# Appendix

## Project Structure

Codes for this research is available on Github repository of Computational Medicine Laboratory, Kumoh National Institute of Technology (<https://github.com/kit-cml>). There are some versions of the code in our Github profile, depending on the cell models and research scenario. This section will discuss deeply the code’s version for ORd 2011 cell model (<https://github.com/kit-cml/MultiConcGPU>). Codes explained in this appendix is in same condition as the time this thesis being written, including some files that might be deleted due to redundancy in the future. More recent updates are available in the Github repository. Structure of the repository can be written as such:

MultiConcGPU

├─ .gitignore

├─ Makefile

├─ Makefile\_commercial

├─ bin

│  ├─ CVAR

│  │  ├─ HF\_male.csv

│  │  └─ sens\_healthy\_male.csv

│  ├─ autorun.sh

│  ├─ control

│  │  ├─ IC50\_samples.csv

│  │  └─ init\_state.csv

│  ├─ drug

│  │  ├─ IC50\_terfenadine.csv

│  │  ├─ IC50\_verapamil.csv

│  │  └─ testing

│  │     └─ IC50\_Mexiletine.csv

│  ├─ drug\_sim

│  ├─ ic50\_sens

│  │  └─ mitoxantrone\_sens.csv

│  ├─ input\_deck.txt

│  ├─ mitoxantrone

│  │  └─ mitoxantrone\_100\_samples\_50\_conc.csv

│  └─ result

│     ├─ do\_not\_delete\_this\_folder

│     └─ state\_only.zip

├─ cellmodels

│  ├─ Ohara\_Rudy\_2011.cpp

│  ├─ Ohara\_Rudy\_2011.hpp

│  ├─ cellmodel.hpp

│  └─ enums

│     ├─ enum\_Ohara\_Rudy\_2011.hpp

│     └─ enum\_ord2011.hpp

├─ main.cu

├─ modules

│  ├─ cipa\_t.cu

│  ├─ cipa\_t.cuh

│  ├─ drug\_conc.cpp

│  ├─ drug\_conc.hpp

│  ├─ glob\_funct.cpp

│  ├─ glob\_funct.hpp

│  ├─ glob\_type.cpp

│  ├─ glob\_type.hpp

│  ├─ gpu.cu

│  ├─ gpu.cuh

│  ├─ gpu\_cu\_.backup

│  ├─ gpu\_cuh.backup

│  ├─ gpu\_glob\_type.cu

│  ├─ gpu\_glob\_type.cuh

│  ├─ param.cpp

│  └─ param.hpp

└─ test\_compile.bat

The main structure, such as folder names, will less likely to be revised. Scripts inside the folder, especially in the ‘modules’ folder, more likely to be changed by removing some redundant functions. Next section will deeply discuss each file and their possibility of redundancy.

## Root folder

Root folder is the main folder that contains makefile, gitignore file, main code, and test\_compile.bat. The next sub section will discuss each of them deeply.

## Makefile

This script is used for compiling the whole project in correct order, and enables easy clean-up and re-compilation. Binary files will be cleaned, and the simulator will be re-compiled with ‘make clean all’ command.

## .gitignore

.gitignore file specifies files and folders to be ignored by git, the version control used in this research. The git will ignore simulation results, binary files, CUDA related libraries, log files, and a jupyter notebook used in plotting. Below is the list of files and folder we ignore in the research:

\*.i

\*.ii

\*.gpu

\*.ptx

\*.cubin

\*.fatbin

.DS\_Store

bin/drug\_sim

\*.o

\*.plt

\*.out

bin/result/\*/\*.csv

bin/result/\*/\*/\*.csv

bin/result/parse.ipynb

output.\*

\*.old

bin/result/\*

## main.cu

The main.cu file serves as the primary entry point for executing the drug simulation program. This file orchestrates the interaction between the core modules, manages the GPU-based computations, and handles input and output processes. Every CPU related orchestration for the simulation happen in main.cu. Its primary responsibilities are initialisation, loading input data, initialise GPU environment, core number calculation, executing simulation, output handling, memory clean-up, logging, and making sure all input and output are correct. We found that CUDA debugger does not really help the debugging process due to unique parallelisation in this research. Self-created debugging points were also introduced in the main.cu.

## test\_compile.bat

This file is a result from previous iteration. There was a moment that I wanted to develop the simulator for Windows Operating System outside of Linux. It is finally decided this repository will be containerised using Docker instead of separately develop Windows version of this simulator. This script will more likely to be deleted in the next update.

## ‘bin’ Folder

The bin folder serves as the central directory for storing various input files, intermediate data, and simulation outputs. This folder is organised into subdirectories that categorise data for ease of access and maintainability.

## CVAR

This folder will be used to store inter-individual conductance variability file for future development of this research.

## Control

The control subfolder contains files that are fundamental to running simulations under control conditions (without drug effects). It typically includes:

* IC50\_samples.csv: This file provides a baseline reference for the IC50 values of various ionic currents, which are used for comparison in simulations involving drug-induced conditions. IC50 file formatted as:

drug\_name,conc,ICaL\_IC50,ICaL\_h,IK1\_IC50,IK1\_h,IKs\_IC50,IKs\_h,INa\_IC50,INa\_h,INaL\_IC50,INaL\_h,Ito\_IC50,Ito\_h,hERG\_IC50,hERG\_h

bepridil,0,2704,0.6954,NA,NA,NA,NA,2371,1.984,1947,1.473,NA,NA,139.1,3.199

bepridil,0,2818,0.6409,NA,NA,NA,NA,2734,1.225,1802,1.212,NA,NA,181.4,2.77

bepridil,0,3939,0.718,NA,NA,NA,NA,3064,1.108,1921,1.421,NA,NA,194.8,0.8339

* init\_state.csv: This file contains the initial states of all variables in the cell model, such as membrane voltage and ion concentrations. These states serve as the starting point for the simulations, ensuring consistent and reproducible results.

By isolating control data in its dedicated subfolder, the simulation framework ensures clarity when comparing control and drug-altered conditions.

## drug

The drug subfolder stores input files related to simulations involving specific drugs. These files typically include the IC50 values and Hill coefficients for the drugs under study, which define their effects on various ionic currents. For example, IC50\_terfenadine.csv and IC50\_verapamil.csv: These files describe the pharmacological properties of terfenadine and verapamil, respectively.

## result

The result subfolder is used to store simulation outputs, ensuring that data generated during the computational runs is systematically archived. Simulation output classified as two types, the init file, and post-processing folder. Init file is the output from the first phase of the simulation. It contains the simulation’s state when gradient of action potential is at its steepest. Named \_state\_only.csv in a folder named from the IC50 file, this initial state usually shaped like:

0,-89.14848,0.01218,0.00007,12.14017,12.14051,144.11277,144.11273,1.56242,1.55949,0.00008,0.00074,0.83611,0.83590,0.68309,0.83534,0.00015,0.53126,0.28205,0.00093,0.99964,0.56097,0.00047,0.99964,0.61777,0.00000,1.00000,0.92678,1.00000,0.99980,0.99996,1.00000,1.00000,0.00051,0.00087,0.00070,0.00083,0.99790,0.00002,0.00060,0.27721,0.00017,0.00000,0.00000,999.00000

1,-89.14444,0.01211,0.00007,12.12804,12.12838,144.10139,144.10135,1.55818,1.55521,0.00008,0.00074,0.83604,0.83582,0.68296,0.83526,0.00015,0.53107,0.28182,0.00093,0.99963,0.56062,0.00047,0.99963,0.61738,0.00000,1.00000,0.92684,1.00000,0.99980,0.99996,1.00000,1.00000,0.00051,0.00086,0.00070,0.00083,0.99789,0.00002,0.00060,0.27735,0.00017,0.00000,0.00000,999.00000

With the first column as sample ID number, and the last column acts as number of pace recorded. This file will become the input of the second phase. Post-processing folder contains a biomarker file and time-series data per sample. The biomarker file will be used for next researches that requires data analysis. In the research, I validate the result, and manually analyse action potential and ion channels with the time-series files.

This subfolder's structure allows for easy retrieval of outputs for validation and analysis, ensuring that results from different runs are preserved without overwriting.

## ‘cellmodels’ Folder

This folder contains mathematical models of the cell. The folder is created to make cell models easier to change and modify. I put the converted C file in here, with an enum file as an addition.

## Ohara\_Rudy\_2011.hpp

This file declares all the function in ORd 2011 cell model. This header is required because the main parallelisation will be handled by gpu.cu, hence requiring an object-oriented programming approach. Notice that every function here is a kernel function, since each cell model will be simulated in parallel using each computing thread of the GPU.

## Ohara\_Rudy\_2011.cpp

This file contains the mathematical cell model and its solver. I added the solver and drug effect simulation function in this script. The rest of the script follows the conversion from CellML

## Cellmodel.hpp

This file contains the general interface for the cell models. It declares common functions that all cellmodel should have.

## enums/enum\_Ohara\_rudy\_2011.hpp

This header act as a ‘translation dictionary’ for each variable in the cell model file. Enumeration required to easily track each variable, so instead of looking at numbers, I looked at pre-defined variable names corresponds to each correct values in the cell model.

## ‘modules’ Folder

The modules folder is a vital component of the project, containing some essential files that implement the core functionalities of the simulation. These files are modularised into source (.cpp or .cu) and header (.hpp or .cuh) formats, ensuring that the codebase is both organised and extensible. Each module serves a specific purpose, contributing to the overall framework of the simulation pipeline. As in the writing of this thesis, We found that some scripts were redundant and will be removed in the next updates.

## cipa\_t.cu and cipa\_t.cuh

These files handle data type related to the formatting of CiPA (Comprehensive in vitro Proarrhythmia Assay) metrics, which are critical and become standard for assessing drug-induced proarrhythmia risks. All implementation in cipa\_t.cu has been migrated to its header file (cipa\_t.cuh) and will be removed in the next updates.

## drug\_conc.cpp and drug\_conc.hpp

These files manage the drug concentration data used in the simulations. These act as utility file to lookup if user did not declare the concentration value and drugs used is in the scope of CiPA. Lookup process is happening in the CPU domain.

## glob\_funct.cpp and glob\_funct.hpp

Global function or glob\_funct encapsulate global utility functions that are used across various parts of the codebase. At the time this thesis was written, this script regulates the reading of flags and input deck. There are some unused functions that will be removed in the future.

## glob\_type.cpp and glob\_type.hpp

The glob\_type files define and manage the global data types and structures used throughout the project. For now, this script is being used to store IC50 data. The glob\_type.hpp is found unused and will be removed in the next update.

## gpu.cu and gpu.cuh

The gpu files represent the core computational engine of the simulation, leveraging the parallel processing capabilities of NVIDIA GPUs to achieve significant speedups in simulation tasks. These files are essential for the framework's computational efficiency, enabling the processing of thousands of samples in parallel.

gpu.cu is the implementation file that contains CUDA-specific kernels and device functions. CUDA kernels are at the heart of the GPU simulation, responsible for solving the differential equations of the cell models, processing drug interactions, and performing large-scale computations for multiple samples concurrently. The kernels in this file are designed to optimise memory usage and minimise latency, ensuring the efficient execution of simulations. Header gpu.cuh serves as the header file that defines the interface for the GPU-specific functionalities implemented in gpu.cu. It declares the prototypes of CUDA kernels and device functions, providing a structured entry point for other parts of the code to interact with GPU-based operations.

## param.cpp and param.hpp

These scripts manages the simulation parameters. It reads the input deck file and put them into the code as variables. The param.cpp implements functions to read the input file, load, parse, validate parameter files, and act as a failsafe if one or more parameters or parameter file is not readable. This script has the default values and path of each required numbers and files. This file ensures that the simulation runs with accurate and user-defined settings. Header of param.hpp declares variables used to store these parameter values and ensuring that they are well defined in the simulation.

## Critical Scripts

This section highlights the key scripts within the framework, focusing on their roles and significance in executing and customising simulations. These critical scripts form the backbone of the codebase, enabling efficient management of computations, GPU operations, and parameter configurations. If there is any issue with the code, 80% of the issue can be solved by looking at these three script first. Most of debugging, adjustement, and modification happen in these scripts. Below is a detailed breakdown:

## Main script

The main script (main.cu) serves as the central controller of the simulation framework. It is the entry point of the program, responsible for managing the simulation's overall workflow, from initialisation to the final output. Below are its primary responsibilities and modifications related to main.cu:

* Simulation Parameter Parsing
  + Reads user-provided inputs, such as input\_deck.txt, initial state files, and IC50/Hill coefficient files.
  + Validates the correctness and presence of required input files.
  + Flags any missing or improperly formatted files, preventing the simulation from starting.
* Simulation Setup
  + Loads and processes input parameters from the provided files.
  + Sets up the pacing protocol, drug concentrations, and simulation-specific configurations.
  + Handles initialisation for multiple samples and conditions.
* Memory Management
  + Allocates and deallocates memory for the CPU and GPU, ensuring efficient resource utilisation using core (thread) per block calculation.
  + Transfers data such as parameters, initial states, and configurations between host (CPU) and device (GPU).
* GPU Kernel Launch
  + Initiates GPU computations by invoking the kernel functions for parallel execution.
  + Ensures proper thread and block configurations to optimise performance for the given workload.
* Result Handling
  + Collects output data from the GPU and processes it for storage.
  + Saves simulation results to the designated output directory.
  + Generates logs for debugging and validation purposes.
* Error Handling and Debugging
  + Includes checks to identify and report common errors, such as missing inputs, memory allocation failures, or kernel launch issues. If something is wrong with the kernel function in the GPU, the code always goes back to this script, assuming the kernel function has done even with no output.
  + Provides detailed logs to help trace and resolve errors during simulation execution.

## GPU control script

The GPU control script (gpu.cu and gpu.cuh) is the core component that drives the high-performance computations. It includes all GPU-specific operations, such as kernel implementations, and parallel execution logic. The kernel simulates the electrical behaviour of cardiomyocyte cells over time, under the influence of a specific drug at a given concentration. This script manages the distribution of workloads across GPU threads and blocks, ensuring that simulations are processed efficiently.

Key responsibilities of this script include:

* Calling the numerical solvers (e.g., Forward Euler, Rush-Larsen) for simulating cardiac cell dynamics.
* Managing steps and loops of simulation, such as calling drug effect applying function.
* Calculating metrics for biomarkers, time series data, and such related to simulation, or calculated during simulation loops.

Most performance issues or GPU errors, such as kernel launch failures or memory overflows, can be traced back to this script. It is also the primary location for implementing modifications to the numerical methods or adding new solvers.

## Parameters

The parameter script (param.cpp and param.hpp) defines and manages the input parameters required for the simulation. It acts as a configuration layer, ensuring that all essential variables are correctly initialised and passed to the simulation workflow. This script is also responsible for providing default values for parameters when they are not explicitly defined in the input files. It ensures that the simulation remains robust against incomplete or inconsistent configurations.

Adjustments to this script are necessary when:

* Adding new features or variables to the simulation.
* Customising the simulation for different biological models or experimental setups.
* Debugging errors related to undefined or mismatched parameters.

## Commands and Flag Usages

This section outlines the essential commands and flags used to execute, customise, and manage the simulation framework. This part tend to face human error as in wrong path. The error usually shown as the file being read as 0, or file not found. Also the code being run without proper compilation previously. The first phase of the simulation’s command and flags run as below:

./drug\_sim -input\_deck input\_deck.txt -hill\_file drug/IC50\_file.csv

Second phase of the simulation can be run as below:

./drug\_sim -input\_deck input\_deck.txt -hill\_file drug/IC50\_file.csv -init\_file initfile.csv

The commands above represent two primary phases of the simulation process. In the first phase, the simulation is initiated with the essential input deck and Hill coefficient file. The -input\_deck flag specifies the configuration file (input\_deck.txt), which contains simulation parameters such as pacing details, cell models to be used, and output settings. The -hill\_file flag points to a CSV file (drug/IC50\_file.csv) containing IC50 and Hill coefficient values for the drug being simulated. This step generates initial conditions and baseline data required for further simulations.

In the second phase, the simulation proceeds with additional parameters specified by the -init\_file flag. This flag allows the user to include an initial state file (initfile.csv), which represents the system's state at the end of the first phase. By leveraging this initial state, the second phase can bypass reinitialisation and focus on computing drug effects or specific pacing protocols. This phase is particularly useful for testing drug interactions or extended pacing simulations without repeating initial computations.

In other cell models, more input is required and uses the same method to be registered in the simulation. ORd 2017 requires additional -herg\_file as input, since this newer cell model needs herg fitting value from CiPA. This update increases the simulation accuracy. A common ORd 2017 simulation will be called as such:

./drug\_sim -input\_deck input\_deck.txt -hill\_file drug/IC50\_file.csv -herg\_file drug/herg.csv

While herg.csv is structured as:

Kmax,Ku,n,halfmax,Vhalf,slope

5594000,0.0001719,0.9374,147200000,-61.34,NA

Each drug will have their own hERG file.

Also in three of the cell models, future update related to inter-individual variability will be applied. In the future, these cell models will require additional -cvar\_file (conductance variability) as input. This update will enrich the simulation capability to simulate drug effect in different population segments such as healthy people compared with people living with heart failure history. This update will increase the simulation accuracy and variability. A common ORd 2017 simulation with inter-individual variability will be called as such:

./drug\_sim -input\_deck input\_deck.txt -hill\_file drug/IC50\_file.csv -herg\_file drug/herg.csv -cvar\_file cvar.csv

## Troubleshooting

This section provides guidance on addressing frequent issues encountered when using the simulation framework. These troubleshooting steps are designed to assist users in diagnosing and resolving problems effectively. Especially in the situation where CUDA debugger were not informative enough due to the uniqueness of the parallelisation.

* File Not Found Errors

Problem: The simulation fails with an error indicating that a required file is missing or cannot be accessed.

Cause:

* + Incorrect file path provided in the command-line arguments.
  + The file does not exist in the specified directory.

Solution:

Verify the file path in the command. Ensure the paths match the directory structure (e.g., drug/IC50\_file.csv). Check if the file exists in the specified folder. If not, create or move the file to the correct location. Ensure file permissions allow read access.

* Compilation Errors

Problem: Compilation fails, showing errors related to missing libraries or incompatible flags.

Cause:

* + Required CUDA or GCC/G++ versions are not installed.
  + Incorrect flags in the Makefile.

Solution:

Verify that the system has the correct version of CUDA (e.g., version 11.x or later).Check that nvcc and gcc are installed and added to the system's PATH. Review and update the Makefile to match the system configuration. For instance, ensure paths to CUDA libraries and headers are correct.

* Zeroes or Incomplete Output

Problem: The output files contain no data or are incomplete.

Cause:

* + Incorrect input parameters or missing initial state files.
  + Errors in the numerical solver or GPU execution.

Solution:

Ensure the input deck (input\_deck.txt) is correctly formatted and contains all required parameters.Verify that the initial state file (initfile.csv) exists and is properly formatted. Check the GPU kernel execution logs (printed in the terminal) for errors (e.g., memory overflows or segmentation faults).

* GPU Kernel Launch Failures,

Problem: The simulation crashes during execution, or right just before interacting with GPU kernel function.

Cause:

* + Insufficient GPU memory to handle the workload.
  + Incompatible GPU architecture flags in the Makefile.

Solution:

Reduce the number of samples or concentrations in the input parameters to fit within available GPU memory. For information, in average it takes 2 MB of GPU memory to simulate one sample. Verify the GPU's compute capability and ensure the Makefile includes the correct -arch flag (e.g., -arch=sm\_86 for NVIDIA RTX 4090).

* Runtime Errors Due to Command-Line Arguments

Problem: The program crashes or produces unexpected results when running with specific arguments.

Cause:

* + Missing or incorrectly formatted command-line arguments.

Solution:

Ensure all required arguments (-input\_deck, -hill\_file, -init\_file, etc.) are included and point to valid files. Refer to the section on "Commands and Flag Usages" for examples of valid commands.

* Mismatched Results

Problem: The GPU simulation results do not match the expected outputs from the CPU simulation.

Cause:

* + Numerical instability or precision issues in the GPU solver.
  + Issue with drug effect implementation

Solution:

Double-check the numerical solver configurations to ensure consistency between CPU and GPU implementations. Compare the input data files to confirm they are identical for both simulations. Start comparison from non-drug, control data first. If control from both CPU and GPU shows no noticeable difference in action potential shape, it is confirmed that the issue comes from drug effect implementation.

Check the IC50 input, current development applies these values to the ratio of the gate instead of the gate. Previous implementation may implement directly to the gate. These two kinds of IC50 files can be distinguished by looking at the orde of the numbers. Old implementation method usually uses thousands or hundred of thousands, while more recent version usually stays at most around the thousands.

If the IC50 uses a more recent version, ensure in the drug effect implementation function, it does not change the gate, but instead the ‘\_b’ of the gate (GNaL\_b for example).

* Slow Performance on GPU

Problem: The GPU simulation runs slower than expected, or it runs more than 24 even 48 hours for 1000 pacing or less.

Cause:

* + Inappropriate workload size or insufficient parallelisation.
  + Suboptimal GPU memory management.
  + Less compatible GPU clock speed.

Solution:

Increase the workload size to fully utilise GPU cores (e.g., simulate more samples or concentrations). Profile the simulation using tools like nvprof or Nsight to identify bottlenecks in memory transfers or kernel executions. Ensure the simulation runs on a more recent GPU generation from NVIDIA, specifically the RTX series (RTX 3080Ti, RTX 4090, etc.). Also for information, this simulation tested and optimised for a gaming grade PC GPU with clock speed around 900-1200 MHz. Some of server-grade GPU has a lot of GPU memory but lacking of clockspeed. I conducted a test on a server grade GPU, NVIDIA T4 with 81 GB of GPU memory, but it was considered too slow. It finished the job simulating 100 pacing of ORd 2011 with drug effect in around 36-40 hours.

* Memory Overflows or Leaks

Problem: The simulation crashes with strange, unreadable error message, makes us cannot access the GPU, or hangs due to excessive memory usage.

Cause:

* + Improper memory allocation or deallocation in GPU kernels.

Solution:

Inspect the GPU control script (gpu.cu) for memory management issues.

Use tools like cuda-memcheck to debug memory leaks or access violations.

* Compatibility Issues

Problem: The code does not work on certain hardware or software configurations. Signed by a fail in running the kernel script with no error at all.

Cause:

* + Dependency mismatches, such as outdated libraries or drivers.

Solution:

Update the GPU drivers to the latest version recommended by NVIDIA. Ensure the CUDA version matches the code’s requirements (CUDA 11.x). Also ensure the Makefile includes the correct -arch flag (e.g., -arch=sm\_86 for NVIDIA RTX 4090). The code is not tested on other than GTX or RTX family. The oldest GPU we have ever tested the code was GTX 1660 Ti.