Enhancing the Efficiency of Animal-Alternative *In Silico* Drug Cardiotoxicity Prediction through CUDA-Based Parallel Processing

Iga Narendra Pramawijaya[1], Ariyadi[1], Aroli Marcellinus[1] , Ali Ikhsanul Qauli[1,2] , Ki Moo Lim[1,3,4*]

[1]: Computational Medicine Lab, Department of IT Convergence Engineering, Kumoh National Institute of Technology, Gumi, 39177, Republic of Korea
[2]: Department of Engineering, Faculty of Advanced Technology and Multidiscipline, Universitas Airlangga, Surabaya, Indonesia
[3]: Computational Medicine Lab, Department of Medical IT Convergence Engineering, Kumoh National Institute of Technology, Gumi, 39177, Republic of Korea
[4]: Meta Heart Inc., Gumi, South Korea
[*]: kmlim@kumoh.ac.kr

Abstract
Traditional Central Processing Unit (CPU)-based simulations are computationally expensive, especially for large-scale studies. By leveraging CUDA programming, this research aims to optimise simulation efficiency while maintaining the accuracy of cellular electrophysiological models. The study employed three well-established cardiac cell models: ORd 2011, CiPAORdv1.0, and ToR-ORd. Simulations were conducted using GPU-based implementations of ordinary differential equation (ODE) solvers, with the Rush-Larsen method applied for ORd 2011 and a Forward Euler approach for CiPAORdv1.0 and ToR-ORd. The simulations were validated against CPU results, with performance evaluated in both drug free and drug induced conditions. GPU simulations demonstrated equivalent accuracy to CPU-based results, replicating action potential dynamics and key biomarkers. However, the Forward Euler solver required more computation time compared to the Rush-Larsen method. Computational performance analysis revealed significant efficiency improvements (up to 49 times in fastest configuration) in GPU-based simulations. This research successfully validates GPU-based parallel computing as a reliable and efficient approach for *in silico* cardiotoxicity prediction. The findings support its potential for accelerating drug discovery processes while reducing reliance on animal testing. Future work will focus on expanding model complexity and variabilities to further enhance the system's applicability.

Keywords: *In silico*, cardiotoxicity, CUDA, GPU Parallel Programming

1 Introduction

Cardiovascular diseases are the leading global causes of death, which emphasizes the importance of effective methods for cardiac drug discovery. Traditionally, drug cardiotoxicity prediction is achieved using animal testing, which takes a lot of financial resource and failure to identify toxicity in some human subject [1]. Modern *in silico* or computer-based methods for drug cardiotoxicity prediction show promising results as an animal-alternative solution due to its higher accuracy demonstration [2]. Nevertheless, at some point, this approach is computationally inefficient due to large amount of sample it needs to compute to mimic natural variations. As the sample size increases, the complexity of the calculations grows, resulting in longer processing times and reduced efficiency.

This efficiency limitation makes it difficult for traditional computational approaches to handle large-scale simulation within a reasonable timeframe. Large scale simulation means when a simulation takes many input values. in the magnitude of thousands or more. Multi-sample scenario refers to the simulation of many drug samples simultaneously, enabling efficient evaluation of varying drug concentrations or conditions. Inter-individual variations, on the other hand, account for differences in biological responses across individuals, ensuring the simulation captures variability in drug effects due to genetic, physiological, or environmental factors. By incorporating these aspects, the study aims to provide a more comprehensive and realistic prediction of cardiotoxicity across diverse populations. This research introduces parallelisation to address current *in silico* drug cardiotoxicity simulations inefficiencies. By implementing NVIDIA's CUDA (Compute Unified Device Architecture)-based parallel programming on Graphics Processing Units (GPU) [3].

Parallel computing method can significantly accelerate overall computational process. The parallelisation enables faster handling of large-scale simulations, makes computational time decreased. Time advancement gained through this approach can reduce drug development cost, by accelerating preclinical test and reliance on animal testing.

Other than biological cell, Cells *in Silico* (CiS) framework developed by Berghoff et al. [4] provides a modular and parallel design for simulating biological tissue growth and development. This flexibility allows researchers to configure various model assumptions for diverse research questions. A key application of CiS was the simulation of a 1000^3 voxel-sized cancerous tissue at sub-cellular resolution, showcasing its ability to handle complex biological processes with high detail.

The use of GPUs in biological cell computing has also been explored extensively in prior researches. Martinez et al. [5] introduced an adaptive parallel simulator to mitigate performance loss in massive parallel membrane computing devices, also known as P systems. By extending an existing simulator for Population Dynamics P systems, their approach improved performance by up to 2.5 times on both GPUs and multicore processors. Similarly, McIntosh-Smith et al. developed BUDE (Bristol University Docking Engine), a drug discovery tool for molecular docking, to work with OpenCL, a standard for parallel programming. Their adaptation enabled peak performance of 46%, or 1.43 TFLOP/s, on a single Nvidia GTX 680 [6]. Amar et al. extended parallelisation to biochemical simulations of metabolic pathways, while Barth et al. leveraged parallel computing in BUDE to handle more complex models with an increased number of chemicals and reactions, achieving more realistic and computationally efficient outcomes [7].

Recent work has increasingly focused on using *in silico* methods to address specific cardiac conditions and evaluate pharmacological interventions. For example, Whittaker et al. employed computational modelling to examine mutations linked to Short QT Syndrome and their effects on atrial arrhythmias. Their study demonstrated how *in silico* simulations could assess drug responses and inform strategies for treating genetic cardiac disorders [8].

Advances in computer hardware and parallel processing have further enhanced the speed and efficiency of *in silico* cardiac simulations, enabling the analysis of larger datasets and more intricate models. However, despite these advancements, optimising *in silico* simulations specifically for cardiotoxicity prediction remains underexplored. Therefore, this study main goal is to improve the efficiency and scalability of drug cardiotoxicity simulations by employing CUDA-based parallel processing techniques. The research aims to:

1. Address computational bottlenecks caused by increasing sample sizes and complex calculations in traditional methods.
2. Optimize GPU resources for faster, large-scale simulations without compromising accuracy.
3. Validate the accuracy and reliability of GPU-based simulations compared to CPU-based, both in single-core and multi-core scenario.

2 Methodologies

This chapter describes the development process of the GPU-based cardiac cell simulation software. The focus is on enabling multi-sample simulations, where each cardiac cell model is simulated in parallel. Additionally, this chapter will briefly explain how ordinary differential equations (ODEs) in the cell models are solved within this framework. The goal is to ensure that researchers or software engineers can follow the steps, to replicate the parallelisation process and build their own GPU-based multi-sample simulation platforms.

This research leverages the flexibility provided by the CellML platform, an XML-based language designed to represent and share mathematical models of biological cells [9]. CellML ensures standardisation across models, enabling translation into various programming languages [10]. It also includes models converted from the Systems Biology Markup Language (SBML), enhancing accessibility for researchers [11-12]. This study utilises three cell models: O'Hara et al. 2011 (ORd 2011) [13], Dutta et al. 2017 (CiPAORdv1.0)[14], and ToR-ORd [15],

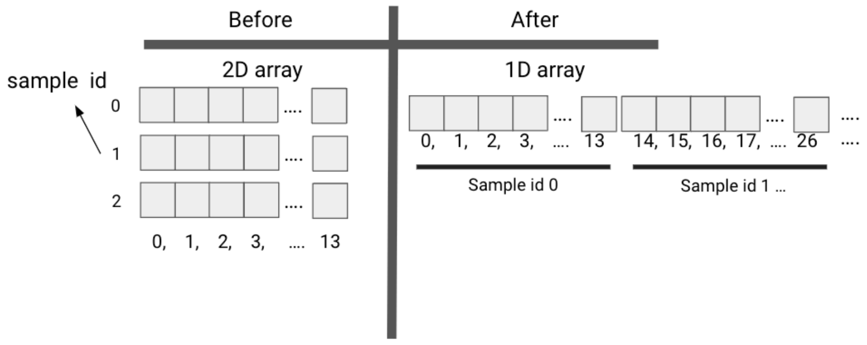


Fig. 1. Offsetting implementation as memory optimisation to generate GPU-compatible cell model.

demonstrating the platform's versatility in supporting diverse simulations.

Once cell model file is acquired in C or C++ format, conversion in terms of GPU memory adjustment and offsetting is needed. CUDA-based application heavily depends on how data is transferred between the host in the central processing unit (CPU) and device (GPU), as well as how it is organized within the GPU's memory.

This research uses three types of GPU memory: global, shared, and constant memory. Proper adjustment of these memory types can significantly enhance computational efficiency. In this research, the global memory is used for storing variables, constant memory for orchestrating commands from CPU, and shared memory used to optimise GPU to CPU feedbacks.

In this research, selecting an optimal number of threads per block is crucial to maximising utilisation of shared memory. Two factors need to be considered to optimise performance when selecting core per block value: thread grouping in CUDA, and number of samples. CUDA executes threads in groups called warps, which consist of 32 threads [16]. Using a block size that is a multiple of 32 ensures that all warps are fully utilised, minimising idle threads and maximising efficiency. It is known that this configuration is not transferable across different GPUs. As the time writing this, NVIDIA 40xx series GPU supports 32 core per block, while older series like the 30xx only support 16 core per block.

The choice of cores per block was also influenced by the number of samples. Each simulation sample usually comes in the multiplication of 2000 (2000, 4000, etc.). To optimise warp utilisation (with warps consisting of 32 threads), To optimize warp utilization, the number of cores should be close to 32 and evenly divisible by 2000. Through trial and error, 20 cores per block were found to provide the most efficient configuration. This adjustment ensures that each sample is allocated its own computing core. Code in this research is configured to use 32 cores per block. However, if hardware limitations issue raises, the number of cores per block decreased to 20, maintaining parallelisation.

Offsetting is a technique used to efficiently manage data indexing, ensuring thread indices map to the correct memory addresses. This reduces memory conflicts and improves performance, particularly when dividing large datasets across multiple threads and blocks. In this research, offsetting was used to simplify multi-dimensional inputs by converting them into 1D arrays. Instead of using a vector of structs as in the CPU version, each sample is identified by a unique `sample_id`, with rows in the 1D array allocated accordingly. For example, there is an array that keeps the value of each ionic current. This array holds 43 values. Index number 0–42 correspond to `sample_id = 0`, index number 43–84 correspond to `sample_id = 1`, and so on. Figure 1 shows offsetting implementation visually. Offsetting approach ensures efficient data access and processing in the GPU-based simulation.

The model relies on algebraic calculations and dynamic functions expressed in the form of ordinary differential equations (ODEs), which are essential for simulating the complex behaviours of biological systems. To efficiently solve these ODEs within a CUDA-based parallel processing framework, two distinct numerical methods were employed depending on the specific cell model: the Rush-Larsen method and a custom implementation of the forward Euler method. These methods were chosen to balance computational efficiency, numerical stability, and compatibility with the CUDA architecture. This research implements the ODE solver inside the cell model code as a function.

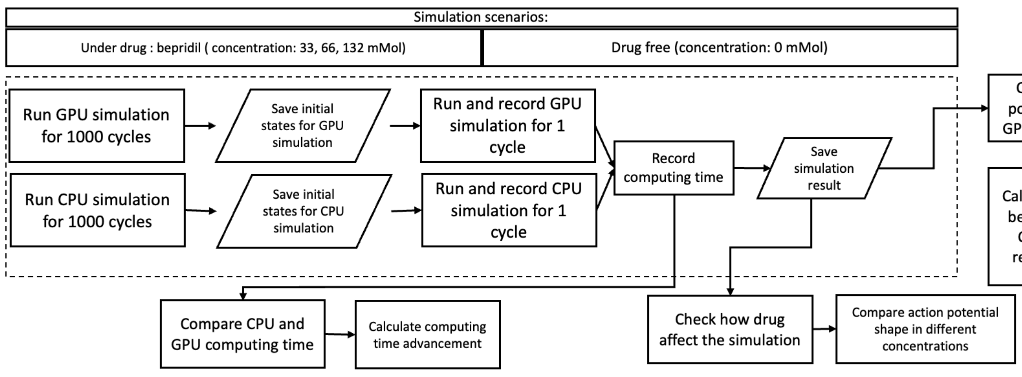

Simulation scenarios:
Under drug : bepridil (concentration: 33, 66, 132 mMol)
Drug free (concentration: 0 mMol)
Run GPU simulation for 1000 cycles
Save initial states for GPU simulation
Run and record GPU simulation for 1 cycle
Run CPU simulation for 1000 cycles
Save initial states for CPU simulation
Run and record CPU simulation for 1 cycle
Record computing time
Save simulation result
Compare action potential shape of GPU and CPU result
Calculate difference between GPU and CPU simulation result using MAE
Compare CPU and GPU computing time
Calculate computing time advancement
Check how drug affect the simulation
Compare action potential shape in different concentrations

Fig. 2. Simulation steps

For the ORd 2011 model, the Rush-Larsen method was utilised due to its computational efficiency and computational stability in this context. This method effectively integrates stiff components of the equations, making it well-suited for the dynamic features of the ORd 2011 model. This method was implemented with a dynamic time-stepping mechanism by adjusting the time step during each iteration, while balancing acceptable numerical errors [17].

Initially, the Rush-Larsen method was meant to be used across all cell models. However, when applied to CiPAORdv1.0 and ToR-ORd models, this approach exhibited instability, failing to provide reliable results. To address this limitation, a simple forward Euler method was implemented for these two models.

Forward Euler is a simple method to solve ODE in particular time. The forward Euler method calculates the next value of a variable by taking its current value (STATES array) and adding the product of the rate (RATES array) of change and the time step, mentioned as *dt*. This straightforward approach makes the method computationally simple and easy to implement. Simply, it is expressed as in Equation 1:

$$x_{n+1}=x_n + rate(x_n) \cdot \Delta t \qquad (1)$$

where x_{n+1} is the current value, $rate(x_n)$ represents the rate of change at x_n, and Δt is the time step. This study uses Δt value of 0.005, acquired after trial and error to balance computational time and simulation's accuracy. In the converted code, a function should be added to implement this calculation.

While the Euler method produced accurate and stable results, especially for the CiPAORdv1.0 and ToR-ORd models, it proved to be computationally intensive, significantly increasing the runtime. To optimise the parallelisation process, algorithm was simplified. The simplification also ensure parallelisation across samples were executed properly. This study initially tries to implement other advanced numerical methods to solve ODEs. Some of them is backward differentiation function and Newton-Raphson method, both of them found hardware limitation due to lack of GPU memory. Hence, the performance of these ODE solvers on GPU parallel platform cannot be concluded. Despite the trade-off between computational speed and stability in forward Euler method, current combination of methods ensures that the CUDA-based framework effectively supports the diverse requirements of different cell models, while maintaining accuracy and to reduce numerical errors.

The simulation protocols are designed to comprehensively evaluate the effects of drugs on cardiac cell models, ensuring accurate representation of both physiological (drug free) and pharmacological (drug affected) scenarios. For the drug free simulations, the models replicate the natural state of ventricular cardiac cells without introducing external agents. Technically, drug free situation runs the simulation with drug concentration set to 0 mMol. This process involves solving the ordinary differential equations (ODEs) governing ion channel dynamics, membrane potentials, and other cellular electrophysiological properties. These simulations serve as a baseline to observe normal cardiac function and provide a reference point for comparison.

For drug affected simulations, specific modifications to constants are made to the cell models to reflect the influence of pharmacological agents. This involves incorporating drug binding effects on ionic currents or transporters. For instance, drugs like bepridil, which block potassium or calcium channels, are simulated by altering corresponding values in the models. The simulations are carried out at varying drug concentrations to capture the dose-response relationship and understand the drug's potency and safety profile.

The simulation consists of two different stages, phase to amplify drug effect by running the simulation for multiple times, and phase to record the drug effect by running for just once. Result from the first phase is a cache file, which represents the initial phase before observation. During this phase, the function runs the simulation for thousands of cycles (referred to as paces) to amplify the drug effects within the model.

After this initial phase, the function to run the second phase is executed, which generates the biomarker file and the time-series data files. This function uses the output from the first phase as input. All output files from the second phase are organised into a dedicated folder within the result directory for efficient storage and retrieval. As a whole, the simulation produces two distinct types of output files, a biomarker file and time-series data files, along with one intermediate cache file. Figure 2 graphically shows overall protocol of simulation.

The biomarker file provides a summary of key features extracted from the simulation for each sample. It includes data such as sample numbers, net charge transfer during the action potential, action potential duration at 50 and 90% repolarisation, action potential triangulation, calcium transient at 50 and 90% including its triangulation, maximum rate of change during peak depolarisation and repolarisation, peak, valley and diastolic value of both membrane potential and calcium levels. These biomarkers represent crucial physiological parameters simulated under drug influence [2]. The time-series file offers a detailed temporal view of each sample's behaviour. Each sample has its own individual time-series file; thus, a simulation involving 2000 samples will result in 2000 time-series files. These files capture parameters such as time, action potential, voltage gradient over time, Cai, INa, INaL, ICaL, IKs, IKr, IK1, and Ito. Using this detailed data, it is possible to plot the drug induced cellular responses over a single cycle, facilitating visualisation and deeper analysis of dynamic behaviours.

3 Results and Discussion

This chapter presents the results of GPU-based simulations for three cardiac cell models: ORd 2011, CiPAORdv1.0, and ToR-ORd. It analyses simulation accuracy, drug induced changes, and computational performance. Two types of simulations were conducted: control (no drug) and drug induced using bepridil at concentrations of 33 mMol, 66 mMol, and 132 mMol. Each simulation was run for 1000 pacing cycles, with each cycle lasting 1000 milliseconds.

Each simulation needs to simulate 8000 drug samples in multi-core situation. Single-core computational time also compared between CPU and GPU. Single-core performance focused on how each configuration (CPU and GPU) simulate only one drug sample. Both single-core and multi-core simulation uses the same parameters, drug, and concentration.

To evaluate the effects comprehensively, simulations examine changes in action potential morphology, duration, and ion current amplitudes compared to the baseline. By combining these simulations with a multi-sample approach, thousands of scenarios, representing different concentrations and inter-individual variations, can be generated efficiently using GPU parallelisation. This protocol ensures that both physiological and pharmacological behaviours are accurately simulated, offering valuable insights into drug induced cardiotoxicity.

3.1 Simulation Validation

Simulation validation is a crucial step in ensuring the accuracy and reliability of GPU-based cardiac electrophysiology simulations. The results of the GPU simulations were validated by comparing them with CPU-based simulations using OpenCOR as the ground truth. Across all three cell models—ORd 2011, CiPAORdv1.0, and ToR-ORd—the validation process involved calculating the mean absolute error (MAE) between the GPU and CPU results and evaluating key electrophysiological biomarkers, including

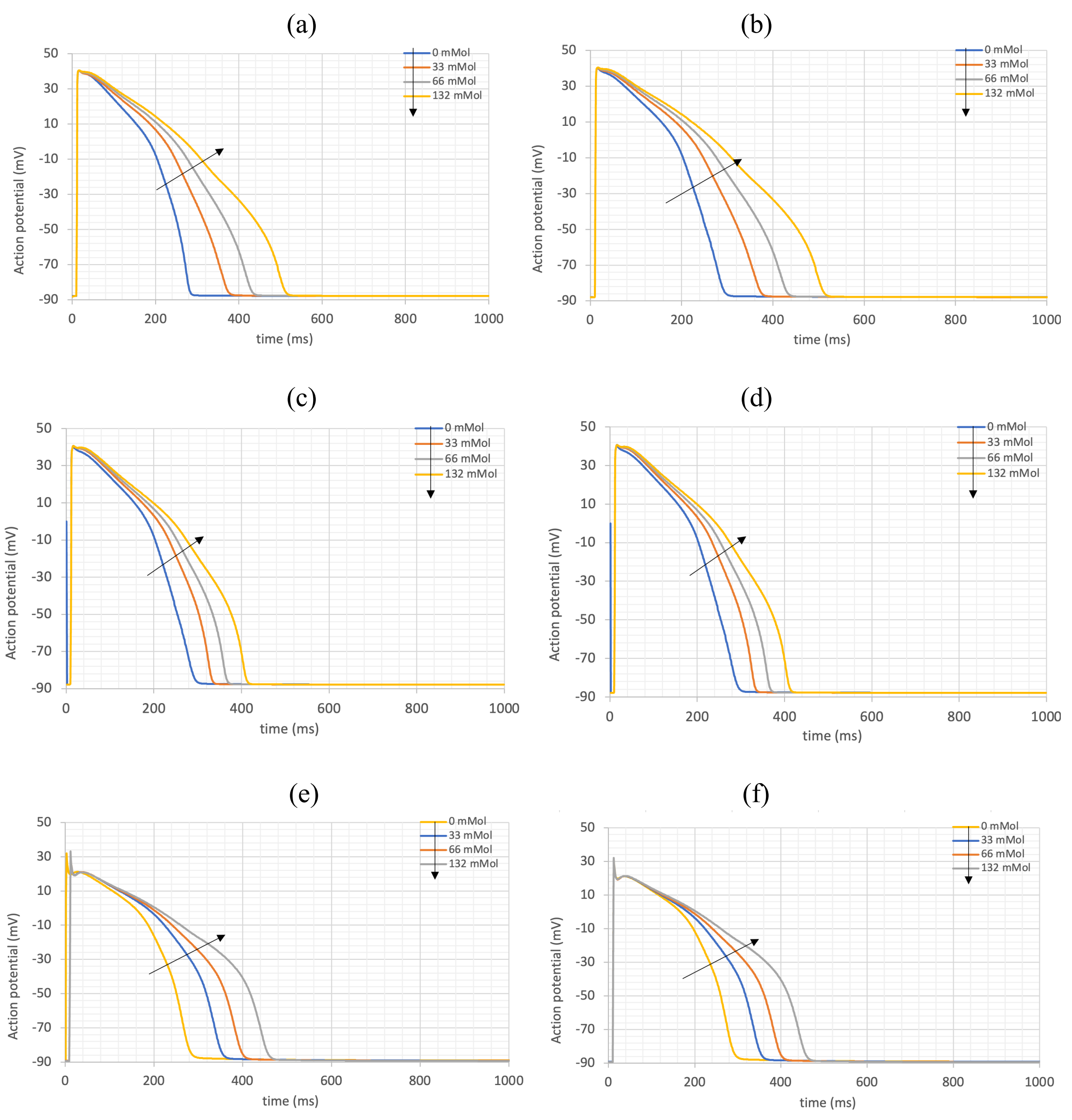


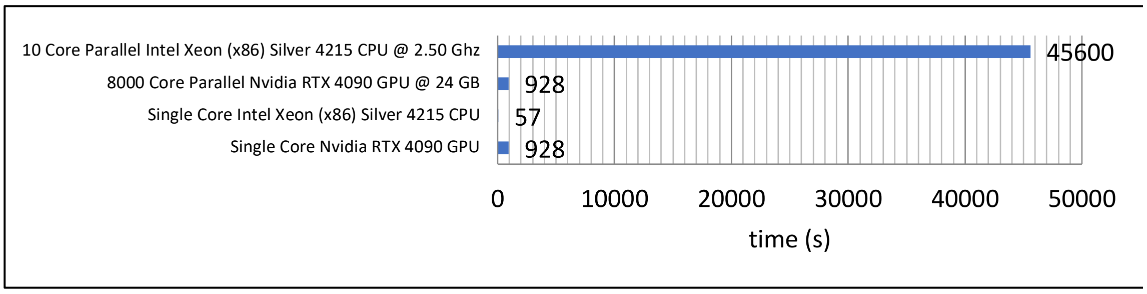
Fig. 3. Action Potential from CPU simulation in ORd 2011 (a), CiPAORdv1.0 (c), and ToR-ORd (e) model, compared to GPU [ORd 2011 (b), CiPAORdv1.0 (d), and ToR-ORd (f)] simulation in no drug (0 mMol) and under drug situation.

action potential duration (APD) and calcium transient properties.

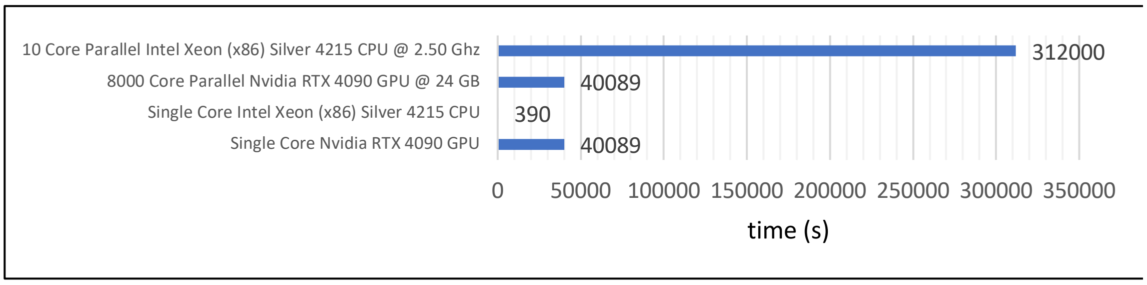
For the ORd 2011 model, the GPU simulation showed an MAE of 0.078 mV, demonstrating a high degree of accuracy. The Rush-Larsen method, used in this model for solving ordinary differential equations (ODEs), contributed to its computational stability and efficiency. The validation confirmed that the GPU accurately captured the physiological behaviours of the cardiac cells, including both control and drug induced conditions, providing confidence in the simulation's robustness. Action potential result from both drug free and under-drug simulation are visible in Figure 3 part (a) and (b).

The CiPAORdv1.0 model achieved an exceptionally low MAE of 0.004 mV, reflecting near-perfect agreement with the CPU results. This model employed a forward Euler method for solving ODEs, which, while computationally more intensive than Rush-Larsen, delivered precise results. Validation under both no-drug and drug induced conditions demonstrated the GPU's capability to reproduce the intricate dynamics of the cardiac cells simulated in this model. Figure 3 part (c) and (d) shows action potential result from CiPAORdv1.0 model, both with and without drug simulation.

(a)



(b)



(c)

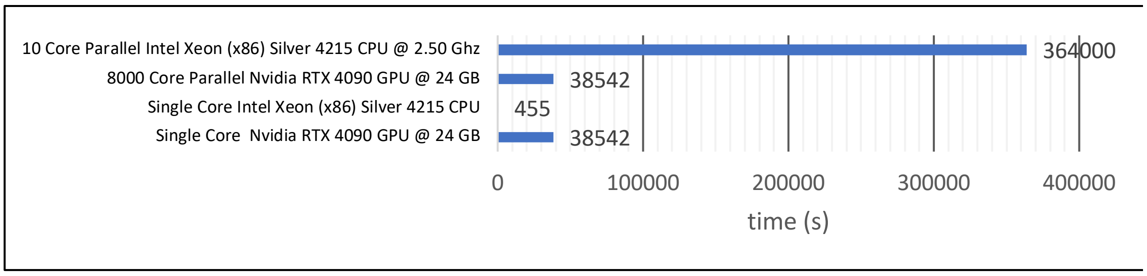


Fig. 4. Simulation time for ORd 2011 (a), CiPAORdv1.0 (b), and ToR-ORd (c) models using CPU and GPU

In the ToR-ORd model, the GPU simulation yielded an MAE of 0.023 mV, indicating strong alignment with the CPU reference. Despite the model's higher complexity and computational demands, the GPU efficiently handled both control and drug induced scenarios. The simulation accurately replicated the cardiac cells' electrophysiological responses to bepridil at varying concentrations, further confirming the GPU's suitability for large-scale, high-fidelity simulations. ToR-ORd simulation result is shown in Figure 3 part (e) and (f). This figure shows action potential curve from drug free, and drug induced simulation.

Overall, the validation results for all three cell models underscore the accuracy and reliability of the GPU-based approach. The low MAE values across models confirm that the parallel computing implementation does not compromise the simulation's fidelity, making it a viable alternative to traditional CPU-based methods. These findings support the broader application of GPU-based simulations in cardiac electrophysiology and drug discovery research.

3.2 Computation Time Performance

The computational time performance of GPU-based simulations was compared against both single-core and multi-core CPU implementations for the three cardiac cell models: ORd 2011, CiPAORdv1.0, and ToR-ORd. This comparison highlights the efficiency gains achieved through CUDA-based parallel processing on GPUs.

For the ORd 2011 model, the GPU achieved significant time savings compared to its CPU counterparts (Figure 4 part (a)). A single-core CPU required 57 seconds to complete the simulation for one sample, while a 10-core multi-core configuration took 45,600 seconds for 8,000 samples. In contrast, the GPU completed the same workload in just 928 seconds, representing a substantial speedup. This efficiency is attributed to the Rush-Larsen method employed in ORd 2011, which is well-suited for GPU parallelisation and reduces computational overhead.

The CiPAORdv1.0 model, using a forward Euler ODE solver, was computationally more demanding (Figure 4 part (b)). A single-core CPU required 390 seconds per sample, and the multi-core CPU setup took 312,000 seconds for 8,000 samples. Despite the increased complexity, the GPU outperformed the CPU significantly, completing the same task in 40,089 seconds. While the GPU's advantage is slightly less pronounced for this model, it still demonstrated its capability to handle large-scale simulations more efficiently than traditional CPUs.

The ToR-ORd model, being the most complex of the three, posed the greatest computational challenges (Figure 4 part (c)). Single-core CPU simulations required 455 seconds per sample, and the multi-core setup took 364,000 seconds for 8,000 samples. In comparison, the GPU completed the same workload in 38,542 seconds, achieving a notable speedup. This performance demonstrates the scalability of GPU-based simulations for handling high-fidelity cardiac models with greater computational demands.

Across all three cell models, the GPU consistently outperformed both single-core and multi-core CPU configurations, particularly as the sample size increased. The results underscore the GPU's efficiency in large-scale simulations, making it a superior choice for computationally intensive applications. Figure 4 summaries all computing time difference between GPU-based simulation compared with CPU-based simulation.

4. Conclusions, Limitations and Suggestions

This research proven its ability to address computational bottlenecks from increasing sample size to *in silico* cardiotoxicity prediction by leveraging CUDA-based parallel processing. CUDA enables this research to do massive parallelisation with up to 40.91 times faster simulation speed for 8000 drug samples, compared to CPU-based 10-core simulation. Validation results confirmed the accuracy of GPU simulation. Little to no difference were found in both GPU and the well-used CPU simulation results. This result highlights not only the GPU capability to produce more samples in shorter amount of time, but also it is similarly reliable to common CPU cardiotoxicity simulation.

While this research demonstrates significant advancement in GPU-based parallel simulations *in silico* cardiotoxicity prediction, several limitations should be noted. First, this research restricts hardware use by requiring NVIDIA GPUs, as it relies on CUDA for its programming. This dependency makes the research incompatible with another brand's GPU architecture. Another limitation is the reliance on simplifying ODE solvers, such as the Rush-Larsen and forward Euler methods. Hence, other ODE solving methods should be heavily modified to be applied into this research. Furthermore, the fixed configurations for threads and blocks may not optimally utilise newer or varying GPU hardware.

This research focuses solely on drug induced simulations using specific cell models and parameters, leaving the generalisation to other cell types or drugs unexplored. This point raises future suggestion to expand the research by incorporating another biological cell models and variables. More testing on diverse GPU hardware models would provide insights into performance variations across systems, ensuring broader applicability. Furthermore, assessing the economic viability of the GPU-based approach, including long-term operational costs and energy efficiency, could ensure its sustainability. Standardising protocols for simulation workflows would help maintain consistency and reproducibility, which are critical for scientific and industrial adoption. With depth assessment to the long-term operational cost and energy, this research might start a more practical and cost-effective drug discovery method, then eventually reducing reliance on live animal testing.

Acknowledgements This research was partially supported by the Ministry of Food and Drug Safety (22213MFDS3922 and (24212MFDS266), National Research Foundation of Korea (NRF) under the Basic Science Research Program (2022R1A2C2006326), and Ministry of Science and ICT Korea (MSIT) under the Grand Information Technology Research Center support program (RS-2020-II201612) supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP).

Declarations

Conflict of Interest: The authors have no financial or any conflict of interest to declare in this research.

References

1. Van Norman G. A. (2019). "Limitations of Animal Studies for Predicting Toxicity in Clinical Trials: Is it Time to Rethink Our Current Approach?". JACC. Basic to translational science, 4(7), 845–854. https://doi.org/10.1016/j.jacbts.2019.10.008
2. Passini, E., Britton, O. J., Lu, H. R., Rohrbacher, J., Hermans, A. N., Gallacher, D. J., Greig, R. J. H., Bueno-Orovio, A., & Rodriguez, B. (2017). "Human in silico drug trials demonstrate higher accuracy than animal models in predicting clinical pro-arrhythmic cardiotoxicity." Frontiers in Physiology, 8, 668. https://www.frontiersin.org/articles/10.3389/fphys.2017.00668.
3. Jason Sanders and Edward Kandrot. 2010. "CUDA by Example: An Introduction to General-Purpose GPU Programming (1st. ed.)". Addison-Wesley Professional.
4. M. Berghoff, J. Rosenbauer, F. Hoffmann, and A. Schug, "Cells in Silico-introducing a high-performance framework for large-scale tissue modeling," BMC Bioinformatics, vol. 21, no. 1, Oct. 2020, doi: 10.1186/s12859-020-03728-7.
5. M. Martínez-del-Amor, I. Pérez-Hurtado, D. Orellana-Martín, and M. J. Pérez-Jiménez, "Adaptative parallel simulators for bioinspired computing models," Future Generation Computer Systems, vol. 107, pp. 469–484, Jun. 2020, doi: 10.1016/j.future.2020.02.012.
6. S. McIntosh-Smith, J. Price, R. B. Sessions, and A. A. Ibarra, "High performance in silico virtual drug screening on many-core processors," International Journal of High Performance Computing Applications, vol. 29, no. 2, pp. 119–134, May 2015, doi: 10.1177/1094342014528252.
7. P. Amar, M. Baillieul, D. Barth, B. LeCun, F. Quessette, and S. Vial, "Parallel Biological In Silico Simulation," Nov. 2014, doi: 10.1007/978-3-319-09465-6_40.
8. D. G. Whittaker, J. C. Hancox, and H. Zhang, "In Silico Assestment of Pharmacotherapy for Human Atrial Patho-Electrophysiology Associated With hERG-Linked Short QT Syndrome" Jan. 2019, doi: 10.3389/fphys.2018.01888.
9. A. Garny et al., "CellML and associated tools and techniques," Sep. 13, 2008, Royal Society. doi: 10.1098/rsta.2008.0094.
10. M. Gómez, J. Carro, E. Pueyo, A. Pérez, A. Oliván, and V. Monasterio, "In Silico Modeling and Validation of the Effect of Calcium-Activated Potassium Current on Ventricular Repolarization in Failing Myocytes," IEEE J Biomed Health Inform, pp. 1–9, 2024, doi: 10.1109/JBHI.2024.3495027.
11. C. M. Lloyd, J. R. Lawson, P. J. Hunter, and P. F. Nielsen, "The CellML Model Repository," Bioinformatics, vol. 24, no. 18, pp. 2122–2123, 2008, doi: 10.1093/bioinformatics/btn390.
12. N. Le Novère et al., "BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems.," Nucleic Acids Res, vol. 34, no. Database issue, 2006, doi: 10.1093/nar/gkj092.
13. T. O'Hara, L. Virág, A. Varró, and Y. Rudy, "Simulation of the Undiseased Human Cardiac Ventricular Action Potential: Model Formulation and Experimental Validation." 2011, PLOS Computational Biology 7(5): e1002061. https://doi.org/10.1371/journal.pcbi.1002061.
14. S. Dutta, K.C. Chang, K.A. Beattie, J. Sheng , P.N. Tran, W.W. Wu, M. Wu, D.G. Strauss, T. Colatsky, and Z. Li. "Optimization of an In silico Cardiac Cell Model for Proarrhythmia Risk Assessment." Front Physiol. Aug 2017 doi: 10.3389/fphys.2017.00616.
15. Tomek, Jakub et al. "Development, calibration, and validation of a novel human ventricular myocyte model in health, disease, and drug block." eLife vol. 8 e48890. Dec 2019, doi:10.7554/eLife.48890.
16. "Parallel Thread Execution ISA Version 8.5" Accessed: Nov. 24, 2024. [Online]. https://docs.nvidia.com/cuda/parallel-thread-execution/index.html.
17. C., Yves, C. D. Lontsi, and C. Pierre. "Rush-Larsen time-stepping methods of high order for stiff problems in cardiac electrophysiology." 2017, arXiv preprint arXiv:1712.02260.