Enhancing the efficiency of animal-alternative in-silico drug cardiotoxicity prediction through CUDA-based parallel processing

CUDA기반 병렬처리를 통한 동물대체 인실리코 약물 심독성 예측 효율성 증대

Iga Narendra Pramawijaya
20236132

Supervisor: Prof. Ki Moo Lim

Department of IT Convergence Engineering, Kumoh National Institute of Technology, Korea
iga@kumoh.ac.kr
December 2024

Introduction

Background

- Cardiovascular diseases are a leading cause of death globally.
- Current drug discovery methods, which rely heavily on animal testing, face ethical concerns and limitations in accuracy to predict human drug safety.
- In recent years, ***in silico*** **(computer-based)** methods have emerged as a promising **alternative**.
- However, their **efficiency** is often hindered by the complexity of simulating biological processes, and **large samples** that being processed.

Main Objectives

- Develop an efficient *in silico* cardiotoxicity prediction method capable of handling large sample sizes.
- Utilise GPU-based parallel processing with CUDA to accelerate simulations while maintaining accuracy.

CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA. It enables general-purpose computing on NVIDIA graphics processing unit (GPU).

Key Features

1. **Massive Parallelism:**
   - Utilizes thousands of GPU cores to process tasks simultaneously.
   - Ideal for large-scale computations like in silico simulations.
2. **Flexible Programming Model:**
   - Built on **C, C++, and Fortran**, making it accessible for developers.
   - Allows for custom thread and memory management.
3. **Hierarchical Threading:**
   - Threads are grouped into blocks, and blocks form grids.
   - Supports efficient task distribution for high-performance computing.

Why CUDA for This Research?

- Accelerates **simulations** by distributing tasks across **multiple threads.**
- Leverages GPU's architecture to handle large datasets efficiently -> Reduces simulation time.

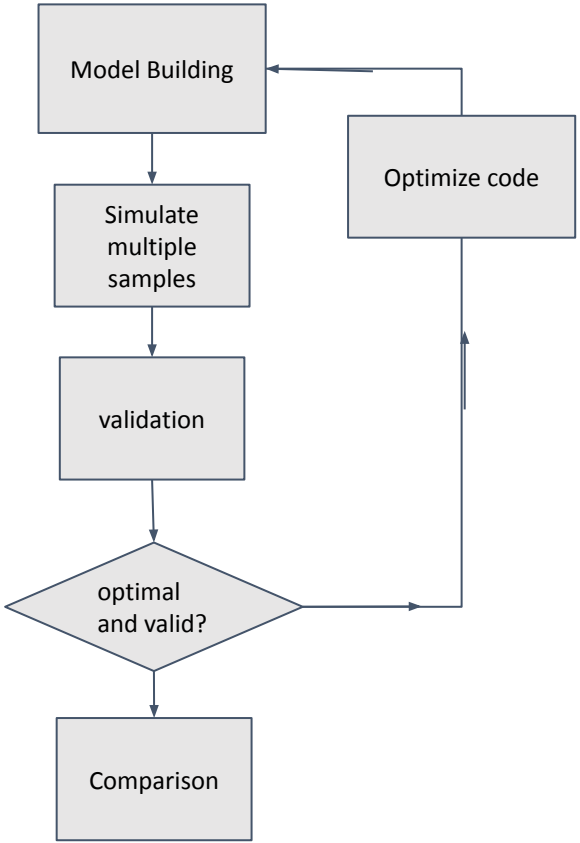
Methodology

Development Overview

The simulation process was designed to efficiently model cardiac electrophysiology and predict cardiotoxicity under drug-induced and control conditions. Each step leverages CUDA-based GPU parallelization to accelerate processing.

Steps:

1. **Parse CellML models and export to C.**
2. **Format C to CUDA**
3. **Apply Ordinary Differential Equation Solver**
4. Compile and run CUDA-enabled simulations.
5. Simulate for 1000 paces, with and without drug effects.
6. Output time-series data and biomarkers for analysis.

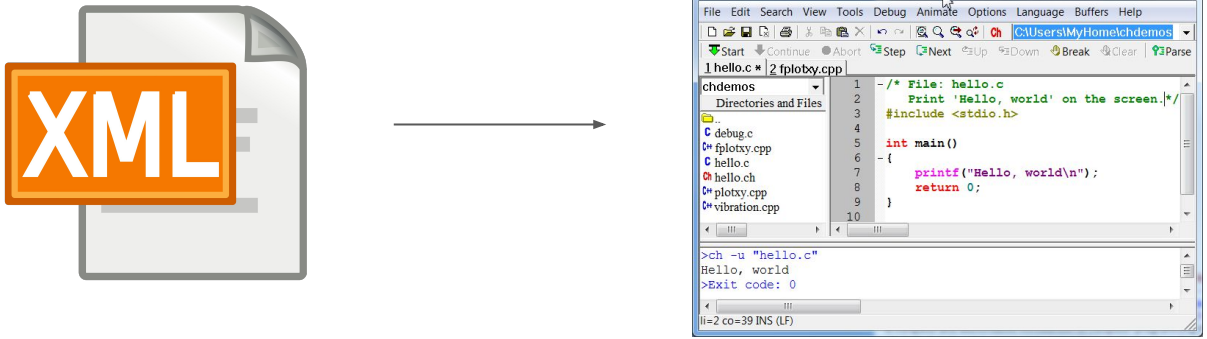


Simulator Development Flow

Methodology

Development Overview

1. Parse CellML models and export to C.



Model Selection: Three well-established cardiac cell models (ORd 2011, ORd 2017, and ToR-ORd).

Methodology

Development Overview

Before

```
Cellmodel *p_cell = new mar_cell_MKII();

void do_drug_sim_analytical(const double conc, std::array<double, 14> ic50,
const param_t* p_param, const unsigned short sample_id, Cellmodel *p_cell)
{
```

After

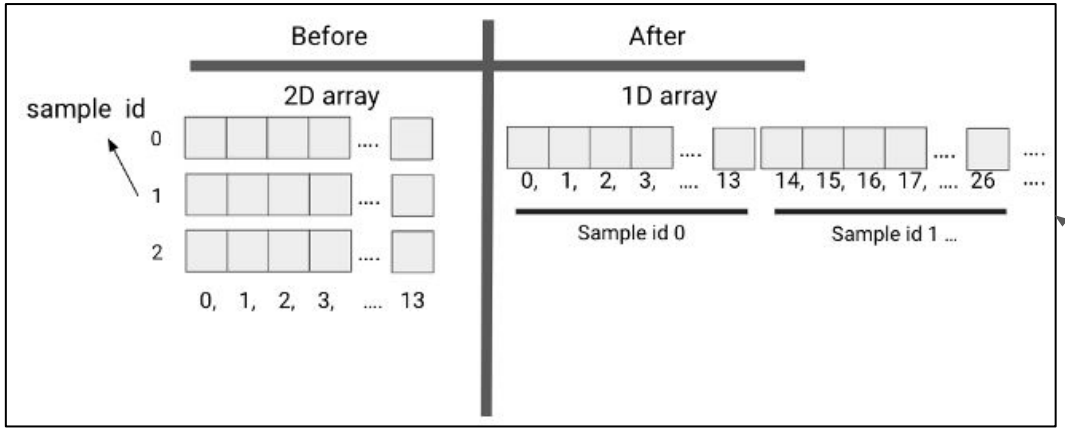
```
__global__ void do_drug_sim_analytical(drug_t d_ic50, double *d_CONSTANTS, double *d_STATES, double *d_RATES, double *d_ALGEBRAIC, double *dt_set){
    unsigned short sample_id;
    sample_id = threadIdx.x;
    int num_of_constants = 146;
    int num_of_states = 41;
    int num_of_rates = 41;
```

CUDA Implementation:

- **Converted CellML models to C, then simplify to CUDA.**
- Optimized GPU memory allocation (global, shared, constant).
  - Flattened multidimensional data structures for efficiency.

Methodology

Development Overview



CUDA Implementation:

- Converted CellML models to C code, then simplify to CUDA.
- **Optimized GPU memory allocation (global, shared, constant).**
  - **Flattened multidimensional data structures for efficiency.**

```
There are a total of 223 entries in the algebraic variable array.
There are a total of 43 entries in each of the rate and state variable arrays.
There are a total of 163 entries in the constant variable array.
```

```
ALGEBRAIC[3] = 1.00000/(1.00000+exp((STATES[0]+87.6100)/7.48800))
```

↓

```
ALGEBRAIC[(sample_id * 223) + 3] = 1.00000/(1.00000+exp((STATES[( sample_id * 43) +
0]+87.6100)/7.48800));
```

Methodology

Development Overview

Rush-Larsen

$$\frac{dy}{dt} = \alpha(1-y) - \beta y, \quad \xrightarrow{A = a\ ;\ B = \alpha+\beta} \quad \frac{dy}{dt} = A - By,$$

$$y(t+\Delta t) = y(t)e^{-B\Delta t} + \frac{A}{B}(1 - e^{-B\Delta t}).$$

Models and ODE Solvers:

1. ORd 2011 (Rush-Larsen solver)
2. ORd 2017 (Forward Euler solver)
3. ToR-ORd (Forward Euler solver)

Forward Euler

$$y_{(t+\Delta t)} = y_t + rate(y_t) \cdot \Delta t$$

```
void solveEuler( double *STATES, double *RATES, double dt, int sample_id)
{
	for(int i=0;i<43;i++){
	STATES[(43 * sample_id) + i] = STATES[(43 * sample_id) + i] + RATES[(43 * sample_id) + i] * dt;
	}
}
```

Simulation Results

Parameters Overview

Simulation Goal:

- Ensure GPU-based simulations replicate physiological outputs of CPU-based simulations.
- **Key Metrics**:
  - **Action Potential (AP).**
  - **Simulation time (seconds)**

- **Common Parameters (Drug-free and Drug-Induced)**
  - cell models: ORd 2011, ORd 2017, and ToR-ORd.
  - 1000 paces
  - 1000ms per pace
  - **Hardware Used**:
    i. **GPU**: NVIDIA RTX 4090.
    ii. **CPU**: 10-core Intel Xeon @ 2.50 GHz.

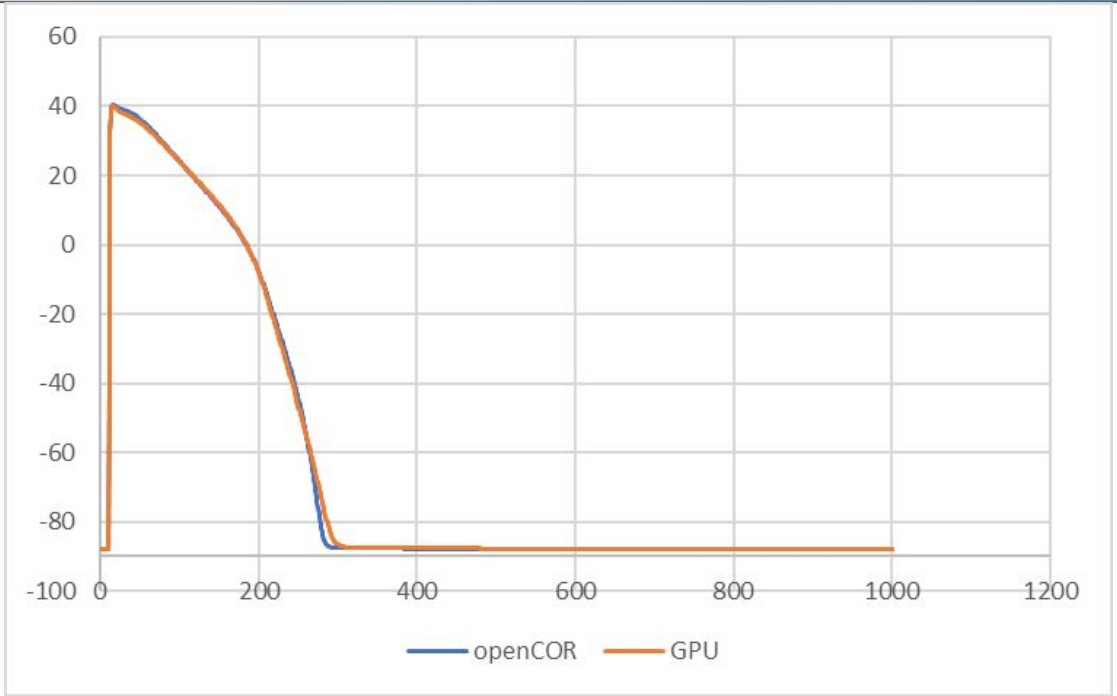
Drug-Free Simulation Parameters

- **Key Observations**:
  - **Compare GPU results with OpenCOR** (CPU-based) benchmarks.
  - Visual alignment of action potentials confirms fidelity.

Drug-Induced Simulation Parameters

- **Test Drug**: Bepridil, simulated at cmax 1 (33 mMol), cmax 2 (66 mMol), and cmax 4 (132 mMol).
- **Key Observations**:
  - Action potentials altered predictably with drug effects.
  - Compare GPU results with OpenCOR (CPU-based) benchmarks.

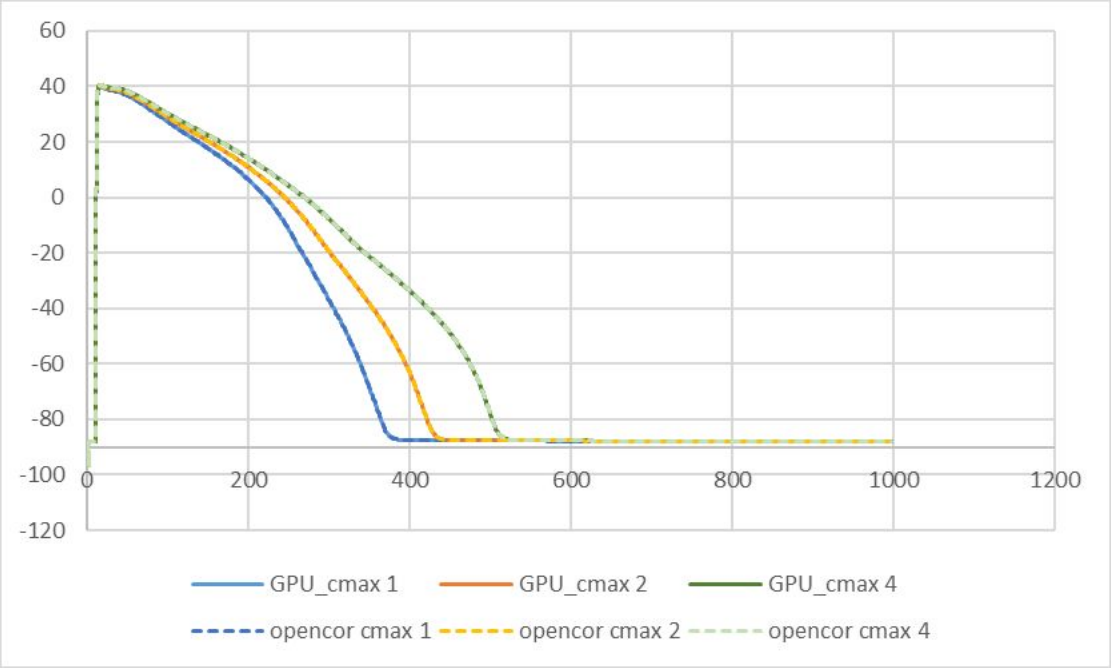
Drug-Free ORd 2011 Result Validation



Action Potential (mV) Shape of both CPU (blue) and GPU (orange) Result Using ORd 2011

The GPU-based simulation for the ORd 2011 model was validated against CPU (OpenCOR) results by comparing action potential plots and key biomarkers like APD under no-drug conditions. The findings showed near-identical outputs, confirming the GPU's accuracy in replicating physiological dynamics.

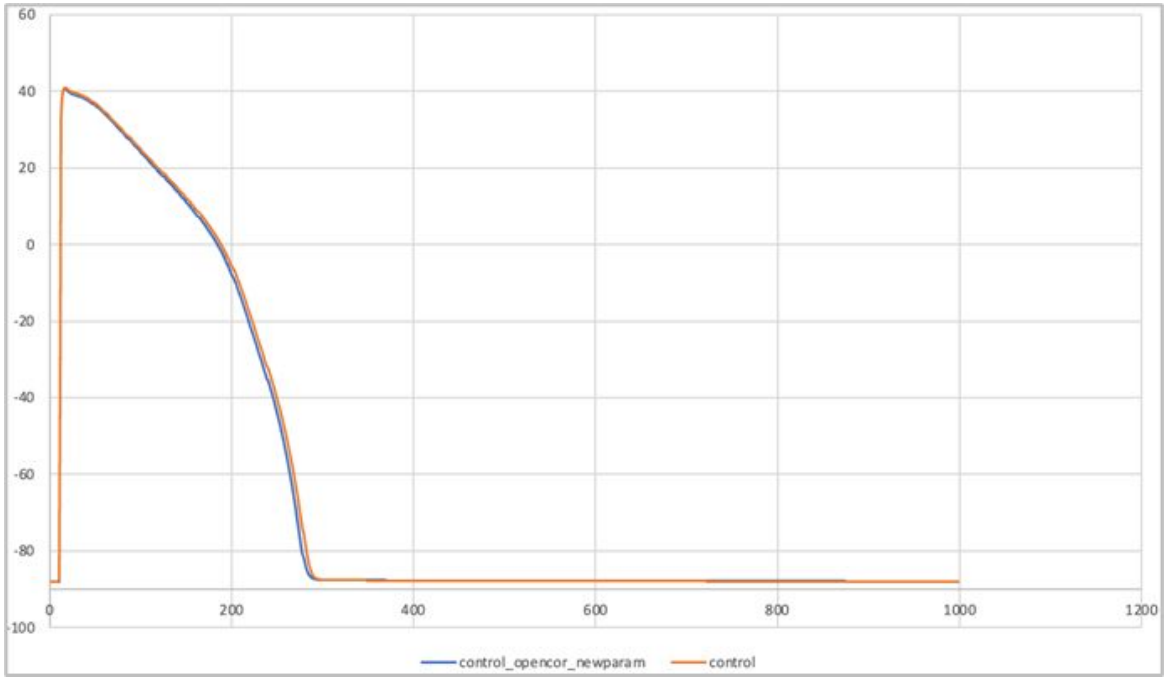
Drug-Induced ORd 2011 Result Validation



Action Potential Shape (mV) of both CPU (dashed) and GPU under drug effect Using ORd 2011

To validate the GPU-based simulation of the ORd 2011 model, this research compared its action potential traces to those generated by a CPU-based simulation (OpenCOR). This research consistently applied drug effects to both platforms by adjusting ionic current parameters using IC50 and Hill coefficients. The results demonstrated that the GPU simulation accurately replicated the CPU's output, confirming its reliability in simulating both normal physiological and drug-induced responses.

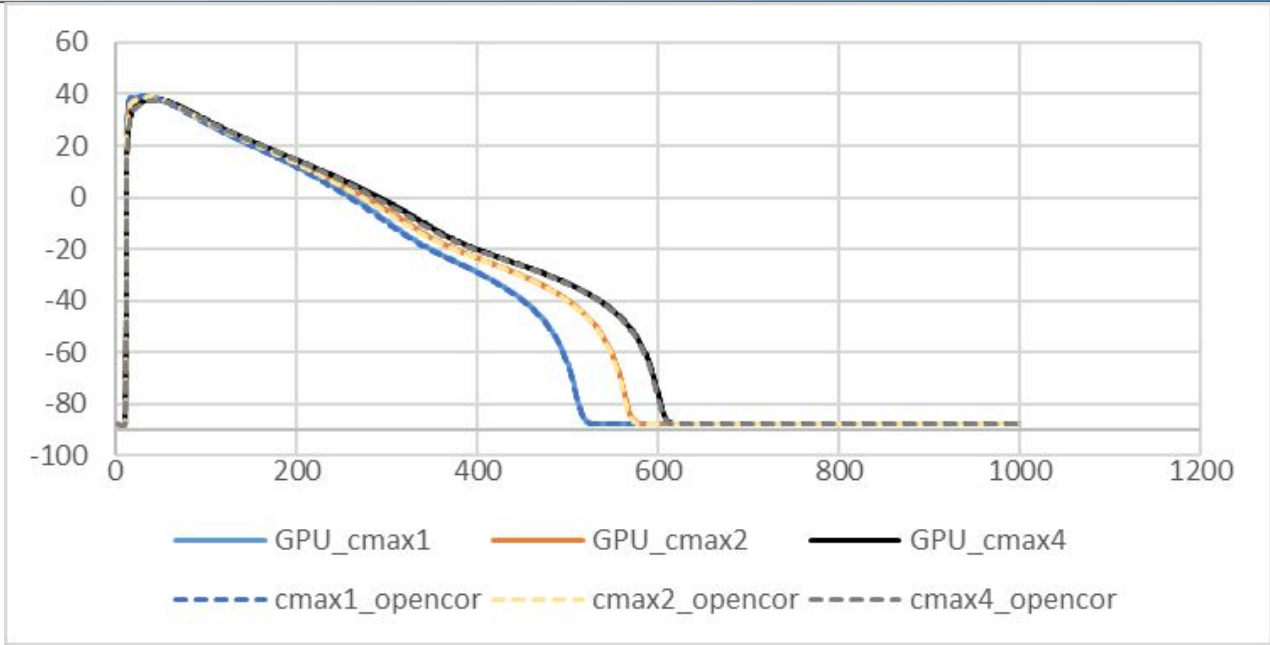
Drug-Free ORd 2017 Result Validation



Action Potential (mV) Shape of both CPU (blue) and GPU (orange) Result Using ORd 2017

The GPU-based ORd 2017 model was validated against a CPU-based reference (OpenCOR). By visually aligning action potential time-series data and analyzing key biomarkers, it is confirming the GPU's accuracy under no-drug conditions. Similar to the ORd 2011 model, the GPU closely replicated the CPU's output, demonstrating its reliability for the ORd 2017 model.

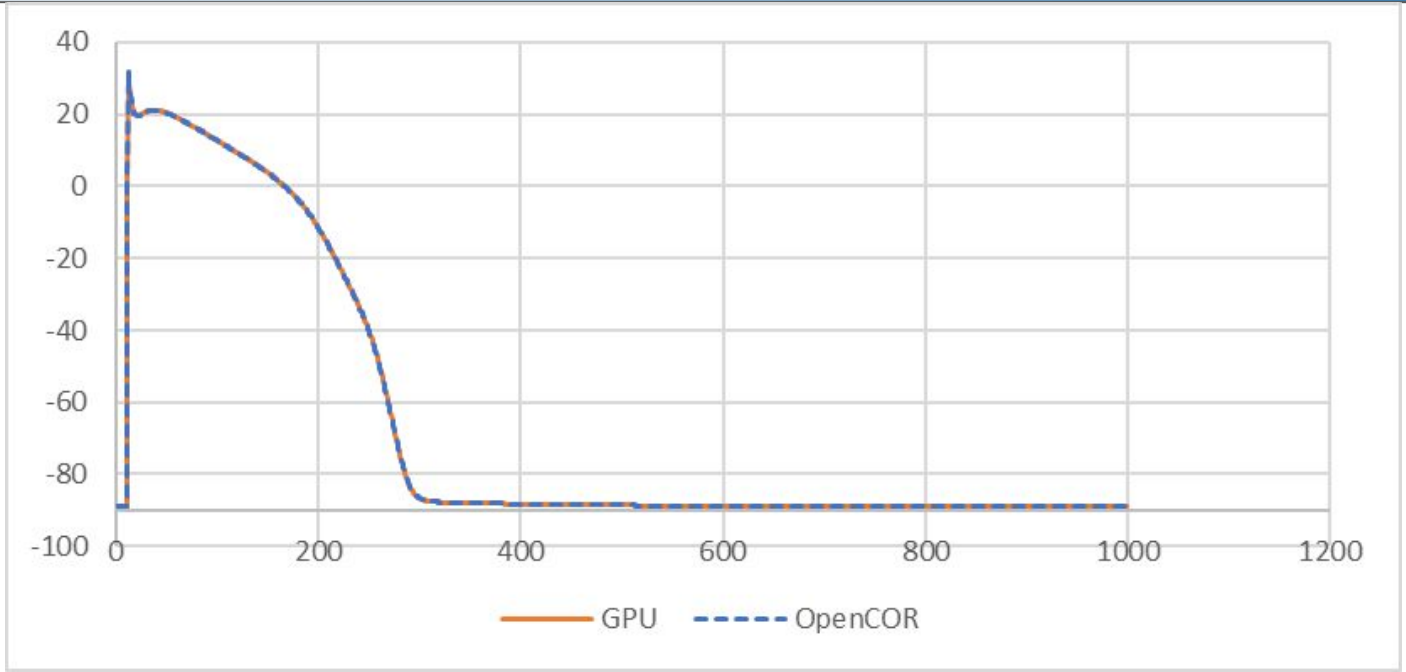
Drug-Induced ORd 2017 Result Validation



Action Potential Shape (mV) of both CPU (dashed) and GPU under drug effect Using ORd 2017

The GPU-based ORd 2017 model was validated under drug conditions by comparing its output to a CPU-based reference (OpenCOR). Drug effects were consistently simulated on both platforms. Analysis of action potential traces confirmed the GPU's accuracy in replicating physiological and pharmacological responses, even with drug effects.

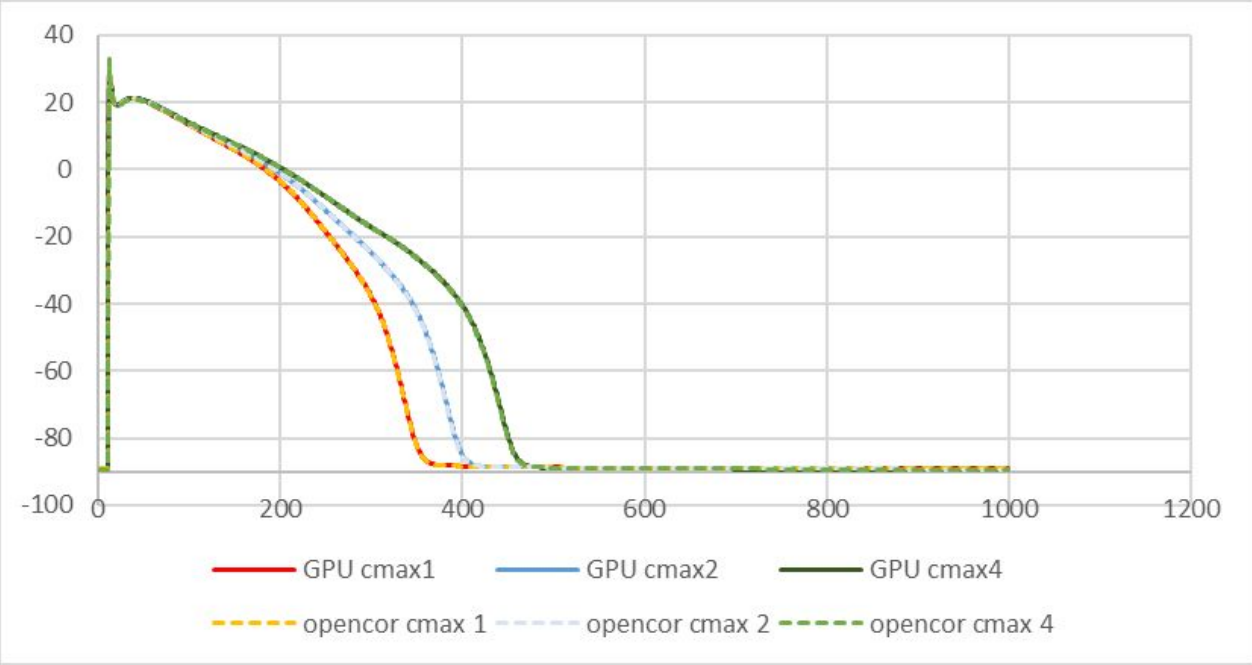
Drug-Free ToR-ORd Result Validation



Action Potential (mV) Shape of both CPU (dashed blue) and GPU (orange) Result Using ToR-ORd

The GPU-based ToR-ORd model was validated by comparing its output to a CPU-based reference (OpenCOR). Time-series plots of action potentials were compared, and key biomarkers were assessed under drug-free conditions. The GPU simulation accurately replicated the CPU results, confirming its reliability for the ToR-ORd model, with the forward Euler method.

Drug-Induced ORd 2017 Result Validation



Action Potential (mV) Shape of both CPU (dashed) and GPU under drug effect Using ToR-ORd cell model

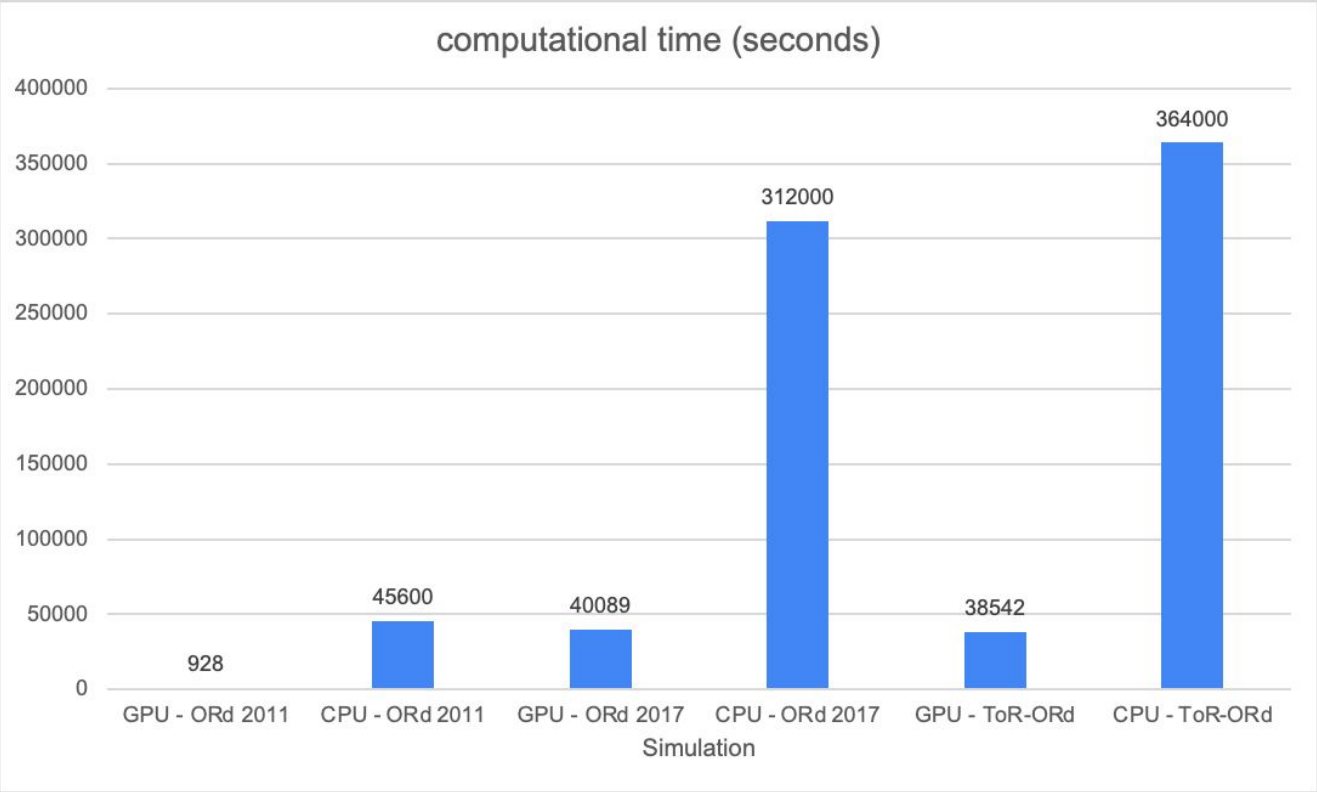
The GPU simulation accurately replicated the CPU-based simulations for the drug-induced ToR-ORd model, with no significant differences in action potential traces or biomarkers. This confirms the validity of the GPU-based simulation, even under drug-induced perturbations. The successful validation of drug effects highlights the GPU method's capability to reliably simulate complex pharmacological scenarios, reinforcing its utility as a powerful tool for investigating drug-induced cellular behaviors.

Computational Advantages

For the ToR-ORd model, GPUs outperform CPUs when simulating 847 or more samples, for the ORd 2011 model, it is 163 samples, and 1,028 samples for ORd 2017.

Beyond these thresholds, GPUs significantly reduce computation times, handling large-scale simulations without substantial performance loss.

For instance, simulating 8,000 samples with the ORd 2017 model takes 40,089 seconds on the GPU, compared to 312,000 seconds on a 10-core CPU.



Conclusion

- **Key highlights:**
  - Demonstrated the effectiveness of CUDA-based GPU parallelization for in silico drug cardiotoxicity prediction.
  - GPU simulations were up to **40.91x faster (ORd 2011)** than 10-core CPU methods, with runtimes remaining constant regardless of sample size.
  - GPU maintained high similarity in action potentials, biomarkers, and drug-induced effects across all models when compared to CPU.
- **Impact**:
  - Accelerates preclinical testing, reduces reliance on animal models, and lowers complexity in drug discovery processes.

This study positions GPU-based simulations as a transformative tool in the pharmaceutical industry, reducing the time, cost, and ethical concerns associated with traditional methods. With continuous development, it has the potential to revolutionize in silico drug discovery.

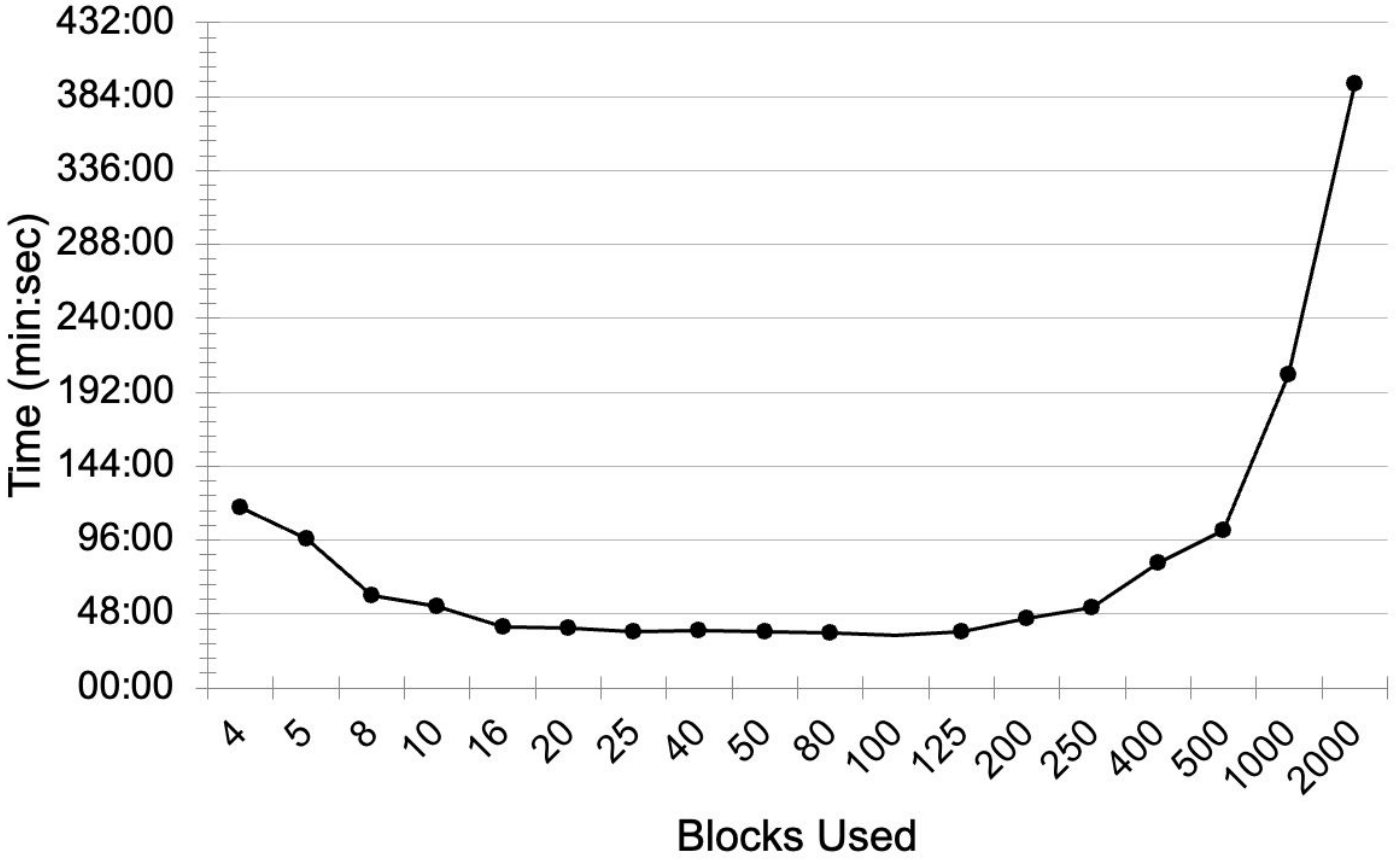
Thank you

Q&A

References

1. Jason Sanders and Edward Kandrot. 2010. CUDA by Example: An Introduction to General-Purpose GPU Programming (1st. ed.). Addison-Wesley Professional.
2. “What is OpenMP?” Accessed: Nov. 10, 2024. [Online]. Available: https://cvw.cac.cornell.edu/openmp/intro/what-is-openmp.
3. R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca and A. Lumsdaine, "Open MPI: A High-Performance, Heterogeneous MPI," 2006 IEEE International Conference on Cluster Computing, Barcelona, Spain, 2006, pp. 1-9, doi: 10.1109/CLUSTR.2006.311904.
4. A. Garny et al., “CellML and associated tools and techniques,” Sep. 13, 2008, Royal Society. doi: 10.1098/rsta.2008.0094.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
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. Available: 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]. Available: 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.
18. G.R. Mirams, Y. Cui, A. Sher, M. Fink, J. Cooper, B. M. Heath, et al. "Simulation of multiple ion channel block provides improved early prediction of compounds' clinical torsadogenic risk." 2011, Cardiovasc. Res. 91 (1), 53-61. doi:10.1093/cvr/cvr044.

Appendix



Trial & errors selecting number of thread per block

```
//
//This CellML file was generated on 8/25/2011 at 12:24:17 at AM using:
//
//COR (0.9.31.1409)
//Copyright 2002-2011 Dr Alan Garny
//http://cor.physiol.ox.ac.uk/ - cor@physiol.ox.ac.uk
//
//CellML 1.0 was used to generate this model
//http://www.cellml.org/
//

def model{Ohara_Rudy_2011} Ohara_Rudy_2011 as
    def unit millisecond as
        unit second {pref: milli};
    enddef;

    def unit per_millisecond as
        unit millisecond {expo: -1};
    enddef;

    def unit millivolt as
        unit volt {pref: milli};
    enddef;

    def unit millivolt_per_millimole as
        unit volt {pref: milli};
        unit millimolar {expo: -1};
    enddef;

    def unit per_millivolt as
        unit millivolt {expo: -1};
    enddef;

    def unit per_millivolt_millisecond as
        unit millivolt {expo: -1};
        unit millisecond {expo: -1};
    enddef;

    def unit millimolar as
        unit mole {pref: milli};
        unit litre {expo: -1};
    enddef;

    def unit per_millimolar as
        unit millimolar {expo: -1};
    enddef;
```

```
def model{ohara_rudy_cipa_v1_2017} ohara_rudy_cipa_v1_2017 as
    def unit millisecond as
        unit second {pref: milli};
    enddef;

    def unit per_millisecond as
        unit millisecond {expo: -1};
    enddef;

    def unit millivolt as
        unit volt {pref: milli};
    enddef;

    def unit millivolt_per_millimole as
        unit volt {pref: milli};
        unit millimolar {expo: -1};
    enddef;

    def unit per_millivolt as
        unit millivolt {expo: -1};
    enddef;

    def unit per_millivolt_millisecond as
        unit millivolt {expo: -1};
        unit millisecond {expo: -1};
    enddef;

    def unit millimolar as
        unit mole {pref: milli};
        unit litre {expo: -1};
    enddef;

    def unit per_millimolar as
        unit millimolar {expo: -1};
    enddef;

    def unit millimolar_per_millisecond as
        unit millimolar;
        unit millisecond {expo: -1};
    enddef;

    def unit per_millimolar_per_millisecond as
        unit millimolar {expo: -1};
        unit millisecond {expo: -1};
    enddef;

    def unit millisecond_per_millimolar as
```

```
// Tomek ... human cardiac ventricular action potential model:
//  - original implementation: 17/11/2018, Alfonso Bueno-Orovio, University of Oxford
//  - Reference:
//    Journal. Year;Volume(Issue):Pages. doi:
//    Title.
//    Authors.
//
// This code makes use of portions of a previous implementation of the O'Hara-Vir

def model{Tomek_model13endo} Tomek_model13endo as
    def unit millisecond as
        unit second {pref: milli};
    enddef;

    def unit per_millisecond as
        unit millisecond {expo: -1};
    enddef;

    def unit millivolt as
        unit volt {pref: milli};
    enddef;

    def unit millivolt_per_millimole as
        unit volt {pref: milli};
        unit millimolar {expo: -1};
    enddef;

    def unit per_millivolt as
        unit millivolt {expo: -1};
    enddef;

    def unit per_millivolt_millisecond as
        unit millivolt {expo: -1};
        unit millisecond {expo: -1};
    enddef;

    def unit millimolar as
        unit mole {pref: milli};
        unit litre {expo: -1};
    enddef;

    def unit per_millimolar as
        unit millimolar {expo: -1};
    enddef;

    def unit millimolar_per_millisecond as
```

CellML's XML for ORd 2011, 2017 and ToR-ORd respectively

Rush-Larsen method

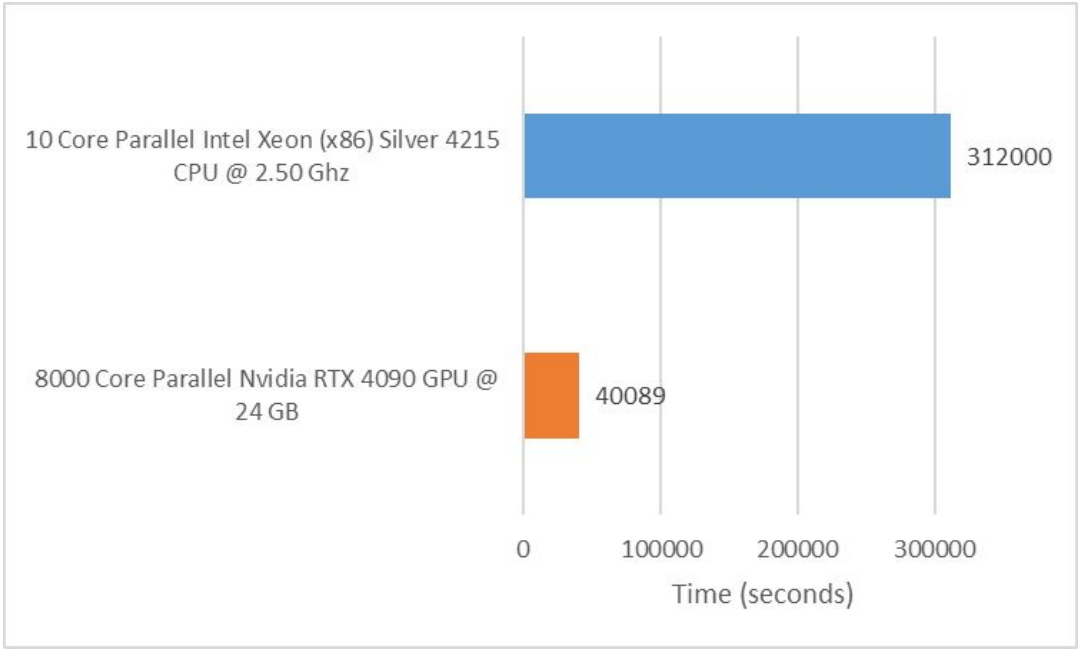
The Rush-Larsen method is a semi-implicit numerical technique designed to solve ordinary differential equations (ODEs) efficiently, particularly in biological and electrophysiological simulations like cardiac cell modeling. ORd 2011 included this solver in their paper and source code.

Key Features

- In cardiac models, gating variables (ion channel states) typically follow equations that resemble exponential decay or growth.
- The Rush-Larsen method analytically solves these exponential components, offering higher accuracy and stability for these variables.
- It is more stable for stiff equations with rapid changes.
- Efficient. By avoiding full numerical integration for the gating variables, -> reduces computational complexity
- Simplifies solving coupled ODEs by separating them into independent equations.

The Rush-Larsen Method, relies on the assumption that during sufficiently small time intervals, a system of differential algebraic equations becomes effectively uncoupled. One can then readily solve uncoupled differential equations one-by-one to obtain expressions for time evolution of state variables

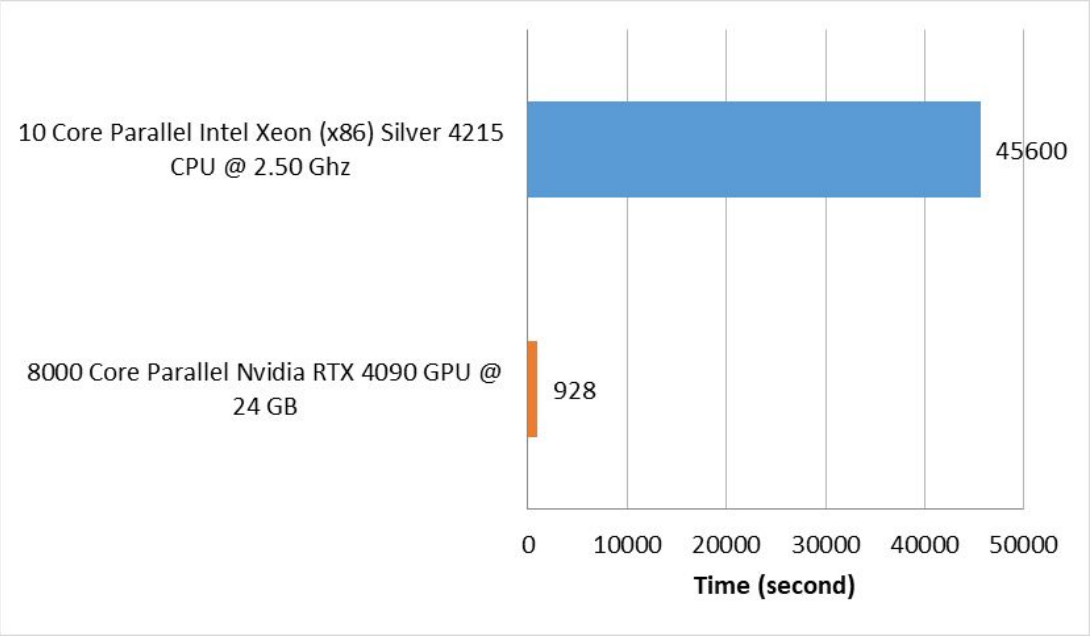
Computational Time in ORd 2017 Simulation



Simulation time comparison between GPU and CPU in ORd 2017

The GPU-based ORd 2017 model was validated under drug conditions by comparing its output to a CPU-based reference (OpenCOR). Drug effects were consistently simulated on both platforms. Analysis of action potential traces and key biomarkers confirmed the GPU's accuracy in replicating physiological and pharmacological responses, even with drug effects. Figure 3.5 illustrates the GPU's reliability for drug simulation scenarios.

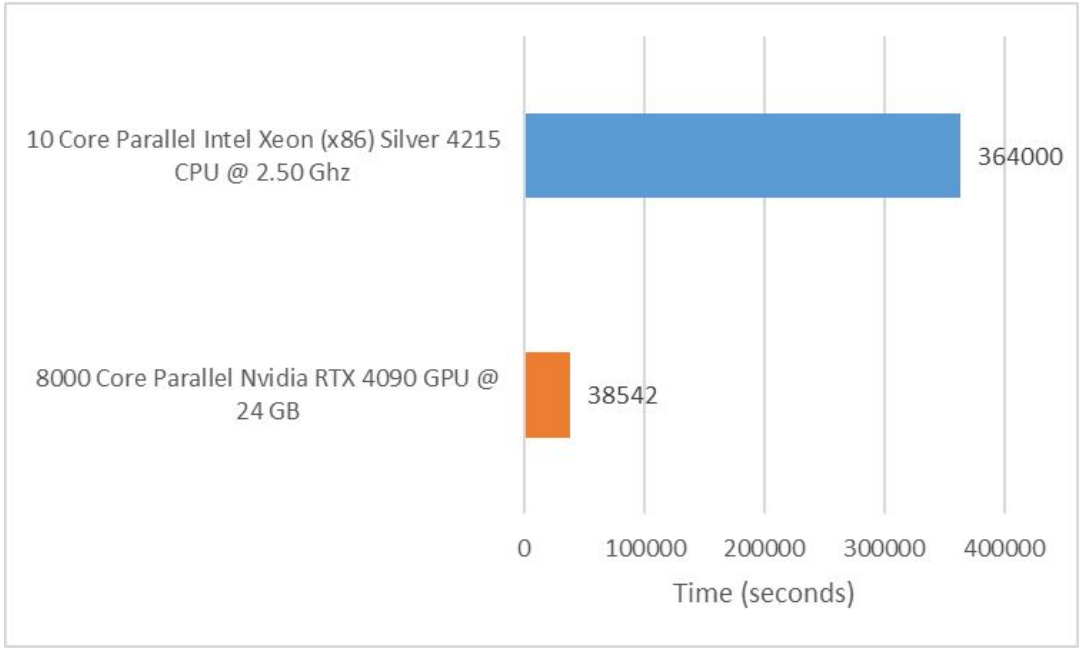
Computational Time in ORd 2011 Simulation



Simulation time comparison between GPU and CPU in ORd 2011

To assess computational efficiency, we compared GPU-based and CPU-based simulations of the ORd 2011 model. The GPU simulation, leveraging an NVIDIA RTX 4090, demonstrated superior performance, completing 8,000 simulations **(across 4 drugs and 2,000 samples per drug)** in a **fixed 928 seconds**. In contrast, the CPU simulation, utilizing a 10-core Intel Xeon Silver 4215, exhibited linear scaling, requiring approximately 45,600 seconds for the same task. The GPU's fixed runtime advantage made it significantly faster for simulations involving 163 samples or more.

Computational Time in ToR-ORd Simulation



Simulation time comparison between GPU and CPU in ToR-ORd cell model

The computational performance of the ToR-ORd cell model was evaluated using an NVIDIA RTX 4090 GPU and a 10-core Intel Xeon (x86) Silver 4215 CPU. Both platforms simulated 8,000 samples under the same conditions. The CPU's computational time increased linearly with the number of samples and pacing due to sequential processing. In contrast, the GPU's parallel architecture ensured consistent performance, highlighting the advantages of parallelization for efficient large-scale simulations.