# Solving Cardiac Electrophysiology Models Based on the Markov-Chain Formulations with Tensor Cores

João Víctor C de Oliveira[1], Johnny M Gomes[1], Marcelo Lobosco[1], Rodrigo W dos Santos[1]

[1] Federal University of Juiz de Fora, Juiz de Fora, Brazil

## Abstract

*This work presents a high-performance GPU-accelerated method for solving cardiac electrophysiology models. Using the uniformization technique to enhance the classical matrix exponential for CTMCs, it improves stability and efficiency. Implemented with NVIDIA Tensor Cores via WMMA, it achieves up to* $148\times$ *CPU speedup and* $1.5\times$ *over standard CUDA. Validation with* 1000–100000 *Bondarenko model instances confirms scalability and accuracy for electrophysiological simulations.*

## 1. Introduction

The modeling of complex biological systems has significantly advanced through the application of mathematical and computational techniques. For instance, Markov chains (MCs) and their extensions play a central role in computational biology, where they enable the simulation of stochastic and state-dependent processes. A prominent example is the use of Markov models to represent ion channel gating dynamics in action potential simulations. One such model, proposed by Bondarenko in 2004 [1], examines the ionic mechanisms underlying action potential behavior in mouse ventricular myocytes. This model utilizes detailed Markov formulations to describe key ionic currents, including the fast sodium current ($I_{\mathrm{Na}}$), L-type calcium current ($I_{\mathrm{CaL}}$), and rapid delayed rectifier potassium current ($I_{\mathrm{Kr}}$), providing a framework for studying the effects of genetic mutations and pharmacological interventions on cardiac electrophysiology. Despite their mechanistic accuracy, MC-based models are considerably more computationally intensive than classical Hodgkin–Huxley-type models, which limits their practical use in large-scale or real-time simulations.

With this in mind, the present work focuses on the efficient numerical solution of electrophysiological models formulated using the Markov chain formalism. To overcome the computational challenges associated with solving large systems of Markovian equations, we extend the matrix exponential method and integrate a uniformization technique [2, 3], improving both numerical stability and computational performance. To further enhance efficiency, we developed two GPU-accelerated implementations: one leveraging conventional CUDA cores, and another exploiting the advanced capabilities of Tensor Cores available in modern NVIDIA GPUs, which are optimized for high-throughput, mixed-precision matrix operations. Our results demonstrate substantial speedups enabled by hardware-aware optimization and parallelization strategies, underscoring the potential of these techniques for scalable and real-time simulation of Markov-based electrophysiological models.

## 2. Methods

### 2.1. Matrix Exponential and Uniformization Method

The uniformization method, also known as the randomization method, is a numerical technique developed to efficiently and accurately compute the solution of continuous-time Markov chains (CTMCs), particularly for transient analysis [3]. This method reformulates the computation of the matrix exponential, which arises in the analytical solution of linear systems of the form [2]

$$\frac{d\vec{Y}(t)}{dt} = \mathbf{A}\vec{Y}(t), \tag{1}$$

with known analytical solution [2]

$$\vec{Y}(t) = exp(t \times \mathbf{A}) \times \vec{Y}(0), \tag{2}$$

where $\mathbf{A}$ is the infinitesimal generator matrix of the CTMC. Direct evaluation of the matrix exponential via its Taylor series, following the Eq. (3), can suffer from numerical instability and poor convergence due to the structure of matrix $\mathbf{A}$, as it typically has negative diagonal entries and non-negative off-diagonal elements, with potentially large magnitudes [3].

$$exp(t \times \mathbf{A}) = \sum_{i=0}^{\infty} \frac{(t \times \mathbf{A})^i}{i!}. \tag{3}$$

To bypass these issues, the uniformization method introduces a normalization parameter $q \geq max_i|\mathbf{A}_{ii}|$, and defines a new matrix:

$$\mathbf{A}^* = \frac{\mathbf{A}}{q} + \mathbf{I}, \tag{4}$$

where $\mathbf{I}$ is a diagonal matrix of the same dimensions as $\mathbf{A}$ and $\mathbf{A}^*$. This transformation gives:

$$\vec{Y}(t) = exp(-q \times t) \times \sum_{i=0}^{\infty} \frac{(q \times t \times \mathbf{A}^*)^i}{i!} \vec{Y}(0). \tag{5}$$

The matrix $\mathbf{A}^*$ now behaves like a stochastic matrix, i.e., non-negative entries, and row sums close to one, which ensures faster and more stable convergence of the series. In practice, the series is truncated at an index $N(t)$ such that the residual error is bounded by the tolerance $\lambda$ [2] described by Eq. (6). This approach guarantees the desired accuracy while enabling efficient reuse of intermediate terms during the computation.

$$\lambda \leq 1 - exp(-q \times t) \times \sum_{i=0}^{N(t)} \frac{(q \times t)^i}{i!}. \tag{6}$$

## 2.2. Tensor Cores

Tensor Cores are specialized processing units found in modern NVIDIA GPUs, specifically designed to accelerate mixed-precision matrix operations, with a focus on matrix-matrix multiplications. Unlike conventional CUDA cores, which handle general-purpose computations, Tensor Cores are optimized for high-throughput linear algebra operations. They exploit the inherent structure of matrix computations to maximize both performance and energy efficiency. Central to their functionality is the ability to rapidly execute fused multiply-accumulate (FMA) operations on small matrix blocks, typically sized at $4 \times 4$, $8 \times 8$, or $16 \times 16$. These operations are performed in mixed precision, commonly utilizing *FP*16 or *BF*16 inputs with *FP*32 accumulation. This approach allows for significant speedups while ensuring adequate numerical accuracy across a variety of scientific and engineering applications. When applied to matrix-matrix multiplication, a fundamental operation in numerous computational problems, including those related to Markov Chains and machine learning, Tensor Cores provide considerable performance enhancements compared to traditional GPU implementations. Conventional methods that rely on general-purpose CUDA cores often encounter memory bandwidth limitations and instruction overhead. In contrast, leveraging Tensor Cores through low-level interfaces such as WMMA in CUDA facilitates specialized hardware acceleration, reducing latency and enhancing arithmetic intensity. This capability proves especially beneficial when computing matrix exponentials via the Taylor series expansion of the transition matrix. Each term in the expansion requires calculating successive powers of the system matrix, and the computational demands increase rapidly with the desired accuracy. Tensor Cores can significantly mitigate this expense by speeding up the necessary matrix multiplications, thereby allowing for the efficient and scalable evaluation of higher-order expansions. As a result, integrating Tensor Core-based operations into the matrix exponential method enhances both computational efficiency and numerical robustness, making it particularly well-suited for large-scale or real-time simulations.

## 2.3. Experiments

To validate the proposed approach, a series of experiments was conducted using the Bondarenko model [1]. The four Markovian chains (MCs) associated with this model were solved using the matrix exponential in conjunction with the uniformization method, and the resulting state probabilities were subsequently passed to the ordinary differential equation (ODE) solver. The first experiment consisted of a serial implementation, in which both the matrix exponential and the uniformization process were computed exclusively on the CPU, along with the numerical integration of the ODEs. In the second implementation both the matrix exponential and the uniformization process were computed exclusively on the CPU, along with the numerical integration of the ODEs, but using the OpenMP API. In the third implementation, the matrix exponential, uniformization and the ODE were solved in the GPU using conventional CUDA. The fourth version employed a Tensor Core-based strategy to accelerate the matrix exponential and uniformization on the GPU, with the ODEs still being solved with conventional CUDA. The CPU code was developed in the `C` programming language, while the GPU implementation was written in CUDA. Specifically, the Tensor Core-based experiment utilized the WMMA (Warp Matrix Multiply-Accumulate) API to exploit the capabilities of Tensor Cores.

In all cases, the ODEs were integrated using the Rush–Larsen method [4], with time steps of 0.01 and 0.05 ms. Preliminary tests were performed to determine the appropriate number of terms required in the truncated series expansion described by Eq. 5, and it was verified that using nine terms provided sufficient accuracy across all configurations. Since the cellular model was not coupled to a spatially explicit tissue-level representation, we simulated a large-scale, tissue-like scenario by solving 1000, 10000, and 100000 independent instances of the model at each time step, and analyzing the increasing in time when the number of instances is increased. To introduce heterogeneity among these instances, small perturbations were applied to the initial conditions described in the original study [1]. This procedure enables a more realistic emulation of the variability expected in a tissue-coupled context. Overall, this strategy serves as a proxy to approximate the computational workload typically observed in tissue-level simulations, where cellular dynamics must be solved concurrently across multiple spatial locations in a discretized domain. All required model parameters, equations, and baseline initial conditions are provided in the original publication [1].

The experiments were conducted on a workstation featuring an AMD Ryzen 9 7950X3D processor with 16 physical cores. Each core is equipped with 32 KB of L1 data cache, 32 KB of L1 instruction cache, and 1 MB of L2 cache, while the processor shares a total of 128 MB of 3D V-Cache (L3). The system is equipped with 64 GB of DDR5 RAM and a Gigabyte GeForce RTX 4080 Gaming OC GPU, which includes 9728 CUDA cores and 304 Tensor Cores. The GPU is equipped with 16 GB of GDDR6X memory on a 256-bit memory interface, delivering high memory bandwidth optimized for intensive parallel computations. For the CUDA implementation, the NVCC compiler (version 12.8) was used with the optimization flag `-O3`. For both the serial version and the CPU-executed code, the `gcc` compiler (version 13.2) was employed, also with the `-O3` optimization flag enabled.

## 3. Results

To assess and compare the performance of the parallel implementations relative to the serial version, the speedup metric was employed. According to this metric, the speedup is calculated as:

$$S = \frac{T_{\text{slower}}}{T_{\text{base}}}, \tag{7}$$

where $T_{\text{slower}}$ denotes the execution time of the baseline (i.e., slower) experiment, and $T_{\text{base}}$ represents the execution time of the experiment under analysis.

The results are summarized in Tables 1 and 2, which presents the performance comparison across different implementations, the two time steps, and instance counts. All the simulations were performed from $t = 0s$ to $t = 20s$, following our previous work [2]. The first column identifies the computational configuration: *Serial* (fully sequential execution on the CPU), *OpenMP* (multi-threaded CPU execution), *CUDA* (GPU execution using standard CUDA cores), and *Tensor Cores* (GPU execution using NVIDIA Tensor Cores). For each configuration, three performance metrics are reported: the average execution time in minutes (*Mean*), the corresponding standard deviation (*Std*), and the *Speedup* for the calculated performance gain relative to the slowest configuration, following Eq. (7). The boldface values highlight the best execution time, i.e., the highest speedup, within each row. The rows indicate the different time step values used in the simulations.

## 4. Discussion

The results from our experiments underscore the substantial impact of leveraging Tensor Cores over conventional CUDA in GPU-accelerated cardiac simulations. Across all tested configurations and time steps, the Tensor Core implementation consistently achieved a speedup of approximately $1.4\times$ to $1.5\times$ over the standard CUDA version, highlighting the meaningful performance gains enabled by specialized hardware. These advantages were especially evident in large-scale scenarios: for 100000 independent instances, Tensor Cores achieved speedups exceeding $144\times$ compared to CPU execution, while conventional CUDA plateaued around $101\times$. This confirms that performance can be significantly enhanced even beyond general-purpose GPU optimization through fine-grained access to low-level architectural features.

Both GPU implementations substantially outperformed CPU and OpenMP baselines. Yet, the consistent additional gain from Tensor Cores across all time steps, even at high temporal resolutions, demonstrates the robustness and scalability of the approach. For instance, at 0.050 ms and 100000 instances, execution time dropped to 0.336 minutes using Tensor Cores, compared to 0.491 minutes with CUDA. These results validate our method's efficiency in handling computationally intensive, parallelizable workloads typical of Markov-based electrophysiological models. However, for simulations involving only 1000 independent instances, the performance gains from parallel strategies (both OpenMP, conventional CUDA, and Tensor Cores) were notably reduced, suggesting that the overhead of GPU initialization and kernel execution may outweigh the benefits in small-scale problems.

Finally, using CUDA WMMA APIs to access Tensor Cores showcases the benefits of low-level hardware-aware programming. While conventional CUDA already delivers strong acceleration, Tensor Cores provide a clear performance edge, particularly for operations dominated by dense matrix multiplications, making them an ideal fit for methods like matrix uniformization in cardiac modeling. Although the implementation using tensor cores can be considered more difficult when comparing with the OpenMP and conventional CUDA approaches, the resulting performance gains justify this additional complexity. Importantly, we also verified whether the use of mixed-precision matrix operations introduced accuracy issues, given the known sensitivity of ionic models to numerical errors. The root mean squared error (RMSE) [5] between

Table 1. Performance comparison for different implementations and independent instances for $h = 0.01$. Each value is the average over 20 runs.

| **Method** | **1000 instances** | | | **10000 instances** | | | **100000 instances** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Mean | Std | Speedup | Mean | Std | Speedup | Mean | Std | Speedup |
| Serial | 0.500 | 0.005 | 1.00 | 5.021 | 0.014 | 1.00 | 50.200 | 0.050 | 1.00 |
| OpenMP | 0.080 | 0.003 | 6.25 | 0.314 | 0.003 | 15.99 | 3.140 | 0.020 | 15.99 |
| CUDA | 0.050 | 0.002 | 10.00 | 0.050 | 0.001 | 100.42 | 0.520 | 96.54 | |
| Tensor Cores | **0.035** | 0.001 | **14.29** | **0.035** | 0.001 | **144.85** | **0.347** | 0.009 | **144.66** |

Table 2. Performance comparison for different implementations and independent instances for $h = 0.05$. Each value is the average over 20 runs.

| **Method** | **1000 instances** | | | **10000 instances** | | | **100000 instances** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Mean | Std | Speedup | Mean | Std | Speedup | Mean | Std | Speedup |
| Serial | 0.495 | 0.004 | 1.00 | 5.001 | 0.008 | 1.00 | 50.010 | 0.045 | 1.00 |
| OpenMP | 0.078 | 0.002 | 6.35 | 0.313 | 0.003 | 15.97 | 3.132 | 0.018 | 15.97 |
| CUDA | 0.049 | 0.001 | 10.10 | 0.050 | 0.001 | 100.02 | 0.491 | 0.009 | 101.85 |
| Tensor Cores | **0.034** | 0.001 | **14.56** | **0.034** | 0.001 | **147.09** | **0.336** | 0.008 | **148.24** |

GPU (mixed-precision) and CPU (double-precision) solutions was found to be at most $10^{-3}$, which we considered acceptable in light of the speedup provided by the GPU implementation.

## 5. Conclusions, Limitations and Future Works

In this study, we proposed and evaluated a high-performance numerical framework for simulating cardiac electrophysiology models based on Markov chains. By integrating the matrix uniformization method with hardware-accelerated computation via NVIDIA Tensor Cores, we aimed to reduce computational cost while maintaining numerical accuracy and stability. The results demonstrate that the Tensor Core implementation consistently outperformed both the CPU and conventional CUDA approaches, achieving up to $148.24\times$ speedup over the CPU and up to $1.455\times$ over standard CUDA, across multiple time steps and problem sizes. These findings support the effectiveness of combining matrix uniformization with specialized GPU hardware for handling large ensembles of independent Markov-based cellular simulations. In future work, we plan to extend our approach to additional Markov-based cellular models and apply the proposed methods and implementations to whole-heart electrophysiology simulations. Moreover, given that the present study focused on cell-level dynamics, future developments will also consider tissue-level simulations, which naturally align with the workload design and would allow assessing the scalability of the proposed framework in more physiologically realistic scenarios.

## Acknowledgments

The authors would like to express their thanks to Wellcome Trust fellowship (214290/Z/18/Z), the EPSRC project CompBioMedX (EP/X019446/1), CompBioMed2 grant agreements No.675451 and No.823712, Minas Gerais State Research Support Foundation (FAPEMIG)-PCE-00048-25; APQ-02752-24, APQ-02445-24, APQ-02513-22, FINEP (SOS Equipamentos 2021 AV02 0062/22), "Coordenação de Aperfeiçoamento de Pessoal de Nível Superior" (CAPES) 88881.708850/2022-01 and 88881.691780/2022-01, "Empresa Brasileira de Serviços Hospitalares" (Ebserh), SINAPAD Santos-Dumond, and Federal University of Juiz de Fora (UFJF) for funding this work.

## References

[1] Bondarenko VE, Szigeti GP, Bett GC, Kim SJ, Rasmusson RL. Computer model of action potential of mouse ventricular myocytes. American journal of physiology heart and circulatory physiology 2004;287(3):H1378–H1403.

[2] Gomes JM, Alvarenga A, Campos RS, Rocha BM, da Silva APC, Santos RWd. Uniformization method for solving cardiac electrophysiology models based on the markov-chain formulation. IEEE transactions on biomedical engineering 2015;62(2):600–608.

[3] Jensen A. Markoff chains as an aid in the study of markoff processes. Scandinavian actuarial journal 1953; 1953(sup1):87–91.

[4] Rush S, Larsen H. A practical algorithm for solving dynamic membrane equations. IEEE Transactions on Biomedical Engineering July 1978;BME-25(4):389–392. ISSN 0018-9294.

[5] Ramachandran K, Tsokos C. Mathematical Statistics with Applications. Elsevier science, 2009. ISBN 9780080951706.

Address for correspondence:

João Víctor Costa de Oliveira
Patamar da Faculdade de Engenharia da UFJF, Prédio Azul, Juiz de Fora - MG, 36036-330
oliveira.joao@estudante.ufjf.br