [8] is highly parallel, typically in terms of the *quasi* Monte-Carlo simulation of EP models and the large matrix operations. GPUs are ideal candidates to process this parallel workload because of their hundreds of simplisitic but highly threaded execution cores. On the other, commodity GPUs also provide a unique opportunity of deployment on desktop workstations and laptop computers in clinical settings. This excludes the needs to manage remote high performance computing resources, which in comparison would render substantial expense and expertise demands. While initial steps have been taken towards GPU acceleration of whole-heart EP simulation using complex, ionic models [13, 14], few effort is reported on GPU acceleration of noninvasive EP imaging.

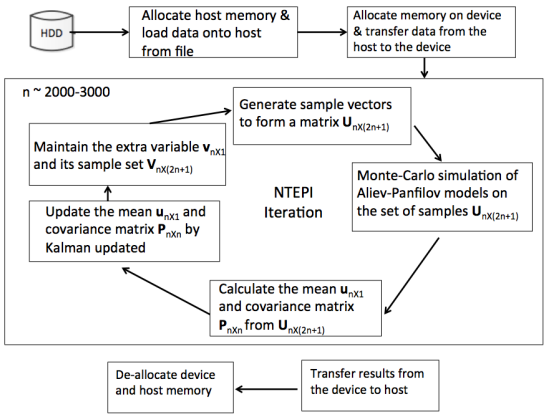

Figure 1: NTEPI diagram describing the algorithm flow, major functioning components and major involved matrix structures.

We will show how we handle the hardware resources available in the GPU and its communication with the CPU for the algorithm, and the preliminary but important $16\times$ speedups compared to a high-end CPU version of TEPI. GPUs count with double precision floating point arithmetic units to fully respect the accuracy of the calculations. In order to provide a guideline for the most cost-effective GPU choices for clinical and research environments, we also focus on benchmarking the speedups versus dollar costs among the wide variety of commodity GPUs. We will compare our GPU implementation of TEPI algorithms on 3 different systems, ranging from low-end cards to a top-of-the-line card. We will benchmark the changes of speedup in these cards and point out the bottlenecks that dollars cannot overcome.

2. Method

GPUs are application specific processors designed to render graphics, a highly data parallel computation. Because of their highly parallel nature, it was soon obvious that not only graphic applications could take advantage of this type of architectures. The Compute Unified Device Architecture (CUDA) was created by Nvidia to better suit general purpose computation on GPUs (GPGPU) [15]. Code runs on the device (GPU) by having the host (CPU) invoke a kernel, a method that executes on the device. When a kernel is called, a grid of blocks is created. The blocks contain light-weight hardware-managed threads.

NTEPI Description and Profiling: NTEPI employs a sequential a MAP estimation of the transmural action potential distributions $\mathbf{u_k}$ given the body-surface potential data up to the current iterations $\phi_{\mathbf{1:k}}$. The main computation of each NTEPI iteration can be broken down into five main components as shown in Fig 1.

Before invoking the sequential MAP estimation, a Cholesky decomposition of the covariance matrix of $\mathbf{u_0}$ is performed. In each iteration, a set of sample vectors $\mathcal{U}_{\|-\infty}$ are generated from the mean and covariance matrix of $\mathbf{u_{k-1}}$. $\mathcal{U}_{\|-\infty}$ individually enters into simulation of the *Alieve-Panfilov* models [12] to predict a new set of sample vectors $\mathcal{U}_{\|}^{-}$, from which the mean and covariance matrix of $\mathbf{u_k^-}$ are predicted.

Next, the input body-surface data ϕ_k enters into the algorithm and, by Kalman filtering rules, updates the predicted statistics into the posterior statistics (mean and covariances). This update computation requires inverting an $m \times m$ matrix where m is the dimension of the body-surface data ϕ_k. In practice m is often of limited size ($< 10^2$). Each iteration repeats between the above prediction and updates of the statistics of $\mathbf{u_k}$. A special condition of EP applications is the existence of additional unknown variables in the EP models. In the case of NTEPI, the *Aliev-Panfilov* model [12] has an additional variable of recovery current. The same number of sample vectors of this variable need to be maintained to pair up with the set of $\mathbf{U_k}$. Its statistics, however, are not tracked or updated.

In a word, common matrix operations in each iteration include addition, subtraction, element-wise and standard multiplication, scaling, inversion, and Cholesky decomposition. In order to further understand the computational cost of these components, the algorithm (CPU verion) is profiled with Microsoft Visual Studio 2010 instrumentation profiler on a synthetic experiment. The results show that the quasi Monte-Carlo simulation on the *Aliev-Panfilov* model [12] is responsible for the majority of the execution time. Since it is the simulation of the same model on a set of sample vectors $\mathcal{U}$, it is expected to be able to take advantage of the GPU's hardware resources.

GPU Acceleration Strategy: Based on the profiling results we can extract that matrix operations compose the majority of this algorithm.

In our implementation we aim to virtually remove all communication between the host and device to avoid its long latency.. The design, as shown in Figure **??**, begins with the host loading data from files and placing the data into device memory. From this point, all the data structures reside in device memory. Then, the host enters the main loop of TEPI. It is here where the device begins to execute. The host mainly instructs the device what to compute. Once the loop terminates, the host will transfer the results from the device to the host and clean up.

Sometimes, the host can execute an algorithm much faster because the algorithm fits the CPUs architecture much better. Cholesky factorization is one such example. But the improvement has to outweigh the added data transfer time. On the other hand, executing an inefficient kernel on the GPU could improve performance since transferring data was avoided.

Standard Libraries and Custom Kernels: Since linear algebra operations dominate the algorithm, an im-

plementation of the Basic Linear Algebra Subprograms (BLAS) library called Compute Unified BLAS (CUBLAS) version 4.1 is used to accelerate the algorithm [16]. Another library used was the Matrix Algebra on GPU and Multicore Architecture (MAGMA) to accelerate those non-basic linear algebra operations such as Cholesky factorization.

Custom written kernels were also used to parallelize the code when CUBLAS did not implement the required operation. In addition, kernels were written to replace multiple calls to the CUBLAS library. To give an example, using a custom kernel compared to three CUBLAS library calls achieved a speedup of 1.81 ($3.42ms$ by the custom kernel *vs.* $6.21ms$ by 3 CUBLAS library calls). The CUBLAS calls were consolidated into these custom kernel calls as much as possible.

3. Results and discussion

In order to evaluate the cost *versus* performance of different GPUs, We have three different CPU+GPU systems that we will use for our evaluation. Table 1 presents the specifications of all three CPU+GPU systems. We ran our CUDA implementation of TEPI in all three systems and compared the performance against the best CPU-only system of the three.

Table 1: Systems' specifications

System A
CPU: Intel i7-2600, 4 cores with hyperthreading, 3.4GHz GPU: Tesla C2070, Compute Capability 2.0, 14 SM cores, 448 CUDA cores, 144 GB/s Memory Bandwidth, 575/1150MHz Graphics/processor clock
System B
CPU: Intel i5, 2 cores with hyperthreading, 3.1GHz GPU: GTX 480, Compute Capability 2.0, 15 SM cores, 480 CUDA cores, 177.4 GB/s Memory Bandwidth, 700/1401 MHz Graphics/processor clock
System C
CPU:AMD Anthlon 64 5600+, 2 cores, no hyperthreading, 2.9GHz GPU: GTX 295, Compute Capability 1.3, 2x30 SM cores, 2x240 CUDA cores, 2x111.9 GB/s Memory Bandwidth, 576/1242 MHz Graphics/processor clock

We focus on evaluating the performance of our GPU acceleration of TEPI on a real-data experiment on a porcine heart with chronic infarction (approximately 5-6 weeks old). The comprehensive dataset includes *in-vivo* electroanatomic voltage and activtion maps acquired on the pig heart by CARTO-XP system (Biosense Webster, Inc., Diamond Bar, CA) when the animals were in sinus rhythm, and *ex-vivo* DW-MRI of the ventricles with $< 1^{mm^3}$ voxel size acquired on a 1.5T GE Signa-Excite scanner. These DW-MR images reveals anatomical data of the ventricles including geometry, fiber, and scar morphology. Here input data for TEPI include MR-derived anatomical data

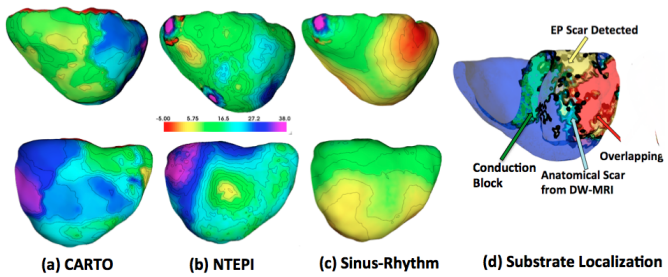

Figure 2: GPU TEPI outputs in forms of detected EP substrates (d) and epicardial isochrones (b) in comparison to CARTO measurements (a) and sinus-rhythm simulation assuming healthy tissues (c).

(geometry and fiber incorporated; scar not incorporated) and CARTO-recorded unipolar electrograms on the epicardium. DW-MR scar data and epicardial and endocardial activation maps are used to validate TEPI outputs.

Fig 2 compares the epicardial isochrone computed by TEPI in GPUs (a) with that measured by the CARTO-XP system (b) as well as that simulated assuming sinus-rhythm excitation with healthy tissue property on the same porcine heart (c). It is evident that TEPI is able to reconstruct two types of electrical dysfunctions that can be validated by the CARTO maps and the DW-MRI. First, there is an evident delay and absence of activation at inferior-lateral LV, consistent to the scar location delineated from DW-MRI and further confirmed by the delay/absence of activation on the CARTO maps. Second, there is a sequential RV-to-LV conduction pattern that replaces the otherwise simultaneous ventricular conduction in the sinus-rhythm conditions and resembles the left bundle branch block mapped by CARTO systems. These two clusters of diseased EP substrates detected by TEPI are shown transmurally in Fig 2 (d), superimposed with the DW-MRI enhanced scar. The difference between GPU and CPU TEPI outputs are in the order of numerical errors (maximum relative error: $1.210^{-}6$).

The execution time of the best CPU system (System A) is 1931 minutes ($>$ 32 hours), and it will be our baseline for comparison. The CUDA implementation on System A, B, and C run 16.1, 8.9 and 3.6 times faster than our baseline respectively. The Tesla C2070 GPU card of system A achieves almost twice the performance of System B with the GTX 480 GPU card, and four times the speedup of System C with the GTX 295 GPU card. The GTX 295 card of Sytem C has limited compute capabilities (its compute capability is 1.3 vs. 2.0 of the other two systems' cards) and for that reason its performance is worse. If we look at the results from the point of view of the cost/performance tradeoff we notice that the cost of the Tesla C2070 card is aproximately five times of the GTX 480, and about ten times the cost of the GTX 295. The performance acceleration of System A is not up to the cost increment, neither it is matching the expectations of its GPU card.

TEPI requires the use of double precision arithmetic,

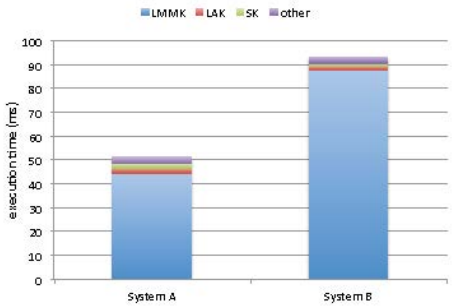

Figure 3: Percentage of device usage time for the *Quasi Monte-Carlo Sim.* function.

due to the small magnitude of the input data, and the requirements of performing matrix inversion per iteration. This has a negative effect on the performance of the GPU The performance of TEPI accelearted with CUDA is very dependent on matrix multiplication in the Monte-Carlo runs of EP models.

To look at this function in more detail we run 3 iterations of it in systems A and B (Tesla C2070 and GTX 480 respectively).We can break this function down into different types of kernels, the main ones being *Large Matrix Multiply Kernel*, *Linear Algebra Kernel*, and *Scale Kernel*. Figure 3 shows the execution time on the device in both systems. The majority of the savings come from the *LMMK* function, highly dependent on the double floating point functional units. Despite the better double precision logic in the Tesla C2070, the time that this specific function takes does not go any lower than half the time of the same opearation in the GTX 480 of B. As long as this operation's performance is not improved, the overal speedup cannot go beyond two. On the other hand, the rest of the operations, less dependent on the double floating point operaions, increase their time in System A. This is due to the fact that System B has one more Stream Multiprocessor than System A, higher memory bandwidth, and higher graphics/processor clocks.

In summary, the Tesla card is considered a research card, specifically aimming for scientific operations that require high precision. Our application is one that requires double precision floating point multiplication, and therefore we expected a dramatic increase in performance of System A vs System B. But as shown in Figure 3 we can see that this large matrix multiplications are still a major bottleneck in System A.

References

[1] Rudy Y, Messinger-Rapport B. The inverse problem of electrocardiography: solutions in terms of epicardial potentials. CRC Critical Reviews on Biomedical Engineering 1988;16:215–268.

[2] Huiskamp G, Greensite F. A new method for myocardial activation imaging. IEEE Transactions on Biomedical Engineering 1997;44(6):446.

[3] Zhang Y, Ghodrati A, Brooks DH. An analytical comparison of three spatio-temporal regularization methods for dynamic linear inverse problems in a common statistical framwork. Inverse Problems 2005;21:357–382.

[4] Liu Z, Liu C, He B. Noninvasive reconstruction of three-dimensional ventricular activation sequence from the inverse solution of distributed equivalent current density. IEEE Transactions on Medical Imaging 2006;25(10):1307–1318.

[5] van Dam PM, Oostendorp TF, Linnenbank AC, van Oosterom A. Noninvasive imaging of cardiac activation and recovery. Annals of Biomedical Engineering 2009; 37(9):1739–1756.

[6] Chauhand VS, Downar E, Nanthakumar K, Parker JD, Ross HJ, Chan W, Picton P. Increased ventricular repolarization heterogeneity in patients with ventricular arrhythmia vulnerability and cardiomyopathy: a human in vivo study. America Journal of Physiology Heart and Circulatory Physiology 2005;290:H79–H86.

[7] Ghosh S JNS, Canham RM, Bowman TM, Zhang J, Rhee EK, Woodard PK, Rudy Y. Electrophysiologic substrate and intraventricular left ventricular dyssnchrony in nonischemic heart failure patients undergoing cardiac resynchronization therapy. Heart Rhythm 2011;8:692–699.

[8] Wang L, Zhang H, Wong K, Liu H, Shi P. Physiological-model-constrained noninvasive reconstruction of volumetric myocardial transmembrane potentials. IEEE Transactions on Biomedical Engineering 2010;5(2):296–315.

[9] Camara O, Sermasant M, Lamata P, Wang L, et al. Inter-model consistency and complementarity: Learning from ex-vivo imaging and electrophysiological data towards an integrated understanding of cardiac physiology. Progress in Biophysics and Molecular Biology 2011;107:122–133.

[10] Wang L, Zhang H, Wong K, Liu H, Shi P. Noninvasive computational imaging of cardiac electrophysiology for 3-d infarct. IEEE Transactions on Biomedical Engineering 2011;58(4):1033–1043.

[11] Wang L, Dawoud F, Wong K, Zhang H, Liu H, Sapp J, Horacek M, Shi P. Mapping the transmural scar and activation for patients with ventricular arrhythmia. Computing in Cardiology 2011; 849–852.

[12] Aliev RR, Panfilov AV. A simple two-variable model of cardiac excitation. Chaos Soliton Fract 1996;7(3):293–301.

[13] Sato D, Xie Y, Weiss JN, Qu Z, Garfinkel A, Sanderson AR. Acceleration of cardiac tissue simulation with graphic processing units. Med Biol Eng Comput 2009;47:1011–1015.

[14] Bartocci E, Cherry EM, Glimm J, Grosu R, Smolka SA, Fenton FH. Toward Real-Time Simulation of Cardiac Dynamics. In Proc. International Conferences on Computational Methods in Systems Biology. 2011; 103–112.

[15] Corporation N. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. NVIDIA Corporation, 2007.

[16] Corporation N. CUDA Toolkit 4.0. CUBLAS Library. NVIDIA Corporation, 2011.