Yecheng Yang

Computer Science

High Honor Thesis

**A System for Simulating Gene Regulation Networks at Tissue Scale with Delay Differential Equations**

**Abstract:**

Complex biochemical systems such as gene regulation networks are often represented as delay-differential equations (DDEs). However, unlike ODEs, there are few large-scale systems available for simulating DDEs, and fewer still for high-performance simulations. To meet the growing need for simulating tissue-level systems of intercellular signaling gene regulation networks, we present a high-performance DDE simulation framework. By using compile-time specialization and optimization, our framework creates highly compact and efficient data structures with a small amount of configuration for the model being simulated. It also supports GPU acceleration of highly parallel models, which can be used to increase the simulation speed of large tissues or increase the search speed among independent simulations in a reaction parameter search. We demonstrate congruence of our simulation results with prior special-purpose simulations, and the superior CPU and GPU performance of our framework over those same frameworks, while also supporting greater flexibility and configurability of the systems being modeled.

**Introduction:**

This honor thesis originates from my previous research on building a computer system for simulating and analyzing a regulatory biological network called zebrafish segmentation clock network. In particular, the computer system builds on top of a list of species related to the biological network (such as mRNAs and proteins of *her1*, *her7*, *her13*) and a set of differential equations describing reactions (such as protein synthesis, protein degradation and mRNA synthesis) in this network. The system workflow consists of several phases: parameter estimation, simulation, feature extraction, testing. Parameter estimation is used to generate parameters though genetic algorithm in hope of finding a “desirable” parameter set whose simulation result “fits” our biological model. Simulation, based on a particular parameter set, happens at tissue level (multiple cells) and is responsible for replicating the entire biological process for a certain period of time and recording concentration levels of each species during this process. Feature extraction goes through concentration levels recorded in the previous step and turns those values into properties such as period, amplitude and synchronization score. At last, during testing, those properties are compared to standards created based on experimental data to determine if results are “desirable” and “fit” the model.

To meet the need of the researches in regulatory biological networks, various systems and packages have been created. Some of such packages provide user-friendly interface, yet they either only conduct simulations of small-scale or focus on ordinary differential equations and simply ignore delay differential equations. Both of the practices effectively prevent researchers from building accurate and comprehensive models. Due to the time delay in certain biological processes, delay differential equations are sometimes crucial in analyzing such network structures. For example, an important section of zebrafish segmentation clock network is gene expression, consisting of two main parts: gene transcription and mRNA translation. Because of the complexity of gene transcription and mRNA translation on the molecular level, they typically take five to ten minutes to finish in the cell of zebrafish, and thus simulation has to rely on DDE to mimic the time delay introduced by such processes in a biological system. Other packages, though never previously applied in this particular field, offer potentially desirable runtime for large-scale simulations through GPU acceleration but those packages are often presented as libraries and thus require a decent understand of GPU programming for researchers to enjoy such benefits. Under such circumstances, many researchers are forced to spend prolonged periods of time creating systems to simulate specific models. However, such systems generally lack flexibility or configurability and, without GPU acceleration, tissue level simulations running on CPU are far from being efficient and extremely time-consuming even on computer clusters.

Here, my study builds a system that (includes DDE’s, adapts to various biological models, supports efficient large-scale simulations through GPU acceleration) is compatible with DDE’s, flexible, easily configurable and, last but not least, (is) accessible for researchers without substantial programming skills. This new system will provide researchers with considerable advantages in three aspects. First, the cost of creating a running system for a comprehensive large-scale model will be significantly lower. When researchers are able to easily configure and simulate their own model, more time can now be devoted to other parts of the research instead of coding. Second, runtime of the new system is five to ten folds faster than CPU only systems. This is important to researchers since parameter estimation usually takes days to complete on computer clusters and much longer on usual workstations. Last of all, requirements on hardware for efficient usage of the systems will be greatly reduced. While previous systems usually require clusters for efficient simulations, the new developed system will run on most workstations with a powerful GPU card. Compared to a cluster of powerful cores that typically cost more than one hundred thousand dollars, a GPU card comes at a lower economic cost (typically under five thousand dollars), and requires less maintenance and is much more accessible to the researchers.

**Related work:**

Aiming to aid researchers in this field, developers have created various packages to provide simulation of such systems yet the current packages fail to fully satisfy the demands of researchers.

COPASI, for example, is one of the most widely used CPU based software for analyzing biological regulatory networks. It includes user–friendly interfaces for model input to cater to the researchers without experience in programming. Simulation and analysis sections of COPASI perform efficiently on small-scale models, namely one-cell models. However, the features included strictly limit a pre-compiled program like COPASI and thus it is impossible for researchers to extend its usability to multi-cell simulation of biological networks. For the same reason, researchers are not able to analyze complex system involving DDEs as COPASI only supports simulations with ordinary differential equations.

On the other hand, simulation software packages that utilize GPU acceleration have shown considerable speedups. For example, cupSODA achieved “a 86× speedup on GPUs with respect to equivalent executions of LSODA on the CPU” and Murray’s package achieved “speedups of up to 115-fold over comparable serial CPU implementations, and 15-fold over multithreaded CPU code” (1,2,3). However, very few of the existing packages support partial differential equations and none of them support DDE. Lack of functionality in this category of systems also excludes a large portion of researchers from using this category of packages. In addition, most of the existing packages are presented as library and are not nearly as user-friendly as COPASI; often written in languages like C/C++, those packages require even higher levels of programming skill, which is not commonly possessed by researchers in this field. Until now, there is virtually no usage of such packages in this field.

Because of the aforementioned limitations of such packages, researchers some are interested in large and complex models are forced to create a specialized simulation system on themselves. Professor Ay and I were part of this group. The system built gradually in the past four years supports DDE and multi-cell simulations particularly for the segmentation clock project. Yet, there are also some obvious limitations in this system. Written for a CPU only environment, the system runs slowly and parameter estimation takes days to execute on Colgate’s computer cluster. Furthermore, all data structures and functions were created based on model information of a particular regulatory network and thus updating the system corresponding to a model update was extremely inconvenient and highly time-consuming, let alone reusing the system for other models.

After observing similar examples and experiencing such circumstances, I decided to construct a new piece of software that could address the limitations of currently available alternatives. The new system will:

1. Accept delay differential equation models as inputs,
2. Includes user-friendly interface for entering model information,
3. Automatically generate necessary data structures and corresponding functions,
4. Simulate using desired numerical solvers, including both deterministic and stochastic methods,
5. Highly modularized and can be extended or updated easily in the future,
6. Support GPU for better performance

The entire software will be developed over the course of the next few years and for the purpose of this high honor thesis, I will mainly focus on design and development of simulation part of the new system. Simulation is the most complicated and time-consuming section of the entire system because it mimics the entire biological process for each species each cell and each time step. In addition, it provides foundation for feature extraction and testing as well as connecting parameter estimation to the rest of the system builds upon completion of simulation. Therefore, simulation section is the essential part to initiate the construction of the entire software and its completion can provide insights for building other parts of the system.

**Preliminary construction of GPU acceleration:**

As mentioned above, the existing system simulates and analyzes a tissue-level model of segmentation clock network. In this stage, I will focus on improving runtime of the existing CPU only system by introducing GPU acceleration to the system.

***Methodology:***

In this study, I chose CUDA by NVIDIA as the platform for GPU computing. As a minimal extension of C and C++, CUDA works well with the existing system and it has been applied to create scalable parallel programs in some similar disciplines such as computational chemistry. There are three key abstractions in CUDA platform: a hierarchy of thread groups, shared memories, and barrier synchronization. Together, those three abstractions create a parallel structure to C/C++ code. This structure allows coarse-grained parallelism on the high level and more fine-grained parallelism on lower levels. It is a perfect candidate for executing simulations in a biological regulatory network since the three levels of parallelism within CUDA platform correspond well to a large computational model. In particular, CUDA platform is organized as grids of blocks where each block contains a set of parallel threads, the lowest level of parallelism in the system. The set of threads in a block cooperates well with barrier synchronization and shared access to a memory space private to the block, and thus each tread is analogous to a cell inside one single simulation since cells are simulated in parallel to each other but the set of all cells in a simulation are connected and need to share common memory. On a higher level, parameter estimation contains a set of simulations that are not related to each other and can execute in parallel, exactly corresponding to a gird of independent block that can be executed independently.

At first, I started by modifying the code to simulate all six mutants for one parameter set in parallel. Under this scheme, the code is broken into three major parts. All inputting parameters, creating new data structures and copying original code to new copies are executed in the original order for each of the mutants, and a for loop is added to iterate and setup for all six mutants before the system moves to the next phase. Next, the system transfers all the data necessary for simulation (particularly data needed by protein synthesis, dimer synthesis and mRNA synthesis) to GPU at once and start one block on GPU for simulating each of the six mutants. During simulation of a single time step, the system transfers only relatively small amount of data between CPU and GPU for purposes such as splitting and updating rates and copying results from baby\_cl to cl. This part of the program is executed in parallel on GPU. After all simulations of this time step have completed on GPU, the system then copy all results back to CPU. The same process repeats for all time steps and, once simulation of all time steps is complete, the system starts to perform feature extraction and testing for all the mutants. Similar to the first phase, all processes in the last phase are also executed in the same order as they were originally except there is a new for loop over six mutants.

The major difficulty in this part of the project is minimizing data transfer between CPU and GPU. Since rates of reaction, concentration level of genes, mRNA and protein all need to be updated and stored for every time step and there are one hundred time steps in one second. Large amount of data transfer will take so much time that may dismiss the benefit of incorporating GPU computing into the system. Therefore, I applied several methods to prevent this to be the bottleneck of the heterogenous computing system.

First, I integrated memory transfer of all six mutants together because essentially transferring all data for six mutants at once is faster than transferring the same amount of data six separate times. Also, I moved data transfer outside of the for-loop over all time steps. Now, instead of transferring all data between two computing environments for each of the time steps, the system transfer all data onto GPU in the beginning of the large for loop over all time steps and transfer only necessary data inside the for loop. Next, I divided the transfer functions into two separate d functions. One will allocate a CPU array on the GPU and the other will transfer the data and swap pointers to the array on GPU and CPU. I then added another function that will only swap pointers to the array. Though those separated functions, the system only needs to allocate the array once and can access the array repeatedly later using the pointer and swapping functions. Now that allocation process is now omitted during each data transfer and the system only needs to overwrite the data at the given location, each data transfer can be accelerated.

Less communication between GPU and CPU will obviously reduce unnecessary overheads and improve runtime of each individual simulation. It can be foreseen that less data transfer between CPU and GPU will require more data to stay on GPU throughout each simulation and will directly affect memory requirement of each process. When I was trying to simulate two or three parameter sets at the same time, I found out that simulation of multiple parameter sets lead to another challenge of the system in this stage, which is to reduce memory requirement of the simulation on GPU. Limited by the structure of the current system, the system keeps a copy on CPU for every copy of data structure existing on GPU. With smaller requirement on memory for each simulation, a larger number of simulations can fit on the GPU card simultaneously and thus utilization of GPU card will approach its fully capacity.

There are two viable ways to reduce the memory requirement of the system on GPU. One involves directly reducing the size of baby\_cl structure that will be copied to GPU. Currently, the number of time steps that every baby\_cl keeps is the same and equals to the maximum delay. However, most of the reactions require data from much fewer time steps. For example, delay in her1 DNA translation ranges from 150 to 590, which is far less than 5500, the maximum delay. It is clear that a data structure that keeps only the necessary number of time steps in baby\_cl can help reduce memory requirement of the program on GPU. If baby\_cl for each of the reaction can be tailored according to the delay size of that reacation, memory requirement on GPU can reduce by a factor of six in the worst case and by a factor of ten in the best scenario. The other way to reduce memory requirement is to increment time step and thus indirectly decrease the size of baby\_cl. If the time step is increased to 0.02 then size of baby\_cl can reduce by half and similarly, if time step if four time larger, then size of baby\_cl will be only a quarter of the original size. However, redesigning the data structure and changing time step sizes for differential equations require various changes over the entire program and thus they were not implemented in this first stage. They will be addressed and incorporated during the second stage of the project.

***Results:***

At the end of stage 1, I obtained the following results: runtime of the system simulating six mutants sequentially are in parallel are 6:15 and 4:18 respectively. Runtime for simulating twelve mutants in parallel is 7:26 (Figure 1). Instead of a linear increase in simulation time, GPU accelerated system has much smaller increase due to initialization and analysis. New simulation, the most time consuming part, is now executed in parallel with other sets and thus does not incur additional runtime increase.

This is a successful experiment and reveals several factors in the current system that may affect the runtime. First, as mentioned above, data structures are larger than they need to be and thus inefficient. Therefore, number of parameter sets we can simulate at the same time is limited. Second, there are two copies of some data structures in the system. Those data structures exist on both CPU and GPU. For example, copies of baby\_cl or active\_rates are originally created on CPU but they are later copied to GPU for simulation. For now, we keep both copies in the system. Those problems provide insights about limiting factors of current system and will be addressed during development of the new system. Next, the goal is to construct a new system that will address four major issues in the previous system: difficulty in model switch, difficulty in system update, inefficiency in memory usage and runtime.

**Model separation:**

At the time of development, the original system was designed to simulate a specific biological model and every aspect, from data structures to numerical solvers, was written with single purpose of fitting that model. Now this characteristic has become one major disadvantage of the original system, if not the biggest. Model information and simulation functions deeply entangle with each other and model updates require changes to be made throughout the entire system. In this section, I aim to separate all model information from simulation methods and data structures, since the ultimate goal is to adapt this prototype system to accept mathematical models as input, dynamically create corresponding data structures and function calls that can be used directly during simulation. Through model separation, user of the system will be able to easily update or switch biological models in the future while keeping the rest of the system unchanged.

***Methodology:***

A new form of model representation is designed for model information in the new system. In this representation, the basis of the model consists of a list of species and list of reactions related to those species, and there are two more structures building on top of the basis: reactions.cpp and model\_impl.hpp. The former one contains data structure describing the relations between species and reactions, which involve input, output and dependent specie(s) of each reaction and their effects, while as the later file describes mechanisms of each reaction, what are the related active rates and where do they stay in the system.

Recall that we are trying to dynamically create related data structures and functions though this representation of model information. A common way to construct such a class is to use virtual functions and then different kinds of reactions will specify different functions in order to calculate reaction rate. Those functions will overwrite the virtual functions. However, this is undesirable to our system because all virtual functions provide only run-time polymorphism. In other words, the system will be much less efficient since those functions cannot be optimized during compile time. To overcome the loss of complier optimization, I introduced x-macros into the system and define reaction list and specie list as an enumerated list. Through x-macros, resulting data structures and functions calls can be generated during pre-processing thus granting polymorphism while enjoying compiler optimization. Details of x-macros usage will be explicated in the next section due to its close interaction with simulation.

***Results:***

In this section, we choose a much higher level of generalization of model information at cost of not continuing to use design and manual optimization of the previous system. But usage of x-macros provides enough compensation to the system to make generalization viable. It is important to note here that manual optimization of this particular section of the system is impractical and reasoning follows below. Manual optimization requires all data structures and functions to be static and written in the system, yet the goal of the system is to first, relieve future users from manually create them and secondly, keep simulation section independent of model input for convenient model switch.

**Simulation structure change:**

The work in this stage involves both re-writing existing functions for more general usage and redesigning the structure the system to further support improvements made on the model side. As illustrated earlier, model part of the system is now declared through x-macros and thus it was necessary to redesign simulation section to connect and well support those changes. In addition, to support a special biological mode, the old simulation section as well as numerical methods has little, if any, flexibility in reusability. To address this problem, I intend to design new classes and functions to generalize existing functions and numerical methods so that the new system is no longer model specific but instead can be re-applied to or easily updated based on other biological models.

***Methodology:***

For further isolation of reactions and other simulation process, a new context class is created. In a biological system, it provides a scope for various simulations and, in this biological system, each instance of context class is equivalent to a cell. It is essentially a wrapper class inside simulation that interacts with reactions and is responsible for two important tasks, calculating rate change and updating concentration levels for every species present.

Context class also serves as a bridge for interactions between model information and simulation, as it encapsulates not only the simulation mechanism but also provides access functions for simulation information, including step size and time step, necessary for calculation and updating concentration levels. To do so, I added unique identifier inside context class and used context as the second template parameter for functions in model\_impl.hpp shown earlier. Now the system easily iterate through all context instances and perform designated tasks with enough information.

First part is to update active rates of each reaction, which describe the rates of change at one particular time step for all reactions and this notion is the same as the idea of derivative in a differential equation. To do so, functions are designed to cooperate well with x-macros used during model declaration; particularly, function is placed inside a for loop over all context instances and is able to automatically generate lines of code to invoke functions for calculating rates for each of the reactions. An example of how x-macros works in the system in shown in \_\_\_. X-macros provides foundation for the reusability of the simulation section as it ensures that the template function is configured to run with any biological models as long as model information is input under our restrictions.

Second part of simulation is to apply numerical methods to predict concentration levels of all species at the next time step and this is where we need active rates at current time step as well as concentration level of each species. Here we continue to implement a simple Euler’s method as the original system but it is important to notice here that two tasks were integrated in form of differential equations in the original system and each reaction may be calculated multiple times since it may contribute to concentration level changes of various species. Thus, to change the numerical solver, or the methodology applied to update concentration levels was extremely complicated. In the new system, I first aggregate all reactions as well as their influences on each of the species and then apply them collectively along with concentration level of current step to estimate the concentration level of next time step.

***Results:***

Through those methods applied, two subsections of updating concentration level is separated from each other, which will then render much high level of freedom in either part. In the new system, implementation of a new numerical solver, such as Runge-Kutta method, will take place inside just one function and can be applied to all reactions with no further adjustments except storing more past concentration levels and active rates (for example Runge-Kutta method requires concentration levels from multiple time steps).

**Memory usage:**

In section, I will discuss the design pattern used in the new system to cover the flaws in memory usage in the original model. The new design aims to further reduce the unnecessary memory requirement for better space efficiency. While space efficiency is an important factor in system design in general, it is particularly important in our system because less memory requirement for each individual simulation will allow us to run a larger number of simulations in parallel later on GPU (or only CPU). Smaller memory requirement for each simulation also means less data transfer between GPU and CPU, another bottleneck identified during stage one of the project. Given those two factors, it is easy to see that overall time efficiency is directly connected to space efficiency.

***Methodology:***

One of the main data structure in the system is a large three-dimensional array named baby\_cl, which holds concentration levels over a number of delayed time steps for all species and all cells. Length of delay is associated with each individual reaction but the concentration level is stored in the system according to each species, which means there is direct attribute that will determine the delay size in baby\_cl. To handle this problem, original systems decides the number of time steps to keep based on the maximum delay size across all reactions to ensure that each species is kept in the system long enough for possible reactions. However, not all species are related in a reaction with the maximum delay. The maximum delay can be as long as 1,200 time steps for some reactions such as gene transcription and much shorter for reactions such as mRNA translation; some reactions in the system may not even have a delay. It is clear to see that certainly not all species need to be stored for maximum delay to provide history data for future simulations.

To decrease the size of baby\_cl, I will redesign data structures so that the system can find related reactions for one species and find the maximum delay size within this subset. Then the system will create a large one-dimensional array to hold all concentration levels and place individual wrappers for each species according to its delay size. For the system to access concentration levels of each species, I created another short one-dimensional array to hold the pointers to the beginning point of each species in the larger array.

***Results:***

This design will reduce size of baby\_cl on running environment greatly and thus allow simulation of more parameter sets at the same time. Using the segmentation clock project for example, the new baby\_cl data structure uses 70% less memory than the original one. Through this improvement, the number of simulations can stay on the GPU the same time increases more than two folds and consequentially, the average runtime for each of the parameter sets will be reduced to one third of its original runtime.

**Time efficiency and final integration of GPU acceleration**

During the preliminary phase of GPU acceleration, several factors prevented us from an efficient GPU implementation. Thus, in the new system, much effort was spent to address those problems, and now with simulation improvements and space efficiency, GPU acceleration is again implemented to the system, free of previous limitations.

***Methodology:***

Methodology in converting CPU code into CPU-GPU code follows a similar pattern as demonstrated in the preliminary phase and thus will not be re-introduced here.

On top of the changes mentioned earlier, another attempt to minimize data transfer between CPU and GPU is made through initiating some data structures on GPU to reduce both memory requirement on CPU and data transfer required during simulation. For example, copies of baby\_cl or active\_rates may be created on GPU and no initial transfer of those structures from CPU to GPU will be needed during simulation process. Other necessary data transfer during simulation is now performed by CUDA managed memory. It allows data structure to be accessed on CPU when required.

***Results:***

All aspects of design have been successfully implemented in the new system. Simulation of CPU only code runs 1.6 to 1.9 times longer than the original system. Simulation of CPU-GPU code runs \_\_ times longer than the original code. However, parallel simulation on GPU environment shows significant improvement in the new system. As a rough test case, it shows that when the simulating twenty parameter sets simultaneously, the original code takes twenty times longer than single simulation, while the new system takes less than double of the time as single simulation. Overall, simulation runtime of twenty parameter sets is five times faster than the same simulation through the original system and, with multiple GPU cards connected to the same work station, the new code can run efficiently on the station instead of relying on cluster.

The slight increase in runtime for CPU only code of the new system was a big concern for us and, after profiling, I found that the extra layer of indirection added to save memory requirement was a major source of time spent. Note that concentration levels are constantly accessed and updated in each time step for all species and all cells and thus number of memory access is very large. This will be a price we need to pay for much smaller memory requirement on CPU or GPU and as discussed above, it is will be compensated through larger numbers of simulations running in parallel on the GPU.

**Additional challenges:**

In order to make the system general enough to benefit as many researchers as possible, I also included a CPU only version of the system. This imposes another low-level challenge to the system, which is to compile the same functions in a different way with different compilers (g++ for CPU and nvcc for GPU).

There are two main parts of my solution. On the simulation side, for functions to be run on both CPU and GPU, I defined a new macro called CPUGPU\_FUNC to declarations of all functions. If the system will run on GPU, it will be compiled with nvcc and when this is the case, I define CPUGPU\_FUNC as \_\_host\_\_ \_\_device\_\_, which is the function declaration necessary for a function to run both the host (CPU) and the device (GPU). When the user wish to compile the code only on CPU, CPUGPU\_FUNC will be defined as blank, essentially not changing anything to function declaration. Another changed was with regard to usage of CUDA managed memory, which is used in the new system to transfer data between CPU and GPU. Following a similar design patter, I added another macro called STATIC\_VAR and declare it as \_\_managed\_\_ on GPU and nothing for CPU only simulations. Through these two strategies, functions are declared based on the platform on which the simulation will run.

Now onto the files, I am forced to create different files for different simulation environments since when a static variable is passed to the compiler, g++ will make assumptions such as the location of the variable, such assumptions are not valid when nvcc is used and thus files have to created in versions, cpp files for g++ and .cu files for nvcc. To work with two versions of the .cpp files, I added macro declarations inside the header file so when the header file is linked to the implementation file, it will be compatible to either of them. For a similar reason, two versions of the test files are created for CPU only and CPU-GPU simulations.

At last, files with the two kinds of macros mentioned above can only be included once among all the object files since multiple inclusions will result multiple declaration of the macros. The location of inclusion is thus an important choice. For the CPU-GPU simulation, we cannot place it in the simulation implementation file since it also serves as the super class for simulation\_cuda implementation file and is thus included in GPU accelerations as well. Therefore, the only place that will determine if a simulation is CPU only or CPU-GPU simulation is the actual test file. One of the header files also includes implementation details and for that file in particular, it was placed inside simulation\_cuda implementation file since that will never be included in a CPU only simulation.

**Reliability:**

One aspect of the new system important to biological users is the reliability of the system. Is the new system conducting all parts of simulation correctly? One way to test this is to see if the new system is able to replicate the simulation results of the original system. Controlled tests are conducted to test the simulation accuracy of this system. I passed the same set of simulation results into both the original system and the new system for simulation. After the concentration level of each species and each cell are recorded along the entire simulation, we compare the concentration levels from two systems to see if they are identical to each other. This controlled tests show that there is a 0.3% discrepancy in final simulation results after 60,000 time steps (equivalent to 600 minutes, a common length for the segmentation clock project).

This slight discrepancy in simulation results may stem from the differences in implementation of the system. Recall from the simulation section that, the original system updates concentration level of a species based on the corresponding differential equation, which only counts the influences of a reaction on that particular species at a time. This process will be repeated for each of the species and if a reaction is related to multiple species, it will be included in multiple differential equations. On the other side, the new system gathers all active rate changes one reactions may impose on various species and then update the concentration level for each of the species collectively. Potential rounding off error may rise since the order of two sub processes is switched. But overall, this is a relatively small error especially after 60,000 time steps and will not affect any system level characteristics of the biological network.

**Possible future work:**

In this new system for simulating biological regulatory networks, I have utilized model separation, simulation mechanism reconstruction, compressed sparse row and GPU acceleration for easier model switch, independent simulation update, reduced memory requirement on CPU and GPU, and improved runtime overall.At this point, the majority of the simulation process is completed and desirable results are attained in various aspects. However, there are potential improvements to the system as a whole and to be completed in future exploration of this system.

One possible future improvement to the system is to add user-friendly interface for model input. Current model is separated from all other parts of the system and organized in a systematic way to represent each piece of information inside the mathematical model. Those four files have rather lower-level representation at this point, and researchers are still responsible for enter model implementation in all files before they can be used in other parts of simulation. A user-friendly interface can further levitate this task from researchers as that will require less programming experience and is much less error-prone.

The second possible improvement can happen inside simulation section. Currently, a deterministic simulation is used mimic the biological process and solved by Euler’s method and updates mRNA and protein levels at each iteration using the rate of changes provided by the model. Another type of simulations, stochastic simulation, can help researchers to build an even more realistic and comprehensive biological model. Probabilistically determined propensities and reaction times are used to decide which reactions fire at each iteration. Reactions with higher propensities are more likely to fire. Since stochastic simulation typically requires even more resources, both memory and computation power, than DDE’s, it was omitted in the original system. With major improvements in space efficiency and time efficiency, stochastic simulations, such as next reaction method, which discretely computes concentration levels based on probabilistic calculations, may now by feasible in the new system.

The last possible improvement involves other sections of the system, namely feature extraction. Profiling of the original model shows that feature extraction from results of simulation was taking up to thirty percent of the whole simulation time (nearly half of the simulation process runtime). Specifically, the original system would first run all simulations and save all concentration levels in memory and then later access and utilize such information for calculating features. Therefore, a new deign of converting feature extraction onto GPU might further improve system runtime. Because feature extraction is necessary for all parameter sets and mutants, every determined block on GPU will have relatively the same amount of work to complete and thus most of the blocks can run in parallel and resource waste will be kept as minimum.

Studies in regulatory biological networks often rely on mathematical modeling and computer simulations on various scales and levels of complexity.