Fundamenta Informaticae XXI (2001) 1001–1014
DOI 10.3233/FI-2012-0000
IOS Press

Efficient simulation of reaction systems on Graphics Processing Units

Marco S. Nobile [†,‡], Antonio E. Porreca [†], Simone Spolaor [†], Luca Manzoni [†],

Paolo Cazzaniga [§,‡], Giancarlo Mauri [†,‡], Daniela Besozzi [†]

[†]*University of Milano-Bicocca – Department of Informatics, Systems and Communication*
Viale Sarca 336, 20126 Milano, Italy
[‡]*SYSBIO.IT Centre of Systems Biology – 20126 Milano, Italy*
[§]*University of Bergamo – Department of Human and Social Sciences*
Piazzale S. Agostino 2, 24129 Bergamo, Italy

Abstract. Reaction systems represent a theoretical framework based on the regulation mechanisms of facilitation and inhibition of biochemical reactions. The dynamic process defined by a reaction system is typically derived by hand, starting from the set of reactions and a given context sequence. However, this procedure may be error-prone and time-consuming, especially when the size of the reaction system increases. Here we present HERESY, a simulator of reaction systems accelerated on Graphics Processing Units (GPUs). HERESY is based on a fine-grained parallelization strategy, whereby all reactions are simultaneously executed on the GPU, therefore reducing the overall running time of the simulation. HERESY is particularly advantageous for the simulation of large-scale reaction systems, consisting of hundreds or thousands of reactions. By considering as test case some reaction systems with an increasing number of reactions and entities, as well as an increasing number of entities per reaction, we show that HERESY allows up to $29\times$ speed-up with respect to a CPU-based simulator of reaction systems. Finally, we provide some directions for the optimization of HERESY, considering minimal reaction systems in normal form.

Keywords: reaction systems, general-purpose GPU computing, high-performance computing, simulation

1. Introduction

The last decades have been characterized by an ever increasing synergy between computer science and biology. Disciplines like bioinformatics, computational biology and systems biology nowadays provide effective tools to manipulate the huge amount of biological data generated via laboratory experiments, to retrieve useful information and knowledge from them, and to predict the behavior of complex biological systems under physiological or perturbed conditions. On the other hand, in the field of computer science, novel algorithms and innovative models of computations – e.g., neural networks [4], evolutionary programming [21], molecular computing [10], cellular automata [27], membrane systems [26] – have frequently taken inspiration from biological mechanisms and processes with the aim of solving computationally difficult problems. In this scenario, reaction systems were introduced in [15, 14] as a theoretical framework based on the working principles of biochemical reactions, which can be assumed to represent the "primitive" information processing events that occur in living cells. In particular, reaction systems consider as fundamental the regulation mechanisms of facilitation and inhibition of reactions. The interest of the scientific community in reaction systems has been growing in various research directions, as demonstrated by the recent literature [9, 7, 22, 13, 12, 8, 11].

Reaction systems are based on two main assumptions concerning the chemistry of the cell. First, the *threshold supply* of molecules implies that either an element is present and its amount is sufficient to allow the execution of the reactions, or an element is not present and no reactions having it as reagent can occur. Second, the *no permanency* assumption of elements states that if an element is not sustained by any reaction, then it ceases to exist. In other terms, reaction systems rely on a qualitative rather than quantitative description of reactions. This basic definition of reaction systems can be extended, for instance by assigning specific measurement functions to the states, in order to add quantitative parameters such as time values, molecular amounts or mass [16].

Despite its high level of abstraction in representing biochemical reactions, the formalism exploited by reaction systems allows to investigate how the state of a given system changes over discrete time steps by considering the execution of a set of reactions. The evolution of a reaction system is regulated by a core deterministic assumption, since all reactions that can take place in the system are actually applied regardless of any possible concurrency on the availability of elements (that is, no conflicts are assumed to exist between the reactions that can take place in a given state). Therefore, the next state is univocally determined by the current state of the system, and the dynamic behavior of a reaction system can be derived by considering the simultaneous application of reactions. Reaction systems differ from other abstract formalisms of natural computing, since they do not work in a well defined or structured environment, but rather they *create* their own environment of interactive processes. A significant notion for reaction systems is the context sequence, which represents the input that the system can receive from outside. So doing, new entities can be introduced in any current state of the reaction system, therefore facilitating or inhibiting its reactions, and possibly changing its dynamics.

The dynamic process defined by a reaction system is typically derived by hand, starting from the set of reactions and a given context sequence. However, this procedure may be error-prone and can be lengthy, notably as the size – that is, the number of reactions and entities – of the reaction system increases. To overcome these limitations and automate this procedure, in this work we present HERESY (Highly Efficient REaction SYstem simulator), a novel software tool accelerated on Graphics Processing Units (GPUs) that allows to run simulations of dynamics processes, especially in the presence of context sequences. GPUs are gaining an ever increasing attention by the scientific community, mainly in life sciences disciplines [25], as they can considerably reduce the running time compared to standard CPU-based software, thanks to their massively multi-core architecture.

HERESY is able to accelerate the simulation of reaction systems by automatically switching from a CPU-based to a GPU-based implementation in case of large-scale reaction systems, consisting of hundreds or thousands of reactions. HERESY was specifically designed to distribute the calculations of a single simulation on the cores of the GPU, when required; otherwise, the simulation is executed on the CPU. To analyze the computational performance of HERESY, in this work we evaluate the speed-up achieved on GPUs by simulating some randomly generated reaction systems of increasing size, as well as a large-scale reaction system describing the signaling pathway of the ErbB receptor family [20].

The paper is structured as follows. In Section 2 we recall the basic notions of reaction systems, GPGPU computing and the CUDA architecture. We describe the fine-grained GPU-based simulator of reaction systems in Section 3, and present the experimental results on its computational performance in Section 4. We conclude with some final remarks and directions for future developments of this work in Section 5.

2. Prerequisites

2.1. Basic notions on reaction systems

We recall the basic notions of reaction systems from the literature [15, 7].

Let S be a finite set of *entities*, called the *background set*. The entities represent the objects of interest in the modeled system, e.g., a set of molecular species with mutual interactions. A *reaction* over the background set S is a triple $a = (R_a, I_a, P_a)$ of (finite) subsets of S with $R_a \cap I_a = \varnothing$. The elements of R_a are called the *reactants* of a, while I_a is the set of *inhibitors* of a and P_a the set of *products* of a. In the literature it is sometimes required for R, S, and P to be non-empty. A *reaction system* is a pair $\mathcal{A} = (S, A)$, where S is a background set and A is a (finite) set of reactions over S.

A *state* of the reaction system is represented by any subset $T \subseteq S$ of entities. We say that a reaction $a = (R_a, I_a, P_a)$ is *enabled* in state T if $R_a \subseteq T$ and $I_a \cap T = \varnothing$, i.e., if all the reactants but none of the inhibitors are present in T. The *result* of reaction a in state T is its set of products when a is enabled, and the empty set otherwise. This defines a *result function* $\mathrm{res}_a \colon 2^S \to 2^S$ over the subsets of S:

$$\mathrm{res}_a(T) = \begin{cases} P_a & \text{if } R_a \subseteq T \text{ and } I_a \cap T = \varnothing, \\ \varnothing & \text{otherwise.} \end{cases}$$

A set of enabled reactions never compete for the entities in the current state: the amount of entities is assumed to be either sufficient for all reactions having them as reactants to be triggered simultaneously, or for none. Thus, given a set of reactions A over S, the result function $\mathrm{res}_A \colon 2^S \to 2^S$ is simply given by the union of the results of the individual reactions:

$$\mathrm{res}_A(T) = \bigcup_{a \in A} \mathrm{res}_a(T). \tag{1}$$

The result function $\mathrm{res}_{\mathcal{A}} \colon 2^S \to 2^S$ for the whole reaction system $\mathcal{A}$ is defined as the result function of its set of reactions A, i.e., $\mathrm{res}_{\mathcal{A}} = \mathrm{res}_A$.

Given a reaction system $\mathcal{A} = (S, A)$ and an initial state $T_0 \subseteq S$, we are interested in analysing the dynamics generated by its result function, that is, the sequence of states $(T_0, T_1, T_2, \ldots)$ – or any of its finite prefixes – where $T_n = \mathrm{res}^n_{\mathcal{A}}(T_0)$ for any $n \geq 0$. Notice that, according to Equation (1), each state $T_n,\ n \geq 1$, is obtained by collecting the products of the reactions enabled in its predecessor state T_{n-1}. This highlights the lack of permanence in reaction systems: if an entity in state $T_n, n \geq 0$, is not produced again by a reaction enabled in T_n, then it disappears. Stated otherwise, the lack of permanence mimics the degradation of entities that are not supported by any reaction.

Of particular interest is the analysis of the dynamics of reaction systems with input, where each non-initial state consists of the result function applied to the previous state *plus* a set of extra entities provided from outside. In this way we obtain a discrete time *dynamic process* [15], defined as a pair $\pi = (\gamma, \delta)$ of finite sequences $\gamma = (C_0, \ldots, C_n)$ and $\delta = (D_0, \ldots, D_n)$, where $C_0, \ldots, C_n, D_0, \ldots, D_n$ are subsets of S such that $D_0 = \varnothing$ and $D_{i+1} = \mathrm{res}_{\mathcal{A}}(C_i \cup D_i)$ for $0 \leq i < n$. The sequence $\gamma = (C_0, \ldots, C_n)$ is the *context sequence* of π and includes the initial state and the aforementioned external input, while $\delta = (D_0, \ldots, D_n)$ is called the *result sequence*. The sequence $(W_0, \ldots, W_n)$, where $W_i = C_i \cup D_i$, is called the *state sequence* of π.

A problem of relevance, for instance in the modeling and analysis of biological systems, is to check whether a given reaction system satisfies certain dynamical properties, for instance the reachability of a certain state from a given initial condition, the existence of attractors such as fixed points or cycles, the establishment of oscillatory regimes, etc. More generally, we might be interested in model checking of properties expressed, for instance, in a formal logic [18, 17, 24]. Unfortunately, it turns out that checking the majority of interesting properties, including all those mentioned here, is intractable, e.g., $\mathbf{NP}$-complete or even $\mathbf{PSPACE}$-complete [15, 18, 17, 24, 5].

2.2. GPGPU computing and Nvidia CUDA

Multi-core Graphics Processing Units (GPUs) are the most pervasive and cheap architectures nowadays available for parallel computation. GPUs are devoted to massive multi-threaded execution, and can be exploited for *general-purpose* computational tasks (the so-called GPGPU computing). Among the various libraries developed for GPGPU computing, in this work we consider Nvidia's CUDA (Compute Unified Device Architecture), which combines the *single instruction multiple data* (SIMD) architecture with *multi-threading*. This specific paradigm leverages the architectural characteristics of Nvidia GPUs, which are based on multiple multi-core streaming multi-processors (SMs).

The main concept of CUDA is the *kernel*, namely a C/C++ function loaded from the host (the CPU) to the devices (one or more GPUs). A kernel applies the SIMD paradigm: it is replicated in many copies named *threads*, all working on different input data. Threads must be organized in three-dimensional structures named *blocks* which, in turn, must be organized in three-dimensional *grids*. CUDA automatically schedules the blocks on the available SMs, allowing the transparent scaling of performances on different GPUs. During normal execution, groups of 32 threads (named *warps*) are executed in lock-step according to a strict SIMD fashion. However, threads belonging to the same warp are allowed to take divergent paths (due to conditional branches). The drawback of this flexibility is that the execution of the warp is serialized until convergence, thus affecting the overall performances of the task execution.

In CUDA, threads can access data from multiple kind of memories, characterized by different features (see Figure 1). The main type is the global memory: it can be read and written by both the CPU and the threads on the GPU, hence representing the main interface between the two architectures. The shared memory can be accessed by threads belonging to the same block, allowing intra-block communication. The constant memory is a special cached and read-only memory that can be written by the host and read by the devices. Finally, registers are private memories, whose access is reserved to the threads.

Besides the different visibilities, the aforementioned memories are characterized by different sizes and latencies. The global memory, for instance, is usually very large (thousands of MBs) but suffers from

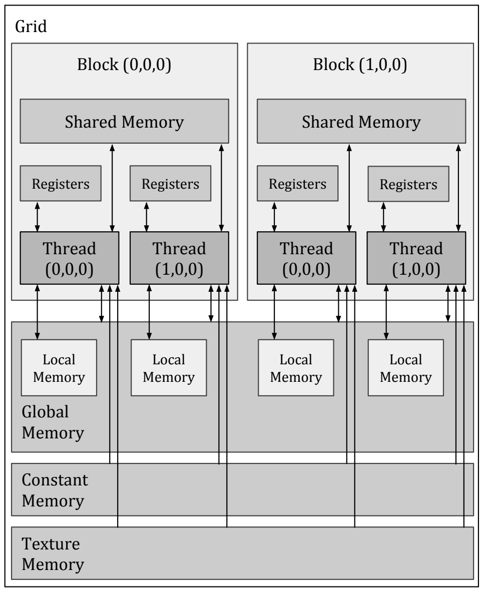


Figure 1. CUDA relies on a memory hierarchy consisting in different memories with different scopes. Registers and local memories are private memories accessible by threads. The low access latency shared memory is used by threads belonging to the same block to communicate. The high access latency global memory is accessible by all threads and it is cached since the introduction of the Fermi architecture. The texture and constant memories are equipped with a cache as well, and all threads can read from these two memories.

high access latencies, whereas the shared memory is faster but much smaller (about one hundred KBs for each SM). To obtain the best performances, high performance memories must be exploited as much as possible; however, being a very limited resource on each SM, they pose limitations on the blocks size.

Due to this complex hierarchy of memories, the existence of different libraries and the use of SIMD paradigm, GPU programming generally represents a challenging task. Indeed, algorithms need to be redesigned to be run on GPUs and any given CPU code needs to be rewritten for GPUs. In what follows, we present a novel GPU-powered simulator based on Nvidia's CUDA that – according to the size of the reaction system – can automatically switch from CPU to GPU execution to the aim of reducing the running time required to generate the dynamic process of the reaction system.

3. HERESY: Highly Efficient REaction SYstem simulator

The Highly Efficient REaction SYstems simulator (HERESY) [3] we propose here allows to automate the derivation of the dynamic process induced by a reaction system. HERESY is characterized by a user-friendly Graphical User Interface (GUI) which allows to enter and test arbitrary reaction systems.

Differently from the existing simulator brsim [6, 2, 1], HERESY is able to accelerate the simulation of reaction systems by automatically switching from a CPU-based to a GPU-based engine. As a matter of fact, small-scale reaction systems characterized by tens of reactions can be efficiently simulated by using the CPU-based engine provided by HERESY, consisting in a novel Python implementation. On the other hand, medium- and large-scale reaction systems, characterized by hundreds or thousands of reactions, can be simulated by HERESY exploiting the GPU-based engine written in CUDA.

A bird's eye view of HERESY's GPU-based engine is shown in Figure 2. Its functioning can be described by the following workflow:

- the simulator starts by loading the set of reactions, the set of contexts and the initial set of entities (*phase A*). The iteration counter $step$ is initialized to 0;
- an empty state T' is created and the entities belonging to the $step$-th context set are added to the system (*phase B*). $|S|$ independent GPU threads are used to distribute the required memory updates across the $|S|$ entities;
- all reactions in A are evaluated and T' is populated with $\text{res}_{\mathcal{A}}(T)$ (*phase C*), using $|A|$ independent GPU threads (i.e., one for each reaction);
- the state of the system is appended to a trace (*phase D*), using $|S|$ independent GPU threads to distribute the memory updates;
- the $step$ counter is incremented.

The aforementioned process is iterated until the requested number of steps is reached (e.g., there are no more context sets in the given context sequence). Eventually, the trace is read back from the GPU to the host in a single memory transfer and then sent to the GUI for further processing using a pipe (*phase E*). All the operations in the gray boxes of Figure 2 are performed on the host, while the rest is parallelized on the GPU by means of specific kernels, namely: `Context` (*phase B*), `Simulate` (*phase C*) and `SaveTrace` (*phase D*).

To be more specific, HERESY implements the so-called *fine-grained* parallelization strategy, meaning that all calculations required to simulate the dynamic process of a reaction system are distributed over the cores of the GPU. To this aim, in *phase C*, HERESY assigns each reaction $a \in A$ to a specific GPU thread, which evaluates whether a is enabled or not in the current state of the reaction system. By simultaneously evaluating all reactions in A, HERESY strongly reduces the running time of the simulation[1], especially in the case of large-scale systems characterized by thousands of reactions. A different parallelization is exploited during *phase B* and *phase D*, in which HERESY launches $|S|$ GPU threads for the memory update operations. Since HERESY requires several accesses to the global memory to fetch the structure of the reactions, we optimized the execution on the GPU by storing this information into the constant memory, a solution that strongly reduces the latencies, thanks to the L1 cache.

Since the contexts require a large amount of GPU's memory, they are allocated in the global memory. Specifically, we use a binary matrix $\mathbf{X} \in \{0, 1\}^{n \times |S|}$, where n is the length of the context sequence (i.e., $n =$MAXSTEPS). An element $x_{ij} \in \mathbf{X}$ is equal to 1 if the j-th entity must be added to T' during the i-th iteration, 0 otherwise. In order to save memory, the matrix is represented on the GPU as an array of `char` data type.

In order to be processed on the GPU, HERESY encodes the set of reactions A as a serialized unidimensional array, where all reactions are concatenated. Each reaction is composed of three sets of unsigned integer numbers, representing the indices of the entities involved in that reaction as reactants,

[1]In principle, if the number of reactions is smaller than the number of available cores, the running time is reduced by a factor of $|A|$.

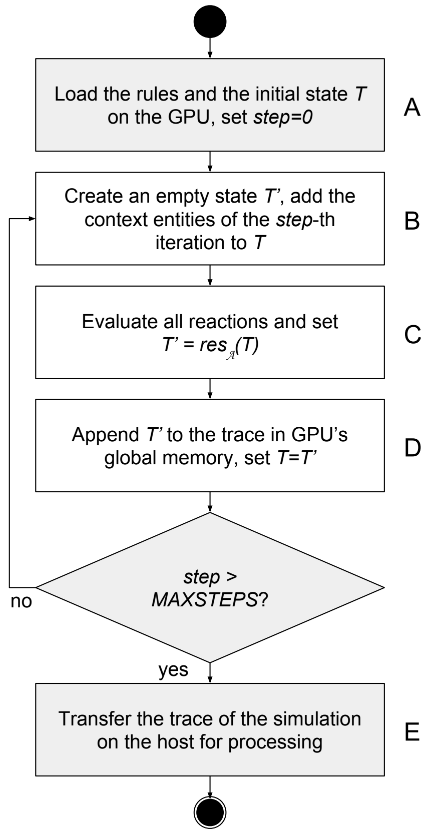

Load the rules and the initial state *T* on the GPU, set *step=0*
A
Create an empty state *T'*, add the context entities of the *step*-th iteration to *T*
B
Evaluate all reactions and set *T' = res*$_{\mathcal{A}}$*(T)*
C
Append *T'* to the trace in GPU's global memory, set *T=T'*
D
step > MAXSTEPS?
no
yes
Transfer the trace of the simulation on the host for processing
E

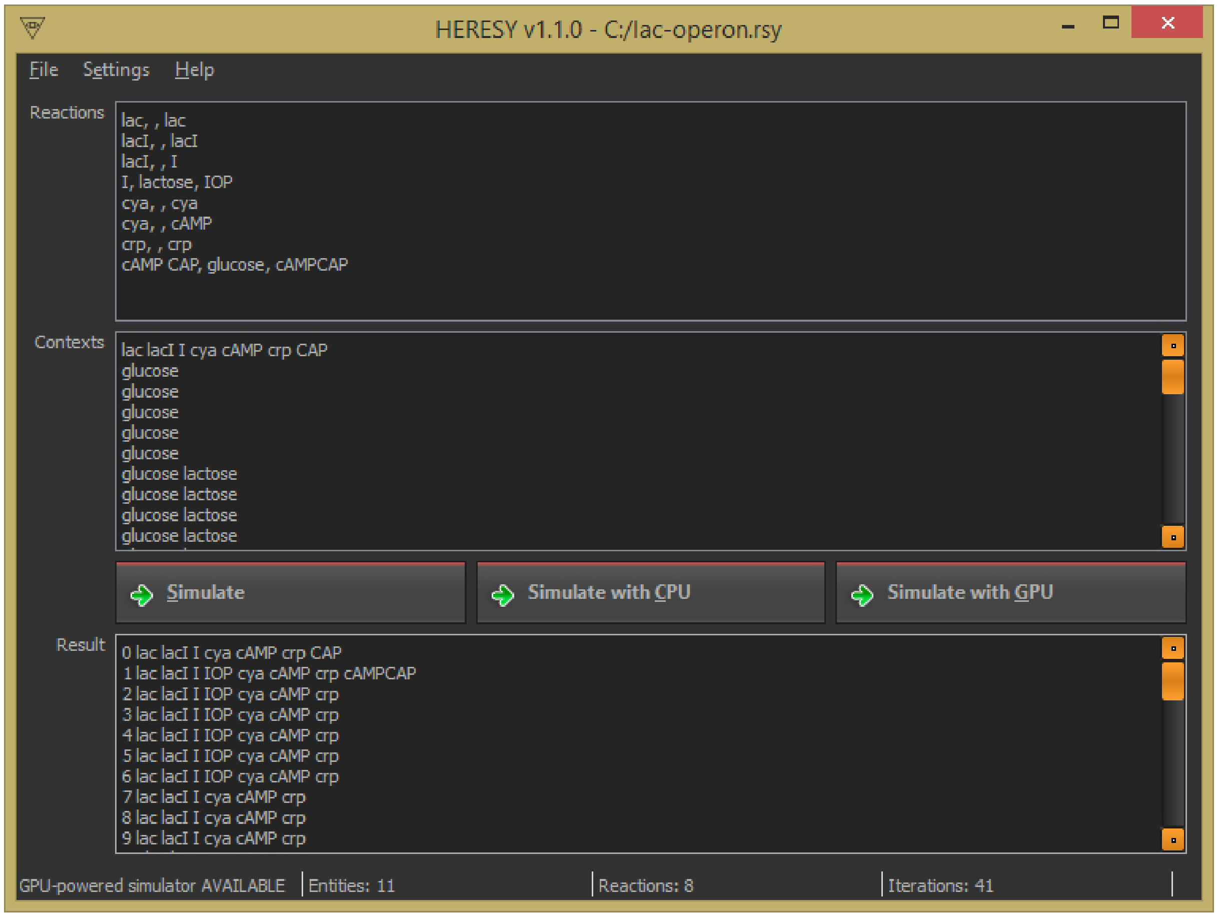
Figure 2. Flow-chart of HERESY functioning. The operations in the gray boxes are performed on the CPU, while the operations in the white boxes are parallelized on the GPU.

inhibitors and products, respectively. The sets are separated using special indices (specifically, the maximum unsigned integer value that can be represented on the machine `UINT_MAX`, `UINT_MAX`-1, and `UINT_MAX`-2)[2]. Using an additional unidimensional offset array, which is also stored in the constant memory, HERESY reconstructs the form of the reactions in each thread.

The output of the simulation is the dynamic process of the given reaction system (i.e., the entities occurring in each state of the resulting state sequence). We highlight that HERESY is able to automatically determine the storage requirements for the whole dynamic process, warning the user in case of a potential memory overflow.

HERESY is provided with a user-friendly GUI, implemented with the PyQT libraries (version 4), which can be exploited by the user to enter the reactions and the context sequence needed to simulate a reaction system (see Figure 3). The reaction system specified by the user by means of the GUI can be saved, in order to be reloaded for further analyses. By pressing the "Simulate" button, the trace of the simulation is calculated and printed in the "Result" box. HERESY will perform a simulation using the

[2]Thus, on 32 bit systems, HERESY cannot simulate reaction systems with more than $2^{32} - 3$ entities.



```
lac, , lac
lacI, , lacI
lacI, , I
I, lactose, IOP
cya, , cya
cya, , cAMP
crp, , crp
cAMP CAP, glucose, cAMPCAP
```

```
lac lacI I cya cAMP crp CAP
glucose
glucose
glucose
glucose
glucose
glucose lactose
glucose lactose
glucose lactose
glucose lactose
```

```
0 lac lacI I cya cAMP crp CAP
1 lac lacI I IOP cya cAMP crp cAMPCAP
2 lac lacI I IOP cya cAMP crp
3 lac lacI I IOP cya cAMP crp
4 lac lacI I IOP cya cAMP crp
5 lac lacI I IOP cya cAMP crp
6 lac lacI I IOP cya cAMP crp
7 lac lacI I cya cAMP crp
8 lac lacI I cya cAMP crp
9 lac lacI I cya cAMP crp
```

GPU-powered simulator AVAILABLE | Entities: 11 | Reactions: 8 | Iterations: 41

Figure 3. The GUI of HERESY.

CPU when $|A| \leq 100$, otherwise it will run on the GPU[3]. To force the execution on the CPU or on the GPU, the user can press the buttons “Simulate with CPU” or “Simulate with GPU”, respectively.

4. Results

In this section we present some experimental results concerning the accuracy and efficiency of HERESY, obtained by simulating both real and synthetic reactions systems, exploiting the CPU-based and the GPU-based simulation methods. In all tests we employed the following hardware: a CPU Intel Core i7-3537U, clock 2.5 GHz; a GPU Nvidia GeForce Titan X, 3584 cores, clock 1417 MHz.

As a first test, we assessed the accuracy of HERESY for the simulation of the dynamic process of the *lac* operon reaction system [11]. To this aim, we provided as input both the reactions and the context sequence given in [11]. As shown in Figure 3, HERESY is able to correctly reproduce the expected dynamic process, as shown in the “Result” box. Afterwards, we performed different tests to investigate

[3]This functionality is automatically enabled as long as the binary CUDA file is contained in the main directory.

the computational performance of HERESY and to compare it to brsim, focusing in particular on the novel GPU implementation presented in this paper and on the simulation speed-up that can be achieved. We consider as test cases some randomly generated reaction systems of increasing size (Section 4.1), and a large-scale reaction system describing the cellular signaling of the ErbB receptors family (Section 4.2).

4.1. Simulation of randomly generated reaction systems

In order to determine the improvement of performances provided by the GPU acceleration, we performed a set of simulations on randomly generated reaction systems. Specifically, we generated reaction systems with an arbitrarily chosen size for the background set and the set of reactions (i.e., $|S| \times |A|$), and a random context sequence with length equal to 1000 (i.e., MAXSTEPS = 1000).

Each reaction was generated as follows. First, three integer numbers r, i, and p were selected according to a binomial distribution $B(|S|, \alpha)$, with $\alpha \in [0, 1]$. The minimum value r' between $r + 1$ and $|S|$ was chosen as the number of reactants; the number of inhibitors was chosen equal to the minimum value between $|S| - r'$ and i, while the number of products was chosen equal to the minimum value between $p + 1$ and $|S|$. Notice that a reaction generated by following this procedure has always at least one reactant and one product, while the set of inhibitors can be empty. This is consistent with the fact that the effect of a reaction with no reactants can be simulated by an appropriate context set, while a reaction without products can be dropped without affecting the dynamics of the system. The generator of reaction systems can be downloaded from [3].

The values of α used to test the performance of HERESY are 0.01, 0.05 and 0.1. The number of reactants, inhibitors and products per reaction is approximately $|S| \times \alpha$, the expected value of the distribution $B(|S|, \alpha)$. The value α was thus used to "control" the length of the reactions in the reaction system, meaning that a higher value of α increases the probability of introducing entities in the reaction.

Table 1 shows a summary of the performed experiments. These results show that the GPU can outperform the CPU in the case of reaction systems having more than a few entities and reactions (Figure 4, gray and black bars), and that brsim is faster than HERESY running on the CPU (Figure 4, hatched and gray bars, respectively). Moreover, the comparison of HERESY with brsim shows that, for both simulators, the running time increases with the size of the reaction system. We highlight that the algorithms used by HERESY and brsim for the CPU simulation are similar, since both are a direct translation of the mathematical definition of the result function of a reaction system. The discrepancy of the CPU execution time can thus be ascribed to the different programming language and the libraries used to implement the algorithm. In particular, brsim is written in Haskell, a language natively compiled, while the CPU-based implementation of HERESY is written in Python, which is interpreted. Finally, since both simulators employ *set* data structures, their running time is greatly affected by the implementations of the operations over this data structure provided by the two languages (i.e., Haskell and Python).

As shown in Table 1 and Figure 4, the highest speed-up value of HERESY running on the GPU is achieved in the case of a reaction system of size $|S| \times |A| = 2000 \times 2000$, generated using $\alpha = 0.01$. In particular, the speed-up is around $29\times$ with respect to HERESY running on the CPU and around $5\times$ with respect to brsim. In the case of a higher value of α (i.e., when the reactions involve a larger number of entities), the speed-up values achieved are reduced due to the higher number of global memory accesses required, which affects the overall execution time on the GPU. For instance, in the case of a reaction system of size 2000×2000, created using $\alpha = 0.1$, the speed-up with respect to brsim is reduced to

Table 1. Comparison of the computational performance between brsim, and the CPU and GPU implementations of HERESY, for randomly generated reaction systems of size $|S| \times |A| \in \{10 \times 10, 100 \times 100, 1000 \times 1000, 2000 \times 2000\}$, with $\alpha \in \{0.01, 0.05, 0.1\}$. The table reports the average running times and standard deviations (given in parentheses) computed over 30 runs.

	Running time [s]			**Speed-up**	
Reaction system size	**brsim**	**CPU**	**GPU**	**brsim / GPU**	**CPU / GPU**
		$\alpha = 0.01$			
10×10	0.02	0.21	0.26	0.07×	0.81×
	(0.01)	(0.03)	(0.02)		
100×100	0.22	1.38	0.26	0.85×	5.31×
	(0.04)	(0.09)	(0.02)		
1000×1000	1.38	8.64	0.43	3.21×	20.09×
	(0.04)	(0.34)	(0.03)		
2000×2000	3.22	18.02	0.62	5.19×	29.06×
	(0.12)	(0.67)	(0.03)		
		$\alpha = 0.05$			
10×10	0.02	0.20	0.26	0.08×	0.77×
	(0.01)	(0.03)	(0.02)		
100×100	0.13	0.86	0.27	0.48×	3.19×
	(0.03)	(0.03)	(0.02)		
1000×1000	1.47	8.63	0.57	2.58×	15.14×
	(0.05)	(0.29)	(0.03)		
2000×2000	3.50	18.11	1.27	2.76×	14.26×
	(0.06)	(0.69)	(0.03)		
		$\alpha = 0.10$			
10×10	0.02	0.19	0.26	0.08×	0.73×
	(0.01)	(0.03)	(0.02)		
100×100	0.13	0.84	0.29	0.45×	2.90×
	(0.03)	(0.04)	(0.02)		
1000×1000	1.51	8.86	0.78	1.94×	11.36×
	(0.04)	(0.45)	(0.04)		
2000×2000	4.01	18.01	2.05	1.96×	8.79×
	(0.07)	(0.63)	(0.05)		

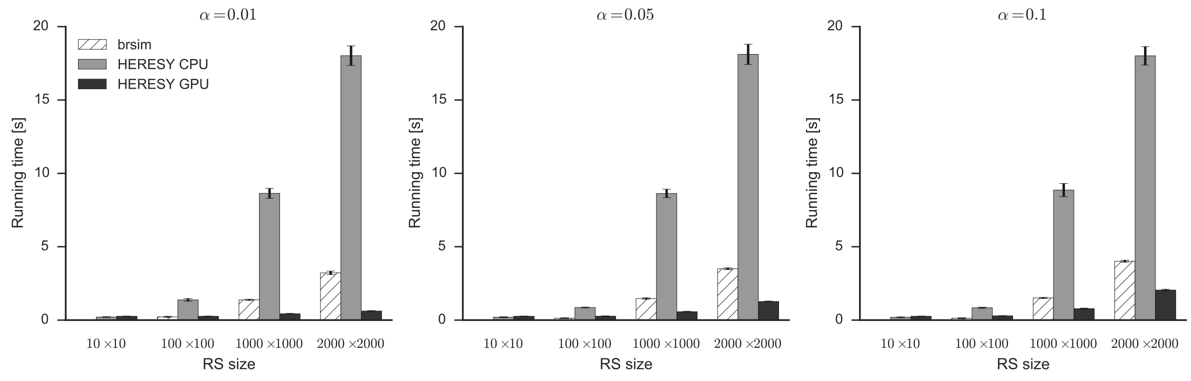


Figure 4. Comparison of the average running time of brsim (hatched bars), HERESY on the CPU (gray bars), and HERESY leveraging the GPU (black bars) for the simulation of randomly generated synthetic reaction systems of increasing complexity. The average was calculated over 30 runs.

approximately $2\times$.

It is worth noting that, when the size of the reaction system is small (e.g., 10×10), the execution on the GPU can be slower than the CPU due to the different clock frequency (1417 MHz and 2.5 GHz for the GPU and the CPU employed in this work). For this reason, HERESY automatically switches on the GPU when the reaction system is characterized by more than 100 reactions. This setting can be changed under HERESY's options menu.

4.2. Simulation of the ErbB system modeled as a reaction system

In order to assess its performance on a *large-scale* biological system, we tested HERESY to simulate a model of the ErbB receptor signal transduction in human mammary epithelial cells [19]. The model comprises all members of the ErbB family of receptor tyrosine kinases, including individually phosphorylated receptor forms, as well as numerous other receptors, signaling and stress response pathways. The ErbB receptors play an essential physiological role in the development and maintenance of epithelial tissues by regulating cell survival, proliferation and differentiation in response to specific ligands and signaling pathways [20]. Pathological behaviors of ErbB receptors are known to be linked to the onset and progression of several human tumors. The analysis of the signaling pathways under control of ErbB receptors has proven particularly challenging due to the cross-talk phenomena existing among the family members, and to the presence of multiple downstream effector pathways and complex regulatory mechanisms, which include several positive and negative feedback loops [28].

The original model of the ErbB receptors presented in [19] consists in a graph with 245 nodes and 1100 arcs that represent, respectively, heterogeneous cellular components and the biochemical regulations existing among them. This model was defined according to a Boolean logic formalism, and was then transformed into an equivalent reaction system according to the method described in [11, 15]. The reaction system of the ErbB model – which can be downloaded from [3] – consists in 6720 reactions involving 246 entities.

In order to evaluate the effectiveness of the GPU-powered tool, we compared the running time required by the CPU and the GPU for the simulation of a context sequence of length 1000, created as described in [19]. The average running time of brsim was 4.28 ± 0.14 seconds, while the running time of HERESY executed on the CPU was 10.25 ± 0.19 seconds (computed over 30 runs). On the contrary, the time required for the simulation of ErbB reaction system was 0.39 ± 0.02 seconds by running HERESY on the GPU GeForce Titan X, resulting in a $10.97\times$ and $26.28\times$ speed-up with respect to brsim and HERESY executed on the CPU, respectively.

5. Conclusions

Reaction systems are typically analyzed by deriving by hand the dynamic process defined by the set of reactions and a given context sequence. In order to automate this procedure, in this paper we introduced HERESY [3], an open-source and cross-platform tool for the simulation of reaction systems. HERESY supports the automatic offload of the calculations to the GPU in the case of the simulation of large-scale reaction systems (characterized by hundreds or thousands of reactions), in order to reduce the computational time otherwise required. According to our tests, HERESY allows to reduce the simulation time with respect to a pure Python implementation up to $29\times$ for randomly generated reaction systems consisting of two thousands reactions and entities.

We envisage that this GPU-accelerated simulator may prove its usefulness for two main purposes. On the one hand, it can be exploited to efficiently simulate the dynamics of any kind of real or theoretical system formalized as a reaction system, especially for those applications where the interaction between the system and the environment (i.e., the presence of a context sequence) plays a relevant role. On the other hand, the availability of an automatic simulator could become profitable also to check the correctness of proofs and theoretical results, mainly in reaction systems consisting of many reactions, or to quickly verify the existence of peculiar behaviors (e.g., reachability of attractor states) starting from different initial states or using different context sequences.

As a future extension of this work, we plan to investigate the standardization of any arbitrary reaction system into a normal form, in which each reaction consists of one reactant, one inhibitor, and one product [23]. In principle, this normalized or standard form should help to reduce GPU warps' divergence due to conditional branches, which forces the serialization of threads' execution and affects the overall performances of the simulator. To take advantage of the standard form of reaction systems described above, HERESY will implement novel and highly optimized kernels, tailored on the simulation of such kind of reaction systems, which is supposed to execute warps in a fully SIMD fashion, possibly providing better performances than the simulation of non-standard reaction systems.

Finally, we are planning a porting of the CPU-bound simulator to the C++ language, in order to have an optimized and compiled simulator able to fill the performance gap with respect to brsim.

References

[1] brsim Web Interface, `http://combio.abo.fi/research/reaction-systems/reaction-system-simulator/`.

[2] brsim GitHub Repository, `https://github.com/scolobb/brsim/`, 2014.

[3] HERESY GitHub Repository, `https://github.com/aresio/HERESY/`, 2017.

[4] Arbib, M. A., Ed.: *Handbook of Brain Theory and Neural Networks*, The MIT Press, 1995.

[5] Azimi, S., Gratie, C., Ivanov, S., Manzoni, L., Petre, I., Porreca, A. E.: Complexity of model checking for reaction systems, *Theoretical Computer Science*, **623**, 2016, 103–113.

[6] Azimi, S., Gratie, C., Ivanov, S., Petre, I.: Dependency graphs and mass conservation in reaction systems, *Theoretical Computer Science*, **598**, 2015, 23–39.

[7] Brijder, R., Ehrenfeucht, A., Main, M., Rozenberg, G.: A tour of reaction systems, *International Journal of Foundations of Computer Science*, **22**(7), 2011, 1499–1517.

[8] Brijder, R., Ehrenfeucht, A., Rozenberg, G.: A note on causalities in reaction systems, *International Colloquium on Graph and Model Transformation On the occasion of the 65th birthday of Hartmut Ehrig (GraMoT 2010)* (C. Ermel, H. Ehrig, F. Orejas, G. Taentzer, Eds.), 30, ECEASST, 2010.

[9] Brijder, R., Ehrenfeucht, A., Rozenberg, G.: Reaction systems with duration, in: *Computation, Cooperation, and Life* (A. Kelemen, J. Kelemenová, Eds.), vol. 6610 of *Lecture Notes in Computer Science*, Springer-Verlag, 2011, 191–202.

[10] Calude, C., Păun, G.: *Computing with Cells and Atoms: An Introduction to Quantum, DNA and Membrane Computing*, CRC Press, 2000.

[11] Corolli, L., Maj, C., Marini, F., Besozzi, D., Mauri, G.: An excursion in reaction systems: From computer science to biology, *Theoretical Computer Science*, **454**, 2012, 95–108.

[12] Ehrenfeucht, A., Main, M., Rozenberg, G.: Combinatorics of life and death for reaction systems, *International Journal of Foundations of Computer Science*, **21**(3), 2010, 345–356.

[13] Ehrenfeucht, A., Main, M., Rozenberg, G.: Functions defined by reaction systems, *International Journal of Foundations of Computer Science*, **22**(1), 2011, 167–178.

[14] Ehrenfeucht, A., Rozenberg, G.: Events and modules in reaction systems, *Theoretical Computer Science*, **376**, 2007, 3–16.

[15] Ehrenfeucht, A., Rozenberg, G.: Reaction systems, *Fundamenta Informaticae*, **75**, 2007, 263–280.

[16] Ehrenfeucht, A., Rozenberg, G.: Introducing time in reaction systems, *Theoretical Computer Science*, **410**, 2009, 310–322.

[17] Formenti, E., Manzoni, L., Porreca, A. E.: Cycles and global attractors of reaction systems, in: *Descriptional Complexity of Formal Systems, 16th International Workshop, DCFS 2014* (H. Jürgensen, J. Karhumäki, A. Okhotin, Eds.), vol. 8614 of *Lecture Notes in Computer Science*, Springer, 2014, 114–125.

[18] Formenti, E., Manzoni, L., Porreca, A. E.: Fixed points and attractors of reaction systems, in: *Language, Life, Limits, 10th Conference on Computability in Europe, CiE 2014* (A. Beckmann, E. Csuhaj-Varjú, K. Meer, Eds.), vol. 8493 of *Lecture Notes in Computer Science*, Springer, 2014, 194–203.

[19] Helikar, T., Kochi, N., Kowal, B., Dimri, M., Naramura, M., Raja, S. M., Band, V., Band, H., Rogers, J. A.: A comprehensive, multi-scale dynamical model of ErbB receptor signal transduction in human mammary epithelial cells, *PLoS ONE*, **8**(4), 2013, e61757.

[20] Hynes, N. E., Lane, H. A.: ERBB receptors and cancer: the complexity of targeted inhibitors, *Nature Reviews Cancer*, **5**(5), 2005, 341–354.

[21] Jong, K. D.: *Evolutionary Computation: A Unified Approach*, The MIT Press, 2006.

[22] Kleijn, J., Koutny, M., Rozenberg, G.: Modelling reaction systems with Petri nets, *Proceedings of the International Workshop on Biological Processes & Petri Nets (BioPPN-2011)* (M. Heiner, H. Matsuno, Eds.), 724, CEUR Workshop Proceedings, 2011.

[23] Manzoni, L., Poças, D., Porreca, A. E.: Simple reaction systems and their classification, *International Journal of Foundations of Computer Science*, **25**(4), 2014, 441–457.

[24] Męski, A., Penczek, W., Rozenberg, G.: Model checking temporal properties of reaction systems, *Information Sciences*, **313**, 2015, 22–42.

[25] Nobile, M. S., Cazzaniga, P., Tangherloni, A., Besozzi, D.: Graphics processing units in bioinformatics, computational biology and systems biology, *Briefings in Bioinformatics*, **-**(-), 2016, –.

[26] Păun, G., Rozenberg, G., Salomaa, A., Eds.: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2009.

[27] Wolfram, S.: *A New Kind of Science*, Wolfram Media, 2002.

[28] Yarden, Y., Sliwkowski, M. X.: Untangling the ErbB signalling network, *Nature Reviews Molecular Cell Biology*, **2**(2), 2001, 127–137.