Cocoro: Fast Simulation of Cardiac Electrophysiology with WebGPU

Ricardo M Rosales[1,2], Ana Mincholé[1,2], Esther Pueyo[1,2]

[1] Aragon Institute of Engineering Research, University of Zaragoza, IIS Aragon, Zaragoza, Spain
[2] CIBER in Bioengineering, Biomaterials & Nanomedicine (CIBER-BBN), Spain

Abstract

As the complexity of in silico cardiac electrophysiological models increases, so do the computational costs. To address this, we present Cocoro, a cutting-edge WebGPU-based solver for efficient computation and rendering. We describe our implementation and benchmark its accuracy, performance, and clinical potential. Our results were compared with those of openCARP (OC) on a porcine biventricular mesh under anisotropic (A) and isotropic (I) setups. Cocoro's execution time (ET) for 5-second simulations was measured on two GPUs with and without extra graphical information (EGI), including visualizations of pseudo-electrocardiograms and cellular data. In addition, we compared the simulated and experimental QRS complexes to showcase clinical applicability. Cocoro simulations closely aligned with those of OC, with the 90th percentile (P_{90}) of node-wise activation time differences of 11 ms (A) and 4 ms (I). For repolarization times, P_{90} remained below 6 ms for both A and I. The five-second simulations ran in 9.54 and 22.13 min on RTX 2070 and Titan V GPUs, respectively. Enabling EGI had a minimal impact on ETs. Furthermore, the simulated QRS complexes reproduced the experimental QRS morphologies and durations. Thus, Cocoro enables fast, portable, and accurate fully GPU-resident cardiac electrophysiological simulations.

1. Introduction

In silico cardiac electrophysiology can accurately reproduce healthy and pathological states [1, 2], allowing the study of cardiac electrical disturbances and therapeutic targets without costly clinical trials [1,3].

Driven by the need to investigate a wider spectrum of physiological conditions and accompanied by experimental breakthroughs, simulation complexity has significantly escalated and now it requires expensive supercomputers running for long hours to answer critical open questions, such as anatomically-based QRS variations [4]. To keep simulations tractable, efficient software is needed that runs on cutting-edge hardware. In this context, graphical processing units (GPUs) have outperformed traditional CPU-based implementations of *in silico* cardiac electrophysiology models [1,5], due to their architecture's ability to handle massive parallelized workloads efficiently.

Increased model complexity compromises the simplicity and portability of solvers aimed at simulating those models, thereby hindering broader usability. To prevent this, an increasingly popular approach is the development of web apps that allow cross-platform compatibility and user-friendly abstraction [1, 6]. However, web apps have extra communication layers, which can hamper performance. Concerning this, a recently developed web standard called WebGPU enables GPU-based acceleration. Unlike its predecessor WebGL, WebGPU not only supports GPU-based rendering but also natively allows GPU-based computations. Briefly, GPU threads executing a main script in parallel are organized into workgroups (WGs), and the total number of WGs defines the workload (WL) sent to the GPU for processing. The number of threads within a single WG defines the WG size (WGS) and its product with the number of WGs in the WL is the total number of threads invoked by an app.

Here, we present Cocoro, a WebGPU-based implementation for fast and portable cardiac electrophysiology simulations. Our solver incorporates GPU-based computation and rendering of extra graphical information (EGI), such as pseudo-electrocardiograms (pECGs) and cellular-level variables. In addition, it allows interactive stimulation and parameter updates as in [1]. In the following, we describe our implementation and the evaluation performed in terms of accuracy, performance, and clinical applicability.

2. Methodology

2.1. Implementation

Figure 1 illustrates the app's workflow (a) and class relations in our object-oriented design (b). We aimed to minimize GPU calls and execute all computations and rendering directly on the GPU. The source code is available at `https://github.com/lino202/cocoro`.

A *Django* server hosts three main interfaces: the Up-

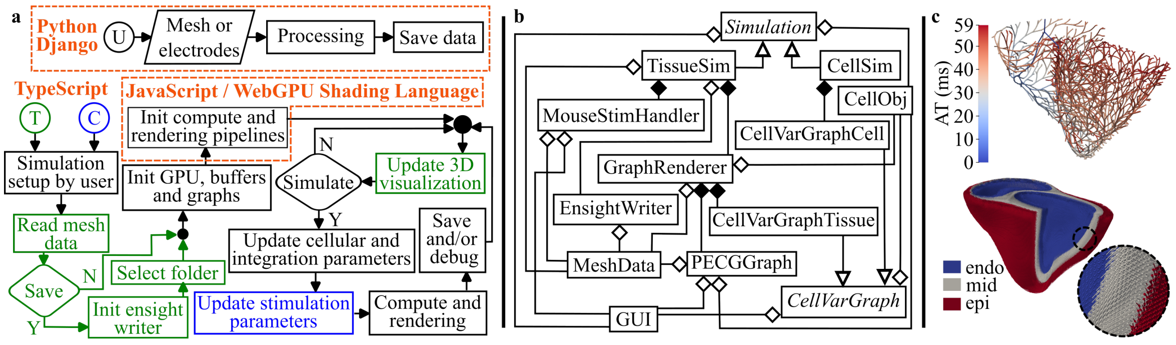

Figure 1: a- Flux diagram. b- Conceptual class diagram. c- CS activation (top) and BiV fiber field (bottom).

load (U), Tissue (T) and Cellular (C) web pages. In U, the user uploads meshes containing electrode positions for pECGs computation or defining the simulation domain, its node-wise stimulation, and fiber orientations. Once saved and processed, these data are available in T for simulation, where users can configure the cellular model, EGI, interactive stimulation, and select options for saving or debugging results. Firstly, a *TissueSim* object is created and, depending on the user selection, it may instantiate *MouseStimHandler* for interactive stimulation and *EnSightWriter* for saving results in the EnSight Gold format with the help of web workers. When any EGI is enabled, a *GraphRenderer* instance handles the *PECGGraph* computation and rendering of pECGs and the *CellVarGraphTissue* plotting of any state variable at a specified location. Cellular simulation is conducted in C, with computation handled by the *CellSim* class and state variable rendering managed by the *CellVarGraphCell* class. *MeshData* and *CellObj* objects store mesh and cellular information, respectively.

After a *Simulation* initiated the GPU, the EGIs and the communicating buffers, the main simulation loop begins. Each iteration involves one rendering step and $plot_{dt}/dt$ computation steps, where dt is the time integration step and $plot_{dt}$ is the rendering interval set by the user. Consequently, the simulation update is constrained by the computation load and the display's refresh rate. For numerical integration, time derivatives are computed using the explicit Euler method, while gating cellular variables are integrated with the Rush-Larsen method. For diffusion, spatial derivatives are solved with a central finite difference scheme and zero-flux boundary conditions are imposed by setting ghost nodes in the domain boundaries, as in [1, 7].

2.2. Data races

In parallel computing, read and write operations must be carefully synchronized to avoid data races. WebGPU offers thread-safe atomic operations and synchronization barriers. Nevertheless, atomic operations are limited to integers, and synchronization barriers are only

Algorithm 1 Thread-safe WG-wise array reduction

```
1: procedure REDUCEARRAY($A$, $A_r$, $WGS$)
2:     $D_{wg}[Id_l] \leftarrow A[Id_g]$
3:     $workgroupBarrier()$                    ▷ wait all WG threads
4:     for ($cs \leftarrow WGS, cs > 0, cs/2$) do
5:         if $Id_l < cs$ then
6:             $D_{wg}[Id_l] \leftarrow \mathbf{F}(D_{wg}[Id_l], D_{wg}[Id_l + cs])$
7:         $workgroupBarrier()$                ▷ wait all WG threads
8:     if $Id_l == 0$ then
9:         $A_r[Id_{wg}] \leftarrow D_{wg}[0]$
```

WG-wise thread-safe, restricting synchronization when floating-point operations and large meshes are used.

In Cocoro, potential data races arise during spatial derivative determination, data shift for graph update, pECG computation, and interactive stimulation. For the first two cases, data races are avoided by using a copy buffer of the array exposed to read-write operations, i.e., the copy is read, the original is written, and then the copy is updated. As in [4], the pECG is obtained from the extracellular potential $\phi(\vec{e}) = \sum_{i=1}^{N} \phi_i(\vec{e})$, where $\vec{e}$ is the position of the electrode, $\phi_i(\vec{e})$ is the node-wise contribution and N is the number of nodes. Thus, a thread-safe calculation of $\phi_i(\vec{e})$ and the posterior sum is required, since N is often much larger than the maximum synchronizable WGS. Thus, we used the reduction algorithm shown in 1, where Id_l, Id_{wg} and Id_g are the local, WG and global thread indices, respectively. Briefly, the array A contains N $\phi_i(\vec{e})$ values. For $Id_{wg} = 0$, a WG-level array D_{wg} is loaded with the first WGS elements of A. Then, an element-wise sum F is applied between the first and second halves of A, with the results stored in the first half. After reduction, $D_{wg}[0]$ is the sum of the first WGS elements of A saved to the element Id_{wg} of the reduced array A_r. Thus, A_r has N_{wg} partial sums, where N_{wg} is the number of WGs dispatched when the maximum WGS is used. Since $N_{wg} << N$, the total sum can be drawn with a simple

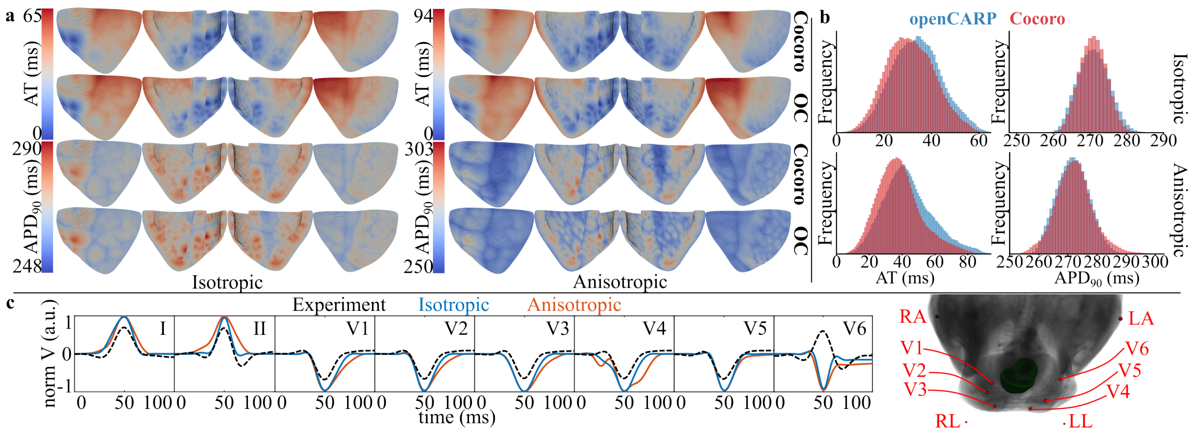

Figure 2: AT and APD$_{90}$ maps (a) and histograms (b) for the I and A setups. c- Experimental and simulated QRS complexes (left) and electrodes-heart relative positions (right).

loop or with additional reductions until $N_{wg} = 1$. During interactive stimulation, F is a minimum operation, which computes the shortest distance between mesh elements and a click-defined ray to determine the stimulation point.

2.3. Accuracy and performance evaluation

To evaluate Cocoro, an *ex vivo* magnetic resonance image (MRI) of a porcine heart was used to reconstruct the biventricular (BiV) geometry at two resolutions (Figure 1c). The high (BiV$_h$) and low (BiV$_l$) resolution hexahedral meshes consisted of 1 334 286 nodes (400 µm edge length) and 387 539 nodes (600 µm edge length), respectively. The fiber field and the species-specific conduction system (CS) were generated as in [3]. Electrical propagation was computed by numerically integrating the monodomain model. BiV nodes within 1 mm of the CS endpoints (Purkinje-muscular junctions) were stimulated using activation times (ATs) from a CS simulation obtained with the gold-standard simulator openCARP (OC). Specifically, stimuli of 80-$\mu\mathrm{A}/\mathrm{cm}^2$ magnitude, 1-ms duration and 1000-ms period were applied. Moreover, the Gaur et al. [2] cellular model was set in the entire BiV domain, considering both isotropic (I) and anisotropic (A) conductivity profiles. The conductivity along the cardiac fibers was set at 0.12 $\mathrm{S/m}$ and its transverse-to-longitudinal ratio was set at 1 and 0.3 for the I and A cases, respectively.

We first performed 3-s simulations using the BiV$_h$ for both I and A cases. The values of AT and action potential duration at 90% repolarization (APD$_{90}$) were drawn from the last beat [3]. The accuracy of our solver was evaluated by comparing these results with those obtained using OC [6]. Both solvers used a *dt* of 20 µs and equal A and I profiles. Secondly, the execution time (ET) was measured for 5-s simulations in Cocoro with BiV$_{h/l}$. Performance was examined using two NVIDIA GPUs (RTX 2070 Super and Titan V) and enabling or disabling the computation and rendering of EGI. Lastly, we qualitatively assessed clinical potential by comparing simulated and experimental QRS complexes after low-pass filtering (55 Hz cutoff frequency). The positions of the electrodes were derived from a thoracic MRI acquired in an anatomically similar pig and aligned to our BiV$_h$ through a rigid transformation (see Figure 2c). Since our BiV model was based on an *ex vivo* MRI that exhibited tissue thickening and blood pool collapse, we evaluated the morphology of the QRS complex after per-lead normalization and peak-based alignment.

3. Results and discussion

Cocoro showed strong agreement with the OC simulations, used as a benchmark, in terms of depolarization-repolarization dynamics. As illustrated in Figures 2a and 2b, Cocoro resulted in only slightly faster depolarization and a somewhat more dispersed repolarization pattern, especially for the A case. Quantitatively, the median difference of node-wise ATs between the two solvers was 5 ms for case A and 2 ms for case I, with the corresponding 90th percentile (P$_{90}$) being 11 and 4 ms. This AT discrepancy is hypothesized to arise from differences in integration methods: OC employed a semi-implicit (forward Euler and Crank-Nicolson) scheme with the finite element method for temporal and spatial integrations, respectively. For repolarization, APD$_{90}$ median differences were negligible, and the P$_{90}$ values reached 6 and 2 ms for A and I, respectively. The slightly higher discrepancies observed in the A scenario may be due to limitations of the finite difference method, particularly when imposing zero-flux boundary conditions in the A cases [7]. Nevertheless, Cocoro mimicked the conductivity-based variations and patterns observed in OC, with reduced dispersions of AT and APD$_{90}$ in the I case. Overall, Cocoro closely reproduced the depolarization-repolarization results, despite their implementation differences, particularly regarding the numerical integration methods.

In terms of performance, a 5-s simulation took 2.92 min for the $\mathrm{BiV_l}$ mesh and 9.54 min for the $\mathrm{BiV_h}$ mesh on the RTX GPU, whereas on the Titan GPU, the ETs were 6.35 and 22.13 min, respectively. Thus, Cocoro achieved fast integration of the monodomain model using a biophysically detailed cellular model across various mesh sizes and hardware architectures. In addition, ETs remained stable even when computing and rendering the EGI. This remarks on the efficiency achieved by our solution, mainly due to the implementation of the reduction algorithm (Section 2.2), which enabled the fast GPU-based pECG calculation while circumventing data-races. Comparison of performance between solvers is challenging and sometimes unfair, due to differences in numerical methods, software technology, target hardware, and scope of development. For instance, on an MSI Raider GE66 laptop, OC required 10 hours on 8 CPU cores for a 3-s simulation with $\mathrm{BiV_h}$, whereas Cocoro completed the equivalent 5-s simulation in 9.54 min. In this direction, a recent study [1] reported similar ETs as our approach when solving a similar problem using a WebGL-based solver. With WebGPU showing potential to outperform WebGL [8], there is room for optimization in our implementation.

To demonstrate clinical applicability, we compared the simulated pECGs with an experimental ECG. As shown in Figure 2c, the *in silico* and experimental QRS complexes aligned closely on nearly all leads, especially for the I case. Discrepancies in leads such as V6 may stem from geometric differences between the simulated and experimental pig data (see Section 2.3). Note that we computed biomimetic QRS complexes in most leads without CS personalization, which could lead to improved results. Future work could include cellular-level spatial heterogeneities, allowing deeper analysis of ventricular repolarization. Moreover, an extended validation and convergence analysis will further strengthen our implementation.

Overall, this work enabled WebGPU-based fast and accurate cardiac electrophysiology simulations. Our implementation differs from current GPU-based solutions by being fully GPU-resident and seamlessly providing web-mediated cross-platform usability, fast clinically relevant pECG calculation, and the potential for straightforward integration with WebRTC and WebXR, enabling multi-GPU partitioning and immersive applications [1, 9].

4. Conclusion

We presented Cocoro, our WebGPU-based implementation for cardiac electrophysiology simulation. Leveraging fully GPU-resident computation and rendering, Cocoro achieved high-performance and interactive simulations that are easily portable across diverse GPU architectures. Deployed as a web application, Cocoro reduces technical barriers, allowing users to run complex simulations without requiring extensive programming expertise.

Acknowledgments

This work was supported by EU H2020 Program under G.A. 874827 (BRAV3), by Agencia Estatal de Investigación - Ministerio de Ciencia e Innovación (Spain) through projects PID2022-140556OB-I00, CNS2022-135899 and TED2021-130459B-I00, by European Social Fund (EU) and Aragón Government through BSICoS group T39_23R. The calculations were performed using ICTS NANBIOSIS (HPC Unit at University of Zaragoza).

References

[1] Kaboudian A, et al. Fast interactive simulations of cardiac electrical activity in anatomically accurate heart structures by compressing sparse uniform cartesian grids. Computer Methods and Programs in Biomedicine 2024;257:108456.

[2] Gaur N, et al. A computational model of pig ventricular cardiomyocyte electrophysiology and calcium handling: Translation from pig to human electrophysiology. PLoS Computational Biology 2021;17(6):e1009137.

[3] Rosales R, et al. In silico assessment of arrhythmic risk in infarcted ventricles engrafted with engineered heart tissues. Computing in Cardiology November 2023;50:1–4.

[4] Mincholé A, Zacur E, Ariga R, Grau V, Rodriguez B. MRI-based computational torso/biventricular multiscale models to investigate the impact of anatomical variability on the ECG QRS complex. Frontiers in Physiology 2019;10:1103.

[5] Neic A, et al. Accelerating cardiac bidomain simulations using graphics processing units. IEEE Transactions on Biomedical Engineering 2012;59:2281–2290.

[6] Plank G, et al. The openCARP simulation environment for cardiac electrophysiology. Computer Methods and Programs in Biomedicine 2021;208:106223.

[7] Heidenreich E. Algoritmos para ecuaciones de reacción difusión aplicados a electrofisiología. PhD Thesis, Universidad de Zaragoza, 2009.

[8] Chickerur S, et al. WebGL vs. WebGPU: A performance analysis for web 3.0. Procedia Computer Science 2024; 233:919–928.

[9] Marins Ramalho De Lima L, et al. MonoWeb: Cardiac electrophysiology web simulator. In Computational Science – ICCS 2024, volume 14835. Springer Nature Switzerland, 2024; 147–154.

Address for correspondence:

Ricardo M. Rosales
University of Zaragoza, Campus Río Ebro, I+D Building, D-5.01.1B, Mariano Esquillor, s/n street, 50018, Zaragoza, Spain
rrosales@unizar.es