Burgas State University "Prof. Dr. Assen Zlatarov"
Faculty of technical Sciences
Department of Computer Systems and Technologies

Extended Abstract of a PhD Dissertation

ON THE DISSERTATION FOR AWARDING THE EDUCATIONAL AND SCIENTIFIC DEGREE "DOCTOR"

by Militsa Toshkova Kostadinova- Gocheva, MEng

Doctoral program: "Computer Systems and Technologies"

Professional field:

5.3 Communication and computer technology

Scientific field: 5 Engineering sciences

TOPIC:

Computational methods for impulse research of genetic neural networks

Scientific advitors:
Prof. Gani Trendafilov Stamov, DSc
Prof. Stanislav Denchev Simeonov, PhD

Burgas, 2026

The dissertation was discussed and approved for defense at an extended meeting of the Department of Computer Systems and Technologies held on May 7, 2026, at Burgas State University "Prof. Dr. Assen Zlatarov" - Burgas.

The dissertation consists of 149 pages, including 36 figures and 8 tables. A total of 128 references are cited. The results have been published in 6 articles.

The defense of the dissertation will take place on ..... 2026 at ..... in room ..... at Burgas State University "Prof. Dr. Asen Zlatarov" - Burgas before a scientific jury composed of:

The defense materials have been made available to interested parties in office ..... at Burgas State University "Prof. Dr. Assen Zlatarov" - Burgas.

Author: Militsa Toshkova Kostadinova- Gocheva, MEng

Topic: Computational methods for impulse research of genetic neural networks

Abbreviations used

ANNs - Artificial neural networks

AMD - Advanced Micro Devices

API - Application Programming Interface

CUDA - Compute Unified Device Architecture

CNN - Convolutional Neural Networks

GPU - Graphics Processing Unit

GRU - Gated Recurrent Unit

GRN - Gene Regulatory Networks

IBM - International Business Machines Corporation

IGRNs - Impulsive Gene Regulatory Networks

MLP - Multilayer Perceptron

MPI - Message Passing Interface

mRNA - messenger RNA

OpenCL - Open Computing Language

RNN - Recurrent Neural Networks

DNN - Deep Neural Network

SNNs - Spiking Neural Networks

ROCm - Radeon Open Compute

HIP - Heterogeneous-computing Interface for Portability

HPC - High-Performance Computing

Chapter 1

Introduction. Aim and tasks.

The field of intelligent systems has recently demonstrated significant breakthroughs with several advancements based on biological neural networks, which provide natural models for developing new systems. Artificial neural networks (ANNs) were created thanks to the collaborative efforts of researchers from various scientific fields to perform various tasks, such as pattern recognition and prediction, optimization, associative memory, and process control.

The aim of this dissertation is to model stable processes in neural networks of the impulsive genetic regulatory neural network type. A software application is to be developed that facilitates the input of parameters, extraction of result data, generation of graphs for the obtained results, and the preparation of reports according to the examined necessary dependencies for model stability.

Implementation of the following tasks:

- 1. To research existing software solutions for Impulsive Genetic Regulatory Neural Networks or a similar model.
- Task 2. To create a mathematical model and investigate the stability conditions for a network of three neurons.
- Task 3. To model the stability conditions of the neural network

model under consideration.

- Task 4. To create and implement a classical algorithm for neural networks of the model under consideration.
- Task 5. To implement and investigate parallel algorithms for stability conditions of an existing configuration.
- Task 6. To implement a parallel algorithm based on OpenMPI.
- Task 7. To implement a parallel algorithm based on ROCm.
- Task 8. To implement a parallel algorithm based on a hybrid OpenMPI and ROCm model.
- Task 9. To perform a comparison between the three technologies.
- Task 10. To investigate the results and their presentation.

The subject of this dissertation is the acquisition, investigation, and analysis of stable processes in neural networks of the impulsive genetic regulatory neural network type with variable impulsive disturbances through software models and libraries. The use of parallel techniques provides a scalability advantage for obtaining results, which leads to the acceleration of processes.

Chapter 2

Neuron and Neural Networks.

This chapter analyzes the evolutionary development of neural networks over the past decades.

2.1 Biological neural networks

The human brain functions as a massively parallel system composed of approximately 100 billion neurons, supported by numerous neuroglial cells. The high computational power of the biological system is due to the vast network of connections—each neuron can interact with up to 10,000 other cells. The total number of synaptic connections in the brain reaches 100 trillion, allowing for information processing at an intensity estimated at approximately 1 trillion bits per second.

2.2 Artificial neural networks

This section discusses the theoretical foundations and architectural principles of artificial neural networks (ANNs). They are presented as mathematical models inspired by biological neural structures, designed to solve complex tasks through approximation, classification, and pattern recognition.

2.2.1 History and development of neural networks

A retrospective analysis of the evolution of ANNs is made - from conceptual ideas in the 1940s to the modern era of deep learning.

2.2.2 The first formal model - the McCulloch and Peets neuron

The 1943 model of Warren McCulloch and Walter Pitts, which laid the foundations of computational neuroscience, is described in detail. Its principle of operation as a logical threshold element is analyzed, demonstrating that a network of such elements can perform arbitrary logical functions, laying the foundation for the binary representation of neural activity.

2.2.3 Network architectures types of artificial neural networks

A classification of the main types of network topologies is presented.

2.2.4 Components of an artificial neural network

The fundamental building blocks of ANNs are described:

- Inputs and weights: representing the strength of the connections between elements.
- Summation function: calculating the weighted sum of the input signals.
- Activation function: determining the output state of the neuron (Sigmoid, ReLU, Tanh, etc.).
- Layers: input, hidden, and output layers, defining the depth of the model.

2.2.5 Training

In this section, neural networks are classified into two main categories based on their structure and how they process information:

- **Static networks:** They are characterized by direct calculation of the output from the input through feedforward connections. A typical example is the multilayer perceptron.
- **Dynamic networks:** They contain delays and recurrent (feedback) connections. Unlike static networks, the output here depends not only on the current input, but also on the previous states or inputs of the network (e.g. Hopfield network).

2.2.6 Stability

Recurrent neural networks (RNNs) are distinguished from feedforward networks by their ability to process temporal and spatial dependencies. While in standard networks the output depends only on the current input vector, in recurrent models it is a function of time. This leads to more complex behavior: for a fixed input the output can stabilize, but it can also oscillate, grow indefinitely, or exhibit chaotic dynamics.

Chapter 3

Mathematical model of impulse gene regulatory neural networks

3.1 Impulse gene regulatory neural networks

This chapter presents the development of impulse gene regulatory neural networks as an interdisciplinary scientific field. A retrospective review of the key stages of their emergence is made and significant scientific contributions in the literature are systematized. Based on the review, insufficiently researched aspects and unsolved problems are identified, which determine the directions of the present work.

3.2 Theoretical model

A theorem and proof part are defined for the equilibrium of the impulse genetic neural network system according to the specified model (3.3) for global exponential stability with respect to the function h.

The following (IGRNs) model is presented, specified by:

$$\begin{cases} \dot{m}_i(t) = -a_i m_i(t) + \sum_{j=1}^{n} w_{ij}(t) f_j(p_j(t)) + B_i(t), & t \neq t_k, \\ \dot{p}_i(t) = -c_i p_i(t) + d_i(t) m_i(t), & t \neq t_k, \\ \Delta m_i(t_k) = \alpha_{ik} m_i(t_k) + \nu_{ik}, \\ \Delta p_i(t_k) = \gamma_{ik} p_i(t_k) + \chi_{ik}, \end{cases} \tag{3.3}$$

that regulates the concentrations of mRNA $m_i(t)$ and protein $p_i(t)$ at time t, where:

i/ i, $i = 1, 2, \ldots, n$ is the number of nodes, a_i, c_i represent the dilution rates, using the dimensionless transcriptional bounded rate $q_{ij}(t)$ at time t of transcription factor j to i, $w_{ij}(t)$ are defined as :

$$w_{ij}(t) = \begin{cases} q_{ij}(t), & \text{when } j \text{ is an activator of gene } i, \\ -q_{ij}(t), & \text{when } j \text{ is a repressor of gene } j, \\ 0, & \text{when there is no link from the node } j \text{ to gene } i, \end{cases}$$

$d_i(t) \in \mathbb{R}$ denotes the translation rate, f_j represents the regulatory (activation) of the protein function and is in the form

$$f_j(p_j) = \frac{(p_j/\beta_j)^{H_j}}{1 + (p_j/\beta_j)^{H_j}},$$

where H_j denote the Hill coefficients and β_j are positive scalar quantities, $q_i(t)$ is defined as $q_i(t) = \sum_{\mu \in I_i} q_{ij}(t)$, where I_i is the set containing all repressors of the gene i;

The validity of the results obtained in Theorem 3.3 - criteria for global exponential stability with respect to sets is demonstrated.

Chapter 4

Parallel algorithms for studying impulse gene neural networks

The human brain functions through complex networks of neurons and synapses that enable parallel information processing and self-learning. Artificial neural networks seek to emulate this biological model through an architecture of interconnected nodes for distributed computing. Although modern computers offer enormous computational speed, they still cannot fully replicate the scale and functional complexity of the human brain [114].

4.1 Basic concepts

4.1.1 Sequential execution time

The sequential execution time T_1 is the time required to execute a given program on a single processor, without parallelization.

4.1.2 Parallel execution time

The parallel execution time T_p is the time required to execute the same program on p processors (cores) using parallel algorithms.

4.1.3 Laws of Scalability

Amdahl's formula for overall speedup in parallelization:

$$S = \frac{1}{(1-p) + \frac{p}{s}} \tag{4.1}$$

където:
p – number of processors,
s – is the speedup of the parallelizable part (often denoted as the number of processors or speedup factor),
$(1-p)$– the inevitable sequel [115].

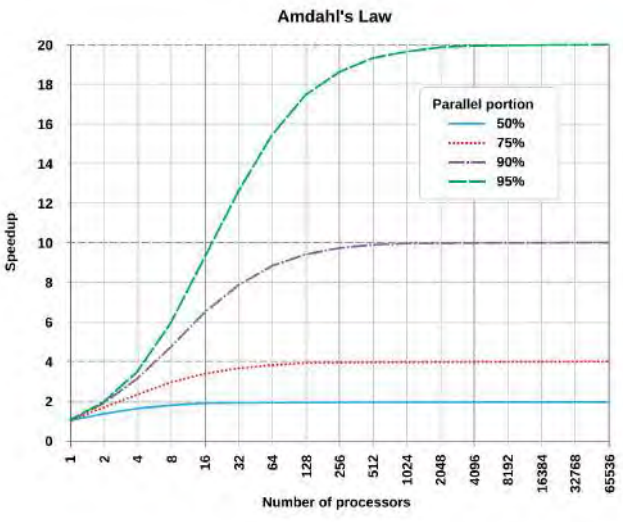

Figure 4.1: Parametric influence of scalability according to Amdahl's law

4.1.4 Laws of parallel efficiency

Parallel efficiency E_n measures how much of the time computing resources are used usefully. It is defined as the ratio of the speedup to the number of processes used [116]:

$$E_n = \frac{S_n}{n} \times 100 \tag{4.2}$$

където:

- E_n is the parallel efficiency of the system;
- S_n is the speedup when using n processors;
- n is the number of processors used;

Ideally, $E\_ n = 100\%$, but in real conditions it decreases due to the so-called Parallel Overhead, which includes: PCIe bus latency during data transfer between Host and Device. Synchronization delays in the MPI_ Allgather operation. Conflicts when accessing the shared system memory (Memory Socket contention)[116].

4.2 Hardware implementation of high-performance and parallel systems

4.2.1 Classification

High Performance Computing (HPC)

This section examines the HIP (Heterogeneous-compute Interface for Portability) programming model as a tool for implementing parallel C/C++ algorithms on massively parallel SIMD architectures. The fundamental difference between central and graphics processors is analyzed, emphasizing the need for a specific approach to programming to achieve optimal performance. The evolution of AMD's software stack is historically traced - from the Boltzmann initiative to the establishment of the open ROCm platform. Special attention is paid to the role of the HIP language as a means of achieving portability, allowing compilation of program code for both AMD and NVIDIA hardware using a syntax identical to that of CUDA. In this way, HIP is established as a key element in modern heterogeneous computing systems. **Hardware Differences: CPU vs. GPU**

Central processing units (CPUs) and graphics processing units (GPUs) are designed for different purposes. Central processing units

(CPUs) execute a single thread quickly, reducing the time it takes to complete a single operation while increasing the number of consecutive instructions that can be executed. This includes fetching data and reducing pipeline bottlenecks where the ALU (assistant logic unit) must wait for previous instructions to complete.

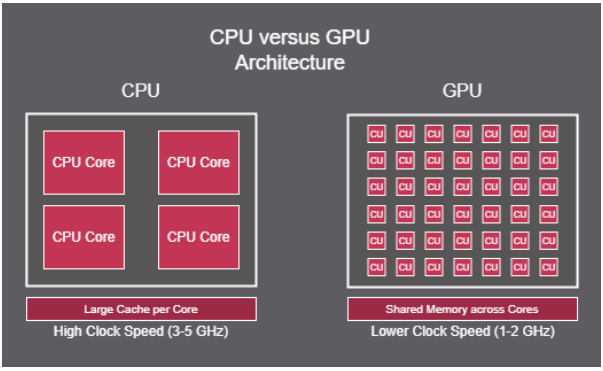

Figure 4.2: Differences in CPUs and GPUs [117].

While central processing units (CPUs) are optimized for low-latency serial processing (fast execution of a single task), graphics processing units (GPUs) focus on high performance by executing multiple threads in parallel. Latency measures the time it takes to complete a single operation, while throughput measures the total number of operations per unit of time.

The GPU architecture is based on compute units (CUs), which function as the basic building blocks for processing program cores. Each compute unit has its own resources, including arithmetic logic units (ALUs), registers, and cache memory, enabling efficient parallel communication [117].

Programming models and software

A key element in HPC is the software stack, which includes:

- MPI (Message Passing Interface) – standard for communication between processes in distributed memory
- OpenMP – shared memory parallelism model

- OpenCL - an open, standardized programming model for parallel computing on heterogeneous systems
- CUDA / OpenACC – GPU acceleration programming models
- Digital libraries – BLAS, LAPACK, ScaLAPACK

Effective use of HPC requires optimization of algorithms with regard to memory, communication costs, and scalability.

4.2.2 High-performance computing system based on AMD GPU

As a primarily open source ecosystem, ROCm™ provides the ability to test, customize, and adapt the software stack to specific requirements, supported by a collaborative developer community. ROCm is a software stack composed primarily of open source software that provides the tools for programming AMD graphics processing units (GPUs), from low-level cores to high-level end-user applications. ROCm provides the tools for Heterogeneous-computing Interface for Portability (HIP), OpenCL, and OpenMP. These include compilers, high-level function libraries, debuggers, profilers, and runtimes [117].

Figure 4.3: Architecture of ROCm [117].

An AMD-based system can be transformed into a high-performance and customizable machine learning workstation. ROCm™ 7.1.1 supports the latest Radeon™ 9000 Series GPUs (RDNA 4) and select 7000 Series GPUs (RDNA 3), and introduces initial support for Ryzen™ APUs, enabling cost-effective local development and inference for researchers and engineers using Pytorch.

The AMD RDNA3 processor [118] implements a parallel microarchitecture that provides a platform for computer graphics applications as well as general-purpose parallel data applications.

The figure below shows a block diagram of the AMD RDNA3 Generation series processors:

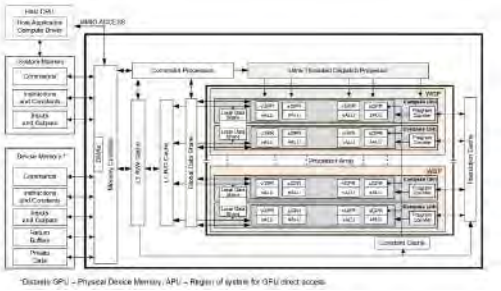

Figure 4.4: AMD RDNA3 Generation Series Block Diagram [118].

The RDNA3 architecture consists of parallel processors, a command processor, and a memory controller. The command processor manages communication with the host via registers and interrupts, while the memory controller provides direct access (DMA) to local and system memory. This structure allows for automated calculation of address offsets and efficient real-time data processing. In the RDNA3 environment, a complete application consists of two parts:

- program running on the host processor
- programs called shader programs or kernels running on the RDNA3 processor.

4.2.3 OpenMPI

MPI is a standard defining a model for message passing in parallel systems, while OpenMPI is its practical software implementation. The relationship between them is like that between an architectural plan and a finished building – MPI defines the abstract rules and interfaces, while OpenMPI provides the libraries and environment for their actual implementation on hardware. In this way, the standard guarantees compatibility, and the concrete implementation allows for the practical development of distributed applications.

MPI standard

This section examines the Message-Passing Interface (MPI) as a fundamental standard for parallel programming in distributed memory environments. MPI is defined as a library specification, not a language, which allows its implementation through various functions and methods in languages such as C, C++, and Fortran. The historical evolution of the standard is traced from the creation of the MPI Forum in 1992 to the modern versions MPI-3 and MPI-4, which are optimized for work with heterogeneous architectures and accelerators (GPU). Special attention is paid to the mechanism of message transmission between the address spaces of individual processes and the role of the operating system. In this regard, the importance of the kernel is defined as a privileged intermediary between the application software and hardware resources, providing the necessary abstraction and control for the execution of parallel processes.

4.2.4 A hybrid parallel architecture with ROCm and OpenMPI

Critical to building exascale supercomputers is the integration of ROCm, the UCX communication framework, and GPU-aware MPI. These technologies enable direct access to accelerator memory via RDMA, which eliminates intermediate data copying and reduces latency. The practical effectiveness of this model has been demonstrated on the Frontier supercomputer, based on AMD hardware. In this architecture, the HIP language is established as a universal solution for supporting a single and portable code base. This ensures high performance on a variety of heterogeneous systems.

One key element is Unified Communication X (UCX). Since

OpenMPI version 5.0.5 and later, including the current ROCm 7.x builds for 2025/2026, UCX is available by default as a transport layer that recognizes ROCm pointers, enabling RDMA (Remote Direct Memory Access) and a GPUDirect-like interface via PeerDirect, which dramatically reduces the latency of communication between nodes [117], [121], [122], [123], [124].

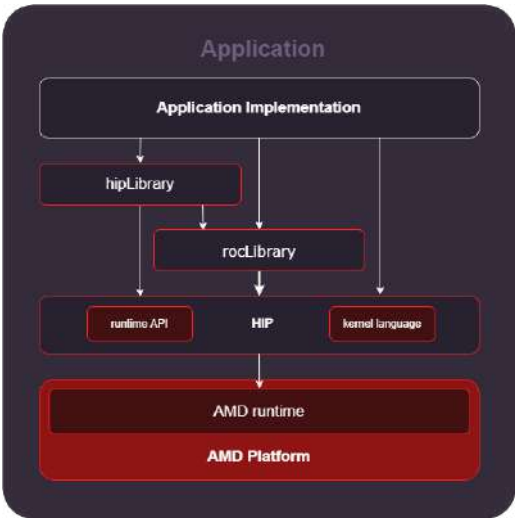

Figure 4.5: The HIP (Heterogeneous-Compute Interface for Portability) architecture in the AMD ecosystem [117].

ROCm is a heterogeneous computing platform that provides tools, libraries, and HIP APIs for CPU and GPU development. Libraries such as hipFFT and hipBLAS provide CUDA compatibility, making it easy to integrate into existing applications. The HIP programming model allows for a single code base across AMD and NVIDIA hardware, and provides automated porting tools (HIPify). The system separates execution into host (CPU) code for data management and kernels for massively parallel processing on the GPU. Despite the high degree of automation, achieving optimal performance often requires additional manual tuning.

Chapter 5

Algorithms and implementation of a robustness model in impulse gene neural networks

The software implementation of the meta-mathematical model was developed in the C++ programming language. The development went through four stages. Initially, the project was implemented without using parallel techniques in order to examine and improve the method by which the calculations are made.
In the second stage of development, the software implementation was achieved using a technology called OpenMPI, on a machine processor with an AMD Ryzen 9 7950X 16-Core Processor (4.50 GHz).
The third step was carried out using ROCm technology on a machine graphics processor with an AMD Radeon RX 7800 XT (2438 MHz). And the fourth step was carried out with the hybrid technology MPI + ROCm.

5.1 Model Summary

To create the various software implementations, a gene neural network model is used, which is based on the generalized Impulse Genetic

Neural Network model having the form of equation (3.3). The model used has the form:

$$\begin{cases} \dot{M}_i(t) = -a_i M_i(t) + \sum_{j=1}^{n} w_{ij}(t) f_j(P_j(t-\sigma_j(t))) + B_i(t), & t \neq t_k, \\ \dot{P}_i(t) = -c_i P_i(t) + d_i(t) M_i(t-\tau_i(t)), & t \neq t_k, \\ \Delta M_i(t_k) = \alpha_{ik} M_i(t_k), \\ \Delta P_i(t_k) = \gamma_{ik} P_i(t_k), \end{cases} \tag{3.3}$$

with impulse disturbances of the type:
$\sum_{i=1}^{n} |B_i(t) + \tilde{B}_i(t)| < \rho, \quad t \geq 0, \quad \frac{\rho}{\mu} < A$
where

(5.1)

Model (3.3) can be considered as an impulse-controlled model corresponding to the drive system

$$\begin{cases} \dot{M}_i(t) = -a_i M_i(t) + \sum_{j=1}^{n} w_{ij}(t) f_j(P_j(t-\sigma_j(t))) + B_i(t), \\ \dot{P}_i(t) = -c_i p_i(t) + d_i(t) M_i(t-\tau_i(t)), \end{cases} \tag{5.2}$$

$i = 1, 2, 3.$

For model (3.3) we can check whether all conditions of Theorem 3.3 are satisfied for

$$l_j = 1, \ j = 1, 2, 3;$$
$$|w_{ij}(t)| \leq 2.3, \quad |d_i(t)| \leq 0.7, \ i, j = 1, 2, 3;$$

$$\alpha_{ik} = -\frac{2}{7}, \ \gamma_{ik} = -\frac{3}{8}, \ i = 1, 2, 3, \ k = 1, 2, \ldots.$$

Fixed values of the arrays a_i, c_i, d_i, f_i, i are only present in the procedural implementation, and they are as follows (5.3). Implementations

using parallel techniques with initial values of the arrays are set with the minimum value passed as a parameter to the program.

$$w_{ij}(t) = \begin{pmatrix} w_{11}(t) & w_{12}(t) & w_{13}(t) \\ w_{21}(t) & w_{22}(t) & w_{23}(t) \\ w_{31}(t) & w_{32}(t) & w_{33}(t) \end{pmatrix} = \begin{pmatrix} 0 & 0 & -2.3\cos(t) \\ -2.3\sin(t) & 0 & 0 \\ 0 & -2.3\cos(t) & 0 \end{pmatrix} \tag{5.3}$$

$t \geq 0$, $0 < t_1 < t_2 < \cdots < t_k < t_{k+1} < \ldots$, $t_k \to \infty$ като $k \to \infty$.

A stable point is sought such that there exists a positive number μ that satisfies the following equation (3.3).

$$\min_{1\leq i\leq n}\{(a_i + \tilde{a}_i), (c_i + \tilde{c}_i)\} - \max_{1\leq i\leq n}\left\{(K_i^{(2)} + \tilde{K}_i^{(2)}),\ \sum_{j=1}^{n}(K_{ji}^{(1)} + \tilde{K}_{ji}^{(1)})l_i\right\} \geq \mu$$

Assuming that all assumptions in Theorem 6 are satisfied, hypotheses 1 to 5 are satisfied. We set the corresponding constants as follows:

$$a_1 = a_2 = a_3 = 2.9, \quad c_1 = c_2 = c_3 = 2.3, \quad B_1(t) = B_2(t) = B_3(t) = 0,$$
$$d_1(t) = d_2(t) = d_3(t) = 0.7\sin(t). \tag{5.4}$$

5.2 Procedural implementation of the impulse model (IGRNs)

For the purpose of numerical study of impulse genetic regulatory neural networks (IGRNs), a procedural implementation of the model based on the fourth-order Runge–Kutta (RK4) method is constructed, executed sequentially on a single processor (CPU). The model describes a connected system of n genes, for which dynamics of mRNA concentration $m_i(t)$ and protein concentration $p_i(t)$ are introduced.

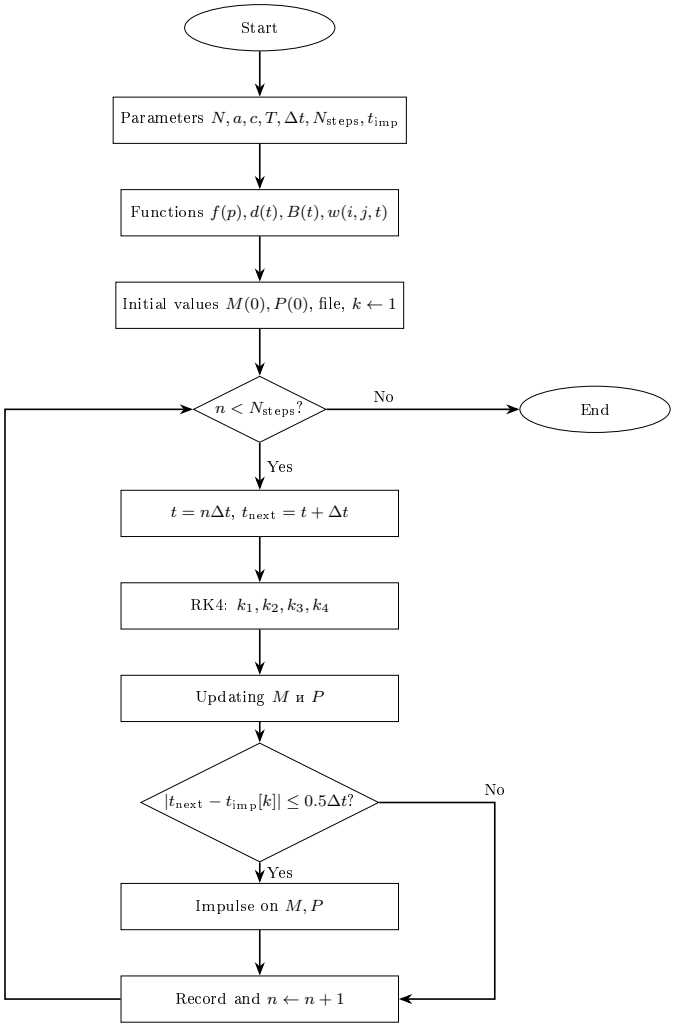

Figure 5.1: Flowchart of the procedural implementation

The resulting text file contains the trajectories (t, M, P) and is used for visualization in MATLAB or Python.

Graphical summary of data

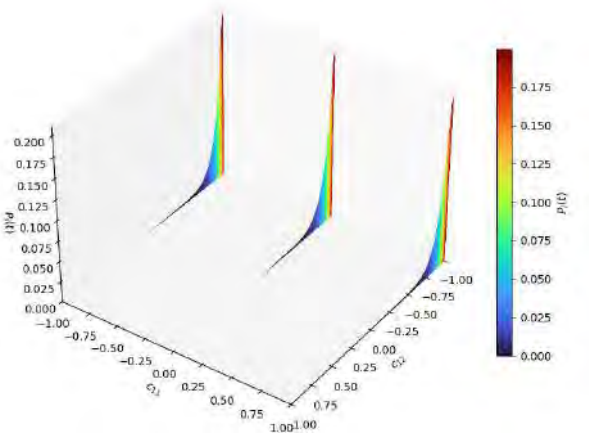

Figure 5.2: Parametric dependence of the function $P_i(t)$ on the parametersc_1 и c_2.

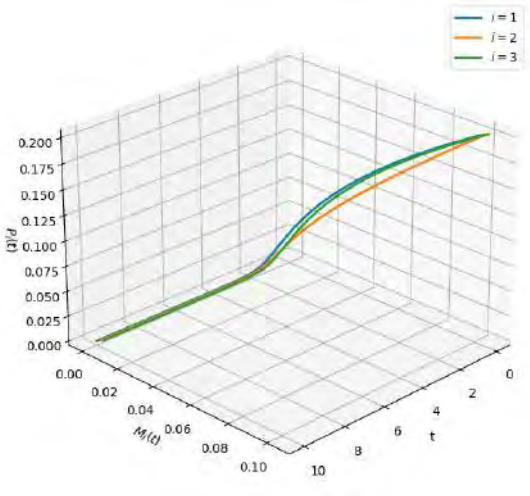

Figure 5.3: Phase dynamics of the system in space (t, M_i, P_i).

5.4 Procedural implementation with OpenMPI

For the numerical study of the IGRNs system, a procedural implementation of a model with impulses and continuous dynamics between them has been developed. The solution is implemented in the C++ language using OpenMPI and the fourth-order Runge-Kutta method (RK4). The implementation is performed in several stages.

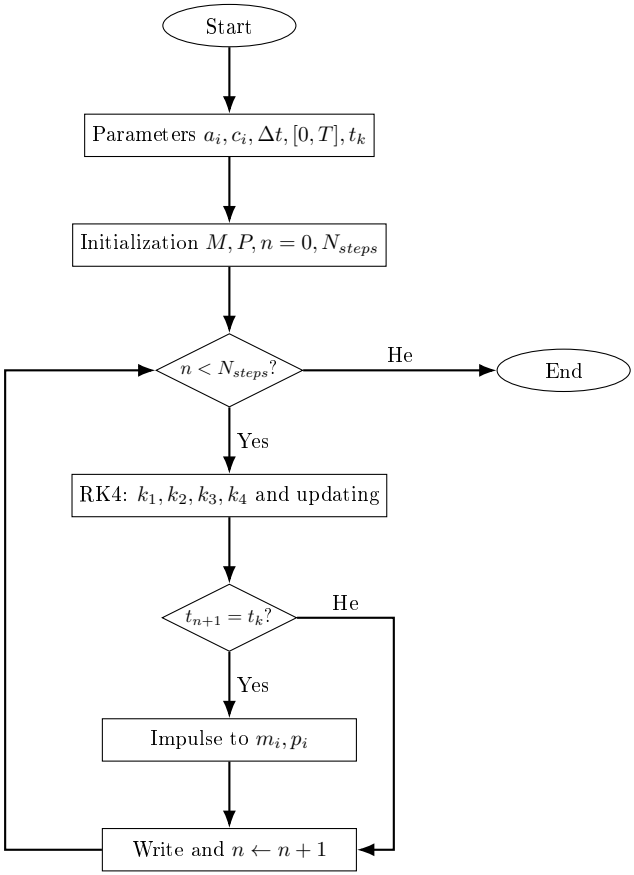

Figure 5.4: OpenMPI implementation flowchart

Parametric influence of scalability according to Amdahl's law

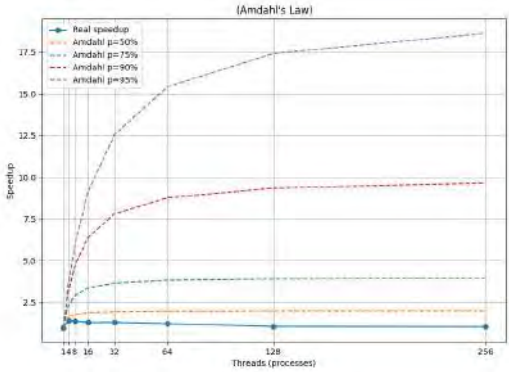

Figure 5.5: Parametric influence of scalability according to Amdahl's law

Figure 5.5 shows the graph of the function based on the verified configuration. The scalability assessment will be performed according to the mRNA processing time and protein concentrations. The algorithmization of the problem focuses on the complexity and optimization of the communication between the different processors.

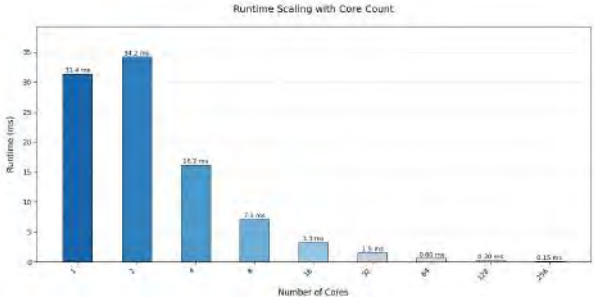

Figure 5.6: Time to complete tasks

Fig. 5.6 shows the execution time of the task as a number of

processors. At the beginning there is a limit to the acceleration of growth, but after a certain number of processors this barrier is barely reached. A change towards lower processing time is observed. The condition starts from the extended initialization time and access to the RAM memory.

5.4.1 Comparative analysis

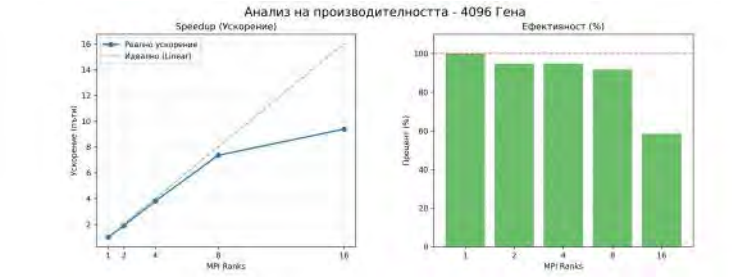

Figure 5.7: Benchmark analysis on 4096 genes

The graph shows an almost perfect linear speedup up to 8 ranks. The blue line (real speedup) moves closely to the dotted gray line (ideal speedup). At 8 ranks, a speedup of 7.35x is achieved. This means that the task is performed over 7 times faster than with a single process.

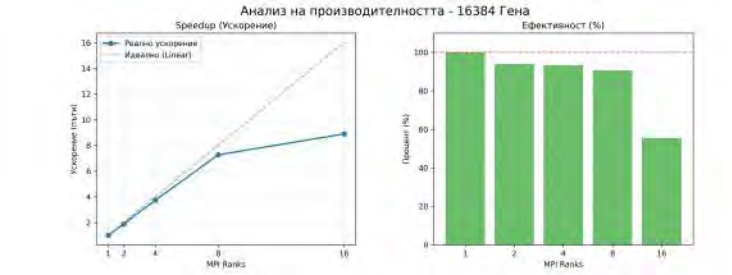

Figure 5.8: Benchmark analysis on 16 384 genes

In Figure 5.8 with 16384 genes, it is noticeable that the speedup up to 8 ranks remains excellent (7.3x), almost identical to that at 4096 genes. This shows that the algorithm is robust to scaling. The critical point is at 16 ranks. Here a clear decline is seen. The speedup increases very little (up to about 8.8x), instead of approaching the ideal 16x. This

is a typical example of Amdahl's Law.

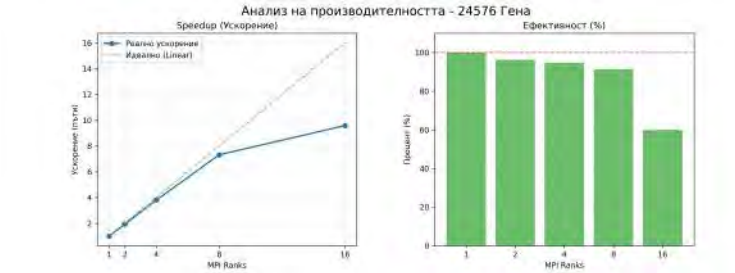

Figure 5.9: Benchmark analysis on 24 576 genes

In these studies, the highest absolute speedup is achieved — 9.59x at 16 ranks. This is an improvement over 16384 genes (where it is about 8.8x). The larger workload (more genes) allows the system to better utilize the available resources, even when the efficiency drops. This is classic evidence that parallel programming is most profitable for large tasks.

5.5 ROCm implementation

In order to accelerate the numerical integration of the impulse model (IGRNs), a GPU-accelerated version of the algorithm using AMD ROCm/HIP has been developed.

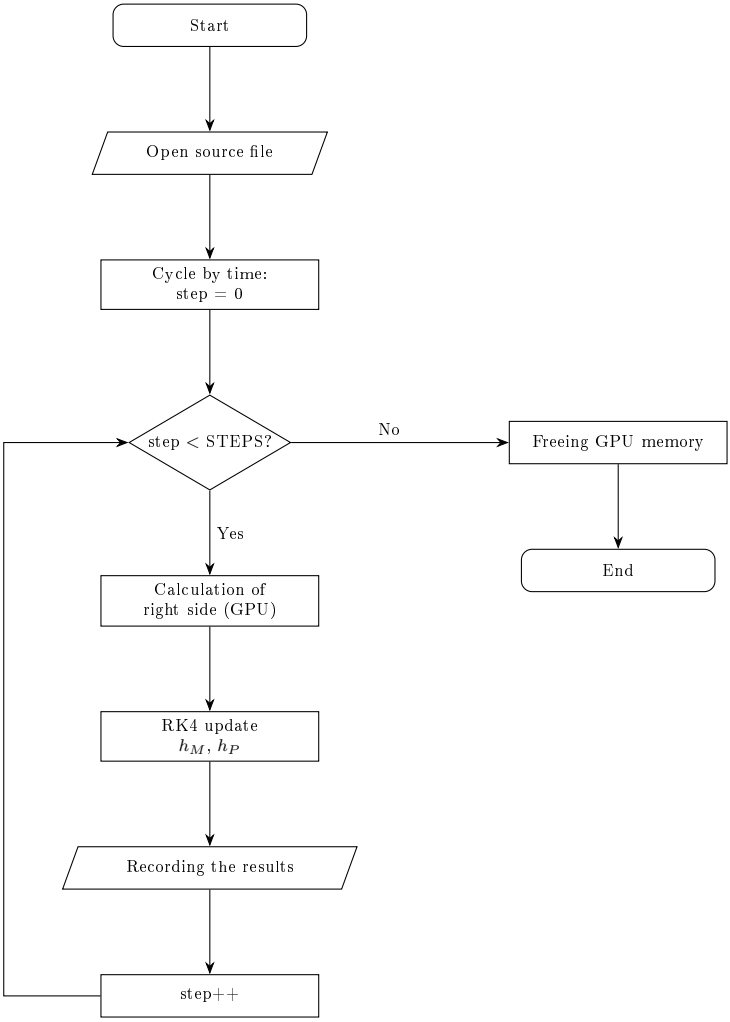

Figure 5.10: Flowchart of ROCm implementation

5.5.1 Comparative analysis

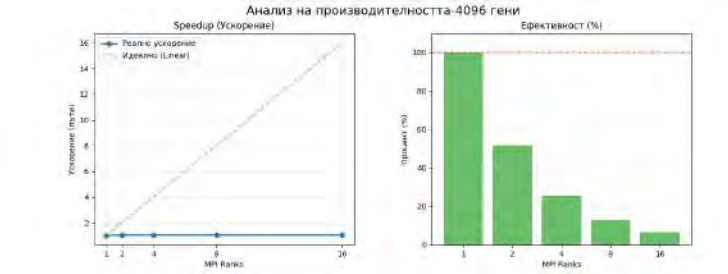

Figure 5.11: Benchmark analysis on 4096 genes

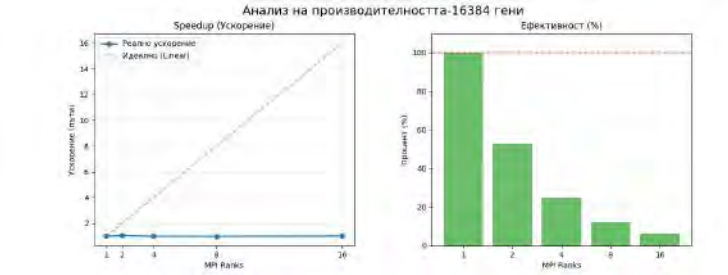

Figure 5.12: Benchmark analysis on 16384 genes

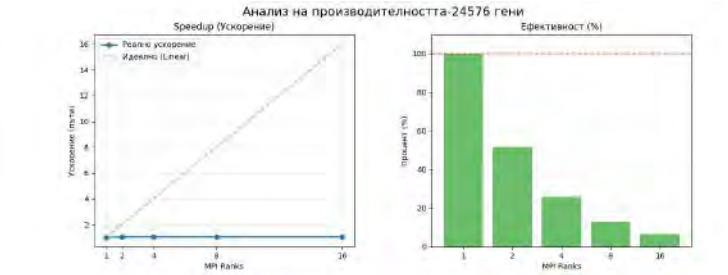

Figure 5.13: Benchmark analysis on 24576 genes

Experimental results show the absence of significant scalability, with the real acceleration curve (Speedup) remaining close to unity, re-

gardless of the increase in the number of parallel ranks. This phenomenon is a direct consequence of the low workload of the computing cores, where the volume of processed data is not sufficient to compensate for the system overhead for synchronization and thread management. An inverse relationship is observed between the parallel efficiency and the number of computing units, with the efficiency reducing to values below 10 at 16 ranks. These data confirm that the current computational task does not reach the critical mass necessary for the full utilization of the computational capacity and throughput of the GPU.

Even with an increase in the number of genes to $24,576$, the nature of the graphs does not demonstrate a transition to linear acceleration, which defines the computing power of the hardware as excessive for the current scale of the problem. In conclusion, the software implementation is limited by the small volume of input data, not by the physical parameters of the accelerator. To achieve high performance, an exponential increase in computational complexity is recommended to minimize the burden of overhead compared to useful computation time.

5.6 Hybrid implementation OpenMPI + ROCm

The program code implements a numerical solution of a system of differential equations modeling an integrated gene regulatory network (IGRNs). A **4th-order Runge–Kutta (RK4)** method is used, combined with parallel processing through **MPI** for distributed memory and **HIP** for GPU computations.

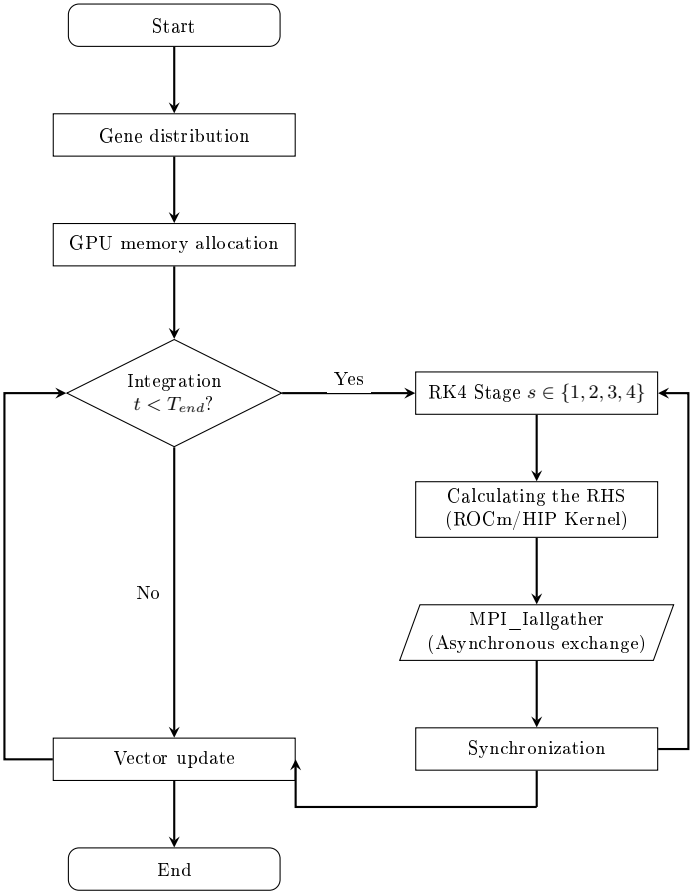

Figure 5.14: Flowchart of the Hybrid implementation.

5.6.1 Comparative analysis

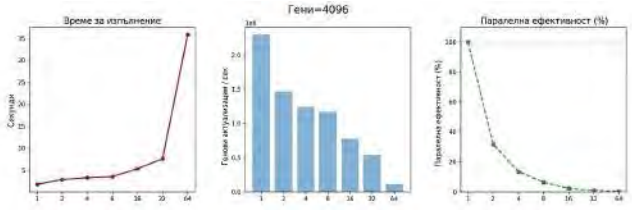

Figure 5.15: Benchmark analysis on 4096 genes

The highest throughput of 2.38 million updates/sec is achieved at 1 MPI rank. This is because 4096 genes are too small a workload for the RX 7800 XT to justify splitting them across multiple processes. The execution time increases as new processes are added. At 64 processes, the simulation is nearly 20 times slower due to the huge latency in $MPI\_Allgather$ and the overload of the GPU command queue. There is a slight improvement when going from 4 to 8 processes, but after that the performance drops sharply.

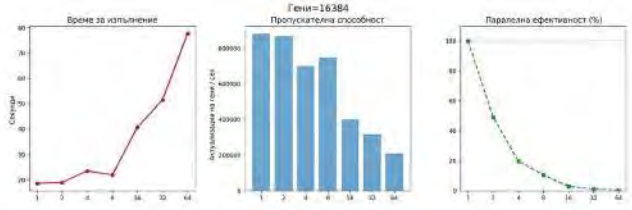

Figure 5.16: Benchmark analysis on 16,384 genes

In benchmark tests conducted on a network of 16,384 genes, the hybrid MPI+ROCm architecture maintains high efficiency (over 98%) with a low level of parallelism between hosts. However, since there is resource sharing on a single GPU (AMD RX 7800 XT), increasing the MPI rank above 8 leads to high communication and task scheduling overhead, which consequently reduces throughput.

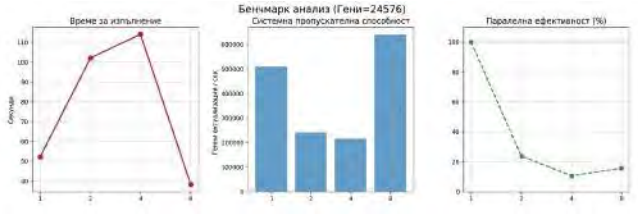

Figure 5.17: Benchmark analysis on 24,576 genes

Eight threads mean eight parallel command queues - the workload of the computational units. The division of the very large W matrix into eight parts/chunks seems to be consistent with the L3 cache/Infinity Cache of the RX 7800 XT. For large systems (N $>$ 20,000), this is also true for the gfx1101 architecture, where eight MPI ranks execute faster than one sequential execution thread.

The optimal operating point is found at 8 MPI ranks, where a balance is achieved between the workload of the computational units and the communication costs.

The present work contributes that the optimal operating point is at 8 MPI ranks, where the hybrid method can achieve a maximum throughput of 639,297 gene updates per second. This capacity is critical when performing large-scale robustness analyses, as required by synthetic biology and genetic engineering.

Chapter 6

Conclusion

In accordance with the goal set in this dissertation, stable processes in pulsed genetic regulatory neural networks with pulsed disturbances are modeled, through the use of high-performance and parallel techniques, the following tasks were accomplished:

- Task 1. To study existing software solutions for impulse genetic regulatory neural networks or a similar model
- Task 2. To create a mathematical model and to study the stability conditions for a network of three neurons
- Task 3. To model the stability conditions of the considered neural network model
- Task 4. To create and implement a classical algorithm for neural networks of the considered model
- Task 5. To implement and study parallel algorithms for stability conditions of an existing configuration
- Task 6. To implement a parallel algorithm based on OpenMPI
- Task 7. To implement a parallel algorithm based on ROCm
- Task 8. To implement a parallel algorithm based on a hybrid model OpenMPI and ROCm

- Task 9. To make a comparison between the two technologies
- Task 10. Yes the results and their presentation are examined.

The present work achieves its goals by fulfilling the tasks set.

Based on the experimental data, the following strategies are recommended to achieve maximum performance in large-scale simulations:

The hybrid technology OpenMPI + ROCm is recommended. It allows the system to use the best of both technologies: MPI for interprocess communication and the enormous computing power of ROCm for data processing.

The system is recommended for tasks with over 20,000 genes. For smaller tasks (e.g. 4096), the administrative costs of MPI and the preparation of GPU queues exceed the time for the calculation itself.

For the specific configuration, the use of 8 MPI ranks is recommended. This provides an optimal balance – enough processes to feed the GPU with data without causing the limited bandwidth of the system bus to the memory bus or overloading the L3 cache.

For contemporary bioinformatics needs, pure OpenMPI is an outdated approach. The future and high performance lies in heterogeneous computing, which results were demonstrated with a 9.59x speedup.

Contributions of the dissertation work

Scientific and applied contributions of the dissertation work

1. Literature review

2. Creation of a mathematical model of an impulse genetic regulatory neural network (IGRNs) consisting of three neurons.

3. Robustness study: The conditions for global asymptotical exponential robustness of solutions with respect to a set for the considered model with impulse disturbances are defined and investigated.

4. Theoretical justification of parallel execution: The dependencies between the software implementation and the scalability of hardware configurations in high-performance ROCm and OpenMPI systems are investigated.

Applied contributions

These contributions relate to the practical software implementation, algorithms and experimental part:

1. Software research: The lack of existing ready-made software solutions for the specific type of impulse genetic regulatory neural networks at the time of development was established.

2. Software implementation in C++ language: Development of a procedural method for calculating the mathematical model for proving the stability criterion.

Development of parallel algorithms:

3. Implementation based on OpenMPI.

4. Implementation based on ROCm.

5. Implementation of a hybrid model OpenMPI + ROCm.

6. Conducting a comparative analysis: A comparison was made between the three technologies on a specific hardware configuration.

Publications

[1] Stamov, G., Stamova, I., Kostadinova M., 2024, Impulsive Gene Regulatory Networks: Stability of Sets, 2024, BioInfoMed'2024 [под печат]

[2] Stamov, G., Stamova, I., Kostadinova M., 2025, Impulsive Delayed Gene Regulatory Networks and Practical Stability of h-manifolds, Continuous and Discrete Dynamics: Theory and Applications, Springer [под печат]

[3] Kostadinova-Gocheva, M. On the Impulsive Gene Regulatory Networks. Announcements of the Union of Scientists – Sliven, Vol. 38, No. 2, 2023, p. 11 - 16

[4] Stamov, G., & Kostadinova-Gocheva, M., 2024, Robust stability of sets for uncertain impulsive gene regulatory networks, Proceedings of the Technical University of Sofia, Vol. 74, No. 3, doi: 10.47978/TUS.2024.74.03.009

[5] KOSTADINOVA-GOCHEVA, M. New Trends in the Study of Impulsive Genetic Regulatory Neural Networks. In: Proceedings of the Student Scientific Conference – 2024. Burgas: Burgas Free University, 2024, ISSN: 1311-221X, pp. 91–95.

[6] Kostadinova-Gocheva, M. The implementation of Impulsive Gene Regulatory Networks Applying the Message Passing Interface, Proceedings of the Technical University of Sofia, Vol. 76, No. 1, 2026. doi: 10.47978/TUS.2025.76.01.007.