Briefings in Bioinformatics, 22(5), 2021, 1–11

https://doi.org/10.1093/bib/bbab104
Problem Solving Protocol

Inference of dynamic spatial GRN models with multi-GPU evolutionary computation

Reza Mousavi, Sri Harsha Konuru and Daniel Lobo

Corresponding author. Daniel Lobo, Department of Biological Sciences, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, USA. Tel.: 410-455-5726; Fax: 410-455-3875; E-mail: lobo@umbc.edu

Abstract

Reverse engineering mechanistic gene regulatory network (GRN) models with a specific dynamic spatial behavior is an inverse problem without analytical solutions in general. Instead, heuristic machine learning algorithms have been proposed to infer the structure and parameters of a system of equations able to recapitulate a given gene expression pattern. However, these algorithms are computationally intensive as they need to simulate millions of candidate models, which limits their applicability and requires high computational resources. Graphics processing unit (GPU) computing is an affordable alternative for accelerating large-scale scientific computation, yet no method is currently available to exploit GPU technology for the reverse engineering of mechanistic GRNs from spatial phenotypes. Here we present an efficient methodology to parallelize evolutionary algorithms using GPU computing for the inference of mechanistic GRNs that can develop a given gene expression pattern in a multicellular tissue area or cell culture. The proposed approach is based on multi-CPU threads running the lightweight crossover, mutation and selection operators and launching GPU kernels asynchronously. Kernels can run in parallel in a single or multiple GPUs and each kernel simulates and scores the error of a model using the thread parallelism of the GPU. We tested this methodology for the inference of spatiotemporal mechanistic gene regulatory networks (GRNs)—including topology and parameters—that can develop a given 2D gene expression pattern. The results show a 700-fold speedup with respect to a single CPU implementation. This approach can streamline the extraction of knowledge from biological and medical datasets and accelerate the automatic design of GRNs for synthetic biology applications.

Key words: systems biology; machine learning; gene regulatory networks; GPU computing; evolutionary computation; gene expression patterns

INTRODUCTION

Gene regulatory networks (GRNs) can precisely control the formation of gene expression patterns through space and time, which can act as regulatory signals leading to the development of complex shapes, anatomical structures and morphologies. Obtaining a mechanistic understanding of GRNs is essential for predicting specific phenotypes and discovering interventions that can lead to desired outcomes, such as a healthy state from a disease one [1]. Furthermore, synthetic biology applications require our ability to construct *de novo* GRNs for the precise regulation and biosynthesis of value-added products and smart materials for industrial and medical applications [2]. However, the discovery of GRNs from experimental phenotypes, disease states, or desired outcomes is an inverse problem without analytical solutions, for which efficient reverse engineering methodologies based on computational optimization are needed [3].

Computational methods have been developed for the reverse engineering of the structure of GRNs. These methods can take as input large-scale transcriptomics data and infer topological models with probabilistic gene–gene regulatory links [4–7]. Unsupervised machine learning approaches—taking as input unlabeled data—have been proposed for the inference of the structure of such networks from both synthetic and

Reza Mousavi is a PhD student in the Department of Biological Sciences at the University of Maryland, Baltimore County.
Sri Harsha Konuru was a Research Associate in the Department of Biological Sciences at the University of Maryland, Baltimore County.
Daniel Lobo is an Assistant Professor in the Department of Biological Sciences at the University of Maryland, Baltimore County.
Submitted: 10 December 2020; **Received (in revised form):** 15 February 2021

experimental nonspatial gene expression patterns, such as microarray or RNAseq data. These methods are mainly based on hierarchical clustering techniques such as the Self-Organizing Tree Algorithm (SoTA) [8] and include tools such as CLR [9], ARACNE [10], GENIE3 [4], iRafNet [11], MRNET [12], TIGRESS [13] and Scenic [5]. Supervised machine learning approaches—taking as input labeled data such as known gene regulatory interactions—have been developed to train classifiers for GRN topology inference also from nonspatial gene expression data, including SIRENE [14], CompareSVM [15] and GRADIS [16]. These supervised approaches can outperform unsupervised methods when carefully trained [17]. In addition, computational methods have been developed that can infer the structure of GRNs from spatial gene expression patterns from *in situ* hybridization images, such as those during early *Drosophila* development [18, 19]. However, these models are static and do not account for the temporal dynamics of gene regulation, which limits their applicability. Indeed, the reverse engineering of dynamic GRNs is a current challenge for which heuristic methods have been proposed as an ideal approach.

The inference of dynamic models of GRNs requires heuristic optimization approaches that can extract a mechanistic model directly from phenotypic outcomes [20]. Among them, evolutionary computation, a population-based methodology for both parameter optimization and solution design search [21–26], is an ideal approach due to their application versatility and high parallelization. These methods have been successfully applied to the inference of Boolean networks for Arabidopsis [27] and of ODE models for nondimensional [28] and one-dimensional [29–31] gene expression patterns of early *Drosophila* development and other embryonic dynamic patterns [32, 33]. Furthermore, evolutionary algorithms have been applied to the inference of two-dimensional spatial phenotypes of planarian body patterns [34–36]. However, the application of these heuristic methods to complex spatial patterns or large datasets is limited due to the high computational requirements needed to evaluate the error of a candidate model. Hence, novel methodologies that can efficiently exploit new parallel technologies are needed for the reverse engineering of dynamic spatiotemporal GRN models.

Graphics processing units (GPUs) are programmable devices that in cooperation with the main central processing units (CPUs) processors can perform massively parallel computation. They consist of a number of streaming multiprocessors, each including a large set of cores operating in a single-instruction multiple-data manner. GPU computing has been demonstrated as a powerful approach to achieve high performance on long-running scientific applications in system biology [37–40], bioinformatics [41–43], data mining [44, 45], machine learning [46, 47] and microscopy image processing [48, 49]. In addition, due to their ability to massively compute in parallel a given algorithm in multiple data, evolutionary computation methods have been proposed based on GPU computation approaches [50–52]. These parallel methods using GPUs can achieve speedups from 8× for complex bioinformatics data mining problems [53] to 7000× for simpler benchmark functions that can run entirely in the GPU [54]. Furthermore, GPU computing has been proposed for accelerating the simulation of dynamic spatial cellular models, including intracellular forces [55], chemical signals [56] and GRNs [57]. Indeed, combining evolutionary computation, spatial dynamic cellular and chemical simulations with GPU computing represents a promising approach for the efficient inference of mechanistic GRNs for spatial morphologies and patterns.

Here we present a novel methodology to efficiently parallelize the inference of spatial, dynamic GRNs using multi-GPU evolutionary computation. The proposed approach exploits the computational power of the CPUs to select and reproduce new candidate GRNs, which are then sent asynchronously in parallel to a single or multiple GPUs. The massively parallel architecture of the GPUs is then exploited for computing the fitness calculation, comprising the computationally demanding spatial numerical simulation and error calculation. We further parallelize and optimize the evolutionary algorithm using an island distribution approach [58], which ensures there are no bottlenecks in the algorithm and improves the diversity of the evolution. We implemented the approach in C++ and tested it with the problem of reverse engineering GRNs that can dynamically produce a gene expression pattern with a given spatial shape. This biological problem is similar to finding GRNs responsible for specific gene expression patterns observed during the development of organs [59, 60] and whole bodies [61, 62], or for synthetic biology applications that seek to design specific spatial gene patterns [63]. Here we focus on testing the methodology to parallelize this problem, instead of finding a solution for a specific biological pattern. Hence, we use a simple circle-like gene expression pattern to evaluate the performance of the methodology. For this, we tested the algorithm with different parallelization parameters and compared the results in a desktop computer with a single GPU and a server computer with one or four GPUs. The results show that this approach can reach a speedup of 200× in a single-GPU desktop computer and 700× in a four-GPU server computer. Indeed, the proposed methodology can efficiently infer GRNs for complex spatial dynamic patterns, paving the way for the mechanistic understanding of developmental and cancer phenotypes and the design of targeted synthetic biology spatial systems.

METHODS

Modeling and simulation of spatial dynamic gene regulatory networks

We define models of spatiotemporal GRNs with a system of nonlinear PDEs. The GRN model is simulated in a homogenous domain representing a multicellular tissue area or cell culture, where all cells are governed by the same GRN. A model is comprised of an arbitrary number of genes, including an input gene defining a particular initial spatial expression pattern and an output gene defining the final target spatial expression pattern. For this work, the initial state is defined as a homogeneous 2D domain with zero concentration for all genes, except for the input gene, which have a square of 2 by 2 cells at the center with a concentration of 1. Each gene can be activated or inhibited by other genes in the network, their products confined intracellularly, or act as intercellular signals through diffusion and decay with time. The dynamics of each gene is defined with three parameters—production, decay and diffusion constants—in addition to their regulatory interactions. Each regulatory interaction between two genes is defined with a nonlinear Hill equation [34], including two parameters—Hill coefficient and disassociation constant. Each regulation can be positive (activating) or negative (inhibiting) and multiple regulations can be grouped as sufficient with a *max* operator or as necessary with a *min* operator. The following generic equation illustrates the PDE for a gene a as regulated by two necessary genes (activator b and inhibitor c) and two sufficient genes (activator d and inhibitor e):

$$\frac{\partial a}{\partial t} = \rho_a \min\left(\frac{b^{\eta_1}}{{\alpha_1}^{\eta_1} + b^{\eta_1}}, \frac{{\alpha_2}^{\eta_2}}{{\alpha_2}^{\eta_2} + c^{\eta_2}}, \max\left(\frac{d^{\eta_3}}{{\alpha_3}^{\eta_3} + d^{\eta_3}}, \frac{{\alpha_4}^{\eta_4}}{{\alpha_4}^{\eta_4} + e^{\eta_4}}\right)\right) - \lambda_a a + D_a \nabla^2 a$$

where ρ_a is the production constant, η_i are the Hill coefficients, α_i are the dissociation constants in the Hill functions, λ_a is the decay constant, and D_a is the diffusion constant. All parameter units in the model are arbitrary. The system of PDEs defining a GRN model is numerically solved in a square domain of 64 by 64 cells using an Euler finite difference method [64]. The model is simulated for 1000 steps after which the expression pattern of the output gene defines the output shape developed by the GRN.

Reverse engineering methodology

A generalized reverse engineering method based on evolutionary computation was implemented for the inference of mechanistic spatiotemporal GRNs that can recapitulate the formation of a particular gene expression pattern. The method takes as input an initial spatial expression pattern and a target spatial expression pattern (e.g. a circle shape) and infers a mechanistic GRN—including the number of genes, their regulatory interactions and their parameters—that starting with the input initial expression pattern develops a steady-state gene expression pattern similar to the input target expression pattern.

The algorithm is based on evolutionary computation, where a population of candidate regulatory networks evolves by combining and mutating existing ones. The initial population is made of simple GRNs with random regulations and parameters. New GRNs are created in each generation by crossing two existent regulatory networks for every two GRNs in the population to produce two new offspring GRNs. A crossover distributes randomly the two parent genes together with their regulatory links to the two offspring GRNs, avoiding gene duplication and without changing the kinetic parameters. Offspring GRNs are probabilistically mutated by adding and removing products and regulations and changing their parameters. Random mutations can duplicate and delete products and links and substitute each of their parameters with new random ones. A parameter mutation substitutes its value with a new one from a random uniform distribution within its range. The input and output genes cannot be deleted. Deletion mutations have a higher probability than duplication mutations, biasing the evolution toward simpler networks and limiting bloating [65]. Selection follows a deterministic crowding approach [66], where offspring replaces its closest parent if its error (fitness) is equal or better than the parent. The algorithm iterates through reproduction, error calculation and selection cyclically until a network with zero error is found, after which 250 extra generations are computed to allow an evolutionary reduction in complexity of the found GRN.

The error of a candidate GRN measures the difference of its developed expression pattern at the last time step of the simulation with respect to the input target expression pattern. This error is defined as the sum of the mean log difference between the developed and target patterns at each domain location, which quantify the distance between the simulated pattern from the candidate GRN and the input target pattern, plus the maximum concentration change at the last time step, which represents a penalty for patterns that are not at equilibrium at the last time step. To avoid overfitting and bloating of the solution, thresholds are defined to limit the difference error and equilibrium penalty. In particular, the error of a developed expression pattern D at the last time step in the simulation with respect to the input target expression pattern T is given by

$$\text{error}\,(D,T) = \frac{1}{w \bullet h}\sum_{i=1}^{w}\sum_{j=1}^{h}\log\left(1+\left(\left|D_{i,j}-T_{i,j}\right|-\alpha\right)^{+}\right)+\left(\Delta_D-\beta\right)^{+}$$

where w and h are the width and height dimensions of the simulation domain, respectively; $D_{i,j}$ and $T_{i,j}$ are the concentrations at location (i,j) of the developed and target pattern, respectively; α is the error concentration threshold; Δ_D is the maximum concentration change for any gene and domain location in D at the last time step in the simulation; and β is the equilibrium penalty threshold. $(x)^+$ denotes the positive part function of x, which outputs x if x is nonnegative and 0 if it is negative.

The parameters of the inference algorithm were fixed for all computations in this work. In particular, the inference parameters were set with a crossover rate of 75%, mutation rate of 1%, link/gene duplication rate of 1%, link/gene deletion rate of 1.5%, error concentration threshold of 0.4 and equilibrium penalty threshold of 10^{-10}. All the model parameters were in the range (0,1), except for the Hill coefficient, which had a range (1,5).

Implementation and hardware

We implemented the presented methodology in C++ using CUDA (NVIDIA Corporation), Eigen (Gaël Guennebaud, Benoît Jacob, and others), Qt (The Qt Company Ltd.) and Qwt (Uwe Rathmann and Josef Wilgen) libraries. The performance of the implementation was evaluated on a PC and a server computer. The PC was a Windows 10 desktop computer equipped with a six-core hyperthreaded Intel Core i7-8700K CPU clocked at 3.70 GHz, 32 GB DDR4 RAM and a 12GB HBM2 NVIDIA Titan V GPU. The server was an Ubuntu Linux rackmount computer with two 18-core hyperthreaded Intel Xeon Gold 6140 CPUs at 2.3 GHz, 384 GB DDR4 RAM and four 32GB HBM2 NVIDIA Tesla V100 GPU accelerators.

INFERENCE ALGORITHM PARALLELIZATION WITH GPU COMPUTING

We developed a GPU-parallelization strategy for the inference of spatiotemporal mechanistic GRNs that can develop a given gene expression pattern in a spatial domain representing a multicellular tissue area or cell culture. Figure 1 shows a diagram of the proposed methodology. To exploit the computational resources of both CPUs and GPUs, the algorithm computes the lightweight crossover, mutation and selection operations in a single thread in the CPU and uses multiple CPU threads to launch multiple instances of the GPU kernel in parallel and asynchronously. Each kernel-calling CPU thread copies a GRN model (topology and parameters) from the CPU to the GPU memory, invokes the GPU kernel call that computes the costly simulation of a GRN model and its error with respect to the input target gene expression pattern, and then copies back the GPU-computed error from the GPU to the CPU memory. Each GPU can compute multiple instances of the same kernel, and hence models, in parallel as launched by different CPU threads each using a different CUDA stream. In this way, kernel-calling CPU threads run in parallel and continuously feed the GPUs with GRNs to evaluate. During a kernel execution, each GPU thread computes the numerical solution of the GRN system of equations for a particular subset of the simulation domain (grid and threads in Figure 1). All the GPU threads computing a GRN model run in a single-GPU block for a fast synchronization at each time step of the simulation and efficient use of shared memory (which is only shared at the level of the block).

To maximize parallelism, avoid idle computation time and improve the robustness and diversity of the candidate GRNs, we implemented an island population distribution approach [58].

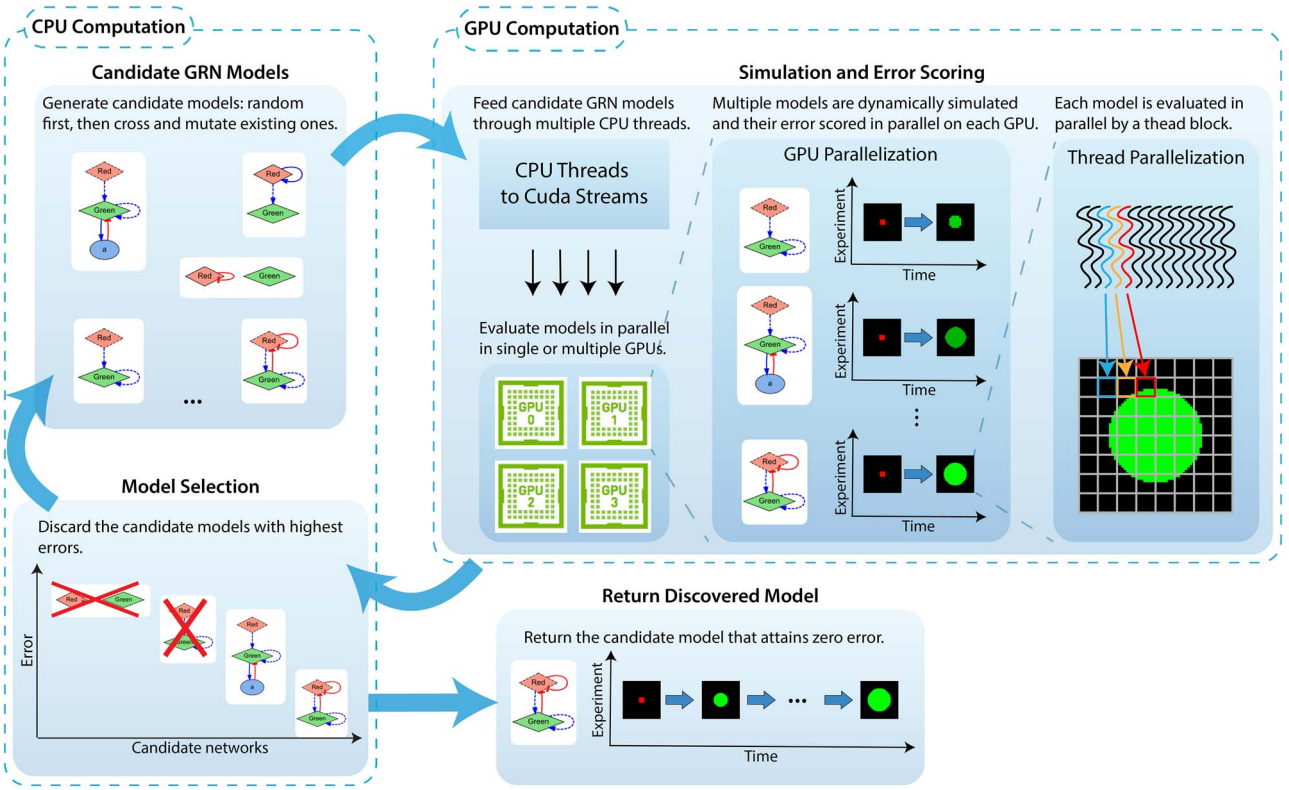

Figure 1. Overview of the proposed methodology based on GPU computing to infer mechanistic genetic regulatory network (GRN) models that can produce a given spatial expression pattern. The method combines an evolutionary computation approach and a dynamic spatial simulator to iteratively (i) generate candidate models in the CPU by crossing and mutating the current population, (ii) copy the models to the GPUs and simulate and compute their errors in parallel at the level of the model and the simulation, and (iii) copy the errors back to the CPU, where the models are ranked and those with the highest errors discarded. The algorithm stops when a model with zero error is discovered. The high asynchronous parallelization and minimal CPU–GPU transfers, which do not include any simulation data, provide a high level of parallel performance.

In particular, the evolving population of candidate GRNs is divided into subpopulations (islands), where crossover, mutation and selection operate independently within each subpopulation. In this way, the individuals on each island evolve without dependencies on other islands and hence the crossover, mutation and selection operators for this subpopulation can be computed in the CPU while the simulation and error scoring of the GRNs in other islands are computed in the GPUs asynchronously and in parallel. Models from any island are queued together for evaluation in their order of their creation and then executed individually but in parallel as GPU resources become available. As soon as all models from an island are evaluated, a new generation is computed by the CPU and their new individuals queued for evaluation in the GPUs. In addition, for every 250 generations on average, the islands are paired randomly and their individuals are shuffled, simulating migrations between the isolated populations. For this work, all evolutionary runs used 32 islands with 64 individuals each.

Data transfers between CPU and GPU memory are computationally costly and need to be minimized to achieve good performance [67]. Toward this, the proposed methodology only transfers the input initial and target gene expression patterns—which size depends on the spatial resolution and dimensions of the simulated domain—once at the beginning of the execution. Each GPU stores this information in global memory, which can be read then by each kernel execution without any memory transfers between CPU and GPU or between GPUs. A single kernel call simulates the model and computes its error, minimizing memory transfers to just the GRN topology and parameters from CPU to GPU memory and then the computed error of the model back from GPU to CPU memory.

The GRN models evolved in the CPU are implemented with two lists, one for genes and one for interactions. Each gene includes an integer label and three real numbers storing their production, degradation and diffusion parameters. Each interaction includes the integer labels of the regulator and regulated genes, the disassociation and Hill coefficient real parameters, and a Boolean indicating if the interaction is of necessary or sufficient type. GRN models are converted to PDE systems before transferring them to the GPUs for simulation and fitness calculation. Importantly, the PDE systems defining GRNs are dynamic in their number of mathematical terms and equations they define, as genes and their interactions can be added to the models during evolution. To numerically solve these variable PDEs in a multi-threaded GPU, an efficient design based on data and function pointers was implemented and the domain divided equally among the number of GPU threads (see Figure 1). In particular, a PDE system is defined as an array of mathematical operation objects, including a pointer to the particular mathematical function they implement (such as Hill activation, Hill repression, etc.), pointers to the corresponding input and output variables and their numeric parameters. In this way, computing the rates of the PDE system consists in just calling all the function pointers sequentially in a loop, which is extremely efficient to execute both in the CPU and GPU. In the case of the GPU implementation,

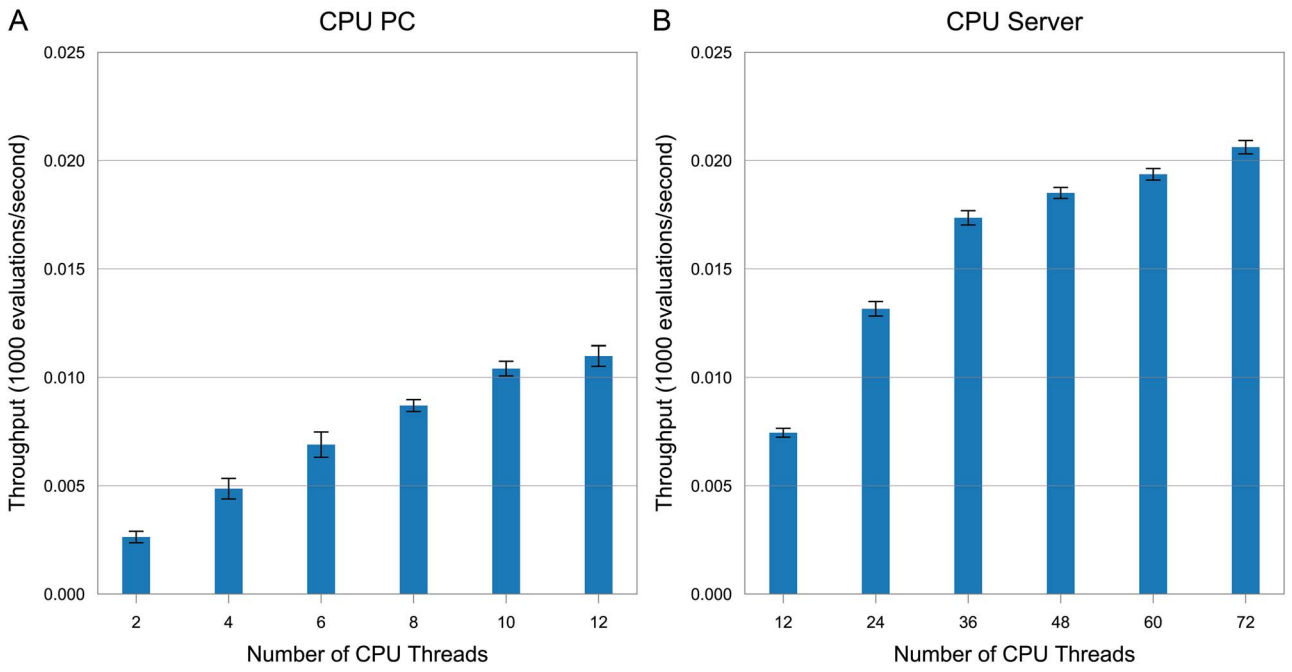

Figure 2. Average throughput of the evaluation of dynamic spatial GRN models with a CPU-based implementation. The simulation and error scoring function were tested with the same model multiple times in parallel with different numbers of CPU threads on a PC (**A**) or a server computer (**B**). Each condition was run three times. Error bars indicate standard deviation.

the pointers are assigned at the beginning of the kernel execution, since they are different for each GPU thread—as each GPU thread is responsible for computing the gene concentrations in a particular area in the domain and hence have different input and output memory locations for the PDE discretization.

In addition, the implementation considers the performance of the different types of GPU memory, including, from slower to faster, the global, shared block, and local thread memory [68]. Simulating a GRN model in the GPU requires shared memory to store the spatial gene concentrations in the domain due to the diffusion terms, for which computation is spatially dependent. Hence, the simulation domain together with the temporary arrays needed for the numerical simulation is stored in global memory (since the size of shared block memory is too low) and reused by each kernel call. Block shared memory is used for reducing arrays, for which the size only linearly depends on the number of GPU threads, and is needed to efficiently compute in parallel the distance between the target and simulated gene expression patterns as well as to compute the equilibrium of the system at the last time step. Finally, fast local thread memory is used for all the variables that do not need to be shared among threads for the numerical simulation and error calculation algorithms.

RESULTS

Performance of the GRN model evaluation parallelization

We first evaluated the performance of the proposed methodology in terms of its capacity to evaluate candidate GRNs in parallel with different implementation configurations running on either a desktop PC with a single GPU or a server computer with one or four GPUs. A model evaluation includes the simulation of the model and calculation of its error with respect to the input target gene expression pattern. To avoid the computational variability between different GRN models, we run all the performance tests in this section by evaluating the same individual iteratively. As a reference, Figure 2 shows the model evaluation throughput obtained in a CPU-only implementation running with different number of CPU threads on either the PC or server. In this case, the CPU threads compute the simulation and error scoring of a GRN model. The results demonstrate how the PC with the maximum number of parallel threads (12 with hyperthreading) runs faster than the equivalent number of threads in the server due to its higher CPU clock speed. However, the maximum throughput of the server is about twice than the PC when using its complete parallel CPU thread capacity (72 with hyperthreading).

Next, we tested the model evaluation performance with the proposed GPU-based methodology. Figure 3 shows the model evaluation throughput obtained with different parallelization parameters on a single GPU in the PC and server and on multiple GPUs in the server. The parallelization parameters tested include the number of kernel-calling CPU threads and the number of GPU threads on each kernel. The results show that the best performance using a single GPU is obtained with 64 kernel-calling CPU threads and 1024 GPU threads per block (the maximum supported by the hardware). This corresponds to a throughput speedup of 20× for the PC (Figure 3A) and 10× for the server (Figure 3B) with respect to the multithreaded CPU implementation (Figure 2). The lower speedup in the server is due to the GPU capacity saturation resulting in CPU idle time. Indeed, when using the four GPUs in the server, the results show a peak performance with 256 kernel-calling threads for a throughput speedup of about 45× (Figure 3C) with respect to the multithreaded CPU implementation (Figure 2).

We then tested the possibility that a mixed configuration of model-evaluation CPU threads (as in the test in Figure 2) and kernel-calling CPU threads (as in the test in Figure 3) could

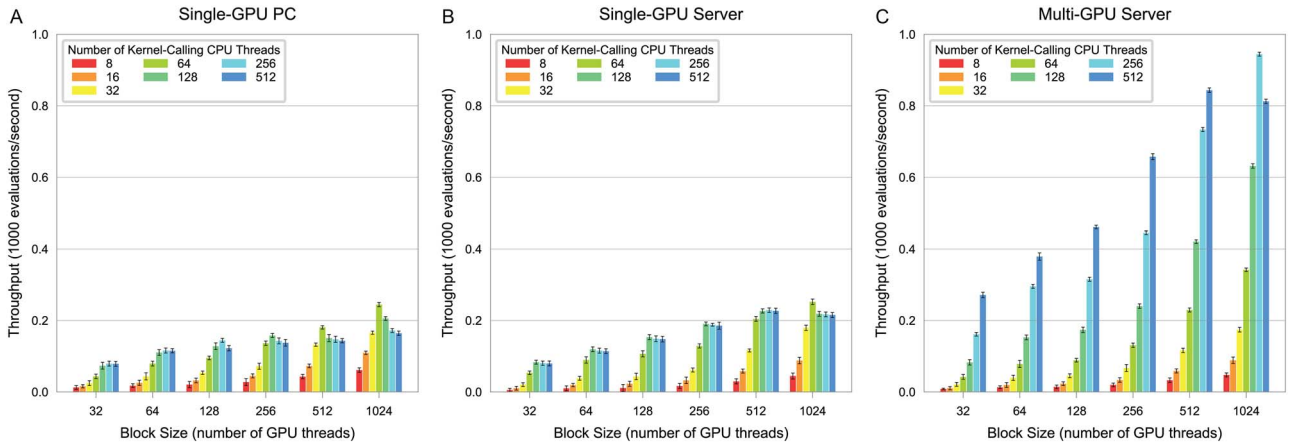

Figure 3. Average throughput of the evaluation of dynamic spatial GRN models with a GPU-based implementation. The simulation and error scoring function were tested with the same model multiple times in parallel with different numbers of kernel-calling CPU threads and GPU block sizes on a PC with a single GPU (**A**) or a server computer with a single (**B**) or four GPUs (**C**). Each condition was run three times. Error bars indicate standard deviation.

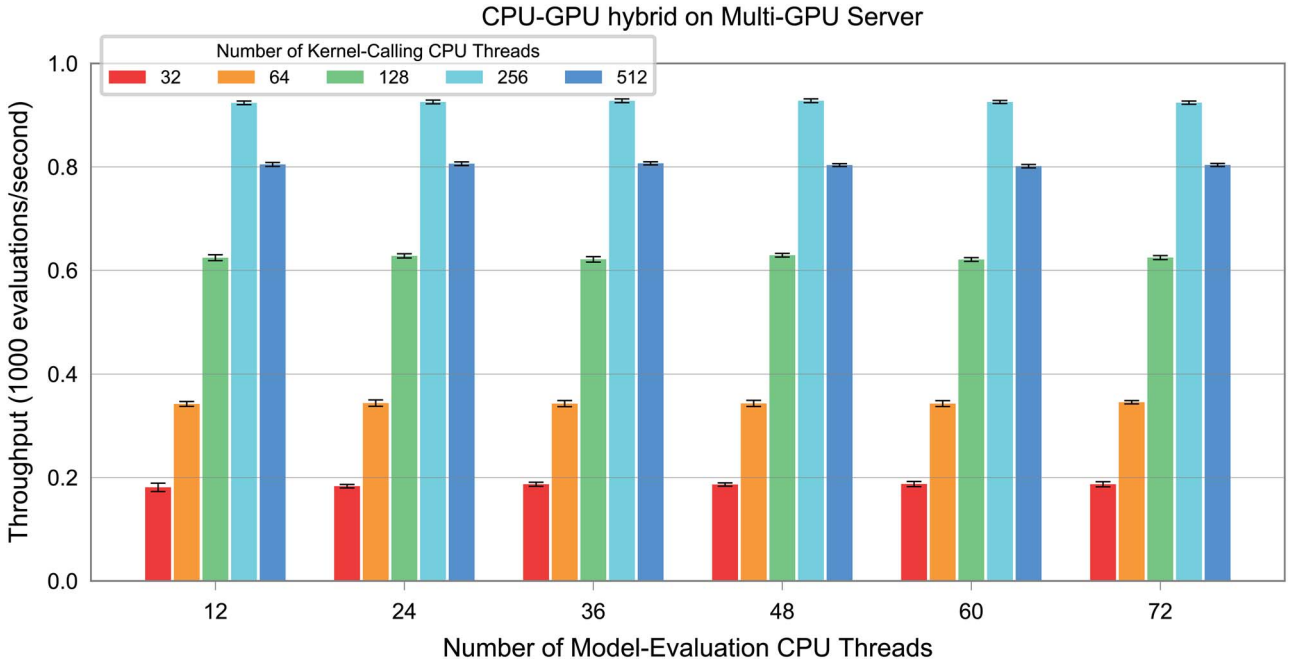

Figure 4. Average throughput of the evaluation of dynamic spatial GRN models with a CPU–GPU cooperative hybrid implementation. The simulation and error scoring function were tested with the same model multiple times in parallel with different numbers of model-evaluation CPU threads and kernel-calling CPU threads on a server with four GPUs. All conditions used a block size of 1024 GPU threads and each condition was run three times. Error bars indicate standard deviation.

result in better model evaluation performance for the multi-GPU server configuration. Figure 4 shows the throughput obtained for different number of model-evaluation and kernel-calling CPU threads. In all cases, the number of GPU threads was 1024. The results show no significant difference in performance for any of the mixed configurations with respect to the configuration with only kernel-calling CPU threads (Figure 3C).

Finally, in order to quantify the speedup of the model evaluation throughput for the best parallel configurations in multi-CPU-threaded and single-GPU PC and multi-CPU-threaded and single- and multi-GPU server, we computed 10 additional runs and performed a statistical analysis comparing the results to a single-CPU-thread configuration. Each run used the best configuration obtained in the previous tests (Figures 2 and 3). Figure 5 shows the model evaluation throughput for each configuration and Table 1 summarizes the average numerical speedups obtained with respect to the single-CPU-thread configuration. The results show a speedup in model evaluation throughput of 186× using a single-GPU PC and of 720× using a multi-GPU server. To analyze these speedup results, a nonparametric Friedman test was used to rank them, followed by a Holm procedure to compute their statistical significance [69]. In this case, the null hypothesis is rejected with a P-value ≤ 0.05 in the Holm test. Table 2 shows the results of the analysis, confirming the ranking of configurations and speedups shown in Figure 5 as statistically significant.

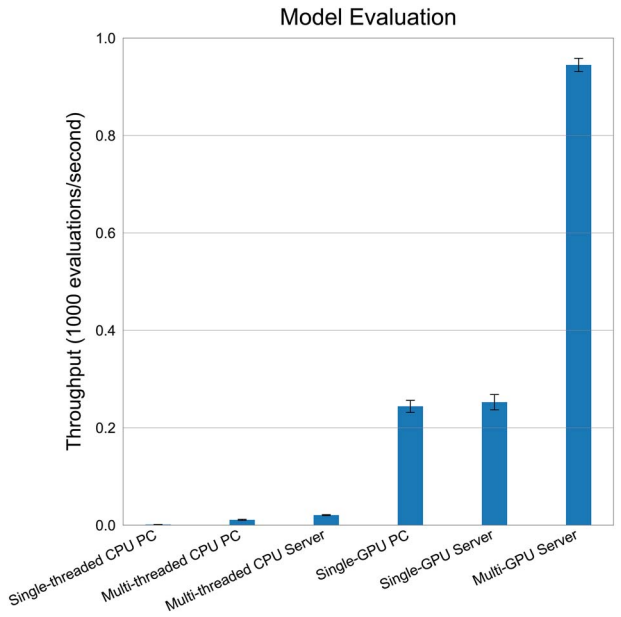

Figure 5. Average throughput of the evaluation of dynamic spatial GRN models with the best configuration for the different implementations. The simulation and error scoring function were tested with the same model multiple times using a single- or multi-threaded CPU implementation or a single- or multi-GPU implementation on either a PC or a server. Each condition used the best number of model-evaluation and kernel-calling CPU threads and GPU block size found with the model evaluation experiments. Each condition was run 10 times. Error bars indicate standard deviation.

Table 1. Model evaluation throughput speedup obtained with the best parallelization parameters in the multi-threaded-CPU, single-GPU, and multi-GPU configurations in the PC and server with respect to a single-thread CPU configuration

	Multi-threaded CPU	Single GPU	Multi GPU
PC	08.36	186.04	•
Server	15.89	192.67	720.71

Performance of the GRN model inference method parallelization

Using the best parallelization parameters obtained for the model evaluation tests, we quantified the performance obtained when running the proposed inference methodology for mechanistic spatiotemporal GRN models. The input to the algorithm was the initial gene expression pattern (a square of 2 by 2 cells at the center of the domain) and the target gene expression pattern (a circle centered in the domain), as shown in Figure 1. We tested the performance of the proposed method in terms of average generation throughput and total execution time to find an optimal model. All runs resulted in a model that could successfully develop a stable gene expression pattern forming the target circle shape. Figure 6 shows the average evolutionary dynamics of the 10 independent runs obtained with the multi-GPU configuration in the server. The results demonstrate how the evolutionary algorithm consistently converges to a GRN model with zero error.

Figure 7 shows the generation throughput and execution time of the inference methodology obtained with a single- and multi-threaded CPU, and single-GPU configuration in both the PC and server as well as a multi-GPU configuration in the server. In addition, Table 3 summarizes the numerical average speedups obtained with respect to the single-CPU-thread configuration. The results demonstrate a speedup in the inference methodology of 204× using a single-GPU PC and of 706× using a multi-GPU server. These speedup improvements result in a decrease in runtime to infer a mechanistic GRN model from about 135 h when using a single-thread configuration to about 10 min when using a multi-GPU server configuration with the proposed methodology. Finally, these speedup results were statistically validated with the same approach used for the model evaluation performance tests. Table 4 shows the ranking resulting from the nonparametric Friedman test and its significance with the Holm test, confirming the ranking of all the configurations and speedups shown in Figure 7 as statistically significant.

DISCUSSION AND CONCLUSION

In order to accelerate the reverse engineering and design of mechanistic spatiotemporal GRNs that can recapitulate a given gene expression pattern in a multicellular domain, we proposed here a methodology combining evolutionary computation and GPU computing. The approach exploits the inherent parallelism of population-based heuristic optimization algorithms and matches it with the massive parallelism of GPUs. Candidate GRNs defined as PDE systems are iteratively generated and selected using the CPU and then distributed asynchronously and in parallel to a single or multiple GPUs. The GPUs then numerically simulate the GRN models and compute their error with a multi-GPU-thread parallel algorithm. In this way, the method can infer both the topology and parameters of the genetic regulatory network as well as the number of genes needed to obtain a stable target gene expression pattern. In this work, we optimized the parallelization parameters of the methodology in both a desktop PC configuration with a single GPU and a server configuration using one or four GPUs. The results showed a speedup improvement of 706× and 204× in the multi-GPU server and single-GPU PC, respectively, when compared with a single-thread implementation.

Parallel computing with GPUs offers a powerful and cost-effective solution for the reverse engineering of mechanistic GRNs. GPUs exploit a single-instruction multiple-data design to obtain massive computational power for a reduced cost. This makes GPUs well-suited for simulating and evaluating in parallel a large set of spatiotemporal mechanistic GRN models defined with PDE systems. Implementing this approach with the parallelization strategy presented here results in excellent speedups, which are independent on the particular solver used or GRN model design. Furthermore, integrating this strategy with a reverse engineering methodology can then result in an efficient machine learning approach for the inference of mechanistic models from target spatial phenotypes. The proposed methodology can be easily adapted to other PDE solvers and machine learning problems.

Future work will exploit the proposed methodology to reverse engineer mechanistic GRNs for complex gene expression patterns. These will include natural expression patterns observed during the development of model organisms, as well as novel patterns for synthetic biology applications. A limitation of the presented inference methodology is that it discovers a single GRN model for each run of the algorithm. However, multiple

Table 2. Nonparametric statistical tests of the model evaluation performance for the five parallel configuration strategies tested

Friedman test	
Configurations	Friedman rank
Multi-threaded CPU PC	5
Multi-threaded CPU Server	4
Single-GPU PC	3
Single-GPU Server	2
Multi-GPU Server	**1**

Holm post hoc procedure (comparing multi-GPU Server with other configurations)

i	Configurations	z-Value	P-Value	Holm	Hypothesis
4	Multi-threaded-CPU PC	5.656854	0.000000	0.0125	Rejected
3	Multi-threaded-CPU Server	4.242641	0.000022	0.0167	Rejected
2	Single-GPU PC	2.828427	0.004678	0.0250	Rejected
1	Single-GPU Server	1.414214	0.157299	0.0500	Rejected

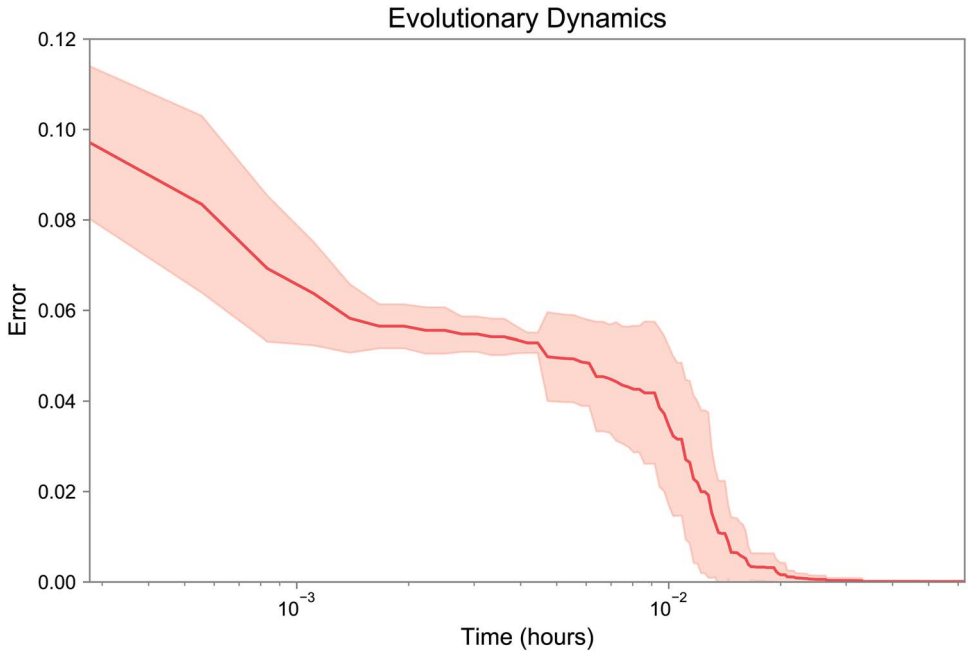

Figure 6. Average error of the best model in the population during 10 different runs of the proposed methodology for the inference of dynamic spatial GRN models. The 10 runs resulted in zero error, validating the convergence of the algorithm. All runs used 256 kernel-calling CPU threads and a block size of 1024 GPU threads in a server with four GPUs. The shaded area indicates standard deviation.

Table 3. Inference methodology speedup obtained with the best parallelization parameters in the multi-threaded-CPU, single-GPU, and multi-GPU configurations in the PC and server with respect to a single-thread CPU configuration

	Multi-threaded CPU	Single GPU	Multi GPU
PC	10.06	204.66	•
Server	23.93	216.66	706.40

GRNs can indeed result in developing the same pattern. Indeed, finding a comprehensive set of GRNs producing a given gene expression pattern—an atlas of GRN designs—is an outstanding challenge for complex phenotypes [70]. Crucially, the presented methodology can be easily extended with multiobjective selection methods that optimize not only the error but also the diversity of the population [71]. This approach could streamline the discovery of GRN design spaces that have either evolved in nature or could be implemented in synthetic biology applications [72].

In conclusion, the increase in performance demonstrated with the presented methodology will pave the way for the inference of mechanistic GRNs that can recapitulate complex spatiotemporal phenotypes. The simulation of PDE-based models of tissue spatial behaviors [73] and whole-body regulatory mechanisms [74] require computationally intensive numerical simulations that represent a bottleneck for the application of machine learning methodologies. Parallelizing and improving the performance of these simulations using GPU computing together with their integration into heuristic machine learning methods with the approach we presented here can lead to the extraction of mechanistic knowledge directly from curated spatial gene expression pattern experimental datasets [75]. Furthermore, these advancements can also streamline the automatic

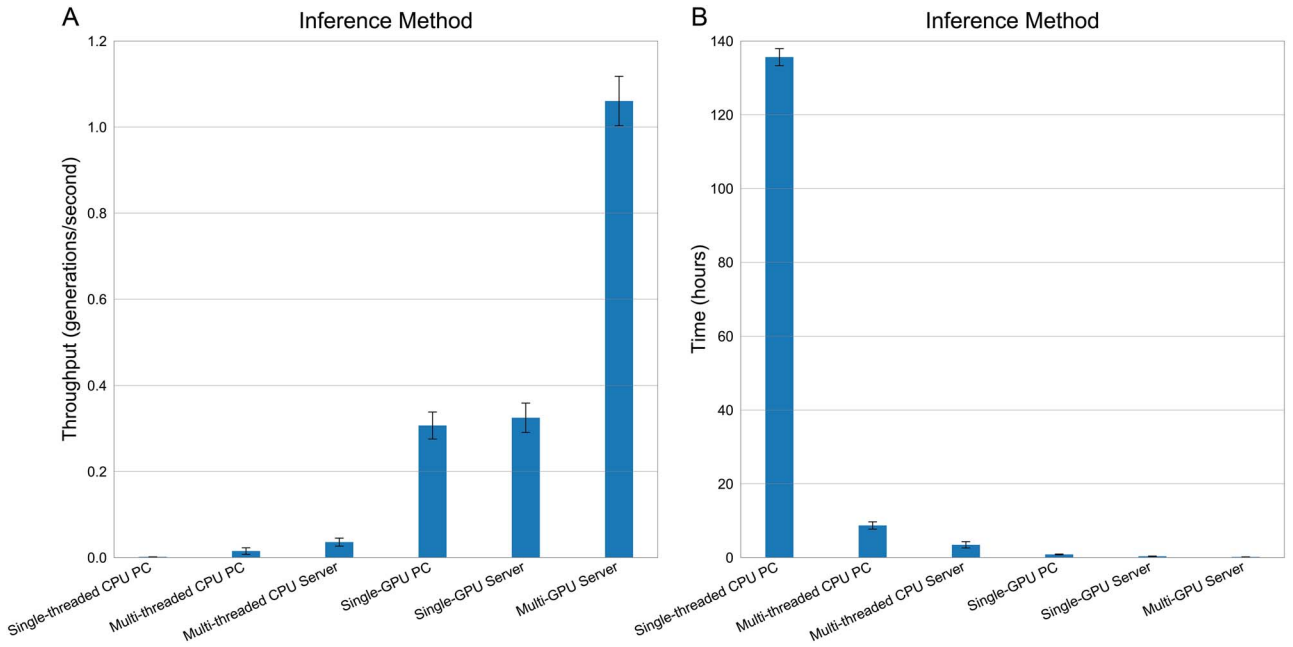

Figure 7. Average throughput (**A**) and execution time (**B**) of the proposed methodology for the inference of dynamic spatial GRN models with the best configuration for the different implementations. The complete reverse engineering algorithm was tested with the best configurations for model evaluations using a single- or multi-threaded CPU implementation or single- or multi-GPU implementation on either a PC or a server. Each condition was run 10 times with different random seeds. Error bars indicate standard deviation.

Table 4. Nonparametric statistical tests of the inference methodology performance for the five parallel configuration strategies tested

Friedman test	
Configurations	Friedman rank
Multi-threaded-CPU PC	4.9
Multi-threaded-CPU Server	4.1
Single-GPU PC	2.7
Single-GPU Server	2.3
Multi-GPU Server	**1.0**

Holm post hoc procedure (comparing multi-GPU Server with other configurations)					
i	**Configurations**	**z-Value**	**P-Value**	**Holm**	**Hypothesis**
4	Multi-threaded-CPU PC	5.515433	0.000000	0.0125	Rejected
3	Multi-threaded-CPU Server	4.384062	0.000012	0.0167	Rejected
2	Single-GPU PC	2.404163	0.016210	0.0250	Rejected
1	Single-GPU Server	1.838478	0.065992	0.0500	Rejected

design and optimization of novel synthetic biology gene circuits for bioproduct production in industrial, pharmaceutical and biomaterial applications [76].

Data Availability

The source code for the implementation of the presented methodology is available at https://github.com/lobolab/grn-gpu-performance.

Key Points

- We presented a methodology to combine evolutionary computation with GPU computing for the efficient inference of mechanistic gene regulatory networks that can develop a given spatial gene expression pattern.
- We evaluated the performance of the methodology, resulting in a speedup of 200× in a desktop computer with a single GPU and of 700× in a server computer with four GPUs.
- This approach can streamline the understanding of biological and medical gene expression pattern datasets and accelerate the design and optimization of GRNs for synthetic biology applications.

Acknowledgments

We thank the members of the Lobo Lab for helpful discussions.

Funding

This work was supported by the National Science Foundation (NSF) under grant IIS-1566077 and the PhRMA Foundation with a Research Starter Grant. Computations used equipment donated by NVIDIA Corporation and the UMBC High Performance Computing Facility (HPCF) supported by the NSF MRI program grants OAC-1726023, CNS-0821258 and CNS-1228778 and the SCREMS program grant DMS-0821311.

References

1. Lobo D, Lobikin M, Levin M. Discovering novel phenotypes with automatically inferred dynamic models: a partial melanocyte conversion in Xenopus. *Sci Rep* 2017;**7**:41339.
2. Kim H, Jin X, Glass DS, *et al*. Engineering and modeling of multicellular morphologies and patterns. *Curr Opin Genet Dev* 2020;**63**:95–102.
3. Delgado FM, Gómez-Vela F. Computational methods for gene regulatory networks reconstruction and analysis: a review. *Artif Intell Med* 2019;**95**:133–45.
4. Huynh-Thu VA, Irrthum A, Wehenkel L, *et al*. Inferring regulatory networks from expression data using tree-based methods. *PLoS One* 2010;**5**:e12776.
5. Aibar S, González-Blas CB, Moerman T, *et al*. SCENIC: single-cell regulatory network inference and clustering. *Nat Methods* 2017;**14**:1083–6.
6. Huynh-Thu VA, Sanguinetti G. Combining tree-based and dynamical systems for the inference of gene regulatory networks. *Bioinformatics* 2015;**31**:1614–22.
7. Zhou X, Cai X. Inference of differential gene regulatory networks based on gene expression and genetic perturbation data. *Bioinformatics* 2020;**36**:197–204.
8. Yin L, Huang C-H, Ni J. Clustering of gene expression data: performance and similarity analysis. *BMC Bioinformatics* 2006;**7**:S19.
9. Faith JJ, Hayete B, Thaden JT, *et al*. Large-scale mapping and validation of *Escherichia coli* transcriptional regulation from a compendium of expression profiles. *PLoS Biol* 2007;**5**:e8.
10. Margolin AA, Nemenman I, Basso K, *et al*. ARACNE: an algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC Bioinformatics* 2006;**7**(Suppl 1):S7.
11. Petralia F, Wang P, Yang J, *et al*. Integrative random forest for gene regulatory network inference. *Bioinformatics* 2015;**31**:i197–205.
12. Meyer PE, Kontos K, Lafitte F, *et al*. Information-theoretic inference of large transcriptional regulatory networks. EURASIP. *J Bioinforma Syst Biol* 2007;**2007**:1–9.
13. Haury A-C, Mordelet F, Vera-Licona P, *et al*. TIGRESS: trustful inference of gene regulation using stability selection. *BMC Syst Biol* 2012;**6**:145.
14. Mordelet F, Vert J-P. SIRENE: supervised inference of regulatory networks. *Bioinformatics* 2008;**24**:i76–82.
15. Gillani Z, Akash MSH, Rahaman MM, *et al*. CompareSVM: supervised, support vector machine (SVM) inference of gene regularity networks. *BMC Bioinformatics* 2014;**15**:395.
16. Razaghi-Moghadam Z, Nikoloski Z. Supervised learning of gene-regulatory networks based on graph distance profiles of transcriptomics data. *npj Syst Biol Appl* 2020;**6**:21.
17. Maetschke SR, Madhamshettiwar PB, Davis MJ, *et al*. Supervised, semi-supervised and unsupervised inference of gene regulatory networks. *Brief Bioinform* 2014;**15**:195–211.
18. Yang Y, Fang Q, Shen H-B. Predicting gene regulatory interactions based on spatial gene expression data and deep learning. *PLoS Comput Biol* 2019;**15**:e1007324.
19. Wu S, Joseph A, Hammonds AS, *et al*. Stability-driven nonnegative matrix factorization to interpret spatial gene expression and build local gene networks. *Proc Natl Acad Sci USA* 2016;**113**:4290–5.
20. Sîrbu A, Ruskin HJ, Crane M. Comparison of evolutionary algorithms in gene regulatory network model inference. *BMC Bioinformatics* 2010;**11**:59.
21. Mousavi R, Eftekhari M. A new ensemble learning methodology based on hybridization of classifier ensemble selection approaches. *Appl Soft Comput* 2015;**37**:652–66.
22. Mousavi R, Eftekhari M, Haghighi MG. A new approach to human microRNA target prediction using ensemble pruning and rotation forest. *J Bioinform Comput Biol* 2015;**13**:1550017.
23. Reali F, Priami C, Marchetti L. Optimization algorithms for computational systems biology. *Front Appl Math Stat* 2017;**3**.
24. Wong K-C. Evolutionary algorithms: concepts, designs, and applications in bioinformatics. *Nature-Inspired Comput* 2016;111–37.
25. Mousavi R, Eftekhari M, Rahdari F. Omni-ensemble learning (OEL): utilizing over-bagging, static and dynamic ensemble selection approaches for software defect prediction. *Int J Artif Intell Tools* 2018;**27**(1–37):1850024.
26. Slowik A, Kwasnicka H. Evolutionary algorithms and their applications to engineering problems. *Neural Comput Applic* 2020.
27. Timmermann T, González B, Ruz GA. Reconstruction of a gene regulatory network of the induced systemic resistance defense response in Arabidopsis using boolean networks. *BMC Bioinformatics* 2020;**21**:142.
28. de Luis Balaguer MA, Fisher AP, Clark NM, *et al*. Predicting gene regulatory networks by combining spatial and temporal gene expression data in Arabidopsis root stem cells. *Proc Natl Acad Sci USA* 2017;**114**:E7632–40.
29. Jaeger J, Blagov M, Kosman D, *et al*. Dynamical analysis of regulatory interactions in the gap gene system of *Drosophila melanogaster*. *Genetics* 2004;**167**:1721–37.
30. Jaeger J, Surkova S, Blagov M, *et al*. Dynamic control of positional information in the early *Drosophila* embryo. *Nature* 2004;**430**:368–71.
31. Verd B, Crombach A, Jaeger J. Dynamic maternal gradients control timing and shift-rates for *Drosophila* gap gene expression. *PLoS Comput Biol* 2017;**13**:e1005285.
32. Francois P, Siggia ED. Predicting embryonic patterning using mutual entropy fitness and in silico evolution. *Development* 2010;**137**:2385–95.
33. Henry A, Hemery M. François P. φ-evo: a program to evolve phenotypic models of biological networks. *PLoS Comput Biol* 2018;**14**:e1006244.
34. Lobo D, Levin M. Inferring regulatory networks from experimental morphological phenotypes: a computational method reverse-engineers planarian regeneration. *PLoS Comput Biol* 2015;**11**:e1004295.
35. Lobo D, Morokuma J, Levin M. Computational discovery and in vivo validation of hnf4 as a regulatory gene in planarian regeneration. *Bioinformatics* 2016;**32**:2681–5.
36. Lobo D, Levin M. Computing a worm: reverse-engineering planarian regeneration. In: *Advances in Unconventional Computing*. **Vol. 2**, Prototypes, Models and Algorithms 2017; 637–54

37. Cecilia JM, Garcia JM, Guerrero GD, *et al*. Simulation of P systems with active membranes on CUDA. *Brief Bioinform* 2010;**11**:313–22.
38. Dematté L, Prandi D. GPU computing for systems biology. *Brief Bioinform* 2010;**11**:323–33.
39. Nobile MS, Cazzaniga P, Tangherloni A, *et al*. Graphics processing units in bioinformatics, computational biology and systems biology. *Brief Bioinform* 2017;**18**:870–85.
40. Abbaszadeh O, Khanteymoori AR, Azarpeyvand A. Parallel algorithms for inferring gene regulatory networks: a review. *Curr Genomics* 2018;**19**:603–14.
41. Biagini T, Chillemi G, Mazzoccoli G, *et al*. Molecular dynamics recipes for genome research. *Brief Bioinform* 2018.
42. Shegay MV, Suplatov DA, Popova NN, *et al*. ParMATT: parallel multiple alignment of protein 3D-structures with translations and twists for distributed-memory systems. *Bioinformatics* 2019;**35**:4456–8.
43. Zhu M, Kang K, Meta-Prism NK. Ultra-fast and highly accurate microbial community structure search utilizing dual indexing and parallel computation. *Brief Bioinform* 2020;1–11.
44. Cano A. A survey on graphic processing unit computing for large-scale data mining. In: *WIREs Data Mining and Knowledge Discovery*, Vol. **Vol. 8**, 2018.
45. Jian L, Wang C, Liu Y, *et al*. Parallel data mining techniques on graphics processing unit with compute unified device architecture (CUDA). *J Supercomput* 2013;**64**:942–67.
46. Orzechowski P, Moore JH. EBIC: an open source software for high-dimensional and big data analyses. *Bioinformatics* 2019;**35**:3181–3.
47. da Cruz MHP, Domingues DS, Saito PTM, *et al*. TERL: classification of transposable elements by convolutional neural networks. *Brief Bioinform* 2020.
48. Eklund A, Dufort P, Forsberg D, *et al*. Medical image processing on the GPU—past, present and future. *Med Image Anal* 2013;**17**:1073–94.
49. Wait E, Winter M, Cohen AR, *et al*. Hydra image processor: 5-D GPU image analysis library with MATLAB and python wrappers. *Bioinformatics* 2019;**35**:5393–5.
50. Banzhaf W, Harding S, Langdon WB, *et al*. Accelerating genetic programming through graphics processing units. *Genet. Program. Theory Pract. VI* 2008;1–19.
51. Langdon WB. Graphics processing units and genetic programming: an overview. *Soft Comput* 2011;**15**:1657–69.
52. Chitty DM. Improving the performance of GPU-based genetic programming through exploitation of on-chip memory. *Soft Comput* 2016;**20**:661–80.
53. Langdon WB, Harrison AP. GP on SPMD parallel graphics hardware for mega bioinformatics data mining. *Soft Comput* 2008;**12**:1169–83.
54. Pospichal P, Jaros J, Schwarz J. Parallel genetic algorithm on the CUDA architecture. *Eur Conf Appl Evol Comput* 2010; 442–51.
55. Richmond P, Walker D, Coakley S, *et al*. High performance cellular level agent-based simulation with FLAME for the GPU. *Brief Bioinform* 2010;**11**:334–47.
56. Germann P, Marin-Riera M, Yaa SJ. GPU-powered spheroid models for mesenchyme and epithelium. *Cell Syst* 2019;**8**:261–266.e3.
57. Christley S, Lee B, Dai X, *et al*. Integrative multicellular biological modeling: a case study of 3D epidermal development using GPU algorithms. *BMC Syst Biol* 2010;**4**.
58. Whitley D, Rana S, Heckendorn RB. The island model genetic algorithm: on separability, population size and convergence. *J Comput Inf Technol* 1999;**7**:33–47.
59. Junion G, Spivakov M, Girardot C, *et al*. A transcription factor collective defines cardiac cell fate and reflects lineage history. *Cell* 2012;**148**:473–86.
60. Potier D, Davie K, Hulselmans G, *et al*. Mapping gene regulatory networks in *Drosophila* eye development by large-scale transcriptome perturbations and motif inference. *Cell Rep* 2014;**9**:2290–303.
61. MacArthur S, Li X-Y, Li J, *et al*. Developmental roles of 21 *Drosophila* transcription factors are determined by quantitative differences in binding to an overlapping set of thousands of genomic regions. *Genome Biol* 2009;**10**:R80.
62. Sandmann T, Girardot C, Brehme M, *et al*. A core transcriptional network for early mesoderm development in *Drosophila melanogaster*. *Genes Dev* 2007.
63. Santos-Moreno J, Schaerli Y. Using synthetic biology to engineer spatial patterns. *Adv Biosyst* 2019;**3**:1–15.
64. Press W, Flannery B, Teukolsky S, *et al*. *Numerical Recipes* 1986.
65. Luke S, Partait L. A comparison of bloat control methods for genetic programming. *Evol Comput* 2006;**14**:309–44.
66. Mahfoud SW. Crowding and preselection revisited. *Parallel Probl Solving from Nat* 1992;**2**:27–36.
67. Van Werkhoven B, Maassen J, Seinstra FJ, *et al*. Performance models for CPU-GPU data transfers. In: *Proc. 14th IEEE/ACM Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2014*, 2014.
68. Borelli FF, de Camargo RY, Martins DC, *et al*. Gene regulatory networks inference using a multi-GPU exhaustive search algorithm. *BMC Bioinformatics* 2013;**14**.
69. Settouti N, Bechar MEA, Chikh MA. Statistical comparisons of the top 10 algorithms in data mining for classification task. *Int J Interact Multimed Artif Intell* 2016;**4**:46–51.
70. Cotterell J, Sharpe J. An atlas of gene regulatory networks reveals multiple three-gene mechanisms for interpreting morphogen gradients. *Mol Syst Biol* 2010;**6**:425.
71. Liu HL, Chen L, Deb K, *et al*. Investigating the effect of imbalance between convergence and diversity in evolutionary multiobjective algorithms. *IEEE Trans Evol Comput* 2017;**21**:408–25.
72. Schaerli Y, Munteanu A, Gili M, *et al*. A unified design space of synthetic stripe-forming networks. *Nat Commun* 2014.
73. Ko JM, Lobo D. Continuous dynamic modeling of regulated cell adhesion: sorting, intercalation, and involution. *Biophys J* 2019;**117**:2166–79.
74. Herath S, Lobo D. Cross-inhibition of Turing patterns explains the self-organized regulatory mechanism of planarian fission. *J Theor Biol* 2020;**485**:110042.
75. Roy J, Cheung E, Bhatti J, *et al*. Curation and annotation of planarian gene expression patterns with segmented reference morphologies. *Bioinformatics* 2020;**36**:2881–7.
76. Hwang J, Hari A, Cheng R, *et al*. Kinetic modeling of microbial growth, enzyme activity, and gene deletions: an integrated model of β-glucosidase function in Cellvibrio japonicus. *Biotechnol Bioeng* 2020;bit.27544.